# Bridging the Gap Between Features and Models

Florian Heidenreich
Technische Universität Dresden
Software Technology Group
01062 Dresden, Germany
florian.heidenreich@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 to specify the problem space of Software Product Lines. To allow for the automatic derivation of a product instance based on a given variant configuration, a mapping between features in the problem space and their realisations in the solution space is required. In this paper we present an approach to define a mapping of features to model fragments specifying the feature realisations. We differentiate collaborative and aspectual features and show how these feature types are supported by a modelling-language independent infrastructure which separates the mapping information and stores it in a dedicated mapping model.

## 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
Feature Modelling, Variability Modelling, Aspect-Oriented Software Development, Product Line Engineering, Domain Specific Languages, Graph Rewriting

## 1. INTRODUCTION
Variability modelling is used to express common and variable parts within Product Line Engineering (PLE) and to explicitly define constraints between different 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.

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 an instance of 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. We differentiate two types of features: features that have realisations connected to specific points in the core and features that add duplicated artefacts scattered across various points in an aspectual manner. An approach to map features to software artefacts has to provide a means for mapping both types of features to their realisation in a concise way.

In this paper we present our work on mapping features of the abovementioned different types from feature models to software artefacts. We provide a feasible approach to bridge the gap between variability modelling and solution modelling that supports features that cross-cut a software system in different ways. This mapping describes how and where specific artefacts are included in the solution models to realise features from feature models.

The rest of the paper is structured as follows. First, we introduce the different feature types and give examples for each of them. In Section 3 we present our approach for mapping features to models which is based on a metamodel for expressing such mappings. The paper concludes with a discussion of related work and a summary.

## 2. FEATURE TYPES
In our work we differentiate two feature types: *collaborative features* and *aspectual features*. The first we call collaborative features, because their realisation describes a self-contained set of artefacts which are woven at predefined places to the core to add a specific increment in collaborative functionality. The latter we call aspectual features due to their quantifying nature [8] where the feature realisation can be applied to the core multiple times based on a pattern over the core. There is no explicit distinction between them when modelling the problem space, but they differ in the means used for their realization in the solution space.

The distinction between different feature types is mainly based on recent research done by Apel et al. [1] where a classification of cross-cutting concerns into *heterogeneous cross-cuts* and *homogeneous cross-cuts* is presented. A heterogeneous cross-cut extends multiple points in the core—each with a different extension. A homogeneous cross-cut extends a core at multiple points by adding the same extension to each of them.

To exemplify our ideas we developed a small example that is based

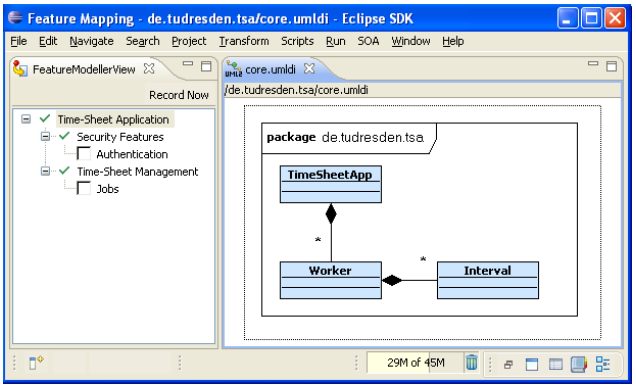


**Figure 1: The core model of the time sheet application**

on a simple time sheet application. A time sheet is used to collect the amount of a worker's time spent on her job. Our initial implementation allows for collecting time intervals for specific workers. Figure 1 depicts a class model for the core application.

A `TimeSheetApp` manages different `Workers` where each of them is associated with `Intervals` that capture the time spent on the `Worker`'s job.

## 2.1 Collaborative Features

Collaborative features have realisations that describe a heterogeneous cross-cut by means of a collaboration. There are specific points in the core that can be localized to connect the artefacts realising a feature. Figure 2 depicts the relationship between feature artefacts and points in the core.

An example of a collaborative feature is the ability to manage different jobs for each worker in our time sheet application as shown in Figure 3. Realisation artefacts are added to specific points of the core, as highlighted by the circles. To realise the *Job* feature the `Job` class is introduced and associated at two specific points with the core artefacts `Worker` and `Interval` to replace their direct composition.

