Towards Systematic Ensuring Well-Formedness of Software Product Lines

Florian Heidenreich
Lehrstuhl Softwaretechnologie
Fakultät Informatik
Technische Universität Dresden, Germany
florian.heidenreich@tu-dresden.de

ABSTRACT

Variability modelling with feature models is one key technique for specifying the problem space of software product lines (SPLs). To allow for the automatic derivation of a concrete product based on a given variant configuration, a mapping between features in the problem space and their realisations in the solution space is required. Ensuring the correctness of all participating models of an SPL (i.e., feature models, mapping models, and solution-space models) is a crucial task to create correct products of an SPL. In this paper we discuss different possibilities for checking well-formedness of SPLs and relate them to their implementation in the FeatureMapper SPL tool.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Computer-aided software engineering, Object-oriented design methods*; D.2.2 [**Software Engineering**]: Software/Program Verification—*Validation*; D.2.13 [**Software Engineering**]: Reusable Software

General Terms

Design, Languages

Keywords

Software product lines, separation of concerns, variability modelling, well-formedness rules, FeatureMapper

1. INTRODUCTION

A software product line (SPL) is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [4]. In addition to the shared core assets, every system of a SPL has features that are specific to the system and that are not shared by all other systems (often called *products*) of the SPL. To express this variability, variability modelling is used to describe the different features available in an SPL and their interdependencies. In the Feature-Oriented Software Development (FOSD) community [2], a widely used approach for variability modelling are feature models [13, 5].

Feature-based variability modelling resides in the *problem space* whereas the realisation of features is part of the *solution space* [6]. To instantiate products from an SPL, feature realisations in the solution space have to be configured according to the presence of the features in a *variant model*; that is, a concrete selection of features from a feature model that describes a product of the SPL. This requires a mapping between features from a feature model and solution-space models or modelling artefacts that realise features (or combinations of those). We focus on model-driven development of SPLs in this paper and refer to solution-space models that are expressed by means of Ecore-based metamodels. To achieve the required mappings, a number of different approaches have been proposed [5, 9, 12] which allow for creating mappings between features from feature models and solution-space models.

While all of the approaches provide means for creating and maintaining required mappings, ensuring the *well-formedness* of all *input models* (i.e., feature models, mapping models, and solution-space models) and all possible *output models* (i.e., solution-space models transformed based on feature selection) is a challenging task often neglected in the aforementioned approaches. In this context, by well-formedness is meant the conformance of a given model with constraints of the underlying metamodel. Note, that well-formedness goes beyond syntactical correctness in the sense that it also takes additional constraints into account that are not directly expressed in the language's metamodel. Also, creating syntax errors in modelling languages is usually not directly possible, since modelling editors work on a different level of abstraction, where it is impossible to create model elements that do not conform to the concrete syntax of the modelling language. To give examples of models, which do not respect well-formedness rules and are, hence, invalid:

- A feature model can become invalid because of contradicting cardinalities, that is, if cardinalities of child features do not comply with the cardinality of their parent feature (e.g., an alternative feature where child features are mandatory).
- A mapping can reference invalid or non-existing model

FOSD'09, October 6, 2009, Denver, Colorado, USA.

elements, e.g., model elements that were removed or changed due to refactoring of the problem space or the solution space.

- An output model can be ill-formed due to mappings that do not take into account the constraints of the language used for modelling the solution space.

We argue, that for creating valid products of an SPL, the well-formedness of input and output models needs to be ensured in a systematic way and, possibly, (automatically) validated during modelling the SPL.

In this paper we discuss well-formedness of the different participating models in an SPL and present possibilities for ensuring the well-formedness of those models. Furthermore, we want to foster discussion of open and not yet addressed issues in well-formedness of SPLs motivating the FOSD community to address them in their research and development. During discussion of the identified possibilities for validation we refer to their realisation in the FeatureMapper [12, 21] SPL tool.

The rest of the paper is structured as follows: We introduce our tool FeatureMapper in Sect. 2 and provide necessary context. In Sect. 3 we discuss various possibilities for validating and enforcing well-formedness in SPLs and relate them to their implementation in FeatureMapper. We present open issues and possibilities for further research and development in Sect. 4 and refer to related work in the FOSD community in Sect. 5. Section 6 concludes the paper.

