The CVM Framework —
A Prototype Tool for Compositional Variability Management

(Tool Presentation)

Andreas Abele (Continental AG, DE), Rolf Johansson (Mentor Graphics, HU), Henrik Lönn (Volvo Technology, SE), Yiannis Papadopoulos (University of Hull, UK), Mark-Oliver Reiser (Technische Universität Berlin, DE), David Servat (Commissariat à l’Energie Atomique, FR), Martin Törgren (Kungliga Tekniska Högskolan, SE) and Matthias Weber (Carmeq GmbH, DE)

Abstract—This article announces the first public release of an experimental research tool for variability management, called “CVM framework” and provides an overview of the tool’s capabilities and architecture.

I. INTRODUCTION

Over the past few years, an experimental variability management tool was developed at Technische Universität Berlin, called “CVM framework”. It was mainly intended for the evaluation of several research approaches developed in a number of industrial cooperations, esp. with Daimler AG and Carmeq GmbH / Volkswagen AG and the European research project ATESST2 (the name “CVM” originated from one of these approaches called “compositional variability management” [12]).

This paper announces the first public release of the tool [4]. We provide an overview of the tool’s capabilities (Section II) and architecture (Section III) and briefly report on recent applications (Section IV) before concluding with a discussion of related work and an outlook.

II. KEY CAPABILITIES OF THE FRAMEWORK

The key functionality of the tool can be divided into the four areas of (a) basic feature modeling and configuration, (b) multi-level feature modeling, (c) configuration links, and (d) configuration graphs.

A. Feature Modeling & Feature Configuration

The CVM framework is heavily based on feature modeling [9], [5]. It was an important design goal to not reinvent the wheel by defining a novel feature modeling approach, but instead making the tool as compatible as possible to classic feature modeling techniques from the literature. Its main basis was cardinality-based feature modeling of Czarnecki et al. [6], but with several extensions from other authors. We cannot provide a comprehensive list here, but an excellent overview of feature modeling techniques with a multitude of further references can be found in [14]. The editing support for feature modeling comprises:

- essential feature editing capabilities (creating and deleting features, moving features within the feature tree hierarchy, ...)
- optional, mandatory and cloned features
- feature groups
- feature inheritance (often called “feature references” in the literature)
- feature links (for simple dependencies between a single start and a single end feature, such as “needs” and “excludes”)
- feature constraints (for more complex dependencies beyond the expressiveness of feature links)
- parameterized features (sometimes called “feature attributes” in the literature)
- editing of configurations of feature models
- a simple type system for checking the validity of values of parameterized features during configuration
- customizable user attributes for project-specific meta-information (which can be attached to most elements)

Figure 1 shows a simple feature model in both the tool’s explorer-style model editor and the graphical feature diagram editor. For feature diagrams, the implementation follows the common model/diagram pattern: for a single feature model, several feature diagrams may be defined that provide different views on the model. This means that some parts of the model, i.e. some features, may show up in a particular diagram while others do not, and if they appear in more than one diagram, all the diagrams show the same model objects.

As mentioned above, a configurator is provided to support configuration of feature models, i.e. to select and deselect their features and provide values for parameterized features if they are selected. Figure 2 shows this for the feature model from Figure 1. Optional features are presented with a check box (e.g. CruiseControl or Radar). The check box has three states: undecided, included and excluded. For example, feature CruiseControl is currently undecided as indicated by the small question mark in the check box. Optional features that are in a feature group of cardinality [1] are presented with radio buttons to indicate that they are mandatory alternative (e.g. Standard and Adaptive below the cruise control). Cloned features with a maximum cardinality greater than 1 (e.g. Wiper[0..2]) are special in that they cannot be selected or deselected. Instead, instances have to be created for them. To achieve this, the user right-clicks on the cloned feature and selects “Create instance ...” from the context menu. In the example, two instances of Wiper were created: frontWiper and rearWiper. The value of a parameterized feature...
can be set by right-clicking the feature and then selecting “Set Value ...” from the context menu. In the example, Radar was not yet supplied with a value which is indicated by the label “<undefined>”.

