

Aspect-oriented Feature Models

Position Paper

Marko Bošković
SCIS, Athabasca University
1 University Drive
Athabasca, AB T9S 3A3,
Canada
marko.boskovic@athabascau.ca

Gunter Mussbacher
SITE, University of Ottawa
800 King Edward
Ottawa, ON, K1N 6N5,
Canada
gunterm@site.uottawa.ca

Ebrahim Bagheri
SCIS, Athabasca University
1 University Drive
Athabasca, AB T9S 3A3,
Canada
ebagheri@athabascau.ca

Daniel Amyot
SITE, University of Ottawa
800 King Edward
Ottawa, ON, K1N 6N5,
Canada
damyot@site.uottawa.ca

Dragan Gašević
SCIS, Athabasca University
1 University Drive
Athabasca, AB T9S 3A3,
Canada
dgasevic@acm.org

Marek Hatala
SIAT, Simon Fraser University
Surrey
13450 102 Avenue
Surrey, BC V3T 5X3, Canada
mhatala@sfu.ca

ABSTRACT

Software Product Lines (SPLs) have emerged as a prominent approach for software reuse. SPLs are sets of software systems called families that are usually developed as a whole and share many common features. Feature models are most typically used as a means for capturing commonality and managing variability of the family. A particular product from the family is configured by selecting the desired features of that product. Typically, feature models are considered monolithic entities that do not support modularization well. As industrial feature models tend to be large, their modularization has become an important research topic lately. However, existing modularization approaches do not support modularization of crosscutting concerns. In this position paper, we introduce Aspect-oriented Feature Models (AoFM) and argue that using aspect-oriented techniques improves the manageability and reduces the maintainability effort of feature models. Particularly, we advocate an asymmetric approach that allows for the modularization of basic and crosscutting concerns in feature models.

Categories and Subject Descriptors

D.2.1 [Software Engineering]: Requirements/Specifications

General Terms

Languages

Keywords

Software Product Lines, Feature Models, Aspect-oriented Modeling

1. INTRODUCTION

The increase in the number of software systems containing extensive sets of common requirements has led to widespread interest in Software Product Line Engineering (SPLE) [19]. Software Product Line Engineering is a methodological framework for engineering Software Product Lines (SPLs), or *software families*. SPLs are sets of software systems with an extensive number of common functional and non-functional properties. SPLs are developed as a whole and share many assets, herewith increasing reusability.

An SPL generally consists of three kinds of artifacts, representing the problem space, the solution space, and the mappings between problem and solution spaces [8]. Artifacts in the solution space represent design and implementation of all members of the family. The problem space, on the other hand, is comprised of all the features for the family members. Typically, the problem space is captured with feature models. A *feature model*, first introduced by Kang [13], is represented with a feature diagram, a tree-like structure whose root represents the whole SPL and whose descendant nodes represent potential features of its members. A particular product of the family is defined by selecting the desired features from the feature model. Based on such feature selection and with the help of defined mappings, artifacts of the solution space are composed to form the desired product. The process of selecting desired features is called *configuration* and the set of selected features a *feature model configuration* [9].

Contemporary feature models often tend to grow very large in size and different groups of experts are usually dedicated to different parts of feature model development [11]. These parts must be assembled into one large feature model. Treating such large feature models as monolithic entities makes them very hard to develop, manage, understand, and evolve. Typically, every change performed on a feature model must be verified by experts with different expertise [17], which is time consuming and costly.

