

# Using Graph-Rewriting for Model Weaving in the context of Aspect-Oriented Product Line Engineering

Florian Heidenreich  
Dresden University of Technology  
Software Technology Group  
01062 Dresden, Germany

florian.heidenreich@inf.tu-dresden.de

Henrik Lochmann  
SAP Research CEC Dresden  
Chemnitz Str. 48  
01187 Dresden, Germany

henrik.lochmann@sap.com

## ABSTRACT

In this paper we present the concept of combining feature models and solution models through aspect-oriented graph rewriting systems (AO-GRS) in the context of product-line engineering (PLE). Variable parts of software systems are often modelled by feature models in PLE. In Model Driven Development a feature is represented by means of model elements in a solution model, e.g. classes, methods or attributes in the context of UML models. The inclusion of a feature in the resulting software system may change the solution model of the product on multiple points and thereby crosscut the solution model. We use GRS-based model weaving to include specific features in a solution model based on the presence or absence of the feature in the variant model. We demonstrate the feasibility of our approach in a case study, which uses story diagrams as GRS.

## 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

AOSD, Product Line Engineering, Feature Models, Graph Rewriting

## 1. INTRODUCTION

Variable parts of software systems in PLE are often modelled by feature models [14], which describe the variabilities of the products in a product line. Enabling a feature often has a direct impact on the resulting solution model of the software product, because it may require including new model elements in the solution model. These changes are often not localised to one specific part of the model but are scattered

over different packages or classes and thereby crosscut the solution model. For example, including a new attribute in a business class may also require changes on the graphical user interface of the application. This brings up the need for defining the necessary changes on the model as pluggable, traceable and well localised transformation aspects on model level.

Models, e.g. UML class diagrams, are graphs describing software systems. These models can be manipulated by graph rewrite systems (GRS) which have a strong formal background, such as criteria for termination, confluence and unique normal forms. GRS have been recognised as a powerful technique for specifying complex transformations that can be used in various stages of the software development process. Aßmann uses GRS for program optimisation [5], Radermacher for application of design patterns [30] while Christoph does transformation of software designs in the context of the *Model Driven Architecture* (MDA) via GRS [11]. Aßmann and Ludwig are using GRS for constructing aspect weavers that work on implementation level [7]. We use GRS to construct weavers to integrate aspectual features chosen from the feature model to the core solution model of a software system to build products in a product line.

We start with a simple example, which demonstrates that graph rewrite rules can construct model weavers. After an introduction to feature modelling and model-driven software engineering in Section 3, we present our idea of model weaving with GRS in Section 4. We explain the usefulness of our approach with a case study on using GRS for model weaving within the Fujaba tool suite [18] in Section 5. Section 6 presents some related work, before we draw our conclusion and discuss future work in Section 7.

## 2. INFORMAL EXAMPLE

Before we explain our approach in detail, we describe the idea of model weaving via GRS by means of a small example.

Assuming that a feature from a feature model requires the inclusion of a new method in a specific class of the solution model. For example, in a banking system, implementing a minimum balance rule may require a method for calculating the available balance in addition to extra attributes for representing the minimum balance. A model weaver should be able to find the specific class, create the required method and insert this method in the class. Figure 1 depicts a simple weaver, consisting of a graph rewrite system with only one

rule. The graph rewrite rule is composed of a pattern, which matches a class with a specific name, and a pattern for creating a new method. Hence, it represents the concrete aspectual feature from the feature model. Whenever a class in the underlying model is found, whose class name corresponds to the name mentioned in the pattern, the method from the aspect pattern is linked to the methods of the matched class. The underlying graph is shown in Figure 2. We can use the graph rewrite rules to automatically generate model weavers as described in detail in Section 4.

