Abstract. Reengineering a legacy product line has been addressed very little by current product line research activities. This paper introduces a method to investigate feature dependencies and interactions, which restricts the variants that can be derived from the legacy product line assets. Reorganizing the product line assets with respect to new requirements requires more knowledge than what is easily provided by the classical feature-modeling approaches. Hence, adding all the feature dependencies and interactions into the feature tree results in unreadable and unmanageable feature models that fail to achieve their original goals.

We therefore propose two complementary views to represent the feature model. One view shows the hierarchical refinement of features similar to common feature-modeling approaches in a feature tree. The second view describes what kind of dependencies and interactions there are between various features. We show two examples of feature dependencies and interactions in the context of an engine-control software product line, and we demonstrate how our approach helps to define correct product configurations from product line variants.

1 Introduction

Automotive products are rich with variants to meet the special needs of different customers. Applying the Product Line Approach (PLA) for the embedded systems in automotive products is therefore quite natural. A product line is an important system development paradigm in the automotive industry to amortize costs beyond a single product. The paradigm is well established in the mechanical and electrical engineering practice in automotive companies like Bosch. To make the car more secure, economical, clean, and comfortable, more functionality is moving from mechanical to electrical, and from electrical to software solutions. Therefore, today’s automotive products are software-intensive systems that are developed with the PLA paradigm [6,9,17,29,31].

Feature modeling provides an industrial strength method for managing the inherent variability of product lines. The variability modeling is often done using two techniques:
tables and graphical models. In this paper, we concentrate on graphical modeling using feature trees and interaction diagrams. Based on our experience, graphical representation allows the easy understanding of complex structures and their dependencies, which is especially important in the reengineering context. On the other hand, using tables does not require any special tools, and it provides easy processing of information by automatic tools. In fact, these methods are complementary. We often model the first version of variability by listing all entries into a table. Later, we collect the variability information into a feature tree and use the graphical representation as a basis for creating the dependency and interaction models. Structuring features in trees allows to model requirements in different granularities. In practice, it is difficult to model features with all their implications and dependencies with current feature-oriented methods. In the real-life context, relations between features often become very complex without a clear way to model features with different dimensions or aspects, leading to a very complex graph of features. These graphs have nodes that represent the features and different types of edges between the nodes where the edges connect nodes in other parts of the feature hierarchy. The feature-oriented domain analysis (FODA) method \cite{19} and successors of FODA \cite{7} require a tree-like structure of the feature model which may be very difficult to generate, as there are no clear criteria to align the structure to a tree-like form. Moreover, you cannot capture all of the information contained in a (general) graph in a tree, leaving you with the possibility of omitting dependencies or losing control of the model\cite{17}.

Feature dependencies and interactions play an important part in complex software-intensive systems, since features often need other features to fulfill their tasks. There are feature dependencies, which are imposed by the domain, and other dependencies that exist due only to a certain design decision made during development or to the way that features are realized in the solution. Categorizing types of interactions by distinguishing all possible feature dependencies in a domain greatly helps to reengineer product line assets. This is because some domain-enforced dependencies among features are difficult if not impossible to change while dependencies that are based on design decisions can often be removed during the reengineering process. The feature-dependency analysis is a crucial part of the product line development, where the features are used to distinguish between product line variants. In this context, feature interaction can have very strong implications on the possible configurations of product line members. Knowing and managing the dependencies becomes very important, especially in highly configurable product lines.

We suggest different views on the feature model like architectural views on the architecture to represent the desired information. In this paper, we show two examples of feature views. One view captures the FODA-like concept of hierarchical refinement of features and their variability assumptions. In the second view, we present (static and dynamic) feature interactions and dependencies. This dependency and interaction view was used to reengineer features from legacy specifications and implementations of a product line.

The next section of this paper reviews current feature-modeling methods and motivates our extension, which is described in detail in Section\ref{sec:extension}. In Section\ref{sec:examples} we apply our method to two examples. Our findings are related to other research in this area as