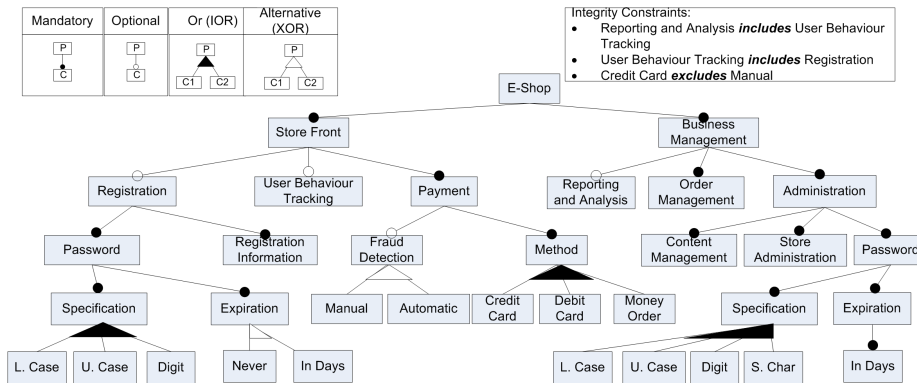


Figure 1: An E-shop Feature Model

To address this problem, recent work by Mannion et al. [17], Hubaux et al. [11] and others introduce approaches for the modularization of feature models. However, separation of concerns can still be improved as these approaches lack support for the modularization of *crosscutting concerns* such as security and other non-functional properties. In this paper, we leverage principles of Aspect-oriented Modeling (AOM) [7] to address this issue. In dealing with similar problems, AOM has been applied in several requirement formalisms like use cases [12], problem frames [16], URN [18], and UML models [21]. By applying these principles, we therefore aim at enabling better understandability, maintainability, and scalability of feature models.

This paper presents Aspect-oriented Feature Models (AoFM) which introduce the notion of concerns in feature models. Particularly, we distinguish between non-crosscutting concerns called *base concerns* and crosscutting concerns called *aspects*. AoFM consist of feature models for these two types of concerns as well as the specification of their *composition rules* with the aim to reduce the impact of changes in one concern's feature model on other concerns' feature models.

The remainder of the paper is as follows. Section 2 describes the basics of feature models. Our extension of basic feature modeling, i.e., AoFM, is introduced in Section 3. Section 4 summarizes identified contemporary related work. Finally, Section 5 concludes the paper and discusses future work.

2. FEATURE MODELS

Feature models are widely accepted means of capturing commonality and managing variability within SPLs. They are modeled as feature diagrams, i.e., tree-like structures consisting of nodes that represent features of modeled SPL and their interrelationships. Figure 1 shows an example of a feature diagram of an E-shop SPL, inspired by Lau [15].

Typically, a feature diagram consists of features, specified as nodes, and of inter-feature relationships. The latter are either *mandatory* and *optional* parent-child feature relationships or *alternative feature groups* and *or feature groups*, graphically presented in Figure 1. *Mandatory* parent-child relationships specify that, if a parent feature is included in a certain feature model configuration, its mandatory child features also have to be included (e.g., the relationships be-

tween the **E-shop** and **Store Front** features and the **E-shop** and **Business management** features specifying that every E-shop consists of the interface that the customer uses for accessing the E-shop and one concerned with back office operations, respectively). An *optional* parent-child relationship specifies that when a parent feature is selected in a configuration, its optional child features may but do not have to be included (e.g., the relationship between the **Store Front** and **User Behaviour Tracking** features). *Alternative feature groups* consist of features from which exactly one must be selected (XOR) (e.g., the group consisting of the **Manual** and **Automatic** features - fraud detection in an E-shop application can be either manual or automatic, but not both). An *or feature group* specifies a group of features from which at least one must be selected (IOR) (e.g., the group consisting of the **Credit Card**, **Debit Card**, and **Money Order** features - to enable customers to pay for selected items, at least one payment method must be selected).

Finally, feature models also contain feature relationships that cannot be captured with a tree structure. Such relationships are called cross-tree constraints, or integrity constraints [5]. Most often, these constraints are includes and excludes constraints. An *includes* constraint specifies a relationship between two features that ensures that one feature is selected when the other is. In the feature model in Figure 1, includes constraints exist between the **Reporting and Analysis** and **User Behaviour Tracking** features and between the **User Behaviour Tracking** and **Registration** features. The *excludes* integrity constraint specifies a relationship between two features that ensures that one feature is not selected when another one is. Such a constraint exists between the **Credit Card** and the **Manual** features.