2. BACKGROUND

FeatureMapper [12, 10, 21] is an Eclipse-based tool that allows for mapping features from feature models to arbitrary modelling artefacts that are expressed by means of an Ecore-based language [19]. These languages include UML2 [15], domain-specific modelling languages (DSLs) defined using the Eclipse Modelling Framework (EMF) [19], and textual languages that are described using EMFText [11]. The mappings can be used to steer the product-instantiation process by allowing the automatic removal from the final product being generated of modelling artefacts that are not part of a selected variant.

An overview of defining an SPL and deriving a concrete product in FeatureMapper is shown in Fig. 1. To associate features or logical combinations of features (*feature expressions*) with modelling artefacts, the developer first selects the feature expression in the FeatureMapper and the modelling artefacts in her favourite modelling editor (e.g., TOPCASED [22]). Next, she applies the feature expression to the modelling artefacts via the FeatureMapper user interface (Step 1). During product derivation, this mapping is interpreted by a FeatureMapper transformation component. Depending on the result of evaluating the feature expression against the set of features selected in the variant (Step 2), the modelling elements are preserved or removed from the model (Step 3). Model elements that are not mapped to a specific feature expression are considered to be part of the core of the product line and are always preserved. In addition to product derivation, the mappings are used for visualisation purposes [10].

FeatureMapper uses cardinality-based feature models [7]. These models are instances of an Ecore-based feature metamodel. Features from these feature models are related to

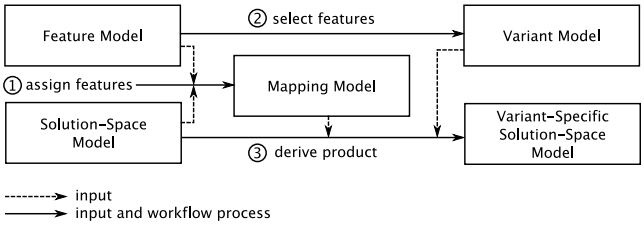

Figure 1: Workflow of defining an SPL and deriving a concrete product with FeatureMapper.

solution-space models through a dedicated Ecore-based mapping model. As depicted in Fig. 2, a mapping in this mapping model basically consists of a feature expression (`Expression`, which directly references features from the feature model or logical combinations of those) and a reference to a solution-space artefact (`EObject`).

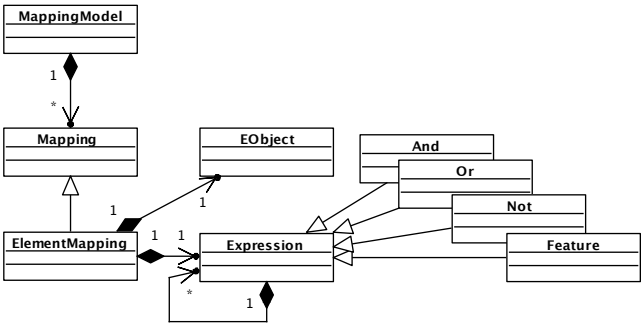

Figure 2: Simplified overview of FeatureMapper's internal mapping metamodel.

3. WELL-FORMEDNESS IN SPLS

As motivated in Sect. 1, ensuring well-formedness of all participating models is crucial in SPL to ensure the validity of the resulting products. Thus, validation of all models used for creating an SPL is needed. This includes validation of problem-space models, validation of mapping models, and validation of all possible solution-space models. In this section we give an overview of various possibilities for validation and relate them to their implementation in FeatureMapper. Our enumeration of possible validation tasks is by no means exhaustive; rather, it is expected to be extended as a result of future discussions.

3.1 Well-formed problem-space models

Cardinality-based feature models and variant models [7] include a set of well-formedness rules that need to be ensured for producing valid feature models. These rules are normally enforced by the feature-modelling tool used for producing the feature models and the variants. While FeatureMapper supports feature models and variant models of the feature-modelling tools pure::variants [3] and fmp [1], it also provides basic means to create those models, and thus, needs to ensure their validity.

FeatureMapper currently imposes the following constraints on feature models:

FM-Mandatory-Root The root feature must be mandatory. This constraint prohibits the creation of empty products by enforcing the inclusion of the root feature in any possible variant.

