Multi-variant Model Transformations — A Problem Statement

Felix Schwägerl, Thomas Buchmann and Bernhard Westfechtel
Applied Computer Science I, University of Bayreuth, Universitätsstr. 30, 95440 Bayreuth, Germany

Keywords: Model-Driven Software Engineering, Software Product Lines, Model Transformations, Code Generation.

Abstract: Model Transformations are a key element of Model-Driven Software Engineering. As soon as variability is involved, transformations become increasingly complicated. The lack of support for variability in model transformations impairs the acceptance of approaches to organized reuse such as software product lines. In this position paper, the general problem of multi-variant model transformations is formulated for MOF-based, XMI-serialized models. A simplistic case study is presented to specify the input and the expected output of such a transformation. Furthermore, requirements for tool support are defined, including a standardized representation of both multi-variant model instances and variability information, as well as an execution specification for multi-variant transformations. A literature review reveals that the problem is weakly identified and often solved using ad-hoc solutions; there exists no tool providing a general solution to the proposed problem statement. The observations presented here may serve for the future development of standards and tools.

1 INTRODUCTION

In *Model-Driven Software Engineering* (MDSE) (Völter et al., 2006), *models* are developed as higher-level abstractions of software systems using well-defined modeling languages such as the *Unified Modeling Language* (UML) (OMG, 2011d). The *Object Management Group* (OMG) have developed the standard *Meta Object Facility* (MOF) (OMG, 2011b) for the definition of *metamodels*, which describe the abstract syntax and the static semantics of modeling languages. *XML Metadata Interchange* (XMI) (OMG, 2011c) serves as a standardized serialization format for MOF models, making them compliant with different modeling tools. The *Eclipse Modeling Framework* (EMF) (Steinberg et al., 2009) provides a basis for many model driven tools and is built around the *Ecore* metamodel, which implements a MOF subset.

Model Transformations (MT) are an indispensable building block for model-driven applications. They describe how a source model, represented as an instance of an input metamodel, is to be converted to a target model, conforming to the output metamodel. Depending on the models' representation, there exist *model-to-model* (M2M) and *model-to-text* (M2T) transformations. The OMG have approved several standards for MTs: *Queries-Views-Transformations* (QVT) (OMG, 2011a) is a family of M2M transformation standards, containing, e.g., the imperative language *QVT Operational* and its declarative counterpart *QVT Relational*. The *MOFM2T* standard (OMG, 2012b) has been implemented, e.g., by the EMF-based tool *Acceleo*. Apart from standardized transformation languages, several alternatives exist, e.g., *ATL* (Jouault and Kurtev, 2006).

In the context of *Model-Driven Architecture* (MDA) (Mellor et al., 2004), MTs are often chained. In MDA, a *platform independent model* (PIM) is refined to a platform specific model (PSM), from which runnable application code is generated. MDA's long-term goal is to completely remove the necessity of writing application code manually, considering modeling languages as an additional layer on top of assembly and programming languages.

Variability is a more and more important property of software to be developed, in order to meet requirements such as customizability, platform independence and maintainability. At source code level, variability is mostly realized by *preprocessor languages* such as the *C preprocessor*. *Software Product Line Engineering* (*SPLE*) systematically exploits variability to achieve organized reuse (Pohl et al., 2005). Core assets of different products are provided as a *platform*. Commonalities and differences among products are captured in *variability models*, e.g., *feature models* (Kang et al., 1990). These features are then connected to the underlying software by *traceability links*. The combination of MDSE with SPLE is subject to many

Schwägerl, F., Buchmann, T. and Westfechtel, B.
Multi-variant Model Transformations — A Problem Statement.
In *Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering (ENASE 2016)*, pages 203-209
ISBN: 978-989-758-189-2