Even in such a small example as in Figure 1, it can be recognized that not all features are of interest to the same stakeholders. While features for password policies are the major concern for security experts, they are not of high relevance to higher management. Furthermore, the developers of user interface functionality are not interested in back office operations and vice versa. Moreover, different parts of a family might require different security policies. Namely, the password policy for back office administration of an E-shop application is usually stricter compared to the user interface's

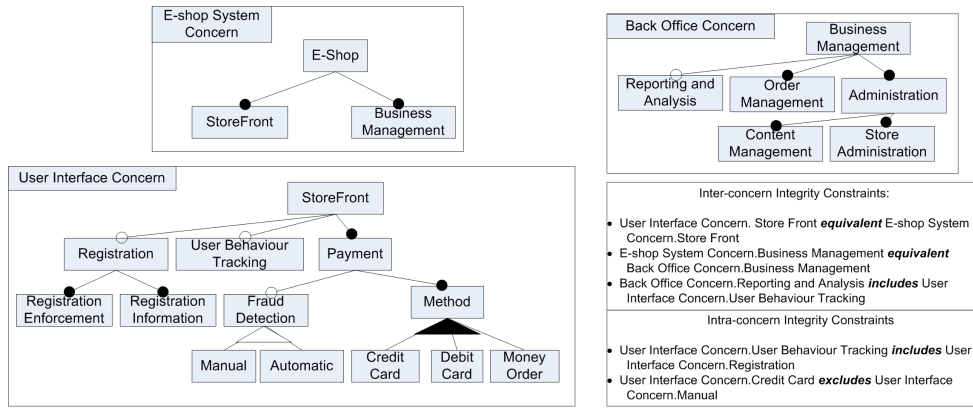


Figure 2: Base Concerns in the E-shop Feature Model

and might require the inclusion of special characters and an expiration after a certain amount of days. For this reason, a subtree for password policy specification of back office administration contains the additional (**S. Char**) and (**In Days**) features. Finally, the addition of new security policies would result in revisions to sub-models related to security policy in the store front and back office sub-parts. Considering the size of contemporary feature models, separation of different concerns is an important task for achieving better understandability, maintainability, and scalability of models. To provide separation of concerns in feature models, we introduce AoFM in the next section.

3. ASPECT-ORIENTED FEATURE MODELS

Aspect-oriented Modeling (AOM) applies concepts originating from Aspect-oriented Programming (AOP) [14] to requirements and design models. AOM emerged as one of the most prominent approaches for software modularization. Particularly, AOM facilitates modularization of aspects, i.e., concerns which cannot be encapsulated properly with other contemporary modularization techniques. Aspects are concerns that often repeat with small variations in other concerns. Examples of aspects are performance and security, each being implemented by different mechanisms such as load-balancing/caching and authentication/encryption, respectively. In AoFM, we distinguish between non-crosscutting concerns, or base concerns, and crosscutting concerns, or aspects. Note that this distinction, however, does not preclude aspects from being applied to other aspects.

3.1 Base Concerns in AoFM

Base concerns in feature models are specific *viewpoints* of a feature model [20]. Viewpoints are different perspectives on the same feature model. A viewpoint contains those parts of a feature diagram that are of interest to the users associated with the viewpoint. In the E-shop case study in Figure 1, we have identified three concerns as illustrated in Figure 2.

The **E-shop System Concern** is a high level business concern representing the major functionalities of the E-shop SPL. This base concern contains the root feature **E-Shop**

and its two child features **Store Front** and **Business Management**. The **User Interface Concern** is a concern of particular interest to developers of user requirements and interactions. It contains the parts of the feature model representing **Registration**, **User Behaviour Tracking**, and **Payment** functionalities. In addition to the features in Figure 1, the **User Interface Concern** contains the **Registration Enforcement** feature which represents actions that mandate user registration such as buying products. Finally, the **Back Office Concern** contains all functionality related to the operation of an E-shop application.

Concerns in AoFM have the same tree-like structure and relationships as the basic feature models introduced in Section 2. However, the modularization of feature models into base concerns introduces two kinds of integrity constraints, namely intra-concern integrity constraints and inter-concern integrity constraints. *Intra-concern* integrity constraints exist between features of one concern. They are essentially the same as the integrity constraints of the basic feature models, i.e., includes and excludes constraints. In the example in Figure 2, there is an includes constraint between the **User Behaviour Tracking** and **Registration** features of the **User Interface Concern** and an excludes constraint between the **Credit Card** and **Manual** features of the **User Interface Concern**.

