Karlsruhe Reports in Informatics 2012,5
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

Towards a generic approach for meta-model- and domain-independent model variability

Zoya Durdik, Klaus Krogmann, Felix Schad

2012

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

Towards a generic approach for meta-model- and domain-independent model variability

Zoya Durdik, Klaus Krogmann, Felix Schad
FZI Forschungszentrum Informatik
Haid-und-Neu-Strae 10-14, 76131 Karlsruhe

Fakultät für Informatik, Karlsruher Institut für Technologie (KIT),
Interner Bericht 2012-5

February 17, 2012

Abstract

Variability originates from product line engineering and is an important part of today's software development. However, existing approaches mostly concentrate only on the variability in software product lines, and are usually not universal enough to consider variability in other development activities (e.g., modelling and hardware). Additionally, the complexity of variability in software is generally hard to capture and to handle. We propose a generic model-based solution which can generally handle variability on Ecore-based meta-models. The approach includes a formal description for variability, a way to express the configuration of variants, a compact DSL to describe the semantics of model variability and model-to-model transformations, and an engine which transforms input models into models with injected variability. This work provides a complete and domain-independent solution for variability handling. The applicability of the proposed approach will be validated in two case studies, considering the two independent domains of mobile platforms and architecture knowledge reuse.

Keywords: Variability, variants, modelling, transformation, features, domain-independent.

1 Introduction

Variability in software engineering originates from product line engineering and is an important part of today's software development practice. Structured handling of variability allows for an efficient creation of variants (e.g. specific products derived from a core). The successful application of model-driven development [17] shifts the need for variability to the model level. A significant number of approaches exists to handle variability (see survey [3]). For example, feature models [5] capture variability. However, these approaches mostly concentrate on the variability in software product lines, imply significant overhead for the description of feature semantics [11], if formal semantics is captured at all, are limited to a single domain, or do not allow for automated creation of variants.

In this technical report, we propose a generic model-based approach, which can generally handle variability on Ecore-based meta-models – independent from the domain model (e.g. not only limited to mobile devices). The approach includes i) a formal description for variability, for this we use extended *feature models* introduced by Czarneki [5], ii) a way to express the configuration of variants (so-called *feature configurations*, iii) a compact DSL (Domain Specific Language) to describe the semantics of features of variability, and iv) an engine which transforms input domain models into domain models with injected variability. An example from the mobile devices domain would be the variability "GPS" or "touch surface" (feature model) of which "no GPS active" is selected (feature configuration), a DSL which formalises feature semantics "remove GPS connector", and a resulting variant of a mobile device which supports touch interaction but not GPS.

Indepedence from the domain model and meta-model is realised by transferring feature semantics to means of the meta-meta-model (i.e. Ecore). The semantics of variability features is translated to elements of the meta-meta-model and can thus be specified and executed for arbitrary meta-models which are defined in Ecore. Formal semantics of model variability can be easily specified due to a compact DSL which furthermore allows for the automated creation of variants.

The main contribution of the approach is its indepence from a single domain meta-model, the aproach's compact notation (DSL) to describe feature semantics, and its ability to create variants at the model-level by means of automatically derived model transformations. The applicability of the proposed approach will be validated in two case studies, considering the two independent domains of mobile platforms and architecture knowledge reuse.

The remainder of this technical report is organized as follows: Section 2

introduces the approach, describes the followed process, and illustrates the approach using an example, Section 3 presents the layout of the planned validation, Section 4 highlights related aproaches, and Section 5 summarises and concludes the technical report.

2 Proposed Approach

This section describes the proposed model-based solution, which can generally handle variability on Ecore-based meta-models. The main idea of the solution is to handle the instantiation of variants (i.e. the actual model transformations) at the meta-meta-model level instead of handling them at the level of domain-specific meta-models (see Fig. 1). Hence the solution is reusable and reduces the overall complexity. A set of *intermediate model transformations* abstracts model variability to the meta-meta-model level. These intermediate model transformations basically provide mappings between the different abstraction levels: meta-model level, model level, and variant instance model. Ultimately, domain models are transformed by the model representation on the level of the meta-meta-model (i.e. Ecore: EClasses, EReferences, etc.). The proposed generic solution is valid for any domain that has a need for variability support, and can be reasonably expressed through a meta-model. The first part of the section provides our proposed process to handle variability, which is demonstrated on an example at the end of the section.

