Towards Separation of Concerns in Model Transformation Workflows

Christoph Elsner
Siemens Corporate Technology
Software & Engineering 2
christoph.elsner.ext@siemens.com

Daniel Lohmann   Wolfgang Schröder-Preikschat
Friedrich-Alexander University, Erlangen-Nuremberg
{lohmann,wosch}@informatik.uni-erlangen.de

Abstract

Model-driven software product lines are an emerging topic in research and industry, as they promise higher development speed and easier adaptability to customer needs. The generation facilities for the products, however, still have a static nature: specification languages for the composition of model transformations sequences (model transformation workflows) up to now only support the specification of simple control flows with conditional execution. Thus, separation of concerns with respect to variable product features is impossible. To flexibly compose model transformation workflows depending on the feature selection, we express them by a dedicated and clearly scoped domain-specific modeling language (DSML). This facilitates to use aspect-oriented model weaving techniques to preprocess the workflow depending on the concrete product configuration and, thus, prevents tangling of concerns in the base workflow artifacts.

This position paper motivates the need for aspect-oriented model transformation orchestration during application engineering and presents our concepts to achieve this goal.

1. Introduction

Model-driven development (MDD) helps to achieve significant productivity gain through model-based abstraction within the analysis, architecture, and design of a software system. Application generation is usually performed in several steps; for instance, the Model-Driven Architecture (MDA) [4] proposes a computational-independent model (CIM), which is gradually transformed: first to a platform-independent model (PIM), then to a platform-specific model (PSM), and finally to application code.

Though this coarse-grained scheme is very useful to classify different stages of application generation, the concrete approach chosen can vary greatly depending on the application domain. For example, there can be intermediate transformation steps and metamodels or additional consistency checks in various stages. However, the approaches considered in this paper are more general than the MDA, which is restricted to MOF [20] metamodeling and QVT [21] transformations.

There already has been research about the specification of model transformation sequences, and the terminology differs depending on the authors. We discovered the notions of *model transformation chain* [25], *composition of model transformations* [16], *modeling workflow* [19], *model refinement line* [30], and *model service orchestration* [10]. In this paper, we will use the term *modeling workflow* or, in short, *workflow*. It emphasizes that there may be other tasks to specify than only model transformations, for example model loading, storing, checking, or code generation.

After evaluating current research and workflow specification tools, we identified the lack of advanced techniques for separation of concerns, what especially hinders the flexible and modularized specification of variability in workflows. This, however, is particularly important in the context of software product line engineering [23], where different software products are generated based on a common set of core assets. Depending on the selected product features [13], which usually denote concerns or parts of concerns, also the concrete workflow will vary. As this is currently usually done by conditional expressions embedded in one workflow artifact, this approach reaches its limits when a lot of features require different workflow behavior. This may finally result in cascades of conditional expressions similar to the *#ifdef* constructs of the C programming language, which are very hard to understand, maintain, change, and reuse.

Furthermore, one feature may require the execution of several actions at different stages of a workflow. A simple security feature, for example, might add an authentication component model element to the PIM, enforce encrypted communication by adapting the PSM, and generate source code to integrate the authentication component into the rest of the application. In this case, the security feature would introduce several conditional constructs into the corresponding workflow and further impair its readability.

By representing the workflow language as a domain-specific modeling language (DSML), it is possible to use model weaving techniques to integrate additional workflow tasks into a base workflow model depending on the actual feature selection. In this paper, we will introduce our concept of workflow modularization by means of aspectual model weaving techniques. In particular, we make the following contributions:

- We analyze current modeling workflow specification languages and show the lack of concepts for specifying feature-based modularization (Section 2).
- We present our concept of aspect-oriented model weaving for workflows, which fosters model-driven software product lines and has feature-based modularization support (Section 3).
- We embed our concept into the upcoming Modeling Workflow Engine (MWE) [18] and the model-driven framework openArchitectureWare [19] (Section 4).