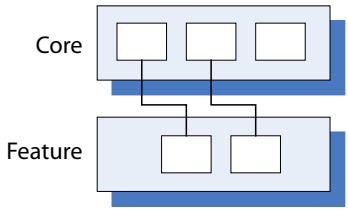


**Figure 2: A collaborative feature that is connected to specific localized points in the core.**

## 2.2 Aspectual Features

Aspectual features are used whenever behaviour or structure needs to be applied to the core multiple times. They describe homogeneous cross-cuts in the sense of Aspect-Oriented Software Development (AOSD) [13]. Normally, the specific points in the core where the feature should be applied are not known at modelling time. Therefore aspect-oriented techniques for quantification of

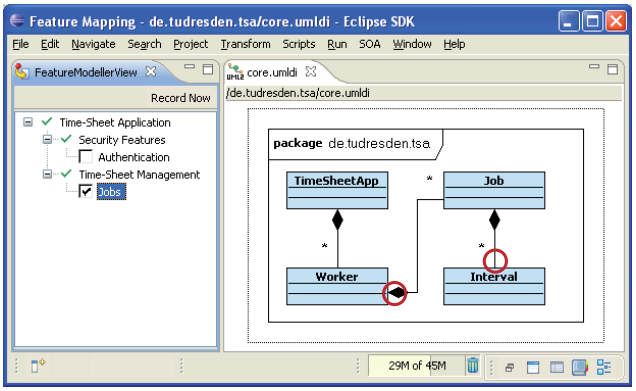


**Figure 3: The core of the time sheet application with the optional feature *Jobs***

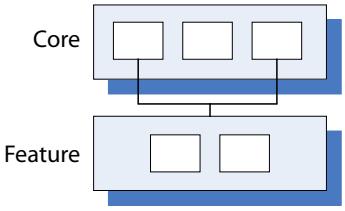


**Figure 4: An aspectual feature that is distributed to various points in the core.**

the aspectual feature are used to apply the feature artefacts at all points in the core that are captured by a pattern that describes the structure of the points of interest (cf. to the notion of *pointcuts* in AOSD). An aspectual feature has a one-to-many relationship to the core, because the whole feature realisation is distributed over the core and connected at possibly multiple points (see Figure 4).

An example aspectual feature is adding authentication to the time sheet application. This requires the worker to login before performing any of the CRUD operations (create, read, update and delete). Since this is difficult to express by static modelling we introduce a state chart that describes the dynamic behaviour of the CRUD operations related to the `Interval` business object. Figure 5 shows the state chart without authentication, where different states and transitions related to interval management of the time sheet application are depicted. The states that change the interval data by CRUD operations can be reached from the *Manage Timesheet* state without authentication. Figure 6 shows the state chart after enabling the aspectual authentication feature. Artefacts added to the core to realise the aspectual feature are shown in red. To realise authentication every transition in the core which is associated with a CRUD operation is augmented by a *Choice* state: The CRUD operation is only executed if the current user has authenticated (`auth == true`), otherwise she is redirected to a `Login` state. We introduce graph-rewrite rules that are used to specify the necessary changes to the core in Section 3.2.

# 3. MAPPING FEATURES TO MODELS

Variability modelling with feature models [5, 12] is used to express the variability of features within a product line. In our work we aim

