A Formal Approach to Software Product Families

Martin Leucker and Daniel Thoma
Institute for Software Engineering and Programming Languages
Universität zu Lübeck

Abstract. Software product line engineering deals with the combined development of a family of similar software systems. These systems provide a similar set of features and should therefore share a large number of common components. We study the user perspective of features and the engineering perspective of components and present a formal notion of features, component-based product families and their interaction. We then demonstrate using Milner’s CCS how our formalism can be applied to extend an arbitrary modelling formalism with support for product lines. To verify that certain products indeed realize certain features, we propose $\mu$-calculus model-checking for multi-valued Kripke-structures. The model checking result in that case no longer is a simple truth-value, but a set of products, conforming to a certain property.

1 Introduction

The vast majority of electronic devices with which we interact is mainly controlled by software—in fact, software-intensive systems pervade our daily life. Typically, not only a single software-intensive system is constructed but rather a family of similar systems that share certain commonalities. Prominent examples of such families of software-intensive systems can be found in a multitude of different application domains, comprising embedded as well as business information systems. For example the model variants of the same model series of a car manufacturer, e.g. the variants of the $\textit{7}$-series $\textit{BMW}$, or the various variants of an operating system, e.g. the various editions of the operating system $\textit{Microsoft Windows 7}$, constitute such families. Typical commonalities for such systems can be found for example in their (conceptual) functionality, their architectural component structure, or code. To enhance the efficiency of the software development and maintenance process, the integrated development of a family of software-intensive systems by explicitly making use of (reusing) their commonalities in a strategic and planned way seems a promising approach. This is the subject of software product family engineering.

Despite its obvious motivation, the way of constructing a family of systems by taking advantage of commonalities is not sufficiently explored—in particular with respect to its theoretical foundation. How can reuse based on commonalities between system variants take place in a systematic way? What are the fundamental concepts behind commonalities and differences of related systems, and
how can we formally represent them? How can commonalities between family members be determined and even schematically computed? How can the relation between family members be modelled, and how are commonalities integrated into the construction of the individual family members? How can we verify correctness properties of a whole software product family instead of looking at the properties of each family member individually?

In this paper we address these questions from a formal point of view and provide an axiomatization of product family concepts using the language of algebraic specification [Wir90]. The axiomatization formalizes the key characteristics of any software product family, where the concept of commonality and the ability to compute the commonalities of an arbitrary subset of family members is the most important aspect for us.

The formal specification may be used as a guidance when defining explicit formalisms supporting the concept of software product families. In this paper, we recall (and slightly simplify) the account of [GLS08] which extends Milner’s CCS by a variant operator yielding the product-line aware calculus PL-CCS. With the help of the specification, we can check that PL-CCS is indeed a reasonable product family extension of CCS.

Finally, to make this overview paper self-contained, we recall the model checking approach for PL-CCS that allows to check a whole family of systems with respect to $\mu$-calculus specifications.

2 Related Work

Most of the related approaches which deal with modelling of software product families are found in the area of Feature Oriented Software Development (FOSD) [CE00]. FOSD deals with the construction of variable software systems. A common specification technique for software product lines in FOSD are so-called feature models [KHNP90]. Feature models are used to model optional, mandatory and variable features, and in particular their dependencies. In that way a feature model allows to restrict the set of possible configurations of a product line, but in general it does not incorporate the information of how to construct the family members, nor does it allow to compute common parts of a given subset of family members. Thus, a feature model serves the same purpose as our dependency model, but does not represent a product family in our sense, i.e. as a construction blueprint that shows how the family members can actually be constructed from the common and variable parts, or how the members are related with respect to reusing common parts. Moreover, feature models usually lack a precise semantics which impedes to reason about features or feature combinations using formal methods.

To make these issues more precise, we recall the concept of features in the next section.

Regarding the algebraic treatment of software product families, there are some approaches which also unify common concepts, techniques and methods of feature-oriented approaches by providing an abstract, common, formal basis.