Inter-concern integrity constraints are specified between features in different concerns. The set of possible inter-concern constraints also contains includes and excludes constraints. To provide a way of representing the same feature in different concerns, we introduce the *equivalent* inter-concern integrity constraint. Such constraints enable the separation of base concerns that are not highly crosscutting but still should be encapsulated into their own modules based on the viewpoint principle mentioned earlier. Hence, inter-concern integrity constraints constitute the composition rules for base concerns. For example, an equivalent constraint exists between the **Store Front** feature of the **E-shop System Concern** and the **Store Front** feature of the **User Interface Concern**, allowing these two concerns to be sufficiently separated. Using equivalent constraints, the feature diagrams of base concerns can be re-composed, if desired, into the original, much larger, and more monolithic feature diagram.

3.2 Aspects in AoFM

To facilitate modeling of aspects in feature models, AoFM must support the specification of feature models for crosscutting concerns and their composition with base concerns. This composition requires the specification of i) patterns defining where the aspect is to be applied, ii) composition rules specifying how the aspect is to be applied at the locations identified by the patterns, and iii) a join point model, i.e., the set of all locations that an aspect is allowed to change and hence may be matched by the patterns. Therefore, we introduce the concepts of a) join point model for feature models, b) aspects, c) aspect feature models, d) pointcut feature models, e) patterns, and f) composition rules.

The *join point model* for feature models contains the feature nodes of a feature diagram, allowing any feature node to be changed by an aspect. *Aspects* are purely organizational units consisting of aspect feature models and pointcut feature models. An example of an aspect is the **Security** aspect of the E-shop case study, a portion of which is presented in Figure 3. It contains one aspect feature model and two pointcut feature models.

Aspect Feature Models (AFMs) are feature models of crosscutting concerns. Structurally, they do not differ from feature models of base concerns. In Figure 3, the **Password Policy** is the crosscutting concern of password policies, a sub-concern of the **Security** aspect. The **Password Policy** aspect defines the feature model for all possible password policies configurations.

Pointcut Feature Models (PFMs) express patterns as well as composition rules. Structurally, they are again not different from feature models of base concerns except that they may be tagged with two kinds of markers. Patterns and composition rules together must enable all changes an aspect in AoFM may possibly want to impose on a base concern, i.e., the adding and removal of feature diagram elements. To achieve this, the *pattern* to be matched in the base concern is first identified by the \circ tag in the PFM. Second, any element from an AFM that is also used on a PFM is, by default, added to the matched base concern. The element from the AFM is added together with its descendant features and their relationships. In addition, we can also add features and relationships that are not specified in the AFM by simply specifying them in the PFM.

Third, any element tagged with \times in the PFM is removed from the feature model. The \times tag may be applied to an element from the AFM or an element already tagged with \circ . In the former case, this tag indicates that the element from the AFM is not to be added to the composed model. The same applies to the descendant features of the element from the AFM and their relationships. In the latter case, the tag may be visualized as \otimes and indicates that the matched element from the base concern is to be removed. Again, this applies to its descendant features and their relationships.

Therefore, the *composition rule* consists of a) the set of links in the PFM that connect elements tagged with \circ and elements from the AFM and b) any remove operations indicated by the \times tag.