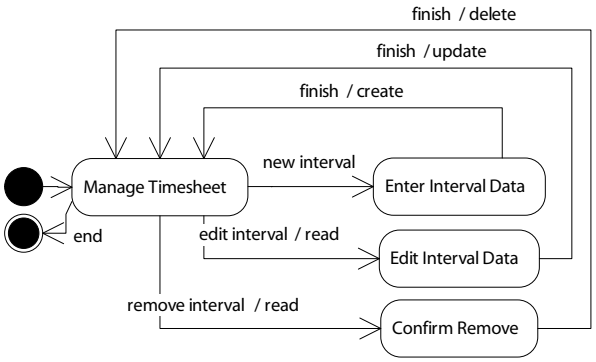


**Figure 5: A state-chart model of the time sheet application.**

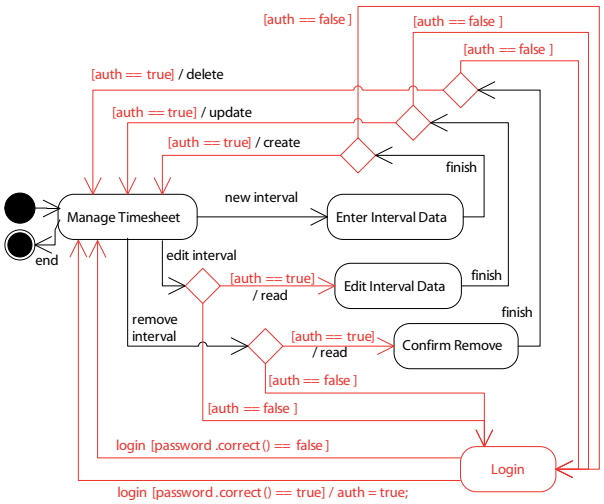


**Figure 6: The state-chart model of the time sheet application with the aspectual feature *Authentification***

at bridging the gap between feature models and solution models to enable the usage of feature models within a Model-Driven Software Development (MDSD) [20] process. We designed a metamodel for expressing generic mappings from features to their realisations that supports both features with heterogeneous cross-cuts and homogeneous cross-cuts. We offer different means to realise the two feature types. This is motivated by previous work that showed that although collaborations can be realised by aspect-oriented techniques it is often not the most convenient way to do so [14].

We developed a plug-in for the Eclipse Platform [17] that supports this mapping. It is based on the Eclipse Modelling Framework (EMF) [4] 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 [18]) can easily be integrated. Our generic mapping model connects features from the problem space—expressed by a feature model based on the feature metamodel developed by the feasiPLe consortium [7]— with changes on artefacts of the solution space. Since these changes reflect the Ecore-based language specifications they preserve the well-formedness of solution space models.

At modelling time the mapping information is used by the plug-in to allow for an interactive adaptation of the solution space in accordance to the feature selection (cf. Figure 1 and Figure 3). Thus, it supports the logical separation of feature realisations from the core as a distinct set of changes applied to a model, but still allows to build views on the combined models. In a further step, the mapping information can be used for automatic product instantiation based on a given variant model.

## 3.1 Mapping Collaborative Features

As described in Section 2, collaborative features affect specific unique points in the core that can be localised at modelling time. Our solution allows to model the artefacts that realise a feature directly onto the core models and provides means to derive the corresponding mapping information automatically. The core models consist of all mandatory features that have to be modelled initially. To realise an optional collaborative feature the user first chooses a specific feature from the feature model and then adapts the core models with all artefacts that are realising the feature. The plug-in provides means to record all changes made to a model while modelling a specific collaborative feature. The change tracking is implemented on top of EMF and is therefore completely language independent. With respect to the relationships found between artefacts of modelling languages (to-one, to-n) we distinguish the following atomic change types in the mapping model:

**Add** An add change adds a set of distinct artefacts of the same type to a specific point that can be bound multiple times with the given artefact type.

**Remove** A remove change removes a set of distinct artefacts of the same type from a specific point that can be bound multiple times with the given artefact type.

**Update** An update change links an artefact to a specific point that can only be bound once with the given artefact type.

After modelling, the mapping model connects every collaborative feature with the set of changes which were applied to the core.

## 3.2 Mapping Aspectual Features

In contrast to collaborative features, aspectual features can not be modelled directly onto the core without duplicating the artefacts, since the individual points in the core where the artefacts of the feature realisation need to be connected are not unique. Therefore, we provide means to specify patterns that describe this points in the core. When the pattern is matched, the artefacts that realise the feature are connected to the matched points. This way, the patterns can be evaluated at product instantiation and modelling time which resolves the redundant duplication of the associated artefacts during modelling the product line.

Previous work [2, 11] shows that graph-rewrite systems can construct aspect weavers and explains the analogy between pointcuts and patterns in graph-rewrite rules (left-hand sides, LHS), and between advice and right-hand sides (RHS) of graph-rewrite rules. Since this technique is very powerful and language independent we use graph-rewrite rules to modularise the realisation of aspectual features and to describe the patterns that describe where to apply the feature realisation.

