FeatureMapper: Mapping Features to Models[*]

Florian Heidenreich
Technische Universität Dresden
Software Technology Group
01062 Dresden, Germany
florian.heidenreich@tu-dresden.de

Jan Kopcsek
Technische Universität Dresden
Software Technology Group
01062 Dresden, Germany
s9730399@mail.inf.tu-dresden.de

Christian Wende
Technische Universität Dresden
Software Technology Group
01062 Dresden, Germany
christian.wende@inf.tu-dresden.de

ABSTRACT

Variability modelling with feature models is one key technique for specifying the problem space of Software Product Lines (SPLs). To allow for the automatic derivation of a concrete product based on a given variant configuration, a mapping between features in the problem space and their realisations in the solution space is required. It is crucial to support the developer in the complex task of defining such mappings. These mappings can also be used to provide visualisations of the variant space that allow to reason over variability in SPLs. In this paper we present *FeatureMapper*, a tool that allows for defining mappings of features to model elements specifying feature realisations. These feature realisations can be defined in arbitrary Ecore-based languages. Furthermore, the tool supports different visualisation techniques that can help developers understand the complex designs of SPLs.

Categories and Subject Descriptors

D.2.2 [**Design Tools and Techniques**]: Computer-aided software engineering (CASE); D.2.2 [**Design Tools and Techniques**]: Object-oriented design methods

General Terms

Design, Languages

Keywords

Software Product Lines, Feature Modelling, Feature Visualisation, Feature Mapping, Model Transformation

1. INTRODUCTION

Variability modelling is used to express common and variable parts within Product Line Engineering (PLE) and to explicitly define constraints between variable parts—so-called *features*. It abstracts from concrete feature realisation through feature models which is a powerful notion to handle the increased complexity in PLE [2, 5]. However, to build concrete products from a product line, features have to be realised using software artefacts shared across the product line. While variability modelling resides in the *problem space*, the realisation of features is part of the *solution space*. To instantiate products from a product line, feature realisations in the solution space have to be included according to the presence of the features in a *variant model*; that is, a concrete selection of features from a feature model.

To support this transition from problem space to solution space in an automated way, a mapping from features to software artefacts that realise the features is needed. Our tool *FeatureMapper* allows for both defining and interpreting such mappings. Furthermore, it supports the developer understand those mappings by providing different visualisation techniques.

The remainder of this paper is structured as follows. Section 2 shortly introduces different ways of mapping features to models and presents our metamodel-based approach to feature mapping. Section 3 presents our tool *FeatureMapper*, its different ways of defining feature mappings and the visualisation techniques provided to support the developer. We conclude in Section 4.

2. MAPPING FEATURES TO MODELS

In our work we aim at bridging the gap between feature models and solution models to enable the usage of feature models within a Model-Driven Software Development (MDSD) [10] process. We designed a metamodel [6] to express a mappings between features from a feature model to their realisation in models defined in arbitrary Ecore-based languages [1].

A mapping model is an instance of our mapping metamodel and contains both links to the features in the feature model and to model elements of one or more solution models. Essentially, it can be created in two different ways.

Manual Mapping A manual mapping is a mapping where the developer manually assigns specific model elements as realisation to one feature or a combination of features in the feature model. A manual mapping is often needed when existing elements from realisation models need to be mapped to features from a feature model. This also includes changes to existing mappings that occur during evolution of the SPL.

Automatic Mapping An automatic mapping is automatically generated while the developer is modelling the realisation of a feature. To this end, all changes to the model are tracked and are mapped to a previously selected feature.

It is obvious that a one-to-one relationship between features and their realisation cannot always be assumed. Different combinations of features may require dedicated solution artefacts. Hence, our mapping metamodel also supports the assignment of logical combinations of features in a mapping. Thus, the mapping of, for example, the conjunction of two features to model artefacts is possible.

[*]This research has been co-funded by the German Ministry of Education and Research (BMBF) within the project feasiPLe.

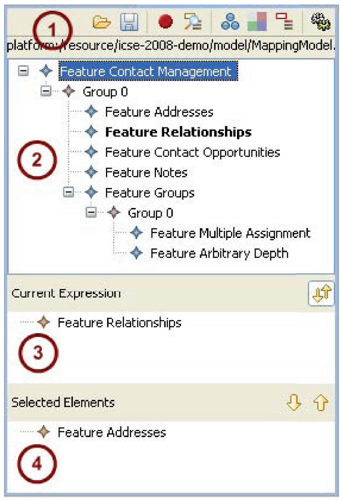

Figure 1: Screenshot of the *FeatureMapper* view.