Finally, we will discuss further related work and relate this paper to our superordinated research topic, which is the decomposition and composition of product lines.

2. Comparison of Modeling Workflow Languages

Workflow languages for model transformation sequences are very common in model-driven development. Implicitly or explicitly, they exist in every model-driven approach. In this section, we will compare only those that offer reasonable tool support and that facilitate the integration of various model-driven tasks (e.g., different model checkers or transformation engines). Related work that does not meet these requirements can be found in Section 5.

We will show that, up to now, most approaches have several shortcomings, either with respect to the *integration effort* necessary to include further transformation technologies or in their *flexibility* of specifying model-driven processes. Most importantly, none of the approaches offers adequate *variability management support* that could provide comprehensive separation of concerns within a software product line. As an outcome of this analysis, we will present a concept for workflow modularization for model-driven product lines in the subsequent Section 3.

We have evaluated the following workflow languages:

Ant: A generic build tool that also can be used for specifying model transformation workflows. [1]

MCC: The MDA Control Center (MCC) presented by KLEPPE. [15]

MOT: The ModelBus Orchestration Tool (MOT) by GUILLOIS ET AL. [10]

UniTI: The Unified Transformation Infrastructure (UniTI) presented by VANHOOFF ET AL. [24]

OAWWL: The openArchitectureWare workflow language.[1] [19]

2.1. Integration Effort

All approaches have tool support and are agnostic with respect to the coordinated model transformation technologies, as this was the precondition for our evaluation. The languages they provide for specifying model transformation workflows either have an XML or XML-like syntax (Ant, MOT, OAWWL), are small scripting languages (MCC), or are based on a dedicated workflow model editor (UniTI).

However, there are differences in the *effort* needed to integrate a new transformation technology (e.g., ATL [11]) or a concrete transformation (e.g., a ComponentModel-to-J2EE transformation in ATL). OAWWL and Ant provide lightweight mechanisms to write Java adaptors for different transformation technologies (OAWWL WorkflowComponents, Ant Tasks). Concrete transformations of the technologies can then be called directly from within the respective workflow language without need for further specifications. This facilitates rapid development of adaptors with reasonable effort.

The other approaches only concentrate on encapsulation of single concrete transformations rather than whole transformation technologies: MCC expects an eclipse plug-in for every new transformation, MOT requires the specification of a model service for the ModelBus middleware, and UniTI requires to adapt a single transformation with a Java adaptor. We thus expect the integration effort for transformations to be higher than for Ant and OAWWL.[2]

2.2. Flexibility

This section describes our analysis results with respect to the flexibility of specifying modeling workflows. Like proposed in [15], we test for the existence of basic combinatorial operators like *sequence*, *parallel*, and *choice*. Then, we compare how flexible and with which structural assumptions other model-driven tasks (model loading, storing, checking, etc.) can be integrated.

[1]The acronym OAWWL is only chosen for convenient comparison and is not used beyond this paper.

[2]We assume that the use of the reflective technologies can be used to make a *concrete transformation adaptor* very generic. It should even be possible to write *one* parameterized adaptor in Java to virtually handle *every* transformation technology and concrete transformation. This is, however, neither intended by the initial specifications of these technologies nor in the scope of this paper.

Combinatorial Operators. All approaches, of course, handle sequential operators, and all but OAWWL support some kind of parallel execution mechanisms (explicit parallelism language operators: Ant, MCC, MOT; implicitly by evaluation of the workflow model: UniTI). Conditional execution of tasks (choice) is supported by all approaches but UniTI: Ant needs to employ a special Ant Task and MCC and OAWWL support merely basic conditional operations. MOT uses embedded Java to evaluate conditions.