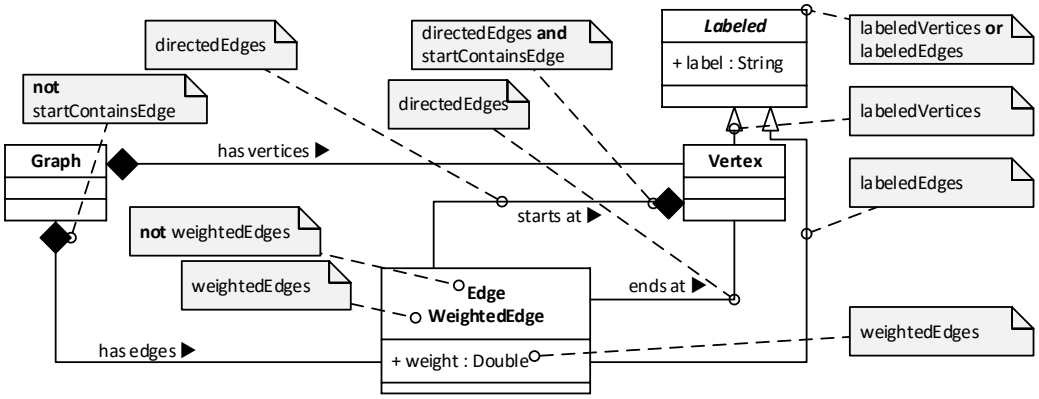

Figure 1: Example source model: multi-variant UML class diagram with traceability links attached as comments. For the reason of clarity, member end names and multiplicities have been omitted.

research activities, resulting in the integrating discipline *Model-Driven Product Line Engineering* (*MDPLE*) (Gomaa, 2004). In MDPLE, traceability links are realized by annotating model elements, e.g., using *presence conditions* (Czarnecki and Kim, 2005).

In this position paper, we investigate limitations and potentials of *Multi-Variant Model Transformations* (MVMT), which assume that the source model contains variability that shall be transferred to corresponding elements of the target model in an adequate way. Our core contribution is the identification of the following requirements being disregarded in current MDSE/M2M standards and tools:

R1. Interchange of multi-variant model instances.

R2. Representation of variability information.

R3. Execution specification of multi-variant model transformations.

The paper does not intend to present complete solutions for these requirements. Instead, we motivate their importance by an example and provide a short literature review where partial solutions are aligned with the problem statement.

Section 2 motivates the problem statement using a simplistic example, a M2T transformation for a multi-variant UML class diagram. In Section 3, the requirements outlined above are detailed. Section 4 contains the literature review. The paper is concluded in Section 5, which also addresses future research.

2 EXAMPLE

In this section, we present a simple example for a multi-variant model transformation, assuming that the requirements defined in the problem statement have already been met. We consider a *model-to-text* transformation, where the source model is a *multi-variant UML class diagram* that represents a cut-out of the static model of a *Graph* library – a standard example in SPLE literature (Lopez-Herrejon and Batory, 2001). Variability is expressed by comments attached to respective model elements. A *multi-variant M2T transformation* is applied to the source model. The expected outcome is provided as *multi-variant Java code*, where variability information is expressed by preprocessor-like constructs.

2.1 Source Model

Figure 1 shows the source model of the example MVMT, a multi-variant class diagram for graphs. In all variants, it is assumed that a Graph combines a Vertex and an Edge set. Edges connect a start with an end vertex. Furthermore, the model is connected to the following options by traceability links:

directedEdges. If selected, the associations *starts at* and *ends at* are included in the product. For the reason of clarity, the realization of *undirected* edges is not shown in the diagram. It could have been realized by an unspecific association connects with multiplicity 2.

labeledVertices. If selected, vertices have a string-valued label assigned. This is realized by an abstract class Labeled, which is specialized by Vertex only in case this option is active.

labeledEdges. Realizes labeled edges in analogy.

weightedEdges. If selected, the attribute weight of the edge class is visible. Furthermore, the class name depends on this option: if deselected, it is simply Edge, if selected WeightedEdge.

startContainsEdge. If selected, class Vertex will be the container for outgoing edges, otherwise class Graph will contain all edges.