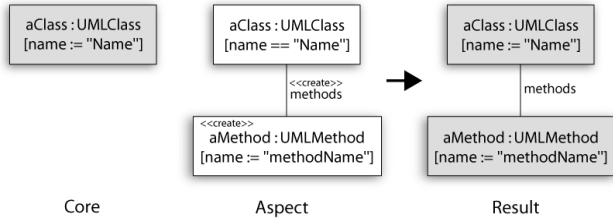


Figure 1: Graph rewrite rule for weaving a method into a class

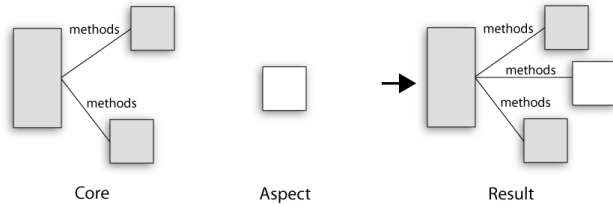


Figure 2: Graph visualisation of the weaving process

### 3. FEATURE AND SOLUTION MODELS

Product-line engineering in a software intensive context focuses on software engineering by developing and using core assets rather than creating products from scratch [24, 25]. To develop such assets, it is necessary to define and describe commonalities and variabilities in product lines. After the introduction of the *Feature-Oriented Domain Analysis* (FODA) [23] these product line variations were often expressed by *feature models*.

#### 3.1 Feature models

A *feature model*, as the result of domain analysis, contains information about mandatory and optional parts of software and provides an abstract, concise and explicit representation of variability in a product line [14]. Feature models consist of concrete features, which are of certain feature types. The FODA feature types are: *mandatory*, *alternative*, *optional* and *or-features*. While *mandatory* features represent required parts of products in a product line, the remaining feature types describe variable ones. The interactions and dependencies between features are described in a hierarchical manner. With the help of Kang’s graphical notation [23], feature models are shown as unranked trees, where the connection between parent and child features illustrates the type and relationship of child features in the same hierarchy level.

Figure 3 shows the very simple feature model we use in the case study described in Section 5. It consists of one *mandatory* root feature *communication*, followed by three *or-features* *email*, *postcard* and *SMS*. The *or-feature* in feature models establishes a dependency between parent and children in a [1-n] manner. That means at least one feature of a group of *or-features* must be included, while the other features of the group remain optional. Applied to the example in Figure 3, a certain product that follows the shown feature model has to implement at least *one* communication feature (e.g. *email*) and is allowed to implement other communication features as well, while this is not obligatory.

While a feature model describes the variability of a complete product line, a *variant model* can be understood as an instance of a feature model. Hence, variant models describe concrete products of a product line. Applied to the example above, a possible variant model would be one which includes the communication features *email* and *SMS* but does not include support for regular mail communication.

#### 3.2 Solution models

In the last few years, omnipresent problems such as heterogeneous platforms, code duplication, and insufficient documentation led to the idea of model-driven software engineering. The key idea is to develop solutions for a general problem domain instead of concrete, context specific scenarios that depend on underlying technologies. Therefore, abstract *solution models* of software systems are created for a certain problem domain. These models represent technology-independent systems, which form the basis for code generation and enable developers to understand unknown software systems more quickly. Different approaches have implemented this concept, the *Model Driven Architecture* [28] and *Model Driven Software Development* [33] in general along with supporting frameworks like the *Eclipse Modeling Framework* [15].

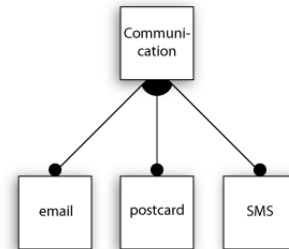


Figure 3: Example feature model in FODA feature diagram notation

In these approaches, system development starts with the creation of a model for a problem domain<sup>1</sup>. This model is either a *meta-model* or an instance of another meta-model. Meta-models form the basis for the creation of solution models by defining syntactic and semantic modelling constraints.

<sup>1</sup>As mentioned by Czarnecki in [12] regarding problem and solution space, the terms problem and solution domain are relative terms. A solution domain can be treated as problem domain on another stage of the software development process.

UML, as an example for a general purpose modelling language, defines the UML meta-model, which constrains the creation of domain specific models that may be instances of class, sequence or collaboration diagrams [29].

### 3.3 Need for integration

The abstract level of model-driven software engineering suits for an application to PLE because it allows a technology independent and high level way of developing software systems, which is demanded by feature models that describe variability in PLE. The realisation of products in a product line postulates the implementation of certain features in designed solution models and thus the combination of both feature models and solution models.