Absence of Structural Assumptions. UniTI and MCC make implicit assumptions on the internal structure of model transformation workflows. In UniTI, the top-level element is the model transformation. It may have checking constraints assigned to its input and output models and model types (platforms), but cannot integrate external model checking facilities. MCC does also not consider model-checking facilities, as it only knows *Creators*, *Transformers*, and *Finishers* as primary workflow elements. Ant, MOT and OAWWL allow the specification of arbitrary workflows and do not imply any internal structure.

2.3. Variability Management Support

We finally analyze the workflow language features that are suitable to meet product line needs. We assume that variability is expressed by feature models and that a single product is generated based on a concrete configuration of the feature model. We identified three different measures how to cope with product variability introduced by features on workflow level: conditional expressions, hooking mechanisms, and advanced modularization mechanisms.

Feature-based Conditional Expressions. All approaches that facilitate conditional expressions can in principle cope with product line variability. However, only OAWWL contains specific constructs to evaluate a feature model in a straightforward way [8]. The other approaches do not address product line engineering specifically.

Hooking Mechanisms. When a product line comprises many features that profoundly influence the product generation process, the numerous conditional expressions in the workflows become hard to understand and maintain. More sophisticated methods of separation of concerns for feature-based modularization become necessary. Only OAWWL facilitates to specify an additional, optional workflow task independent of the execution order as so called *workflow advice*.[3]

[3]A *workflow advice* specification of openArchitectureWare only loosely corresponds to the *advice*-construct known from aspect-oriented programming.

An advice specification is linked to an ordinary workflow task, which is executed in the order it is specified in the workflow file together with the additional logic introduced by the advice specification. This mechanism already has some aspect-oriented properties. However, the “pointcut” is implicitly integrated into the advice specification and can only reference one “join point” (a normal workflow task). For every type of workflow task, a corresponding type of workflow advice is necessary. Thus, workflow advice can rather be classified as a kind of hooking mechanism.

Although the flexibility and scalability is limited due to that, this mechanism, together with feature-based conditional expressions, can already achieve some separation of concerns within product line engineering. All other approaches, in contrast, require arranging all workflow tasks strictly in processing order.

Advanced Modularization Mechanisms. Although OAWWL already constitutes a great advance, workflow tasks and advice that both must be aware of their AO functionality have to be written for every technology (e.g., ATL, QVT, model checkers, etc.) needing AO support. They are supposed to work only pairwise: a workflow advice specification for technology X can, thus, not extend a workflow task implemented in technology Y. So, more complex and unforeseen workflow changes usually occurring during product line evolution become tedious and hard to manage, raising the need for more advanced structural mechanisms.

We consider the flexible introduction of workflow tasks as extremely useful for product lines, as this enables the orchestration of the whole model-driven product generation process independent of the capabilities of the underlying technologies. We will analyze promising approaches for that purpose in detail in Section 3.1.

2.4. Evaluation

A summary of the evaluation can be seen in Table 1. It is out of the scope of the paper to elect a clear winner. There still is no common consensus, if specific workflow DSLs are necessary or if general build tools like Ant are equally apt to serve the purpose. It also depends on the context, if lightweight workflow engines (like OAWWL) are sufficient or if a heavyweight model middleware (like MOT) is necessary.

We believe that a specific, lightweight workflow DSL should serve most purposes, while facilitating a concise notation with rich semantics. However, apart from this discussion, the evaluation shows that product line variability cannot be addressed thoroughly by any of the approaches. In the following section, we will develop a concept to meet this shortcoming.