Model Le el	**Model T pe**	**Use**
Me a Me a M del	c e	T an f ma i n e ec i n
Me a M del	main Me a M del	Va iabili ea e Seman ic
M del	main n ance	Va ian ea i n

Figure 1: Model levels and use

2.1 Process

To generically handle variability on Ecore-based meta-models, we propose a process presented on Fig. 2. It contains five major steps (on the left), and a set of six artefacts (on the right). The process can be executed by any person (from now on "*user*") pocessing sufficient domain knowledge and technical skills to express variability and variants, e.g. by a software architect. Any transformations which create actual variants can then be executed automatically. The process steps and corresponding artefacts are

described in detail in the following.

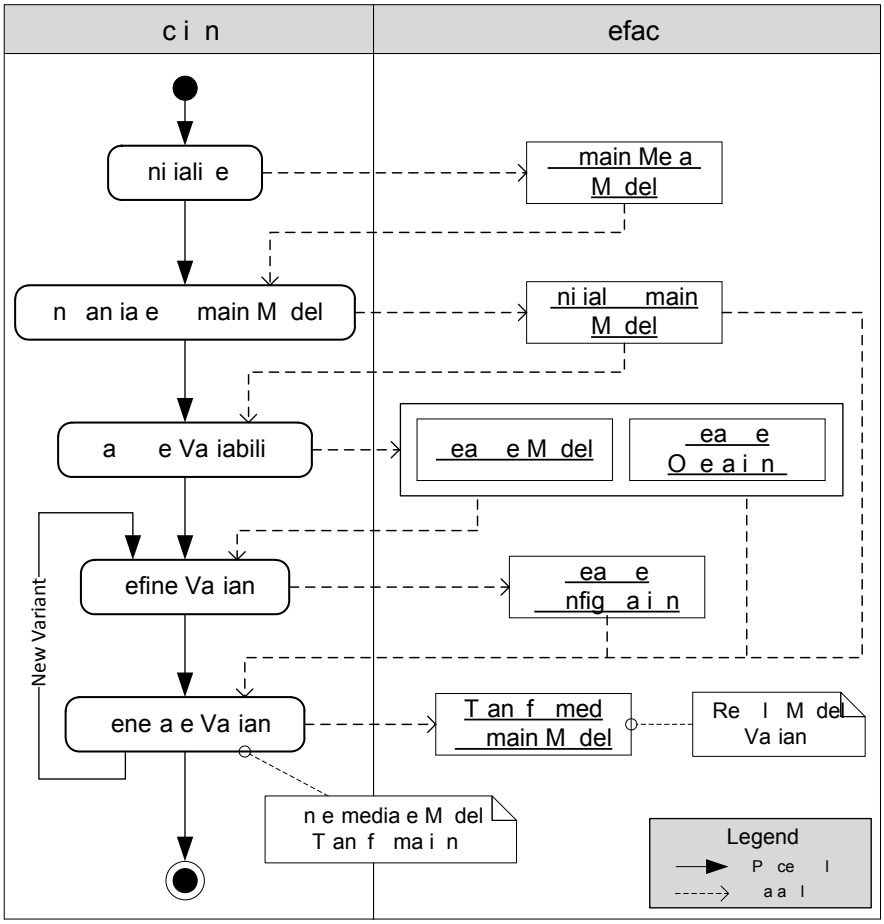

Figure 2: Proposed process for generic variability handling

I. Initialisation. In this step, the *user* initialises the variability support. For this he reuses, defines or develops a *domain meta-model* of the domain. The *domain meta-model* is Ecore-based and describes the domain where variability shall be supported. An example of the *domain meta-model* is a Domain Specific Language (DSL), such as the Palladio Component Model (PCM) [13] which captures the architecture of software systems. The *domain meta-model* serves as input for the next process step "*Modelling*".

II. Domain model instantiation. In this step, the *user* instantiates the *domain meta-model*. For this purpose, he reuses, defines or develops an

initial domain model. The *initial domain model* is an instance of the *domain meta-model*. The *initial domain model* can, for example, be an architecture of a smartphone system modelled in the PCM.