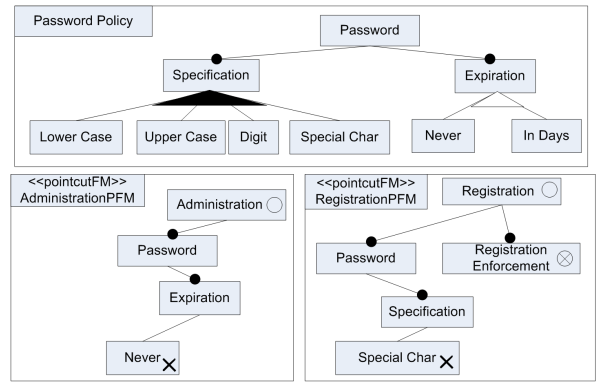


Figure 3: Excerpt of Security Aspect Feature Model for E-shop SPL

It is necessary to allow the specification of elements from the AFM that are not to be added to the matched base concern, because aspects in AoFM should be very generic and applicable to multiple SPLs from very different domains, i.e., an aspect is applied multiple times not to just one SPL but multiple times to multiple SPLs. Therefore, it is very likely that not all features of the aspect are applicable to all domains. Consequently, the \times marker is necessary to indicate such features. Note that a PFM is invalid if the links among features from an AFM are not the same as the links defined in the AFM.

For example, two PFMs are defined in Figure 3. **AdministrationPFM** specifies adding a password policy to the **Administration** operations, while **RegistrationPFM** adds a password policy to the **Registration** feature in the base feature model. For **AdministrationPFM**, the pattern to be matched is the **Administration** feature as indicated by the \circ tag. The composition rule for **AdministrationPFM** states that the complete **Password** feature sub-tree from the AFM except for the **Never** feature is to be attached to the **Administration** feature with a mandatory parent-child relationship. The **Never** feature and its relationship to the **Expiration** feature are excluded because of the \times tag.

For **RegistrationPFM**, the pattern to be matched is the **Registration** and **Registration Enforcement** features connected with a mandatory parent-child relationship. The composition rule for **RegistrationPFM** states that the complete **Password** feature sub-tree from the AFM except for the **Special Char** feature is to be attached to the **Registration** feature with a mandatory parent-child relationship. In addition, the **Registration Enforcement** feature and its relationship to the **Registration** feature must be removed from the base concern, as indicated by the \otimes tag.

The pattern specification for AoFM also allows for regular expressions to increase the matching power of the pattern. For example, if the same password policy were to be applied to both the **Administration** and **Registration** and no elements of the base concerns needed to be removed, only one PFM would be sufficient and the feature tagged with \circ could be named **(Admin|Reg)istration**. This example also demonstrates the modularization power of our

approach. Namely, the number of PFMs and repetition of features from base concerns in PFMs depend only on the differences between subsets of AFM that are added to different join points.

Finally, if there is a need for integrity constraints between features of base concerns and aspects, they are specified as part of the PFM.

4. RELATED WORK

Due to the large size of feature models, the need for modularization has been recognized by several researchers.

Czarnecki et al. [9] introduce a reference in feature model as the main means for modularization. A reference points to one feature of a feature model that defines a subtree which will be copied to the reference location. Compared to this approach, AoFM allow for variability and flexibility in the composition rules of concerns, herewith better encapsulating crosscutting concerns.

Mannion et al. [17] propose a viewpoint-based approach for the development of feature models that facilitates the development of large feature models by merging several smaller ones. Smaller feature models represent viewpoints from the perspective of different stakeholders. This approach allows for the modularization of feature models similar to base concerns in AoFM, but fails at capturing crosscutting concerns, which is the main benefit of AoFM.

Hubaux et al. [11] also introduce an approach for multiview feature models. According to Hubaux et al., a large feature model is supported by multiple views, or concerns, which are used for configuration performed by different stakeholders. Compared to this approach, AoFM additionally supports the specification of crosscutting concerns.

Acher et al. [1] present a textual, domain-specific language for managing and evolving feature models¹. This language includes operators for feature composition (e.g., for inserting and merging models), comparison and analysis (e.g., to determine the validity of the resulting model). Compared to AoFM, their approach does not support crosscutting concerns well as their feature composition targets a single feature in a model where another can be, for instance, inserted. Their feature merging is name-based but is limited to only one operator at a time (e.g., union or difference), whereas AoFM supports a mixture of operators simultaneously.

Dhungana et al. [10] observe that problem and solution space models may both have crosscutting concerns. For this reason, they introduce model fragments, i.e., models of reusable assets and their variability points, as separate modules. In their approach, crosscutting concerns in the problem space are merged with base concerns by means of placeholders. If there is a need for different variants of a crosscutting concern, the concern needs to be copied and adapted for each placeholder. AoFM allows for better modularization as base models are unaware of crosscutting concerns and different variants of crosscutting concerns can be specified in PFMs.

