1 Introduction

In a software product family, a set of products that share similarities but also differ from each other are derived from reusable assets [2, 6]. The ability to differentiate is referred to as variability; different variants are obtained by binding variability in different ways. As a means of managing variability in software product families, several modelling methods have emerged [5, 6, 3]. When variability is expressed rigorously, for example with models with well-defined syntax and semantics, efficient tool support can be developed. Besides hiding concrete syntax behind a graphical user interface, tools can prevent errors and give guidance to the user.

We describe a tool set called Kumbang Tools for managing variability in software product families. Currently, two tools constitute Kumbang Tools. Kumbang Modeller enables creating models of software product family variability from a feature and architecture point of view, whereas Kumbang Configurator enables deriving configurations of individual products by binding variability in the model. Thus, Kumbang Tools support seamless and integrated domain engineering and application engineering activities.

Kumbang Tools rely on capturing product family variability and derived products using features and components with compositional structure and attributes, the interfaces of components and connections between them, and constraints. These concepts are captured in Kumbang ontology with rigorously-defined semantics [1]. That is, Kumbang ontology provides a meta-model for Kumbang models and configurations. Kumbang ontology synthesises previous approaches to modelling variability in software product families as well as in configurable, non-software products. Consequently, Kumbang Tools operate on an integrated conceptual foundation for features, architectural elements, and relations between them.

Kumbang Tools are especially beneficial for software product families with a large amount of variability and complex dependencies. This is because Kumbang Tools check models and configurations automatically and efficiently to prevent errors. Nevertheless, Kumbang Tools are equally applicable for small or simple software product families.

2 Kumbang Tools

Kumbang Modeller (Figure 1) enables creating and modifying Kumbang models to capture product family variability. The user can specify product family features, architectural elements, and relations between them using constraints. Kumbang Modeller hides complexity of concrete syntax behind a graphical user interface and guides the user in the modelling task. Kumbang Modeller checks the model for syntactic correctness. Further, it checks that at least one valid product configuration can be derived from the model. That is, it checks that all required interfaces can be connected to corresponding interfaces, all constraints can be satisfied, and there are no cyclic loops in inheritance or part structures.

Kumbang Configurator (Figure 2) enables deriving configurations of product individuals by binding variability in Kumbang models. Kumbang Configurator is model-independent in the sense that all logic and UI content is based on the model given as input. Every time the configuration is modified, and when a new configuration is initialised, Kumbang Configurator checks the configuration for consistency, completeness, and consequences. A consistent configuration is such that no rules of the Kumbang model are violated, whereas a complete configuration is such that all mandatory selections are made. Finding consequences means that the tool can automatically add selections implied by previous selections and identify selections conflicting with previous selections. In a typical usage scenario, the user selects desired features, and the tool finds the corresponding architecture. However, Kumbang Configurator does not assume that variability is bound in a certain order. Thus, selections concerning architectural elements are reflected back to features.

Checking models and configurations is computationally expensive. In the worst case, checking requires at least an exponential amount of time in relation to the size of
the problem. Therefore, checking is implemented using an efficient smodels inference engine [7], which is a general-purpose inference tool based on the stable model semantics of logic programs. In order to use this inference engine, Kumbang Tools translate models to a form understood by smodels, that is, to programs written in Weight Constraint Rule Language (WCRL) [7].

Kumbang Modeller and Kumbang Configurator are implemented as Eclipse plug-ins using Java programming language. Thus, they can be integrated with other development tools around the Eclipse platform. Kumbang Tools have been released as open source under a GPL license.

Kumbang Tools have been applied to several case studies: a weather station network loosely based on a real-life case; a part of the car periphery system from Robert Bosch GmbH; and a map viewer application for the Nokia 800 mobile device. These cases have been modelled and consequently configured using Kumbang Tools. Further, we are currently conducting a case study within a company called Vaisala in which Kumbang Tools will be applied to an industrial product family.

3 Future Work

Firstly, we are planning to conduct further empirical tests on Kumbang Tools. Kumbang Modeller was tested in lightweight formative usability tests. We are planning more thorough usability tests, which would also cover other Kumbang Tools. Further, we are planning more thorough empirical tests on the scalability of Kumbang Tools. For this, we need to generate models of different sizes and then test the response time of model and configuration checking. Finally, Kumbang Tools need to be tested in practice, for which we are carrying out an initial case study.

Secondly, Kumbang Tools currently address domain and application engineering as well as features and architecture. However, the coverage can be extended. In order to support the full spectrum from requirements to code, Kumbang Modeller should provide a way to map architectural elements to code artifacts, and then Kumbang Configurator could be used for finding the code artifacts constituting one product instance.

Finally, besides studying the product family from a feature and component viewpoint, we seek to add further architectural viewpoints [4]. In particular, we are interested in incorporating behavioural viewpoints to a software product family architecture.

References