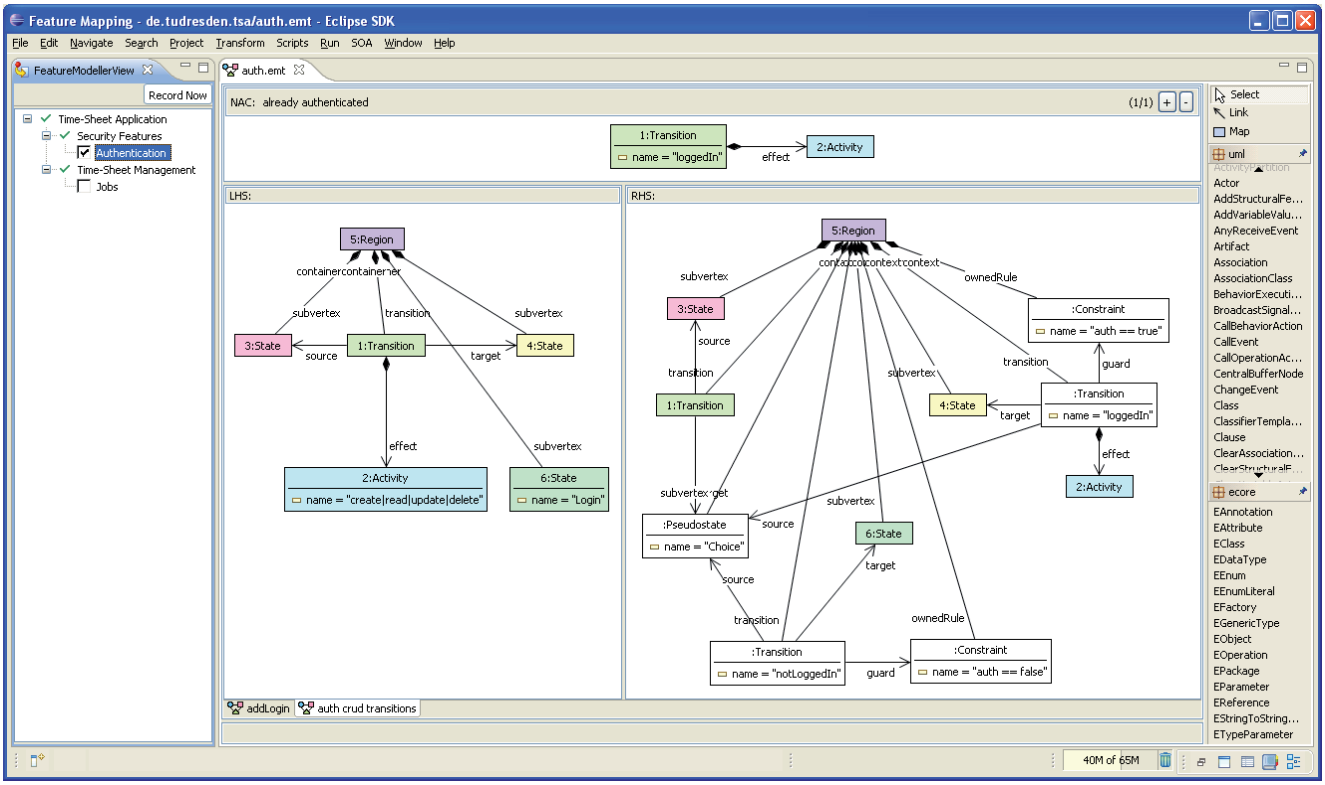


**Figure 7: Specification of the graph-rewrite rule for the realisation of the aspectual authentication feature (using the *Tiger EMF Transformation Project* shown on the right and our Eclipse plug-in shown on the left).**

Figure 7 depicts the *FeatureMapping* plug-in. It provides a *FeatureModellerView* which is used during modelling time to select the feature which is about to be realised and to build views on combined solution models respecting the variability constraints of the feature model. For product configuration it supports the specification of complete and valid variant models. To realise the aspectual feature from Section 2, *Authentication* is selected in the *FeatureModellerView* and associated though the mapping model with the graph-rewrite rule shown on the right. The rule's LHS describes a pattern of state chart artefacts to identify the points in the core which are to be augmented by artefacts of the authentication feature. Therefore, it searches for `Transitions` (1) associated with an `Activity` that performs a CRUD operation (2). Additionally, the `source State` (3), the `target State` (4), and the containing `Region` (5) of the `Transition` are matched. To redirect non-authenticated users the LHS-pattern also refers to the `State` *Login* (6) which have to be contained in the `Region`. Every match of this pattern is adapted by the RHS of the rule: The matched `Transition` (denoted by the same colour of both nodes) is redirected to the newly introduced `Pseudostate` *Choice*. This state is the `source` of two `Transitions` which are associated with `guard Constraints` to check the authentication status of the user. If the user is authenticated the `Transition` *loggedIn* is selected which executes the CRUD operation and leads to the original `target State`. Otherwise the `Transition` *notLoggedIn* redirects the user to the `State` *Login*. To avoid a repeated replacement of the same CRUD transition the *negative application condition* (NAC) *already authenticated* is used which prohibits the adaptation of `Activitys` that are associated with a `Transition` named *loggedIn*.

