# FAMILE: Tool support for evolving model-driven product lines

Thomas Buchmann and Felix Schwägerl

Lehrstuhl Angewandte Informatik 1, University of Bayreuth
D-95440 Bayreuth
*firstname.lastname*@uni-bayreuth.de

**Abstract.** Model-driven development is a well-known practice in modern software engineering. Many tools exist which allow developers to build software in a model-driven way. Unfortunately, these tools do not provide dedicated support for the specific needs in software product line processes. Only recently some approaches tried to combine feature modeling and model-driven development. In this paper we present a new approach that allows for the combination of feature models and Ecore based domain models and keeps both models consistent during evolution.

## 1 Introduction

*Software product line engineering* [1, 2] deals with systematic development of products belonging to a common system family based on organized reuse of software artifacts. Thus, composing products from a library of reusable components, rather than developing each product instance from scratch is preferred. *Model-driven software engineering* [3] puts strong emphasis on the development of higher-level models rather than on the source code. Both techniques promise to increase productivity. In the past, several approaches have been made to combine both techniques to get the best out of both worlds. Only recently, *model-driven software product line engineering* has been established as an integrating discipline. This paper contributes to this discipline by presenting FAMILE (Features and mappings in lucid evolution), a new tool to combine feature models and domain models using an explicit mapping model. Furthermore, textual languages are provided to (a) annotate domain model elements with feature expressions and (b) specify dependencies and (automatically derived) repair actions to ensure the well-formedness of configured domain models.

## 2 Tool support

When developing software product lines in a model-driven way, various models are involved: Features have to be mapped onto corresponding domain model elements realizing them. Figure 1 shows the Ecore-based [4] models and metamodels part of our toolchain. For the *domain model*, arbitrary EMF modeling languages can be employed. In our running example, we use our own UML2

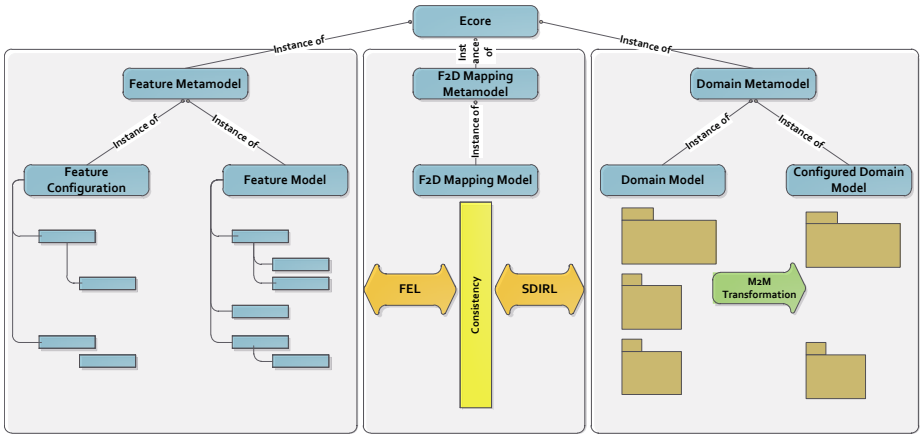


**Fig. 1.** Involved models and meta-models.

based domain modeling tool. The *feature model* [5] consists of a tree of features. A non-leaf feature may be decomposed in two ways: In the case of an AND decomposition, all of its child features have to be selected when the parent is selected. In contrast, for an OR decomposition exactly one child has to be selected. For one feature model, a number of *feature configurations* exist, each defining the selection of features for one product. A mapping model (F2DMM) is used to interconnect both feature model and domain model. The accompanying screencasts (see section 4) demonstrate the mapping editor's use.

In our previous work [6], we used implicit feature annotations and propagation of features to dependent model elements to ensure the well-formedness of configured domain models in case a selected element requires the inclusion of an unselected one. Contrastingly, in our current approach, selection states rather than annotations are propagated. The user can choose between different *propagation strategies*: The *suppress active* strategy propagates the selection state of the required element, resulting in a negative (*suppressed*) selection state of the previously selected context element. Contrastingly, the *enforce inactive* strategy changes the selection state of the unselected required element to *enforced*.

Dependencies are either defined as meta-model specific constraints covered by the SDIRL language based on OCL (see part 2 of the screencast), or generically determined by the Ecore containment hierarchy. Annotations of domain model elements are phrased with our Xtext-based *Feature Expression Language* (FEL) covered in part 3. As soon as a feature configuration is loaded, feature expressions are evaluated and the result is presented to the user: A filled cyan circle indicates that the corresponding model element is directly contained in the configured domain model because its associated annotation evaluates to `true`, whereas orange filled circles mark the exclusion of the respective element. Circles with cyan or orange border depict *enforced* or *suppressed* model elements. Model elements which are not annotated are decorated with a yellow circle. Based upon the user's choice, these elements are included or excluded from each product.