## 4. MODEL WEAVING WITH GRS

The main motivation behind weaving on the model level in the context of product-line engineering is combining the feature model and the solution model. The feature model thereby parameterises the core solution model, the variant-independent model (VIM), with features and extends the variability points of the model to form a variant-specific model (VSM) as shown in Figure 4. We use GRS for weaving aspectual features into the solution model. Before we describe our approach in detail, some basic graph rewriting terminology is presented in the next section.

### 4.1 Basic terminology

We choose *relational graph rewriting* as our graph rewrite system, in which graphs represent both entities of the solution model and aspects [4]. Nodes represent these entities and aspects and are linked by multiple relations, where a *context-free pattern* is a finite connected graph. A *context-sensitive pattern* is a graph of at least two disconnected sub-graphs. For example matching a specific pattern in a graph may require the existence of another pattern. A *host graph* is the graph to be rewritten by a GRS. A *redex* is a subgraph of the host graph injectively homomorphic to a pattern. A *graph rewrite rule*  $r = (L,R)$  consists of a left-hand side pattern L and a right-hand side graph R. L is matched against the host graph, i.e. its redex located in the host graph. In a rewriting step, a redex is replaced by the parts specified in the right-hand side of the rule.

### 4.2 Model Weaving

As explained in [7], GRS can be used to construct aspect weavers. Model-based AO-GRS provide a simple formalism for weaving aspects, which rely on properties directly recognisable from the structure of the aspect specifications and the entities from the model. All nodes and edges as well as their properties form the join-point model of AO-GRS. These join points are addressable through the redexes of a context-free or context-sensitive pattern in the host graph. A graph rewrite rule can be considered as an aspect weaver. Most often, the left-hand pattern of a graph rewrite rule contains at least one pattern addressing join points in the host graph and one pattern describing the aspectual part of the graph rewrite rule, the aspect pattern. The weaving operation attaches the aspect pattern to the redexes of the host graph.

To combine the feature model and the solution model, we

express each feature from the feature model as an aspect of the solution model. Since we use GRS, each aspect is represented by a graph rewrite rule. These rules are formulated on the meta-model level of the solution model, because we need access to the language constructs used to form the solution model. The graph rewrite rules can be just textual representations or built of graphical languages and modelled right beside the core solution model of the product line. We use the latter in our case study, which we describe in Section 5.

In model weaving for PLE, each aspect is woven according to the presence or absence of a feature in the variant model. If a feature is enabled and thereby present in the variant model, the corresponding graph rewrite rule is applied to the solution model. The graph rewrite rule can introduce new model elements by creating the representations of these new elements on meta-model level. This concept can be partially compared to the inter-type declarations of the AspectJ language [1], but furthermore it allows introducing all kinds of elements defined by the target model’s meta-model.

We use the UML meta-model for class diagrams [29] as the graph model for our solution, but it is possible to exchange this by any other meta-model, e.g. the Ecore meta-model [15]. GRS are also not limited to models describing static behaviour of a system like class diagrams. Graph-rewriting can be applied to UML state, activity or sequence diagrams and even to the diagrams of domain specific languages (DSLs), which are based on a meta-model known by the software architect.

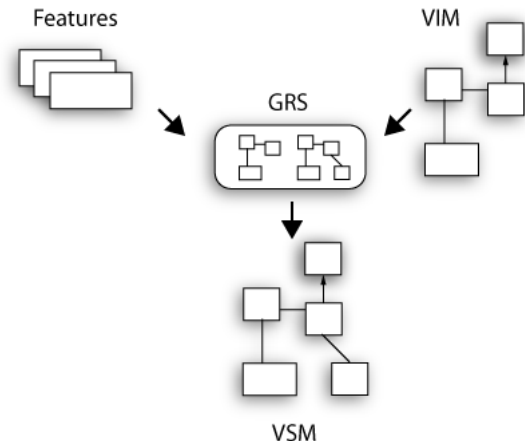


Figure 4: Parameterising the variant independent model (VIM) by features from a feature model to build a variant specific model (VSM)

## 5. CASE STUDY

We use the Fujaba Tool Suite [18] for building a small case study on using GRS for model weaving. Fujaba is a graph-based tool which uses UML for design and realisation of software projects. In the next section, we shortly introduce the concepts of Fujaba and especially its technique to define behavioural software parts with so called story diagrams. We demonstrate the feasibility of our approach by a small sample application.