In our plug-in we use the graph-rewrite system *Tiger EMF Transformation Project* [19] which is a framework for in-place EMF transformations based on graph transformation. It supports the definition of graph-rewrite rules on arbitrary EMF-based metamodels and uses AGG [15, 16] as GRS.

## 4. RELATED WORK

In general two approaches to map between features and realisation artefacts can be distinguished: the *additive* and the *subtractive* approach [10].

In [6] Czarnecki et al. present a subtractive approach, where a monolithic model of the product's solution space is specified. Entities of this model are annotated with presence conditions that refer to features from the problem space. Presence conditions involve several constructs such as checks of a feature's presence/absence in the configuration. If a presence condition is evaluated to `false` the corresponding model entities are removed from the model, else they remain. Thus, the selected features in a variant drive the specialisation of the monolithic solution space model in a subtractive manner. For realistic scenarios the monolithic model gets very complex and hard to understand, because it contains the realisation of all features and gets polluted with lots of presence conditions. Additionally, the realisations of features which cross-cut the solution space gets scattered over the whole model and are therefore hard to recognise. In our work we focused on resolving these issues by externalising the mapping information to a dedicated mapping

model that allows for building dynamic views on the model and by supporting the explicit definition of aspectual features.

In [3] Avila-García et al. present an approach that extends the work of Czarnecki with a Domain-Specific Transformation language for transforming model templates based on a given feature model.

In [11] we present an additive approach where the realisation for each feature—collaborative or aspectual—is modelled separately in an independent fragment. We use graph-rewrite rules to weave them into the core architecture. Consequently, it is necessary to introduce links or pointcut expressions to identify the weaving points. The mapping between features and their realisations is implicitly encoded in the rewrite rules. This reduces the cognitive gap between features and realisation artefacts by modularising the solution space. The architectural design is not polluted with presence conditions which refer to the feature model leading to concise models and a clean separation of problem and solution space. Unfortunately, the strict decomposition makes it hard to analyse the interaction and dependencies between artefacts that belong to different features. The work presented in this paper tries to overcome the abovementioned shortcomings by allowing views on the solution space according to a specific variant model and by offering a more natural way of modelling collaborative features.

XWeave [10] is another approach to weave aspect models for both heterogeneous and homogeneous cross-cuts to a core model and, thus, shares the drawbacks of [11]. It is motivated by the realisation of features in PLE by distinct aspect models but in contrast to the work presented in this paper, it does not allow for feature-model driven building of views during product-line modelling.

# 5. SUMMARY AND FUTURE WORK

In this paper we motivated the distinction of aspectual and collaborative features to allow for a concise and modular specification of feature realisations in the solution space. We presented our language-independent approach which allows for mapping both feature types to arbitrary realisation artefacts in the solution space and separates mapping information in a dedicated mapping model by means of an Eclipse plug-in. Furthermore, we discussed how an interactive evaluation of the mappings during modelling time helps to analyse the interaction and dependencies between artefacts that belong to different features and motivated the importance of the mapping model for product instantiation.

In our future work we would like to investigate the performance-wise limitations we noticed while working with AGG which is also confirmed by the performance comparison presented in [9]. Exchanging the GRS we used in our work would also require us to extend an existing system with support for arbitrary EMF-based languages which is a very promising goal.

We will address interactions between feature realisations which strongly influence the product instantiation. These interactions can be determined through an analysis of feature realisations and mapping information. Hence, we aim to extend our mapping approach to automatically identify interacting features and provide means to resolve interaction problems for using the mapping models in an automated product instantiation process.

Additionally, we want to use our approach in a case study to validate the completeness of the classification into collaborative and aspectual features and to show the scalability of the mapping.

# 6. ACKNOWLEDGEMENTS

This work is supported by the feasiPLe project financed by the German Ministry of Education and Research (BMBF).