¹See also their FAMILIAR Web page at: <http://nyx.unice.fr/projects/familiar/>

Recently, Zhang [22] introduced a notion of aspect-oriented feature modeling. According to Zhang's method, feature models should be extended with aspectual features representing system concerns. Aspectual features are used for adapting the base feature model according to selected system concerns. AoFM is somewhat different as AFMs capture commonalities and manage variabilities of a crosscutting concern, and rules for their integration into major concerns are provided. Furthermore, a crosscutting concern is not necessarily a system concern in AoFM, but may be a part of a subsystem.

At this time, AOM techniques are used mostly for solution space models instead of feature models. Zschaler et al. [23] present VML*, which addresses variability management across several modeling notations such as feature models, use cases, activity diagrams, and UML 2.0 architectural models. While AOM techniques are used to ease the engineering of new family members from a given feature model, feature models themselves are not structured in an aspect-oriented way. An example related to SPL is the DiVA Project² which intends to manage dynamic variability in adaptive systems with the help of AOM techniques. Again, variability models in the problem space are not structured in an aspect-oriented way but solution space artifacts are. AoFM, on the other hand, structure feature models based on aspect-oriented principles.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have introduced the notion of Aspect-oriented Feature Models (AoFM). We argue that AoFM further enhance the modularization of feature models by enabling encapsulation of crosscutting concerns. However, full integration of AoFM in the process of SPLE is still in its early stage. Particularly, there is a need for extending composition rules, providing tool support, and automatically analyzing SPLs consisting of AoFM and their aspect-oriented implementations for the solution space.

To further extend the composition rules of the presented AoFM, we intend to introduce a *substitution* composition rule. This rule allows for features to be substituted while all relationships of the substituted and of the substituting feature are retained. Furthermore, we also intend to enable the specification of ordering of compositions in the case where several aspects are added to the same feature.

For tool support, we intend to extend *FeaturePlugin* [2], an Eclipse plug-in for feature model development. To avoid overwhelming users of the tool with new concepts, we plan to perform a lightweight extension to the existing notation used by the FeaturePlugin. The extensions will facilitate only the annotation of existing elements of the *FeaturePlugin* notation with concepts needed for AoFM.

Due to the large size of feature models, there is a need for automated analysis of feature model designs [4]. A comprehensive set of analysis operations can be found in [5]. To provide analysis operations and explanations for different stakeholders, and to reduce the time required for the analysis, we plan to use Distributed Description Logics (DDL) [6].

²<http://www.ict-diva.eu>

The base model and the aspects including their composition rules are formalized in DDL for modularized conceptual modeling and reasoning with DDL. In our previous work [3], we have applied DDL for verification of inconsistencies in modular feature model specifications and irregular FM configurations. Our initial results show a significant reduction of reasoning time, because only needed information from other modules is imported when reasoning about a particular module. To evaluate the claimed benefits of using DDL, we intend to analyze the time needed for the verification of feature models specified by AoFM and their DDL representation, and compare it to the equivalent, monolithic feature model represented in basic description logic. For example, the analysis of AoFM could aim at ensuring consistency of the whole composed feature model and determine whether constraints of the base model or an aspect are not violated by other aspects.

Typically, in an SPL, AoFM are connected to the solution space implementation of the family. During design time, there is a need for ensuring that interdependencies of cross-cutting concerns in the problem space are not contradictory to mappings and crosscutting concerns in the solution space, i.e., we need to ensure that every valid configuration produces a valid product. We also intend to use DDL for this kind of verification.

Acknowledgments. This research was in part supported by Alberta Innovates – Technology Futures through the New Faculty Award program, the Discovery Grants program of the Natural Sciences and Engineering Research Council of Canada, and the Ontario Graduate Scholarship Program.