## 5.1 The Fujaba Tool Suite

Fujaba, which means “From UML to Java and back again”, was developed by the software engineering group of the University of Paderborn [2] and is supported by other German universities. It offers forward engineering as well as reengineering techniques, and code generation and was developed with the aim of helping non-professionals to develop software applications. The main concept, which shall help to reach this aim, is the concept of *Story-Driven Modelling* (SDM) [17]. With an extended type of UML collaboration diagrams combined with activity diagrams, the behaviour of methods can be defined by modelling complete method bodies, which are used for source code generation. With story diagrams, a UML-based graph rewrite language was supplied that should enable mainly students in an educational context to familiar with the paradigm of object-oriented software development more quickly and easily.

## 5.2 Sample application

To demonstrate our GRS-based approach, we created a short sample application, which implements the core concepts of communication systems. Our example focuses on the structural changes needed on the model level. Behavioural modelling can be done with appropriate models but is not covered by the example. The main idea is to realise several communication technologies which can be used by communication partners to exchange messages with each other. Figure 3 illustrates the feature model for this example, which contains only several communication features. As described above, the semantics of the feature model dictates at least one communication type but also allows the implementation of additional ones. According to the correspondent feature, the communication partners can exchange messages via email, SMS or regular mail. The latter can be understood as a kind of automatic submission of postcards in this software intensive context. A corresponding core solution model for a software system which covers these tasks, could look like the one depicted in Figure 5.

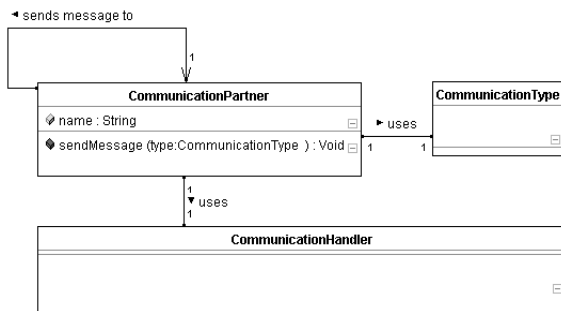


Figure 5: Core solution model before weaving

Besides the enumeration *CommunicationType*, the core solution model accommodates basically two classes, which shall be used for message exchange. The first one is the class *CommunicationPartner* which represents a communication party and contains information that is necessary to send messages according to a certain communication type. The message exchange via email for example needs information about the recipient’s email address, while for SMS-based communi-

tion a mobile phone number has to be supplied. The only behavioural part in the class *CommunicationPartner* is covered by the *sendMessage()* method. This method triggers the message submission according to a certain communication type, which could be supplied e.g. by user interaction.

Due to the fact, that the solution model in Figure 5 illustrates just the core model of the system, the communication details for the class *CommunicationPartner* are not yet included. The same applies to the class *CommunicationHandler* and the enumeration *CommunicationType*, which are discussed below after showing the solution model as a result of a weaving transformation, as described above.

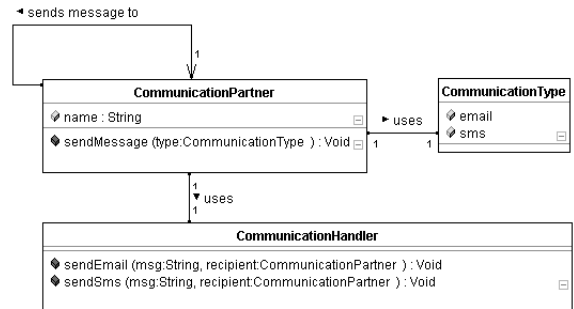


Figure 6: Solution model after weaving

Figure 6 depicts the solution model woven according to a certain variant model, assuming that the correspondent variant model for the shown solution model includes the communication types email and SMS but does not include regular mail sending. Hence, each aspect corresponding to the included features was introduced into the solution model. Now the necessary communication details are modelled in form of attributes in the class *CommunicationPartner* (e.g. *phoneNumber*). The class *CommunicationHandler* now contains methods to send messages via email or SMS. Additionally, the enumeration *CommunicationType* contains constants that refer to all communication types that are possible and hence supported by the system.