# 7. REFERENCES

[1] S. Apel, D. Batory, and M. Rosenmüller. On the Structure of Crosscutting Concerns: Using Aspects or Collaborations? In *Proceedings of the 1st Workshop on Aspect-Oriented Product Line Engineering (AOPLE'06) co-located with the 5th Int'l Conf. on Generative Programming and Component Engineering (GPCE'06)*, Portland, OR, USA, Oct. 2006. Online Proceedings. URL http://www.softeng.ox.ac.uk/aople/aople1/.

[2] U. Aßmann and A. Ludwig. Aspect Weaving with Graph Rewriting. In *Proceedings of the 1st Int'l Symposium on Generative and Component-Based Software Engineering (GCSE'99)*, volume 1799 of *LNCS*, pages 24–36, Erfurt, Germany, Sept. 1999. Springer.

[3] O. Avila-García, A. E. García, and E. V. S. Rebull. Using software product lines to manage model families in model-driven engineering. In *Proceedings of the 22nd Annual ACM Symposium on Applied Computing (SAC'07)*, pages 1006–1011, Seoul, Korea, 2007. ACM Press.

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

[5] K. Czarnecki. Overview of Generative Software Development. In *Proceedings of the Int'l Workshop on Unconventional Programming Paradigms 2004 (UPP'04)*, volume 3566 of *LNCS*, pages 326–341, Le Mont Saint Michel, France, Sept. 2005. Springer.

[6] 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, Tallinn, Estonia, Sept. 2005. Springer.

[7] feasiPLe Consortium. feasiPLe Research Project, Aug. 2007. URL http://feasiple.de.

[8] R. Filman and D. Friedman. Aspect-oriented programming is quantification and obliviousness. In *Proceedings of the 1st Workshop on Advanced Separation of Concerns co-located with the ACM Conf. on Object-Oriented Programming, Systems, Languages, and Applications 2000 (OOPSLA'00)*, Minneapolis, MN, USA, Oct. 2000.

[9] R. Geiß and M. Kroll. On Improvements of the Varro Benchmark for Graph Transformation Tools. Technical Report 2007-7, Universität Karlsruhe, IPD Goos, July 2007. ISSN 1432-7864.

[10] I. Groher and M. Völter. XWeave: Models and Aspects in Concert. In *Proceedings of the 10th Workshop on Aspect-Oriented Modeling (AOM@AOSD'07) co-located with the 6th Int'l Conf. on Aspect-Oriented Software Development (AOSD'07)*, Vancouver, Canada, Mar. 2007. ACM Press.

[11] F. Heidenreich and H. Lochmann. Using Graph-Rewriting for Model Weaving in the context of Aspect-Oriented Product Line Engineering. In *1st Workshop on Aspect-Oriented Product Line Engineering (AOPLE'06) co-located with the 5th Int'l Conf. on Generative Programming and Component Engineering (GPCE'06)*, Portland, OR, USA, Oct. 2006. Online Proceedings. URL http://www.softeng.ox.ac.uk/aople/aople1/.

[12] 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, USA*, 1990.

[13] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In M. Akşit and S. Matsuoka, editors, *Proceedings of the 11th Europ. Conf. on Object-Oriented Programming (ECOOP'97)*, volume 1241 of *LNCS*, Jyväskylä, Finland, June 1997. Springer.

[14] M. Mezini and K. Ostermann. Variability management with feature-oriented programming and aspects. In *Proceedings of the 12th ACM Int'l Symposium on Foundations of Software Engineering (FSE-12)*, pages 127–136, Newport Beach, CA, USA, 2004. ACM Press.

[15] G. Taentzer. AGG: A Graph Transformation Environment for System

Modeling and Validation. In T. Margaria, editor, *Tool Exhibition at Formal Methods 2003*, Pisa, Italy, Sept. 2003.

[16] The AGG Project Team. AGG: The Attributed Graph Grammar System, Aug. 2007. URL http://tfs.cs.tu-berlin.de/agg/.

[17] The Eclipse Foundation. The Eclipse Platform, Aug. 2007. URL http://www.eclipse.org.

[18] The Topcased Project Team. TOPCASED, Aug. 2007. URL http://www.topcased.org.

[19] Tiger EMF Transformation Project Team. Tiger EMF Transformation Project, Aug. 2007. URL http://tfs.cs.tu-berlin.de/emftrans/.

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