FM-Cardinality-Match Cardinalities of child features must comply with the cardinality of their parent feature. This constraint ensures that no contradicting cardinalities exist between child features and their parent features (e.g., an alternative feature that has mandatory child features).

FM-Sound-Reference References such as `requires` or `conflicts` must be non-contradicting. This constraint also includes the parent-child relationship between features during validation.

FM-Existing-Reference Referenced features must exist. This constraint ensures that any of the referenced features in `requires` or `conflicts` references are features in the feature model.

Variant models are seen as a subset of feature models in FeatureMapper. For ensuring valid variant models, the following constraints are enforced:

VM-Mandatory-Parent If a child feature is selected, the parent feature must be selected too.

VM-Mandatory-Child If a feature is selected, all its mandatory child features must be selected too.

VM-Alternative If an alternative feature (a parent feature with a cardinality `[0..1]`) is selected, at most one of its child features must be selected.

VM-Or If an Or feature (a parent feature with a cardinality `[n..m]`) is selected, at least `n` and at most `m` of its child features must be selected.

VM-Requires If the selection of a feature requires the selection of another feature, the latter feature must be selected too.

VM-Conflicts If the selection of a feature excludes the selection of another feature, the former feature cannot be selected.

Due to the added complexity involved when validating feature references (cf. constraints *FM-Sound-Reference*, *VM-Requires*, *VM-Conflicts* listed above), we enhanced our initial OCL-based approach for validating feature models and variant models in FeatureMapper to an Web Ontology Language (OWL) based approach similarly to what is described in [23]. This validation is exposed as a validator to the EMF Validation Framework. Additionally, FeatureMapper checks for invalid feature combinations while creating feature expressions and reports possible violations of the constraints listed above to the user.

3.2 Well-formed mapping models

Mapping models are models that relate features from feature models to their realisation in solution-space models. This mapping works directly on the referenced objects and not only on symbolic representations. Since FeatureMapper intentionally uses a generic mapping model to be independent of the modelling languages used, there exist basically two well-formedness rules that need to be ensured while creating and managing a mapping model:

MM-Existing-Feature Referenced features of a mapping must exist.

MM-Existing-ModelElement Referenced solution-space artefacts of a mapping must exist.

FeatureMapper ensures these constraints automatically during loading and saving of mapping models. If a constraint violation is detected, FeatureMapper informs the modeller and provides interactive means for correcting the model. Thus, FeatureMapper prohibits the creation of invalid mapping models. Note, that this does not imply the well-formedness of the solution-space models which will be addressed in the next subsection.

3.3 Well-formed solution-space models

The most challenging task in creating model-based SPLs is to ensure that all output models conform to the well-formedness rules of the language used for creating the solution-space models. In FeatureMapper, model elements are removed from the model during product derivation if the corresponding feature expression in the mapping does not evaluate to `true` against a given variant model. Checking the well-formedness of a output model is usually done by evaluating OCL constraints on the output model. For an SPL this would imply that any possible variant needs to be created to ensure the well-formedness of the complete SPL. This is not feasible because of the large amount of possible variants that can be created out of an SPL [16]. We focus on checking the well-formedness of an SPL, not its individual products. This includes the following constraint classes:

SM-Multiplicity Multiplicities of modelling artefacts must match the multiplicities of the respective constructs described in the metamodel of the language used for modelling the solution-space models.

SM-Typing Solution-space models must conform to the modelling language's type system.

SM-Semantics Solution-space models must conform to specific semantic constraints exposed by the used modelling languages which do not fall in any of the aforementioned classes. This also applies to domain-specific languages, which can imply constraints on solution-space models that are intrinsic to a specific domain.

To our knowledge, there currently exists no approach that allows for ensuring constraints of all three constraint classes for models created in arbitrary Ecore-based modelling languages. In [8], Czarnecki and Pietroszek presented an approach for verifying feature-based model templates against well-formedness OCL constraints. In their work, they described how the well-formedness of UML models annotated with stereotypes containing feature expressions (so-called *presence conditions*) can be ensured for all possible variants of an SPL without creating any of those variants. They described how OCL well-formedness rules can be interpreted in a way that takes the feature expressions into account and provided a set of evaluation patterns for various OCL constructs in form of propositional formulas.