III. Capture variability. In this step, the *user* captures domain variability with the help of *feature models*, and establishes connections between features in the *feature model* and the *initial domain model* through *feature operation* definitions. The *feature model* describes variability (in our approach we do not discern feature diagrams and feature models, as explained in the related work Section 4). The *feature model* support is based on a meta-model derived from the state-of-the-art (we have used Pure::Variants[1]). It can be directly used by the *user*, and does not require additional development. With the help of the feature model, the user only needs to capture variability of the domain, define dependencies and constraints between variation points (features). An example of a feature model can be seen in Fig. 4. The *feature model* elements are extended with annotations that we call *feature operations*.

Feature operations formalise transformation semantics of single features using a simplified domain specific language (DSL). The DSL expresses model-to-model transformations and is specifically suitable to express model variability. These commands describe how the *initial domain model*, and its elements in particular, should be modified to support the associated feature if it will be selected in a certain variant. Examples of such commands would be: "define A = component", "A = initialStructureModelElementID" or "delete A"; which replaces a component instance by another component.

Basically, the order in which the *initial domain model* and *feature model* are developed or defined can be exchanged. However, these both artefacts need to exist before the *feature operations* can be defined. Therefore, we propose to create the domain meta-model before capturing the variability and the associated *feature operations*.

IV. Variant definition (configuration). In this step, the *user* configures specific variants selecting a valid set of features from the *feature model*. This selection is captured in a *feature configuration*. The *feature configuration* mechanism will be provided and does not require domain-specific development. An example of *feature configuration* model is presented in Fig. 5.

The previous process steps might seem heavyweight, but step I and II are typcially carried out in all model-driven development scenarios. The variant configuration step does not require complicated actions, and is executed

[1] see http://www.pure-systems.com/pure_variants.49.0.html

every time the *user* would like to generate a new variant. Previous steps are only required to prepare variant generation and executed only once.

V. Variant generation. At this step, the *user* retrieves a specific variant. The generation of a specific variant is performed automatically on the *initial domain model* by the execution of the transformations associated with selected features of the *feature configuration*). The result is a *transformed domain model*, which has the same meta-model as the *initial domain model*. To execute the feature operations (i.e. transformations) we will provide an engine, which transforms input models into models with injected variability. The engine itself comprises the intermediate transformations, introduced above, which translate the feature operations expressed in the DSL, into transformations referring to the Ecore-model. The latter transformations can be regular QVT-O transformations [12].

In the example of product lines, the *transformed domain model* would result out of a core model (*initial domain model*) enriched with the selected features. However, our approach also provides other means for *initial domain model* modification. Thus, elements and connections can be added and removed, the existing elements and connections between them can be modified (e.g. with additional attributes or changing connection direction and order).

The "*variant generation*" step is technically and conceptually the most challenging step to develop in our proposed solution.

This proposed solution is domain-independent, as it handles variability on Ecore-based meta-models, and thus, domain models can be simply exchanged. Feature models assure formal description for variability, and feature configurations are the proposed way to express the configuration of variants. In the next subsection we demonstrate the proposed process on an example from the mobile application development domain.

2.2 Smartphone Example

This subsection demonstrates the proposed process on a simple example from the mobile application development domain.

I. Initialisation. A reference architecture of a mobile application is used as the domain model for this example.

II. Domain model instantiation. The *user* instantiates the *domain meta-model* in form of a reference architecture of the mobile phone application. The instance of the reference architecture of the application is presented in Figure 3, where a Picture Controller Component accesses different services, such as Language and Payment Services. The Payment Service

requires two other different services: PayPal and a Credit Card. The Language component can be supported by a corresponding language service, e.g. English or German.

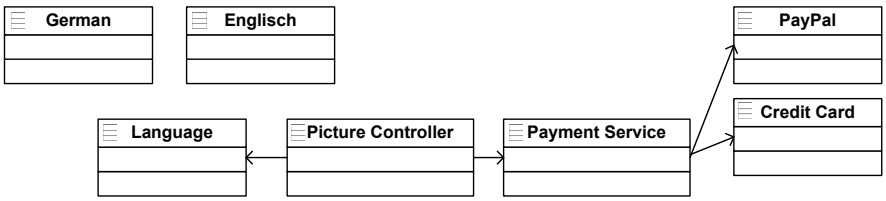

Figure 3: Inital domain model or reference architecture

III. Capture variability. The *user* captures domain variability with the help of a *feature model*, presented in Figure 4. It captures the requirements and the desired variability of the domain. The mobile application "PhotoApp" supports PayPal or Credit Card (PayPal OR Credit Card) as payment methods. However, the application supports only one language, either English, or German (English XOR German). The application can execute either on an iOS, or an Android device (iOS XOR Android). Additional variation points, such as GPS, display resolution, camera, multi-touch are not considered in this simplified example.