B. Multi-Level Feature Modeling

Multi-level feature modeling is an approach to pragmatically manage several related product lines as a global, composite product family without introducing a rigid product line infrastructure on the global level [13]. It is supported by CVM through the following core functionalities:

- defining reference feature models
- propagating features from a referring model to a reference model and vice versa
- finding deviations in a referring model with respect to its reference model and determining the conformance state of a referring model

C. Configuration Links

Configuration links are a concept for defining a relation between two or more feature models with respect to their configuration. In other terms, a configuration link defines how to configure one or more target feature models depending on a given configuration of one or more source feature models. With this information, it is then possible to automatically derive configurations of the target feature models whenever configurations of the source feature models are provided. This concept can be used to manage variability within a complex product line by (a) defining orthogonal views on feature models and by (b) consistently managing the variability in component hierarchies, called compositional variability management, [12], [11].

A configuration link is defined as a set of so-called configuration decisions; each such configuration decision represents a single, conditional rule on how to configure the target feature models, e.g. “if feature USA is selected in the source feature model, then select feature CupHolder in the target feature model”.

Providing prototypical tool support for the evaluation of this approach was the main motivation and driver for implementing the CVM framework tool. Key functionalities for supporting configuration links are:

- essential editing of configuration links and their configuration decisions (creation, deletion, etc.)
- exploring configuration decisions and their impact on the target configuration(s) comfortably, including
special markers that highlight the areas where configuration decisions affect the target configurations
- testing the interaction of several selected configuration decisions or a complete configuration link seamlessly while editing is in progress (“test-drive mode”)
- basic supportive analyses to spot contradictions between the configuration decisions within a single configuration link
- automatic configuration derivation, i.e. application of a configuration link on existing configurations of the link’s source feature models, resulting in one or more target configurations

In addition to manually editing the configuration rule captured in a configuration decision, the tool provides means to conveniently edit this information in a special view, called “Configuration Preview”, as presented in Figure 3. The left side of the configuration preview shows configurations of all source feature models of the configuration link while the right side shows configurations of the target feature models.

This preview has two distinct modes of operation: an editing mode that allows to conveniently edit configuration decisions and a test-drive mode to test the current configuration definition by experimentally configuring the source feature models (left side of the preview) and verifying that the automatically generated configuration of the target feature models (right side of the preview) is correct.

D. Configuration Graphs

Several configuration links can be combined by using the target feature models of a first configuration link as source feature models to a second configuration link, and so on. This way, it is possible to form chains and networks of inter-related feature models, called configuration graphs (or more precisely, directed acyclic graphs, in which nodes represent feature models and the edges are realized by configuration links, [11]). A configuration of such a graph, called a graph configuration, is a set of ordinary feature model configurations, one for each feature model / node in the corresponding graph. The tool supports:

- definition and management of configuration graphs of arbitrary complexity
- support for reuse of configuration graphs with a class-instance concept
- editing and automatically deriving graph configurations

It may seem as if configuration graphs are merely a consecutive application of configuration links. However, sophisticated modeling elements and tool support was required to support a feasible management of such configuration graphs and graph configurations, which is the reason for treating these capabilities in a dedicated section, here.

III. CONSTITUENTS AND ARCHITECTURE

The CVM framework is divided into five main constituents, which are illustrated in Figure 4.

Data Model. The data model for variability management is the core of the entire framework. It provides the functionality to programatically manage variability-related information in memory, for example setting of an object’s values, management of bidirectional associations and containment, event notification. In addition, it provides an XMI import and export.

Variability Specification Language (VSL). A formal language to textually specify all variability-related information supported by CVM. A parser is provided that transforms VSL specifications to instances of the data model. Also the reverse direction, exporting information from the data model to VSL, is supported.

Model Editor. This is the main editor of CVM and provides a means to interactively navigate and manipulate feature models and other variability-related information.

Diagram Editor. Closely connected to the model editor is the diagram editor for visual editing of feature diagrams and configuration graphs.
**Configurator.** A configurator is provided to create and edit configurations of feature models. It supports partial configurations with undecided states and management of entire networks of feature model configurations, called graph configurations (cf. Section II-D).

**VSL Editor.** A text editor with specific support for VSL (e.g. syntax highlighting)

The framework is designed as an Eclipse plug-in. The data model is based on the Eclipse Modeling Framework EMF [3], i.e. an EMOF meta-model of all information entities of CVM was used to generate the Java code of the data model with the EMF code generator. In addition, several manual adaptations and extensions were required. The graphical editing functionality was implemented using the Graphical Editing Framework GEF [7].