While this approach and its implementation only addresses well-formedness rules of UML, well-formedness rules differ significantly depending on the modelling languages used. E.g., in UML each `DecisionNode` in an activity model must

at least have one outgoing edge whereby the guards on the outgoing edges need to be unambiguous—otherwise a race condition may occur (cf. constraint-class *SM-Semantics*). Figure 3 depicts a part of an example activity diagram from a recent case study we performed in FeatureMapper, where modelling elements are coloured according to the feature expression assigned to them. The outgoing edge coloured in red is removed whenever the respective features are not part of the given variant. The same applies to the outgoing edge coloured in blue.

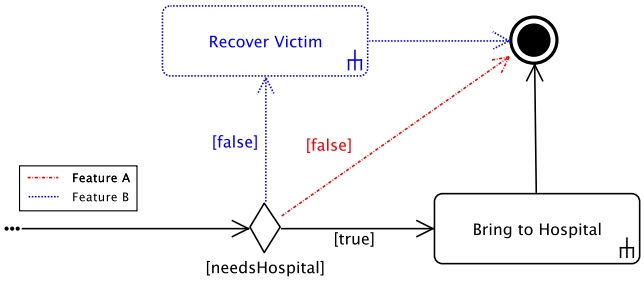

Figure 3: Example of an ambiguous activity model with two outgoing edges with the same guard condition.

Another example is, that an `Association` must have at least two `memberEnds` (cf. constraint-class *SM-Multiplicity*).[1] The detail of the class model depicted in Fig. 4 shows an `Association` where this constraint is not fulfilled in case the class `Blanket` is removed from the model. Of course, fixing those errors can be fairly easy (in this case the association needs to be removed too) but detecting those errors is a task that is not easy to perform, especially when considering cross-model constraints (e.g., each method called in a UML activity model must exist in a related UML class model; cf. constraint-class *SM-Typing*).

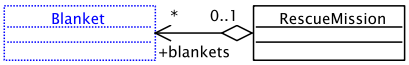

Figure 4: Example of a possible violation of the well-formedness rules of the UML `Association` concept.

Similarly to the `Association` in UML, the `eReferenceType` of an Ecore `EReference` must exist. The specifications [15, 19] contain numerous well-formedness rules based on multiplicities and OCL constraints. Presenting all of those is beyond the scope of this paper. In addition to established and widely-used modelling-languages, the trend towards defining and using DSLs in model-driven development results in numerous new languages, where each of those languages has their own set of well-formedness rules. For example, a language for creating forms can include the concept of depending questions (i.e., a question only has to be answered if a specific other question has been answered). This again involves the concept of references, where each of the referenced questions in a form description must exist.

[1]This is the running example of Czarnecki and Pietroszek in [8].

We are currently investigating possible extensions to FeatureMapper for a modelling-language independent realisation of checking the entire SPL. Since FeatuerMapper is intentionally agnostic to the modelling-languages used, any existing Ecore-based modelling-language can be used for creating solution-space models. This also means that the well-formedness rules of these different languages need to be ensured for each particular language to be supported. Our aim is at creating a generic framework that can be parameterized with those language-specific rules. To this end, ensuring well-formedness of SPLs built of arbitrary Ecore-based modelling languages becomes possible.

4. DISCUSSION

An open issue in ensuring well-formedness of SPLs is the lack of completeness of formally described well-formedness rules. The UML specification contains a lot of explicitly described multiplicities, well-formedness rules, and additional constraints but also contains implicit information (e.g., the need for unambiguousness of multiple outgoing edges on `DecisionNodes` as described in Sect. 3). To our knowledge, no catalogue of formalised descriptions (e.g., described using OCL) of those well-formedness rules exists at the moment. Even current modelling tools have a fairly relaxed interpretation of those rules and effectively allow for creating ill-formed models. Creating a complete catalogue of those well-formedness rules seems to be a complex but also very profitable task, because to this end, checking the well-formedness of all participating models of a given language is possible. To extend this idea, having such catalogues for different languages can foster reusing certain rules that are shared across languages whenever language semantics are appropriate. This is especially promising for model-driven development including multiple DSLs, where certain domain-specific constraints need to be ensured across language boundaries.