3. TOOL SUPPORT

Figure 1 shows a screenshot of the *FeatureMapper*. It consists of four parts. The tool bar (1) provides means for loading and saving feature mappings and for different visualisation options. The upper compartment (2) contains the feature model that is associated with the current mapping model. The example describes the variability options in a basic contact management application. Compartment (3) contains the feature or feature combination that is currently active. Compartment (4) contains the feature or feature combination that has already been applied to currently selected model elements of the solution model.

Our tool consists of multiple plug-ins for the Eclipse Platform. It is based on the Eclipse Modelling Framework (EMF) [1] that provides the Ecore metamodelling language which is used to specify the abstract syntax for arbitrary modelling languages. Thus, the modelling of the solution space is not bound to any concrete language and existing EMF-based modelling tools (e.g. TOPCASED [9]) can easily be integrated. For creating feature models, we use the feature metamodel developed by the feasiPLe consortium [4].

In the following subsections we will describe the different means for mapping features to model elements supported by our tool. We will present the provided visualisation techniques and will explain, how the tool can be used to create concrete products of a product line.

Manual Mapping. The manual mapping of features to model elements is straightforward. First, the developer activates the feature from the feature model and thereby changes the current active feature or feature combination. It can now be applied to selected model elements in one or more solution models using the down arrow on the right of the *Selected Elements* compartment.

Automatic Mapping. The automatic mapping is enabled by the recording button from the tool bar. While the tool is in recording mode, all changes to one or more solution models are automatically associated to the current expression. This way, it is also possible to associate changes to properties of model elements—e.g. changing names of association ends or changing cardinalities—to features from the feature model. Changing properties is a very powerful concept that is comparable to the notion of *meta expressions* in [3].

Visualisation. It is crucial that the different mappings are visualised in a way, that the developer understands the participation of specific elements in a feature and can verify the correctness of a given variant of the SPL. Therefore, our tool provides different means for visualisation—filtering and colouring in graphical editors. The *realisation filter* helps at understanding which parts of a model are mapped to a specific feature by greying out all model elements that do not participate in the realisation of the current feature. The *variant filter* shows all model elements that are included in a specific variant of the product line. Colouring specific feature combinations and the corresponding model elements helps at reasoning about feature interactions. Our tool works in a *non-invasive* way with any graphical editor that is based on the Graphical Editing Framework (GEF) [7]. That means, that the graphical editor does not need to be adjusted to work with our tool.

Instantiation. In addition to creating mappings between features and models and visualising them, our tool can interpret those mappings to create models that only include the parts that are needed for a given variant. A transformation component transforms one or more solution models to a concrete variant based on feature selection in the variant model and the information in the mapping model. This transformation can be triggered from our tool and is also available as an openArchitectureWare [8] workflow component.

4. SUMMARY

In this paper we presented our tool *FeatureMapper* that allows for mapping features to model elements that are defined in Ecore-based languages. These mappings are crucial for automatically creating concrete products from a product line. The tool supports both *automatic mappings* and *manual mappings* and offers various means for defining views on the mappings (e.g. filtering and colouring). In our future work we want to evaluate this approach in a real-world case study to analyse usage patterns and improve the tool. We also want to investigate different ways of visualising property value mappings and explore possibilities to extend the approach for supporting arbitrary solution artefacts.

The *FeatureMapper* and more documentation are available at `http://fheidenreich.de/work/fm/`.

5. REFERENCES

[1] F. Budinsky, S. A. Brodsky, and E. Merks. *Eclipse Modeling Framework*. Pearson Education, 2003.

[2] K. Czarnecki. Overview of Generative Software Development. In *Proceedings of the Unconventional Programming Paradigm*, volume 3566 of *LNCS*, pages 326–341. Springer, 2005.

[3] K. Czarnecki and M. Antkiewicz. Mapping Features to Models: A Template Approach Based on Superimposed Variants. In R. Glück and M. Lowry, editors, *Proceedings of the 4th Int'l Conf. on Generative Programming and Component Engineering (GPCE'05)*, volume 3676 of *LNCS*, pages 422–437. Springer, 2005.

[4] feasiPLe Consortium. feasiPLe Research Project, Feb. 2008. URL http://feasiple.de.

[5] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-oriented Domain Analysis (FODA) Feasibility Study. *Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA*, 1990.

[6] J. Kopcsek. Design and implementation of a tool for feature-driven modelling using domain-specific languages, Nov. 2007. Großer Beleg. Technische Universität Dresden (available in German).

[7] The Eclipse Foundation. Graphical Editing Framework, Feb. 2008. URL http://www.eclipse.org/gef/.

[8] The openArchitectureWare Project Team. openArchitectureWare, Feb. 2008. URL http://www.openArchitectureWare.org.

[9] The Topcased Project Team. TOPCASED, Feb. 2008. URL http://www.topcased.org.

[10] M. Völter and T. Stahl. *Model-Driven Software Development*. John Wiley & Sons, June 2006.