2.2 Generated Code

Listing 1 shows the expected output of the example multi-variant model-to-text transformation. UML classes were mapped to Java classes; UML attributes and associations correspond to Java object variables, where special collection types are used to represent multi-valued non-containment and containment association ends. Furthermore, the traceability links were translated into corresponding preprocessor directives[1].

```
1  public class Graph {
2    public ContainmentList<Vertex> vertices;
3    public
4    #if startContainsEdge
5      ContainmentList<Edge>
6    #else
7      List<Edge>
8    #endif
9    edges;
10 }
11
12 #if labeledVertices or labeledEdges
13   public abstract class Labeled {
14     public String label;
15   }
16 #endif
17
18 public class Vertex
19 #if labeledVertices
20   extends Labeled
21 #endif
22 {
23   public Graph graph;
24   #if directedEdges
25     public
26     #if directedEdges and
       startContainsEdge
27       ContainmentList<Edge>
28     #else
29       List<Edge>
30     #endif
31     outgoing;
32     public List<Edge> incoming;
33   #else ...
34   #endif
35 }
36
37 public class
38 #if weightedEdges
39   WeightedEdge
40 #else
41   Edge
42 #endif
43 #if labeledEdges
44   extends Labeled
45 #endif
46 {
47   public Graph graph;
48   #if directedEdges
49     public Vertex start;
50     public Vertex end;
51   #else ...
52   #endif
53   #if weightedEdges
54     public double weight;
55   #endif
56 }
```

Listing 1: Expected output of the example MVMT in a condensed Java syntax using preprocessor-like directives for conditional elements.

[1]Unlike C/C++, Java does not provide a built-in preprocessor. For this example, without the loss of generality, we assume that conditional compilation is performed by an external tool that understands preprocessor directives containing boolean combinations (and/or/not) of options.

2.3 Desired Properties

From the example provided above, we may informally derive three properties to be satisfied in a best possible way by MVMT: *reuse* of transformation specifications, *orthogonality* of transformed models and traceability links, and *commutativity* of the transformation and a filter operation.

P1: Reuse. Above, we have tacitly assumed that the expected output has been produced based upon an existing MOFM2T specification, which describes the transformation of UML class diagrams into Java code. Thus, we argue that every MT should be *reusable* as a MVMT. Reuse does not refer to the specific *transformation engine*, which is in general not aware of variability, but only to the transformation specification itself. Thus, to enable reuse, a variability-aware transformation engine is desirable.

P2: Orthogonality. Furthermore, we assume that there exists an additional M2T transformation to convert traceability links of the source model into a form understood by the target model, e.g. preprocessor-like directives. This transformation is referred to as *tracelink transformation* below. We assume that the tracelink transformation is *orthogonal* to the source-to-target transformation: Different modeling languages and variability languages can be mixed arbitrarily, without affecting each other.

P3: Commutativity. The third property is concerned with the interaction of the MVMT itself and a *filter* operation (which may be considered as and realized by another model transformation). When assuming the special case of SPL using *negative variability*, the filter operation takes as input a multi-variant model and a *configuration*, which binds each

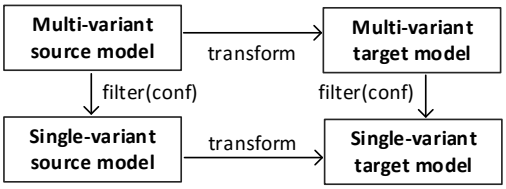

Figure 2: Commutativity of MVMT and product derivation.

option/feature to a boolean selection value, and produces a single-variant model where those elements are filtered out whose tracelinks evaluate to a negative value when applying the configuration. The property of *commutativity* states that the order of application of transformation and filter, always assuming an equal configuration, is immaterial (cf. Figure 2). In SPLE, it is preferable to filter the product as late as possible, since filtering marks the transition from *domain* to *application engineering*. The later the filter is applied, the more reusable the applied refinement steps are.