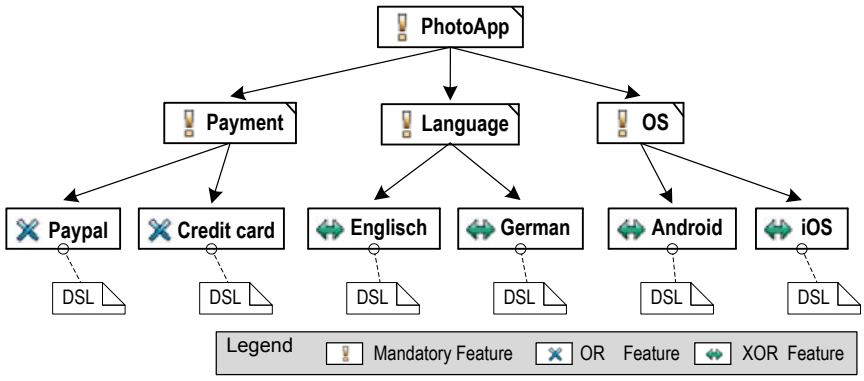

Figure 4: Feature model of a mobile phone application

The DSL annotations attached to features in Figure 4 represent *feature operations*, which formalize transformation semantic of single features in the *feature model* and their connection to the elements of *initial domain model*. For example, the *feature operations* of the Credit Card feature can

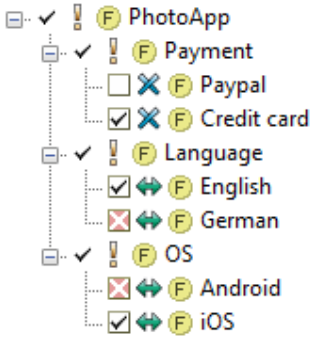

Figure 5: Feature configuration

contain actions to add this feature to the reference architecture, in case this feature is selected. Vice versa, feature operations could also remove feature which are present in a reference architecture but are not desired in a certain variant. The *feature operations* can also contain information, which other service is necessary to support "Credit Card" feature, e.g. a bank service "Sparkasse".

Please note that *feature operations* simplify typical variability operations. They are expressed in a domain-meta-model independent language which is internally mapped to Ecore operations. For example, on the instance level of that DSL, "define A = component", "delete A" creates a reference to a domain meta-model type and deletes all instances of that types, while "define A = component", "A = initialStructureModelElementID" would delete a single instance including all references. Further operations can replace on instance by another (e.g. connect to another bank service) while keeping references intact.

IV. Variant definition (configuration). The *user* configures a specific variant of the mobile application selecting a set of features from the *feature model*, as shown in Figure 5. The features "Credit Card", "English" and "iOS" have been selected for this particular variant (i.e. represented by the *feature configuration*).

V. Variant generation. In the first four steps, all preconditions for the variant generation step are complied. The transformation works as follows: The *feature configuration* contains which features are selected. The transformations within each feature are collected from the *feature operations*, and are executed on the *initial domain model* (mobile application reference architecture). The result is a variant of the mobile application with an ar-

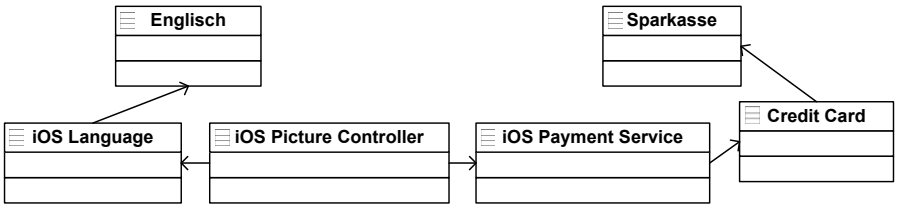

Figure 6: Transformed domain model or variant architecture

chitecture that supports the required features. Figure 6 shows the generated architecture and contains the features "Credit Card", "English" and "iOS". As the *feature operations* defined for the feature "Credit Card" contained information on the bank service required to be implemented together with this feature, the additional service "Sparkasse" was added to the application architecture variant during the transformation.