6. REFERENCES

- [1] M. Acher, P. Collet, P. Lahire, and R. France. Composing feature models. In *2nd Int. Conf. on Software Language Engineering (SLE 2009)*, pages 62–81. LNCS 5969, Springer, 2010.
- [2] M. Antkiewicz and K. Czarnecki. Featureplugin: feature modeling plug-in for eclipse. In *Eclipse '04: Proc. of the 2004 OOPSLA WS on Eclipse technology eXchange*, pages 67–72. ACM, 2004.
- [3] E. Bagheri, D. Gasevic, and F. Ensan. Modular Feature Models: Representation and Configuration. Technical report, Athabasca University, 2009.
- [4] D. Batory, D. Benavides, and A. Ruiz-Cortés. Automated Analysis of Feature Models: Challenges Ahead. *Communications of ACM*, 49(12):45–47, 2006.
- [5] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: A lit. review. *Inf. Systems*, 35(6):615 – 636, 2010.
- [6] A. Borgida and L. Serafini. Distributed description logics: Assimilating information from peer sources. *Journal of Data Semantics*, 1:153–184, 2003.
- [7] R. Chitchyan, A. Rashid, P. Sawyer, A. Garcia, M. P. Alarcon, J. Bakker, B. Tekinerdogan, S. Clarke, and A. Jackson. Survey of analysis and design approaches. Technical report, AOSD-EUROPE, Del. 11, 2005.
- [8] K. Czarnecki and U. W. Eisenecker. Components and generative programming (invited paper). *SIGSOFT Softw. Eng. Notes*, 24(6):2–19, 1999.
- [9] K. Czarnecki, S. Helsen, and U. W. Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.
- [10] D. Dhungana, P. Grünbacher, R. Rabiser, and T. Neumayer. Structuring the modeling space and supporting evolution in software product line engineering. *Journal of Systems and Software*, 83(7):1108–1122, 2010.
- [11] A. Hubaux, P. Heymans, P.-Y. Schobbens, and D. Deridder. Towards multi-view feature-based configuration. In R. Wieringa and A. Persson, editors, *Requirements Engineering: Foundation for Software Quality*, volume 6182 of *LNCS*, pages 106–112. Springer, 2010.
- [12] I. Jacobson and P.-W. Ng. *Aspect-Oriented Software Development with Use Cases*. Addison-Wesley, 2004.
- [13] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University SEI, November 1990.
- [14] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP*, pages 220–242, 1997.
- [15] S. Q. Lau. Domain analysis of e-commerce systems using feature-based model templates. Master’s thesis, University of Waterloo, Waterloo, 2006.
- [16] M. Lencastre, J. Araujo, A. Moreira, and J. Castro. Analyzing crosscutting in the problem frames approach. In *IWAAPF '06: Proc. of the 2006 Int. WS on Advances and Applications of Problem Frames*, pages 59–64. ACM, 2006.
- [17] M. Mannion, J. Savolainen, and T. Asikainen. Viewpoint-oriented variability modeling. *Int. Comp. Softw. and Applications Conf.*, 1:67–72, 2009.
- [18] G. Mussbacher and D. Amyot. Extending the user requirements notation with aspect-oriented concepts. In *SDL'09: Proc. of the 14th Int. SDL Conf. on Design for notes and mobiles*, pages 115–132. Springer, 2009.
- [19] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.
- [20] I. Sommerville and P. Sawyer. Viewpoints: principles, problems and a practical approach to requirements engineering. *Annals of Soft. Eng.*, 3:101–130, 1997.
- [21] J. Whittle, P. K. Jayaraman, A. M. Elkhodary, A. Moreira, and J. Araújo. MATA: A Unified Approach for Composing UML Aspect Models Based on Graph Transformation. *Trans. on AOSD*, 6:191–237, 2009.
- [22] G. Zhang. Aspect-oriented feature modelling for software product line. web: <http://ecs.victoria.ac.nz/twiki/pub/Events/ACDC2010/Zhang.pdf>.
- [23] S. Zschaler et al. VML*—A Family of Languages for Variability Management in Software Product Lines. In *2nd Int. Conf. on Software Language Eng. (SLE 2009)*, pages 334–353. LNCS 5969, Springer, 2010.