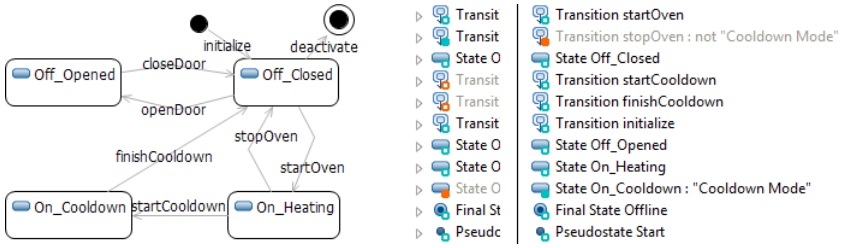


**Fig. 2.** Left: UML 2 state machine diagram from multi-variant input domain model. Right: Excerpts from F2DMM mapping model for two feature configurations.

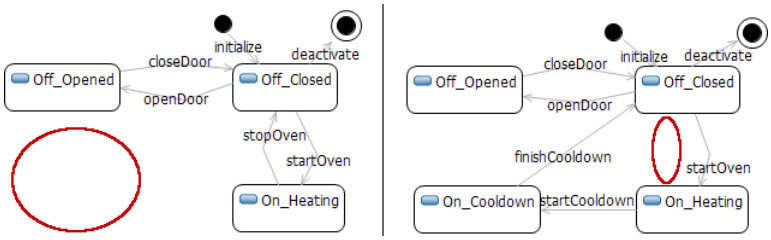


**Fig. 3.** Products derived from mapping using both feature configurations.

Figure 2 shows the annotation of two elements of a UML2 state machine, a state (`On_Cooldown`) and a transition (`stopOven`), positively or negatively associated with the Feature `Cooldown Mode`, which is only included in one of two example configurations. The derived products are visualized in Figure 3: In the left configuration, the adjacent transitions of state `On_Cooldown` are indirectly excluded (suppressed) due to the propagation of `On_Cooldown`'s selection state.

An advanced concept realized by our mapping editor are *alternative mappings*. These pseudo domain model elements are either defined in-place or reference objects located in a different resource. They are merged into the derived product in case the associated feature expression evaluates to true and no conflicts with existing domain model elements occur. This enables the definition of *variation points* even in case of a single-valued structural domain meta-model feature, e.g. a class name (see parts 4 and 6 of the screencast). During the mapping process, the consistency between feature model and configuration is preserved as well as the tree structure of the mapping model which is dependent on the domain model's containment structure except for alternative mapping elements. Changes in feature names may affect feature expressions. In this case, the user is free to accept or deny automatically derived renaming proposals.

## 3 Related work

Due to space restrictions, we limit our comparison to FeatureMapper [7], as it also allows to map features to Ecore-based domain models. For a more detailed comparison of model-driven software product line approaches, the reader is referred to [6]. FeatureMapper [7] is a tool that allows for the mapping of features to Ecore based domain models. Like our approach, it is very general and does

not have any knowledge about the domain meta-model. In contrast to our approach, it provides only basic capabilites of checking well-formedness constraints of the Ecore metamodel [8] and does not provide automatic repair actions. In our previous work [6] we used the UML profile mechanism to implicitly annotate the domain model. Contrastingly to our new approach, the mapping information was persisted within the domain model rather than in a dedicated mapping resource. Automatic feature propagation in a suppress active way was employed, while feature expressions were limited to conjunction of features only.

## 4 Conclusion

In this paper we gave a short overview of our new tool support for the development of evolving model-driven product lines. We extended our previous approach by different directions of automatic feature propagation and feature expressions allowing arbitrary combinations of features. Alternative mappings allow the expression of variation points in domain models. To install our tool in a clean Eclipse Modelling distribution, please make sure to download Xtext 2.2.1, Acceleo 3.2.0, OCL Tools 3.1.2 and ATL 3.2.1 first. The update site can be found at: http://btn1x4.inf.uni-bayreuth.de/famile/update and the accompanying screencast is located at
http://btn1x4.inf.uni-bayreuth.de/famile/screencasts. Please note that the tool distribution also comprises our UML2 based domain modeling tool *Valkyrie*. However, Famile can be used for arbitrary Ecore based domain models.

## References


1. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns, Boston, MA (2001)
2. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer Verlag, Berlin, Germany (2005)
3. Völter, M., Stahl, T., Bettin, J., Haase, A., Helsen, S.: Model-Driven Software Development : Technology, Engineering, Management. John Wiley & Sons (2006)
4. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF Eclipse Modeling Framework. 2 edn. The Eclipse Series, Boston, MA (2009)
5. Batory, D.S.: Feature models, grammars, and propositional formulas. In Obbink, J.H., Pohl, K., eds.: Proceedings of the 9th International Software Product Line Conference (SPLC'05). Volume 3714 of Lecture Notes in Computer Science., Rennes, France, Springer Verlag (September 2005) 7–20
6. Buchmann, T., Westfechtel, B.: Model-driven Engineering of Software Product Lines with Feature Models and Graph Transformations. Software and Systems Modeling (2011) submitted for publication.
7. Heidenreich, F., Kopcsek, J., Wende, C.: FeatureMapper: Mapping features to models. In: Companion Proceedings of the 30th International Conference on Software Engineering (ICSE'08), Leipzig, Germany (May 2008) 943–944
8. Heidenreich, F.: Towards systematic ensuring well-formedness of software product lines. In: In Proceedings of the 1st Workshop on Feature-Oriented Software Development, ACM Press (October 2009)