3 REQUIREMENTS

Obviously, the example presented in Section 2 cannot be conducted using state-of-the-art model transformation languages and tools. In this section, we discuss the missing pieces, i.e., *requirements*, which must be addressed by future research in order to maintain the properties **P1** until **P3** listed above. These requirements include a standard for the *interchange of multi-variant model instances* (**R1**, see Section 3.1), a standard for *representation of variability information* (**R2**, Section 3.2), and *MVMT execution specificaton* (**R3**, Section 3.3). This section details those requirements without aiming to propose a standard at this early stage of understanding the problem of MVMT.

3.1 Interchange of Multi-variant Model Instances (R1)

With XMI (OMG, 2011c), the OMG have issued an XML-based standard for the serialization of MOF-based model instances. XMI is understood by the majority of modeling tools, allowing to interchange model instances across them. Listing 2 shows a cut-out of the XMI-serialization of the example class Edge, excluding traceability links.

Listing 2 ignores one detail shown in Figure 1: The alternative class name WeightedEdge. Basically, when using XMI, we are constrained by the metamodel. In our example, the UML2 metamodel assigns the upper bound 1 to the multiplicity of the attribute name of uml:Class. We argue that single-variant modeling restrictions such as cardinality or

```
1 <packagedElement xmi:type="uml:Class"
      name="Edge">
2   <generalization general="Labeled"/>
3   <ownedAttribute name="graph"
      type="Graph" association="has edges"/>
4   <ownedAttribute name="start" type="Edge"
      association="starts at"/>
5   <ownedAttribute name="end" type="Edge"
      association="ends at">
6 </ownedAttribute>
7 </packagedElement>
```

Listing 2: XMI serialization of class Edge using the Eclipse UML2 metamodel.

```
1 <packagedElement xmi:type="uml:Class"
      name="Edge,WeightedEdge">
2 ... </packagedElement>
```

Listing 3: Multiple alternative values for a class.

typing constraints must be weakened when serializing multi-variant model instances. In our example, the assignment shown in Listing 3 should be allowed.

Generally, multi-variant models will comply only with tools understanding the multi-variant format. However, single-variant tools may transparently access multi-version models through a *filter*.

3.2 Representation of Variability Information (R2)

Let us assume that the problem of multi-variant XMI serialization has been solved. Then, there is still no variability defined in XMI documents. The problem of representing variability information, i.e., traceability links, must be treated as a separate requirement.

Different tools capable of variability management handle traceability links in a non-uniform way, e.g., using UML comments (as in the example above), UML profiles and/or distinct mapping models (see literature review in Section 4). We argue that variability information must be serialized in a uniform and standardized way, which should be independent of the concrete modeling language used. In Listing 4, we suggest a tentative solution based on a new *meta-attribute* mvxmi:tracelink[2]. The range of this attribute must be precisely defined in future research. Here, we stick to the "least common denominator" of propositional logical expressions.

By intention, we have omitted the multi-variant attribute name, which raises an additional challenge: How to attach traceability links to attribute values, which are considered as atomic strings in the low-level XMI serialization? Among others, this technical question must be answered by future research.

[2]mvxmi would stand for *multi-variant XMI*.

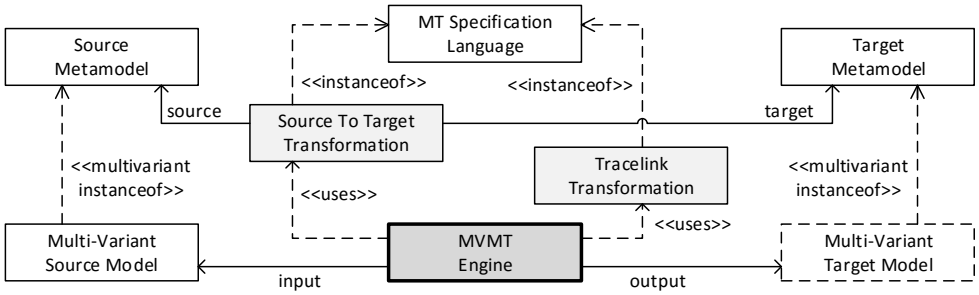

Figure 3: Sketch of a multi-variant model transformation engine.

```
<generalization general="Labeled"
    mvxmi:tracelink="labeledEdges"/>
<ownedAttribute name="graph" type="Graph"
    association="has edges"/>
<ownedAttribute name="start" type="Edge"
    association="starts at"
    mvxmi:tracelink="directedEdges"/>
<ownedAttribute name="end" type="Edge"
    association="ends at"
    mvxmi:tracelink="directedEdges">
</ownedAttribute>
</packagedElement>
```