The weaving applied to the core model was defined by correspondent graph rewrite rules, according to each included feature, as described in Section 4. In this case study we used the visual graph rewrite language of story diagrams, which was introduced in Section 5.1, to define the rewrite rules. Figure 7 shows the rule for including the email aspect.

As described in Section 4, the aspect-weaving behaviour has to be modelled on meta-model level, to be able to manipulate the underlying solution model which just instantiates the meta-model. Hence, the referenced classes in the story diagrams of our example refer to the UML meta-model. Based on the semantics of story diagrams, the upper part of Figure 7 defines the selection of the *UMLClass* with the name “*CommunicationPartner*”, which is contained in the given solution model. This pattern defines the left-hand side of a graph rewrite rule. The connection between the objects *clazz* and *attribute* is stereotyped with the value *create*, which means, that a new attribute named “*eMailAddress*” has to be created for the selected class. This short story

forms a rewrite rule that is sufficient to add the necessary details for email communication to the *CommunicationPartner* class. The story below implements the addition of the “sendMail” method to the class *CommunicationHandler*.

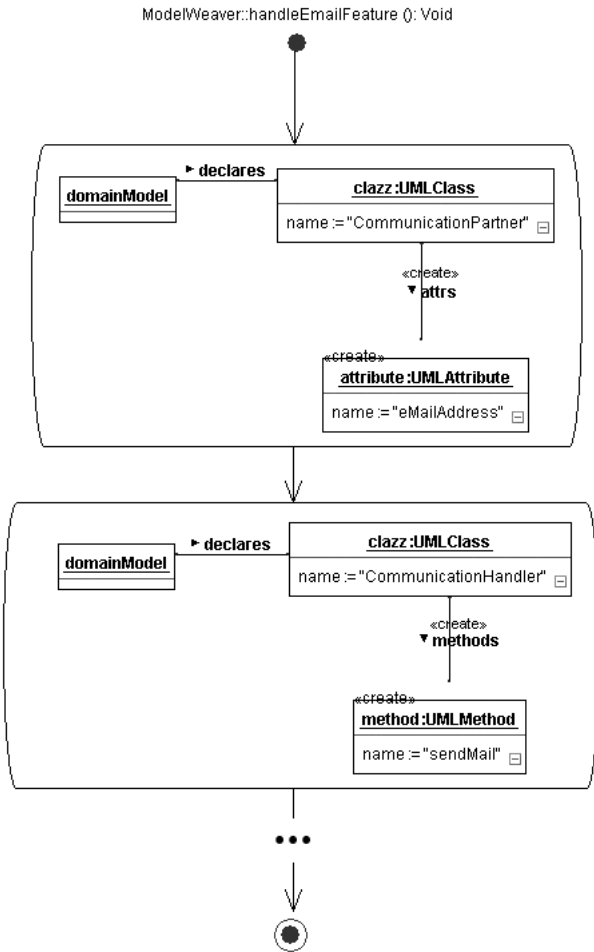


Figure 7: Partial story diagram for the email aspect

The story diagrams described above are now used to generate the actual aspect weaver, which transforms the solution model and thereby weaves the aspectual features. The weaving code is compiled on the fly and loaded into the Fujaba tool suite. Our solution also covers the integration of feature models by using a UML-based feature notation. Based on a specific variant model, the aspect weaver can be triggered via menu item and directly updates the solution model in the CASE tool. The whole solution is deployable as Fujaba plugin.

## 6. RELATED WORK

Many researchers applied aspect-oriented techniques to non-code artifacts, reaching from aspect-oriented requirements engineering (AORE) [32, 31] to aspect-oriented domain modelling (AODM) [20]. They invented methods to use and express AO concepts in the context of the UML [16] [9] and thereby raised the level of abstraction in AOSD.

It is well known that variabilities in product lines can be

realised with aspect-oriented techniques. [19] analyzes different practices for implementing variabilities in PLE and also considers AOP as one possible solution. [26] uses feature oriented programming with collaborations in CaesarJ [27] while [22] introduces UML for Aspects, which supports aspectual collaborations as described in [21].