There exist a whole range of opportunities for performing additional checks on SPLs that go beyond well-formedness rules. Possible checks include detection of bad smells, i.e., violations of modelling conventions. Examples of those are direct communication between components instead of using dedicated interfaces in component models, huge inheritance hierarchies, or strong coupling of classes in class models. Possible extended checks on problem-space models can include ensuring that all features of a given feature model are actually mapped to solution-space artefacts (i.e., ensuring that there exists a realisation of a given feature in the solution-space models).

An open issue of ensuring the validity of an SPL based on different interpretation of OCL well-formedness rules is the performance of checking all propositional formulas. As described by Czarnecki and Pietroszek [8], checking those rules cannot be instantly performed due to the processing time of creating and checking those rules (in their experience, checking is performed in terms of seconds rather than milliseconds). Possible enhancements are incremental checks, where the engine detects which constraints and which parts of a model need to be verified in case of changes in the participating models. Another implication of the approach is the semantics of the mapping. It seems that this approach is feasible for mappings that relate to modelling artefacts and remove those depending on a given feature selection. Approaches that apply complex transformations based on

feature selection actually change the model in ways that are not easily verified using propositional formulas. Further research in this direction is needed.

Another widely unexplored field—which is not addressed in this paper—is detecting semantic errors for all products of an SPL. Since it is already difficult to ensure the semantic correctness of a single product, checking the semantic correctness of all possible products on an SPL is an open issue.

5. RELATED WORK

There is a whole body of work that addresses quality and safety of product lines. Some of the existing works in this field do not check the SPL itself but the distinct products that can be created out of an SPL [17]. The problem with testing all products is a large number of different products that are possible with already a fair amount of independent optional features (for n optional features, 2^n distinct variants are possible). This implies that in those cases not all possible variants are checked. Instead, only products that are actually created out of the SPL or combinatorial samples are considered which again means that a lot of repetitive inspection is needed compared to ensuring the well-formedness of the SPL itself.

As already mentioned in Sect. 3, some approaches check the SPL itself. Czarnecki and Pietroszek [8] address the problem of ensuring the validity of any possible solution-space models by checking those models against well-formedness OCL constraints. Their solution describes how well-formedness OCL constraints can be interpreted based on propositional formulas by taking into account feature expressions mapped to modelling elements. Similarly, Thaker et al. [20] use propositional formulas and SAT solvers to ensure safe composition of feature modules in the AHEAD system. In this paper we proposed extending those existing solutions to models defined in arbitrary Ecore-based languages.

In [14], Kästner et al. present an approach for guaranteeing syntactic correctness of all possible variants of an SPL. In contrast to what we discussed in this paper, their work is based on programming languages where syntax errors (such as omitting a necessary closing bracket) can easily occur. This is not the case for modelling languages since modelling editors work on a different level of abstraction where it is not possible to create model elements that do not conform to the concrete syntax of the modelling language.

In [18], Seifert and Samlaus present an approach for static analysis of source code using OCL. They present *RestrictED*, an extensible editor for textual modelling languages based on EMFText [11] that can be parametrized with language-specific constraints for checking source code modelled with EMFText languages. Our idea of creating a generic framework that can be parameterized with modelling-language specific well-formedness rules is an extension this idea, taking into account mapping information between feature models and solution-space models. Furthermore, we aim at creating a framework that abstracts from the concerte-syntax representation of the specific modelling languages (i.e., graphical or textual concrete syntax) and handles them in a uniform way.

6. CONCLUSION

In this paper we discussed various possibilities to check all participating models of an SPL against well-formedness rules defined on the metamodels that are used to create those models. We discussed existing approaches for ensuring the validity of models for all the concrete products that can be created out of an SPL as well as open issues and future work. Throughout the paper, we related the identified possibilities for ensuring well-formedness to their implementation in the FeatureMapper SPL tool and outlined our plans to integrate existing work to uniformly check well-formedness of SPLs with this tool.

Acknowledgements

We thank Ilie Şavga and Christian Wende for their valuable comments on earlier drafts of this paper. This research has been partly co-funded by the German Ministry of Education and Research (BMBF) within the project feasiPLe.

7. REFERENCES

[1] M. Antkiewicz and K. Czarnecki. FeaturePlugin: Feature Modeling Plug-In for Eclipse. In *Proceedings of the OOPSLA workshop on Eclipse technology eXchange (ETX)*, pages 67–72, New York, NY, USA, 2004. ACM.