Listing 4: Multi-variant XMI serialization of class Edge, including variability information as traceability links.

3.3 MVMT Execution (R3)

Only after having found solutions for requirements **R1** and **R2** may we ask the question of execution semantics: How to produce the target model from the source model while ensuring the desired properties **P1** until **P3** listed in Section 2.3? Obviously, MVMT support may only be achieved by a (partial) re-implementation of an existing MT execution engine. The interface of such a hypothetical *MVMT engine* is sketched in Figure 3.

The engine takes as input a multi-variant instance of the input meta-model and produces a multi-variant instance of the output metamodel. The relationship <<multivariantinstanceof>> must be specified in accordance with Requirement **R1**. As in a single-variant MT, the transformation engine processes the input using a transformation specification which refers to the source and target metamodels. In addition, in the multi-variant case, a transformation specification for the traceability links is needed, e.g., in order to define how traceability links originating from several source elements are merged. Both transformation specification are described by means of an existing MT language specification, which need not be aware of its instances being executed in a multi-variant context.

4 LITERATURE REVIEW

In this section, we outline state-of-the-art solutions to parts of our problem statement, structured by the requirements **R1** until **R3** identified in Section 3.

R1: Multi-Variant Model Interchange. Since there exists no standard for the representation of multi-variant model instances, different tools dealing with MDSE and variability apply custom solutions. An established compromise in the context of MDPLE is that the multi-variant domain model must be instance of the single-variant metamodel. This removes the necessity of an explicit multi-variant interchange format but constrains variability; for instance, alternative class names as shown in Figure 1 would be disallowed. Nevertheless, this approach is realized in the majority of MDPLE approaches, including (Gomaa, 2004; Heidenreich et al., 2008).

The approach presented in (Buchmann and Schwägerl, 2015a) mitigates this restriction by moving alternative values into a distinct *mapping model*, which also contains traceability links. However, this solution is specific to the concrete tool.

Another category of MDPLE tools relying on *positive variability* (Zschaler et al., 2010; Völter and Groher, 2007) describe multi-variant models as a set of model transformations to be conditionally applied to a core model, a valid single-variant model instance. This approach lacks generality since specific transformation tools are needed.

The problem of multi-version model representation also arises in *model versioning* (Altmanninger et al., 2009). In approaches relying on *directed deltas*, different versions are described by means of conditional *graph modifications* (Taentzer et al., 2014). In (Schwägerl et al., 2015), *versioned model graphs* are introduced as a form of representation for *symmetric deltas*. These approaches, however, lack generality, since they are restricted to version control and not designed as interchange format, but merely for internal, tool-specific representation.

R2: Variability Representation. Many SPLE tools, including (Buchmann and Schwägerl, 2015a; Heidenreich et al., 2008), use their own, non-standardized tools to represent variability and traceability links. Variability itself is commonly represented by *feature models* (Kang et al., 1990), for which many derivatives can be found in literature, leading to a quite heterogeneous tool landscape. The expressiveness of approaches trying to unify traceability links ranges from simple *conjunctions* (Gomaa, 2004) over *propositional logic* (Westfechtel and Conradi, 2009) to *feature logic*, which has been suggested as a foundation for SCM in (Zeller and Snelting, 1997).

With the *Common Variability Language* (CVL) (OMG, 2012a), the OMG are working on a standard for representing variability information within models. In contrast to multi-variant models as defined here, CVL assumes that *variation points* are explicitly included in the model and expressed by specific modeling constructs. This is in contradiction with the desired property **P2** (orthogonality). However, CVL's OCL[3]-based *variability constraints* provide a higher expressiveness than all approaches mentioned above.