As mentioned above, all variability modeling supported by the framework can be performed on a textual level by way of the built-in Variability Specification Language (VSL). Figure 5 shows the feature model from Figure 1 defined as a textual VSL specification. A single VSL file can contain several feature models, configuration links and configuration graphs. As can be seen, the syntax was inspired by programming languages such as Java, but several optimizations were introduced to better fit the application area of variability specification. For example, all names such as those of features may contain special characters or white-space, which is simply achieved by enclosing them in quotation marks (e.g. "Body Electronics System"). The tree hierarchy in a feature model can be specified using a straightforward notation: each feature may be followed by a comma-separated list of child-features in brackets; each child feature may also have a list of children, and so on. Cardinalities can be specified following the feature name (e.g. Wiper[0..2]); the default cardinality is [0..1], i.e. optional. A cardinality without a name denotes a feature group (e.g. in Figure 5 below feature CruiseControl).

**IV. APPLICATIONS**

The CVM framework has been applied in several smaller industrial experiments, mainly at Daimler AG and Carmeq GmbH / Volkswagen AG. In addition, it has been used in lectures and various student projects at Technische Universität Berlin. Its main application, however, is in the European FP7 project ATESST2 [2], where it forms an integral part of the project’s tool platform. ATESST2 is aimed at defining a comprehensive architectural description language for the automotive domain, called EAST-ADL2, comprising artifacts on several abstraction layers, thus providing a seamless transition from early development phases (features and requirements on vehicle level), via intermediate levels (functional analysis architecture and design architecture) down to implementation level.

The variability modeling approach of ATESST2 is heavily based on feature modeling. One or more feature models are used on vehicle level to define the variability of the complete system, i.e. the vehicle, on a high level of abstraction: in the so-called core technical feature model the system’s global variability is defined with a technical perspective; other vehicle-level feature models can be added to realize orthogonal views on technical variability, such as a marketing-driven packaging of variability for immediate end-customer configuration. Configuration links are used to realize the mapping of these orthogonal views onto the core technical feature model, thus enabling an automatic derivation of a configuration of the core technical feature model from any given configuration of the orthogonal views.

In addition, EAST-ADL2 supports variability within the functional analysis architecture and design architecture. This means the artifacts on these levels can be defined in variable form, mainly by marking individual elements as optional. Configuration graphs, as presented in Section II-D above, can then be used to manage the variability in these artifacts all across the system’s component hierarchy. In the end, CVM’s graph configuration functionality then provides a means to derive an entire system configuration from given configurations of the vehicle level feature model(s).

EAST-ADL2 and CVM are also being applied in the European project HAVEnit [8] to model the architecture of various advanced driver assistance functions, such as an automatic queue assistance (AQuA) and a temporary auto pilot (TAP).
featureModel BodyElectronicsFM {
  "Body Electronics System" {
    CruiseControl {
      [1] {
        Standard,
        Adaptive {
          Radar : float[16.38..*] = 24.0
        }
      }
      Wiper[0..2] {
        Constant[1], // mandatory
        Adaptive ( RainControlled )
      }
    }
    link Radar < excludes > Wiper.Adaptive;
  }
}

Figure 5. A sample VSL specification.

V. RELATED WORK — OR: YET ANOTHER FEATURE EDITOR?

There already exist several tools that provide feature editing functionality, available both as commercial products (e.g. pure::variants [10]) as well as in the form of research prototypes (e.g. CaptainFeature and FMP [1] or XFeature [15]). It therefore only makes sense to come up with a new feature editing tool if the implementation effort is justified by factual necessities.

The primary motivation for implementing the prototypical CVM framework with its own feature editor was the fact, that the feasibility of the underlying concepts, especially of multi-level feature modeling and of configuration links, is closely linked to how they are embedded in the overall feature-editing facility and it was therefore necessary to experiment with different forms of editing functionality. In addition, when building on an existent research demonstrator there would always be the risk that its development is broken off or that the team working on the demonstrator decides to introduce fundamental changes to the tool’s architecture or concepts which could lead to irreconcilable conflicts with assumptions made in the own project. Finally, building basic editors for your own newly introduced domain specific methodologies became a lot easier thanks to latest achievements in model driven development, for example the Eclipse Modeling Framework (EMF), the Graphical Editing Framework (GEF), and associated projects.

VI. DISCUSSION AND OUTLOOK

Though not intended for application in mission-critical projects, the CVM framework as presented in this article may be of interest for use in teaching or other experimental applications; also the diagramming functionalities might prove useful when drawing models for presentations or publications. The tool is available for download from [4]. In the future, we will continue to use the tool in our own projects and will extend and adapt it accordingly.

Acknowledgments. The research leading to the CVM tool as presented in this article was partly supported by a research scholarship of Daimler AG and received funding from the European Community’s 7th Framework Programme under grant agreement no. 224442.

REFERENCES