[2] S. Apel and C. Kästner. An Overview of Feature-Oriented Software Development. *Journal of Object Technology (JOT)*, 8(5):49–84, July/August 2009.

[3] D. Beuche, H. Papajewski, and W. Schröder-Preikschat. Variability Management with Feature Models. *Science of Computer Programming*, 53(3):333–352, 2004.

[4] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[5] K. Czarnecki and M. Antkiewicz. Mapping Features to Models: A Template Approach Based on Superimposed Variants. In *Proceedings of the 4th International Conference on Generative Programming and Component Engineering (GPCE'05)*, pages 422–437, 2005.

[6] K. Czarnecki and U. W. Eisenecker. *Generative Programming – Methods, Tools, and Applications*. Addison-Wesley, June 2000.

[7] K. Czarnecki and C. H. P. Kim. Cardinality-Based Feature Modeling and Constraints: A Progress Report. In *Proceedings of the OOPSLA'05 International Workshop on Software Factories*, 2005.

[8] K. Czarnecki and K. Pietroszek. Verifying Feature-Based Model Templates Against Well-Formedness OCL Constraints. In S. Jarzabek, D. C. Schmidt, and T. L. Veldhuizen, editors, *Proceedings of the 5th International Conference on Generative Programming and Component Engineering (GPCE'06)*, pages 211–220. ACM, 2006.

[9] Ø. Haugen, B. Møller-Pedersen, J. Oldevik, G. Olsen, and A. Svendsen. Adding Standardized Variability to Domain Specific Languages. In *Proceedings of the 12th International Software Product Line Conference (SPLC'08)*, pages 139–148. IEEE, 2008.

[10] F. Heidenreich, I. Şavga, and C. Wende. On Controlled Visualisations in Software Product Line Engineering. In *Proceedings of the 2nd International*

Workshop on Visualisation in Software Product Line Engineering (ViSPLE'08), collocated with the 12th International Software Product Line Conference (SPLC'08), Sept. 2008.

[11] F. Heidenreich, J. Johannes, S. Karol, M. Seifert, and C. Wende. Derivation and Refinement of Textual Syntax for Models. In R. F. Paige, A. Hartman, and A. Rensink, editors, *Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA'09)*, volume 5562 of *LNCS*, pages 114–129. Springer, 2009.

[12] F. Heidenreich, J. Kopcsek, and C. Wende. FeatureMapper: Mapping Features to Models. In *Companion Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, pages 943–944, New York, NY, USA, May 2008. ACM.

[13] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-0211990, Software Engineering Institute, 1990.

[14] C. Kästner, S. Apel, S. Trujillo, M. Kuhlemann, and D. Batory. Guaranteeing Syntactic Correctness for all Product Line Variants: A Language-Independent Approach. In *Proceedings of the 47th International Conference Objects, Models, Components, Patterns (TOOLS EUROPE'09)*, volume 33 of *Lecture Notes in Business Information Processing*, pages 175–194. Springer Berlin Heidelberg, June 2009.

[15] Object Management Group. UML 2.2 infrastructure specification. OMG Document, Feb. 2009. URL http://www.omg.org/spec/UML/2.2/.

[16] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005.

[17] K. Pohl and A. Metzger. Software Product Line Testing. *Communications of the ACM*, 49(12):78–81, 2006.

[18] M. Seifert and R. Samlaus. Static Source Code Analysis using OCL. In J. Cabot and P. Van Gorp, editors, *Proceedings of the Workshop OCL Tools: From Implementation to Evaluation and Comparison (OCL'08), co-located with the 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS'08)*.

[19] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *Eclipse Modeling Framework, 2nd Edition*. Pearson Education, 2008.

[20] S. Thaker, D. Batory, D. Kitchin, and W. Cook. Safe Composition of Product Lines. In *Proceedings of the 6th International Conference on Generative Programming and Component Engineering (GPCE'07)*, pages 95–104, New York, NY, USA, 2007. ACM.

[21] The FeatureMapper Project Team. FeatureMapper, July 2009. URL http://www.featuremapper.org.

[22] The Topcased Project Team. TOPCASED, July 2009. URL http://www.topcased.org.

[23] H. H. Wang, Y. F. Li, J. Sun, H. Zhang, and J. Pan. Verifying feature models using OWL. *Web Semantics*, 5(2):117–129, 2007.