R3: MVMT Execution. Specialized forms of MVMT have been realized within SPLE tools. Though, we haven't found a general solution having the qualities of our proposed MVMT engine.

The solution presented in (Buchmann and Schwägerl, 2015b) is specific to transformations from Ecore models to Java source code. A MVMT is simulated by applying single-variant code generation to the multi-variant model and then propagating back generated code into the source model. This way, lower-level source code elements such as object variables and accessor methods are provided with the same variability information as their higher-level correspondences, e.g., Ecore references. However, this solution does not address *applied occurrences* of variable parts in manually implemented method bodies.

(Salay et al., 2014) agree with our observation that MDPLE is currently impeded by the lack of support for variability in model transformations. They propose a *lifting algorithm* which interprets existing model transformation rules, which are assumed to be based on a graph rewriting formalism, on a model that contains variability. Properties **P1** (*reuse*) and **P3** (*commutativity*) are guaranteed. The solution is fully variability-aware; However, variability is constrained by single-version metamodels, disallowing "unconstrained" input/output models (e.g., Figure 1).

[3] *Object Constraint Language* (OMG, 2014)

An approach combining the aforementioned CVL with *domain specific languages* is presented in (Haugen et al., 2008). As required by CVL, variation points are contained in the abstract syntax tree of model instances. However, the approach does not include a transfer of variability information to the target model as requested by our problem statement.

5 CONCLUSION AND OUTLOOK

This position paper has asked initial questions referring to the problem of *multi-variant model transformations*, including a discussion of the requirements of a standardization of (**R1**) multi-variant model interchange, (**R2**) representation of variability information and (**R3**) execution semantics. The long-term research goal is to develop a variability-aware model transformation engine that (**P1**) reuses existing single-variant transformation specification, (**P2**) separates the core transformation from the transformation of variability information, and (**P3**) maintains as well as possible the intuitively desirable property of *commutativity* with a *filter* operation. Furthermore, we have provided a simple yet challenging example for MVMT, which may serve as a starting point for the evaluation of an eventual MVMT engine.

From a literature review, we have learned that all of the three requirements defined here have already been addressed by existing tools and approaches, but not to a satisfactory extent. Most solutions lack generality and do not identify MVMTs as an explicit research problem. However, we are convinced that a general solution for MVMTs may raise the acceptance of disciplines such as Model-Driven Product Line Engineering, which explicitly defines the goal of organized reuse. An MVMT engine would lift the goal of organized reuse up to the tooling level, allowing to more easily adapt single-variant tool chains to multi-variant scenarios. Especially when combined with reverse or round-trip engineering, new challenges will soon arise for MVMT, e.g., support for *bidirectional* or *incremental* model transformations. Furthermore, the requirement of being "variability-aware" is not restricted to model transformations, but will gain importance in, e.g., model validation or version control.

REFERENCES

Altmanninger, K., Seidl, M., and Wimmer, M. (2009). A survey on model versioning approaches. *Interna-*

tional Journal of Web Information Systems (IJWIS), 5(3):271–304.

Buchmann, T. and Schwägerl, F. (2015a). Developing heterogeneous software product lines with FAMILE — a model-driven approach. *International Journal on Advances in Software*, 8(1 & 2):232 – 246.

Buchmann, T. and Schwägerl, F. (2015b). On A-posteriori Integration of Ecore Models and Hand-written Java Code. In Pascal Lorenz, M. v. S. and Cardoso, J., editors, *Proceedings of the 10th International Conference on Software Paradigm Trends*, pages 95–102. SCITEPRESS.

Czarnecki, K. and Kim, C. H. P. (2005). Cardinality-based feature modeling and constraints: a progress report. In *International Workshop on Software Factories at OOPSLA'05*, San Diego, California, USA. ACM.

Gomaa, H. (2004). *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Addison-Wesley, Boston, MA.