With AHEAD a more generalised approach is presented by Batory et al. [10] by introducing an equation-based model for refinement of code and non-code artifacts based on feature inclusion. The work on mapping features to models of Czarnecki et al. [13] is also related to our work, whereby the presented superimposition of variants requires all possible features - also conflicting ones - being modelled in the model.

In the domain of GRS, [7] describes the usage of GRS for aspect weaving and gives an overview of different GRS-based weaver classes. The research results presented in [8] are pointing towards integrating GRS tools in the standard software development tools using sublanguage projection. This refers to our work in terms of the integration of the GRS in the software tool as seen with Fujaba and our plugin. [3] uses graph-rewriting for applying transformation rules on domain-specific models defined in an UML-based meta-modelling language. [11] uses OPTIMIX [6] as GRS for the transformation of software designs in the context of the Model Driven Architecture, especially for transformation of platform independent models (PIM) to platform specific models (PSM). This approach has probably the closest relation to our work, because it also uses GRS for model transformation. It differs in regard to the amalgamation of feature models in PLE with solution models through generated GRS-based aspect weavers.

## 7. CONCLUSION AND FUTURE WORK

In this paper we showed that GRS can be used to weave aspectual features on model level. We explained AO-GRS and described the semantics of joint point, pointcut and advice to terms of graph rewriting. We pointed out, that our approach is not limited to models based on UML class diagrams, but can be used for UML state and sequence diagrams as well as for DSLs with an accessible meta-model. The usefulness of AO-GRS on the model-level was shown by a case study in the context of the Fujaba tool suite. We developed a plugin for Fujaba, which constructs model weavers based on story driven graph rewrite rules and applies them to solution models.

Our future work in this area will address the dependency between model-level aspects and implementation-level aspects in model weaving. Therefore, we plan to extend our approach to models describing dynamic behaviour of software systems and want to demonstrate the usefulness of our contribution in this area too.

Another possible research area is the integration of GRS in repositories, where both feature models and solution models reside together and are accessible to the GRS. This will reduce the effort in mapping the different model types, like feature models and solution models to graph models understandable by the GRS.

## 8. ACKNOWLEDGEMENTS

This work is supported by the feasiPLe project financed by the German Ministry of Education and Research (BMBF). We would like to thank Falk Hartmann, Steffen Zschaler, and the anonymous reviewers for their very valuable comments on previous versions of this paper.