Thus, in this example we demonstrated how, following the proposed process, an initial domain model of the mobile application could be transformed into a variant of the application with the help of the information and transformations from the feature model.

3 Validation

This work proposes a generic meta-model independent and domain-independent solution for variability handling. After the development of the proposed solution is completed, we plan to validate the approach in two case studies in different domains with different domain meta-models.

The first case-study will validate the approach for variability handling in mobile application development. Mobile phones and applications contain variability at different levels, such as hardware, operation system (OS) and application levels. Thus, the hardware variability includes different devices with different features, such as GPS, display resolution, camera, and multi-touch. The variability at the OS level includes various operation systems and system libraries, for example iOS, Android, or Windows Phone. Finally, the variability at the application level includes variation points, such as language, UI-Elements, data persistence, and implementing components. We plan to provide a domain meta-model describing software components to apply our approach in this domain.

The second case study will validate the approach in the domain of architecture knowledge reuse. A common practice in software design and development is to reuse architectural solutions, such as design or architecture patterns to solve common software engineering problems. Beside having multiple solution variants to one problem, such as different design patterns, most of the patterns do have several implementation variants to solve deviations of the main problem. These variants can be described as features to generate the desired pattern variant. This case study might be more complex as the first one, as patterns introduce many additional requirements on the variability support (e.g. per instance binding of features to model elements). So the actions for component modification might be more complex. Additionally component interfaces and roles (i.e. multiple types of references per model element) shall be considered.

These two case studies will help to check the applicability of our approach, as we as validate our claim that the approach is generic and domain-independent. We also plan to validate possible advantages of our DSL over the direct use of QVT-O, such as compact description of transformations, maintainability, and simplicity. For this purpose we plan to develop a questionnaire with the transformation example, one in our DSL, and another one in QVT-O, and let participants compare the transformation size, understandability, and expected maintainability.

4 Related Work

There is a significant number of research dedicated to the variability, but most of the proposed approaches target product line engineering, or are domain dependant. In this section, we present approaches the most related to our work.

The fundamental work [15] exposes important themes and issues in variability, such as notation for feature models, definition of abstraction levels and description of pattern of variability. Bosch et al. [2] claim that "the first-class representation for features and variation points" does not exist, and that "implicit dependencies between architectural elements and features are seldom made explicit". Our approach solves these issues by developing a formal description for variability with explicit places for connections between feature diagram and domain model (architecture).

Feature models are a commonly used mechanism to capture variability. A lot of extensions and modifications of the original feature model, which were defined in "Feature-Oriented Domain Analysis (FODA)" [10], has been developed. However, the most approaches do not provide a formal description of a feature model, except the [7, 6, 8, 14]. Thus, Czarnecki et al. proposes staged configuration using feature models in [7], staged configuration through specialization and multi-level configuration of feature models in [6], and formalization of cardinality-based feature models and their specialization in [8]. The work by Schobbens et al. handles generic semantics of feature diagrams in [14]. These approaches by Czarnecki et al. provide an UML based meta-model for feature model, which is partly reused in our approach.

The relation between the feature model and the domain model has been described in [16] and [9]. A similar approach is provided by Czarnecki and Antkiewicz in "Mapping Features to Models: A Template Approach based on superimposed variants" [4]. The both approaches are limited to negative variability, which means that the whole system with all possible variations has to be modelled in the domain model. We provide an approach, which supports both, negative variability and positive variability. This provides the opportunity to extend and modify the structure of the system without predefining all variants in the domain model, and thus keeps the domain model compact and maintainable.

In [1] Bak et al. provide a meta-modelling language that has first-class support for feature modelling. Although, the approach reduces the number of artefacts that are required to express variability during meta-modelling, the different concerns are mixed in a single artefact. Another disadvantage

is that already existing compact meta-models can not be reused, as they have to be translated into Clafer (a concept modeling language for software product lines) and the proposed notation. Our approach is more generic and supports any Ecore-based meta-model and meta-model notation.