Haugen, O., Møller-Pedersen, B., Oldevik, J., Olsen, G. K., and Svendsen, A. (2008). Adding standardized variability to domain specific languages. In *Proceedings of the 2008 12th International Software Product Line Conference*, SPLC '08, pages 139–148, Washington, DC, USA. IEEE Computer Society.

Heidenreich, F., Kopcsek, J., and Wende, C. (2008). FeatureMapper: Mapping features to models. In *Companion Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, pages 943–944, Leipzig, Germany.

Jouault, F. and Kurtev, I. (2006). Transforming models with atl. In *Proceedings of the 2005 International Conference on Satellite Events at the MoDELS*, MoDELS'05, pages 128–138, Berlin, Heidelberg. Springer-Verlag.

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990). Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Carnegie-Mellon University, Software Engineering Institute.

Lopez-Herrejon, R. E. and Batory, D. S. (2001). A standard problem for evaluating product-line methodologies. In *Proceedings of the Third International Conference on Generative and Component-Based Software Engineering*, GCSE '01, pages 10–24, London, UK. Springer.

Mellor, S. J., Kendall, S., Uhl, A., and Weise, D. (2004). *MDA Distilled*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

OMG (2011a). *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.1*. Object Management Group, Needham, MA.

OMG (2011b). *Meta Object Facility (MOF) Core*. Object Management Group, Needham, MA.

OMG (2011c). *OMG MOF 2 XMI Mapping Specification, Version 2.4.1*. Object Management Group, Needham, MA.

OMG (2011d). *UML Infrastructure*. Object Management Group, Needham, MA.

OMG (2012a). *Common Variability Language (CVL) OMG Revised Submission*. Object Management Group, Needham, MA.

OMG (2012b). *MOF Model to Text Transformation Language, Version 1.0*. Object Management Group, Needham, MA.

OMG (2014). *Object Constraint Language (OCL)*. Object Management Group, Needham, MA.

Pohl, K., Böckle, G., and van der Linden, F. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*. Berlin, Germany.

Salay, R., Famelis, M., Rubin, J., Sandro, A. D., and Chechik, M. (2014). Lifting model transformations to product lines. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 117–128.

Schwägerl, F., Uhrig, S., and Westfechtel, B. (2015). A graph-based algorithm for three-way merging of ordered collections in EMF models. *Science of Computer Programming*, 113, Part 1:51 – 81. Selected and Revised Papers from MODELSWARD 2014.

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2009). *EMF Eclipse Modeling Framework*. The Eclipse Series. Boston, MA, 2nd edition.

Taentzer, G., Ermel, C., Langer, P., and Wimmer, M. (2014). A fundamental approach to model versioning based on graph modifications: From theory to implementation. *Software & Systems Modeling*, 13(1):239–272.

Völter, M. and Groher, I. (2007). Handling variability in model transformations and generators. In *Companion to the Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2007)*. ACM.

Völter, M., Stahl, T., Bettin, J., Haase, A., and Helsen, S. (2006). *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons.

Westfechtel, B. and Conradi, R. (2009). Multi-variant modeling - concepts, issues and challenges. In Mezini, M., Beuche, D., and Moreira, A., editors, *Proceedings of the 1st International Workshop on Model-Driven Product Line Engineering (MDPLE 2009)*, pages 57–67. CTIT Proceedings.

Zeller, A. and Snelting, G. (1997). Unified versioning through feature logic. *ACM Trans. Softw. Eng. Methodol.*, 6(4):398–441.

Zschaler, S., Sánchez, P., Santos, J., Alférez, M., Rashid, A., Fuentes, L., Moreira, A., Araújo, J., and Kulesza, U. (2010). VML* - A Family of Languages for Variability Management in Software Product Lines. In van den Brand, M., Gaevic, D., and Gray, J., editors, *Software Language Engineering*, volume 5969 of *Lecture Notes in Computer Science*, pages 82–102. Springer Berlin / Heidelberg, Denver, CO, USA.