Feature and Meta-Models in Clafer:
Mixed, Specialized, and Coupled

Kacper Bąk\textsuperscript{1}, Krzysztof Czarnecki\textsuperscript{1}, and Andrzej Wąsowski\textsuperscript{2}

\textsuperscript{1} Generative Software Development Lab, University of Waterloo, Canada
\textsuperscript{2} IT University of Copenhagen, Denmark
\{kbak,kczarnec\}@gsd.uwaterloo.ca
wasowski@itu.dk

Abstract. We present \textit{Clafer}, a meta-modeling language with first-class support for feature modeling. We designed Clafer as a concise notation for meta-models, feature models, mixtures of meta- and feature models (such as components with options), and models that couple feature models and meta-models via constraints (such as mapping feature configurations to component configurations or model templates). Clafer also allows arranging models into multiple specialization and extension layers via constraints and inheritance. We identify four key mechanisms allowing a meta-modeling language to express feature models concisely and show that Clafer meets its design objectives using a sample product line. We evaluated Clafer and how it lends itself to analysis on sample feature models, meta-models, and model templates of an E-Commerce platform.

1 Introduction

Both feature and meta-modeling have been used in software product line engineering to model variability. Feature models are tree-like menus of mostly Boolean—but sometimes also integer and string—configuration options, augmented with cross-tree constraints [22]. These models are typically used to show the variation of \textit{user-relevant} characteristics of products within a product line. In contrast, meta-models, as supported by the Meta Object Facility (MOF) [28], represent concepts of—possibly domain-specific—modeling languages, used to represent more detailed aspects such as behavioral or architectural specification. For example, meta-models are often used to represent the components and connectors of \textit{product line architectures} and the valid ways to connect them. The nature of variability expressed by each type of models is different: feature models capture simple selections from predefined (mostly Boolean) choices within a fixed (tree) structure; and meta-models support making new structures by creating multiple instances of classes and connecting them via object references.

Over the last eight years, the distinction between feature models and meta-models (represented as class models) has been blurred somewhat in the literature due to 1) feature modeling extensions, such as \textit{cardinality-based feature modeling} [15, 4], or 2) attempts to express feature models as class models in Unified Modeling Language (UML) [11, 16]; note that MOF is essentially the class modeling subset of UML. A key driver behind these developments has been the
desire to express components and configuration options in a single notation [14]. Cardinality-based feature modeling achieves this by extending feature models with multiple instantiation and references. Class modeling, which natively supports multiple instantiation and references, enables feature modeling by a stylized use of composition and the profiling mechanisms of MOF or UML.

Both developments have notable drawbacks, however. An important advantage of feature modeling as originally defined by Kang et al. [22] is its simplicity; several respondents to a recent survey confirmed this view [23]. Extending feature modeling with multiple instantiation and references diminishes this advantage by introducing additional complexity. Further, models that contain significant amounts of multiply-instantiatable features and references can be hardly called feature models in the original sense; they are more of class models. On the other hand, whereas the model parts requiring multiple instantiation and references are naturally expressed as class models, the parts that have feature-modeling nature cannot be expressed elegantly in class models, but only clumsily simulated using composition hierarchy and certain modeling patterns.

We present Clafer (class, feature, reference), a meta-modeling language with first-class support for feature modeling. The language was designed to naturally express meta-models, feature models, mixtures of meta- and feature models (such as components with options), and models that couple feature models with meta-models and their instances via constraints (such as mapping feature configurations to component configurations or to model templates [13]). Clafer also allows arranging models into multiple specialization and extension layers via constraints and inheritance, which we illustrate using a sample product line.

We developed a translator from Clafer to Alloy [19], a class modeling language with a modern constraint notation. The translator gives Clafer precise translational semantics and enables model analyses using Alloy Analyzer. Different strategies are applied for distinct model classes. They all preserve meaning of the models, but speed up analysis by exploiting the Alloy constructions.

We evaluate Clafer analytically and experimentally. The analytic evaluation argues that Clafer meets its design objectives. It identifies four key mechanisms allowing a meta-modeling language to express feature models concisely. The experimental evaluation shows that a wide range of realistic feature models, meta-models, and model templates can be expressed in Clafer and that useful analyses can be run on them within seconds. Many useful analyses such as consistency checks, element liveness, configuration completion, and reasoning on model edits can be reduced to instance finding by combinatorial solvers [7, 9, 12]; thus, we use instance finding and element liveness as representatives of such analyses.

The paper is organized as follows. We introduce our running example in Sect. 2. We discuss the challenges of representing the example using either only class modeling or only feature modeling and define a set of design objectives for Clafer in Sect. 3. We then present Clafer in Sect. 4 and demonstrate that it satisfies these objectives. We evaluate the language analytically and experimentally in Sect. 5. We conclude in Sect. 7, after having compared Clafer with related work in Sect. 6.