## 9. REFERENCES

- [1] AspectJ. <http://www.eclipse.org/aspectj/>.
- [2] University of Paderborn, Germany. <http://www.uni-paderborn.de/home/en/>.
- [3] A. Agrawal, G. Karsai, and A. Ledeczi. An end-to-end domain-driven software development framework. In *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 8–15, New York, NY, USA, 2003. ACM Press.
- [4] U. Aßmann. Graph rewrite systems for program optimization. Technical Report RR-2955, INRIA Rocquencourt, 1996.
- [5] U. Aßmann. How to uniformly specify program analysis and transformation with graph rewrite systems. In *CC '96: Proceedings of the 6th International Conference on Compiler Construction*, pages 121–135, London, UK, 1996. Springer.
- [6] U. Aßmann, A. Christoph, and J. Lövdahl. Optimix. <http://optimix.sf.net>.
- [7] U. Aßmann and A. Ludwig. Aspect weaving with graph rewriting. In *GCSE '99: Proceedings of the First International Symposium on Generative and Component-Based Software Engineering*, pages 24–36, London, UK, 2000. Springer.
- [8] U. Aßmann and J. Lövdahl. Integrating graph rewriting and standard software tools. In J. L. Pfaltz, M. Nagl, and B. Böhlen, editors, *AGTIVE*, volume 3062 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2003.
- [9] E. Baniassad and S. Clarke. Theme: An approach for aspect-oriented analysis and design. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 158–167, Washington, DC, USA, 2004. IEEE Computer Society.
- [10] D. Batory, J. N. Sarvela, and A. Rauschmayer. Scaling step-wise refinement. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 187–197, Washington, DC, USA, 2003. IEEE Computer Society.
- [11] A. Christoph. Graph rewrite systems for software design transformations. In *NODE '02: Revised Papers from the International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World*, pages 76–86, London, UK, 2003. Springer.
- [12] K. Czarnecki. Overview of generative software development. In J.-P. Banâtre et al., editor, *UPP '04: Unconventional Programming Paradigms*, volume 3566 of *Lecture Notes in Computer Science*, pages 313–328, 2005.
- [13] K. Czarnecki and M. Antkiewicz. Mapping features to models: A template approach based on superimposed variants. In R. Glück and M. R. Lowry, editors, *GPCE*, volume 3676 of *Lecture Notes in Computer Science*, pages 422–437. Springer, 2005.
- [14] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley Professional, June 2000.
- [15] Eclipse Foundation. Eclipse Modeling Framework Ecore meta-model. <http://www.eclipse.org/emf/>.
- [16] T. Elrad, O. Aldawud, and A. Bader. Aspect-oriented modeling: Bridging the gap between implementation and design. In *GPCE '02: The ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering*, pages 189–201, London, UK, 2002. Springer.
- [17] T. Fischer, J. Niere, L. Torunski, and A. Zuendorf. Story diagrams: A new graph rewrite language based on the unified modeling language. In *Proceedings of the 6th International Workshop on Theory and Application of Graph Transformation (TAGT)*, 1998.
- [18] Fujaba Tool Suite Developer Team, University of Paderborn. Fujaba tool suite. <http://www.fujaba.de/>.
- [19] C. Gacek and M. Anastasopoulos. Implementing product line variabilities. In *SSR '01: Proceedings of the 2001 symposium on Software reusability*, pages 109–117, New York, NY, USA, 2001. ACM Press.
- [20] J. Gray, T. Bapty, S. Neema, D. C. Schmidt, A. Gokhale, and B. Natarajan. An approach for supporting aspect-oriented domain modeling. In *GPCE '03: Proceedings of the second international conference on Generative programming and component engineering*, pages 151–168, New York, NY, USA, 2003. Springer.
- [21] I. Groher, S. Bleicher, and C. Schwanninger. Model-driven development for pluggable collaborations. In O. Aldawud, T. Elrad, J. Gray, M. K. J. Kienzle, and D. Stein, editors, *7th International Workshop on Aspect-Oriented Modeling*, Oct. 2005.
- [22] S. Herrmann. Composable designs with UFA. In *Workshop on Aspect-Oriented Modeling with UML at 1st Intl. Conference on Aspect Oriented Software Development*, 2002.
- [23] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Software Engineering Institute, Carnegie Mellon University Pittsburgh, Pennsylvania 15213, 1990.
- [24] K. Kang, J. Lee, and P. Donohoe. Feature-oriented product line engineering. *Software, IEEE*, 19:58–65, Jul/Aug 2002.
- [25] K. Lee, K. C. Kang, and J. Lee. Concepts and guidelines of feature modeling for product line software engineering. In *Lecture Notes in Computer Science*, volume Volume 2319, page Page 62, Januar 2002.
- [26] M. Mezini and K. Ostermann. Variability management with feature-oriented programming and aspects. In *SIGSOFT '04/FSE-12: Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering*, pages 127–136, New York, NY, USA, 2004. ACM Press.
- [27] M. Mezini, K. Ostermann, and et al. CaesarJ. <http://caesarj.org>.
- [28] Object Management Group. Model Driven Architecture. <http://www.omg.org/mda/>.
- [29] Object Management Group. Unified Modeling Language 2.0. <http://www.uml.org/>.
- [30] A. Radermacher. Support for design patterns through graph transformation tools. In *AGTIVE '99: Proceedings of the International Workshop on Applications of Graph Transformations with Industrial Relevance*, pages 111–126, London, UK, 2000. Springer.
- [31] A. Rashid, A. Moreira, J. Araujo, P. Clements, E. Baniassad, and B. Tekinerdogan. Early aspects portal. <http://www.early-aspects.net/>.
- [32] A. Rashid, A. Moreira, and B. Tekinerdogan. Special issue on early aspects: aspect-oriented requirements engineering and architecture design. *IEE Proceedings - Software*, 151(4):153–156, 2004.
- [33] M. Völter and T. Stahl. *Model-Driven Software Development*. John Wiley & Sons, June 2006.