Thus, the existing approaches mostly concentrate only on one specific domain to model variability, such as variability in software product lines, and are usually not universal enough to consider variability in other development activities, as it is the case in our approach. Our approach can generally handle variability on models.

5 Conclusion

The approach proposed in this technical report enables a meta-model-independent handling of variability for models. It includes means to capture variability and semantics associated with variability features. The approach also introduces a compact DSL suitable to describe model semantics as a subset of model-to-model transformations specifically selected for model variability. Furthermore, the approach comprises a way to express the configuration of variants and a transformation-based engine which translates input domain models into domain models representing a certain variant. The applicability of the proposed approach will be validated in two case studies, considering the two independent domains of mobile platforms and architecture knowledge reuse.

One outstanding facet of the approach is its completeness (from modelling of variability to the execution of model transformations which implement variants) and its ability to provide a domain-independent solution for variability handling. The proposed generic model-based solution can generally handle variability on Ecore-based meta-models. Hence, there is no need to re-invent the handling of variability from domain to domain. Nevertheless, feature models can directly express domain terminology. The proposed compact DSL captures feature semantics and is directly related to features of the feature model and thus further increases understandability of variability.

The proposed approach focuses on variability for models, but not on source code or other artefacts (except for the case that source code is represented by a model itself). Its DSL does intentionally not represent a full transformation language but focusses on the expression of model variability to stay compact and easy to understand. This is expected to lower the application effort of the approach. Given the increasing importance of model-driven development, the application field and use of the approach is further beneficary. Currently, the approach does not consider ordering effects among features.

The main task for future work is the full implementation of the approach. We plan to implement the approach based on EMF and QVT-O (internal transformation language and engine). The proposed validation will show the applicability and effectiveness of the approach.

References

[1] K. Bak, K. Czarnecki, and A. Wasowski. Feature and meta-models in Clafer: Mixed, specialized, and coupled. In *Proceedings of the Third international conference on Software language engineering*, pages 102–122. Springer-Verlag, 2010.

[2] J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, J. H. Obbink, and K. Pohl. Variability issues in software product lines. *Software Product-Family Engineering*, 2290:13–21, 2002.

[3] L. Chen, M. Ali Babar, and N. Ali. Variability management in software product lines: a systematic review. In *Proceedings of the 13th International Software Product Line Conference*, pages 81–90. Carnegie Mellon University, 2009.

[4] K. Czarnecki and M. Antkiewicz. Mapping Features to Models : A Template Approach Based on Superimposed Variants. pages 422–437, 2005.

[5] K. Czarnecki and U. W. Eisenecker. *Generative Programming*. Addison Wesley, 2000.

[6] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged Configuration Through Specialization and Multi-Level Configuration of Feature Models. *Software Process: Improvement and Practice*, 10(2):266–283, 2004.

[7] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration using feature models. *Software Product Lines*, pages 162–164, 2004.

[8] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.

[9] I. Groher and M. Voelter. Aspect-oriented model-driven software product line engineering. *Transactions on Aspect-Oriented Software Development VI*, pages 111–152, 2009.

[10] K. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA), 1990.

[11] L. Kapova and T. Goldschmidt. Automated feature model-based generation of refinement transformations. In *Proceedings of the 35th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2009.

[12] Object Management Group (OMG). Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification – Version 1.1 Beta 2, December 2009.

[13] R. Reussner, S. Becker, E. Burger, J. Happe, M. Hauck, A. Koziolek, H. Koziolek, K. Krogmann, and M. Kuperberg. The Palladio Component Model. Technical report, Karlsruhe, 2011.

[14] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps. Generic semantics of feature diagrams. *Computer Networks*, 51(2):456–479, 2007.

[15] J. Van Gurp, J. Bosch, and M. Svahnberg. On the notion of variability in software product lines. *Proceedings Working IEEEIFIP Conference on Software Architecture*, pages 45–54, 2001.

[16] M. Voelter. Variantenmanagement im Kontext von MDSD. *System*, pages 1–11, 2005.

[17] M. Völter and T. Stahl. *Model-Driven Software Development*. Wiley, 2006.