	Ant	MCC	MOT	UniTI	OAWWL
General Criteria					
Language	XML	Textual	XML	Tree-Editor	XML-like
Language Base	Parser	Grammar?	XML-Schema	ECore	Parser
Scope	Build Tool	Workflow	Workflow	Workflow	Workflow
Integration Effort					
Integration of New Technologies (Integr. Basis)	+ (Techn)	o (Trafo)	o (Trafo)	o (Trafo)	+ (Techn)
Flexibility					
Combinatorial Operators (Seq., Parallel, Choice)	+|+|o	+|+|o	+|+|+	+|+|-	+|-|o
Absence of Structural Assumptions	+	-	+	-	+
Variability Management Support					
Feature-based Conditional Expressions	o	o	o	-	+
Hooking Mechanisms	-	-	-	-	+
Advanced Modularization Mechanisms	-	-	-	-	-

+: Full Support | o: Limited Support | -: No Support
Techn: One Adaptor per Transformation Technology Necessary
Trafo: One Adaptor per Transformation Implementation Necessary

Table 1. Comparison of Modeling Workflow Langages

3. Feature-Based Workflow Modularization

To modularize workflows based on their variability, as well as to facilitate their flexible composition, we will first analyze promising techniques for this purpose. Then, we present a concept for aspect-oriented workflow modularization and composition based on model-driven weaving techniques.

3.1. Feature-Based Modularization Techniques

There exist several techniques for the separation of concerns. We will present and compare language-agnostic, language-aware and DSML-based approaches.

Language-Agnostic Approach. Basic tooling for feature-based modularization is already integrated into common commercial product line tools [2, 17]. They provide simple means for composing textual artifact fragments to a compound text artifact depending on the current product configuration. They are not aware of the internal semantics of the languages the artifacts are written in and only rely on textual concatenation and pattern matching.

Language Extension Approach. In principle, all workflow languages can be extended with aspect-oriented or multi-dimensional language constructs, similar to the way AspectJ [14] and Hyper/J [22] extend Java. Depending on the configured product features, certain workflow tasks implemented as aspects or hyperslices are part of the product generation workflow or not. As described in Section 2.3, the OAWWL already has basic AO-like features. They could be extended to provide full obliviousness and quantification [6] to facilitate insertion of arbitrary workflow tasks at various positions and even allow more profound structural changes in the workflow process. For business workflows expressed in the Business Process Execution Language (BPEL), there exists such a language extension called AO4BPEL [3]. We will discuss this approach further in Section 5.

DSML Approach. When the language is a DSML and, thus, is based on a metamodel, model weaving techniques [5, 7, 9] can be applied to compose a workflow model from model fragments. A weaving model generated out of the current product configuration then controls which model fragments have to be composed to form the product generation workflow.

Comparing these three approaches, we consider the language-agnostic one as too low-level to meet the specific needs of workflow composition. Simply concatenating artifact fragments or using pattern substitution will usually render the editor support for the workflow language unusable. Simple text fragments cannot provide any means of obliviousness, since every location where a fragment may be inserted must be explicitly foreseen. When using pattern substitution, in contrast, one must consider all possible syntax variations in every pattern. As pattern matching also is not aware of the semantics of the workflow language, expressions usually get quite complex and hard to maintain.

In contrast, we regard both the language extension approach and the DSML approach as equally promising. If the workflow language has no explicit metamodel foundation, AO-like extensions are the viable solution. If the language is a DSML, on the other hand, the latter approach seems to be more convenient. Model weaving languages are specialized on easy processing of data structures, contrary to the transformation of abstract syntax trees, which is often done with normal programming languages. Thus, the rich set of model processing facilities of the MDD weaving environment can be used and should lead to more efficient development.

3.2. Concept for Feature-Based Workflow Modularization

We will now present our concept of feature-based workflow modularization for model-driven product lines. As already indicated, we favor the use of model weaving for this purpose, as we premise that the workflow language is a DSML and, thus, is based on a metamodel. We nevertheless adopt aspect-oriented terminology [14], as there are quite some similarities between both research fields. Actually, model weaving can also provide quantification and obliviousness, as this is just a matter of how a model weaver interprets the input models [9].

Figure 1 outlines the general concept. Optional and alternative workflow fragments are encapsulated in *workflow aspects* containing the information both where and what to manipulate in the base workflow. Mappings from features to aspects ensure that a *workflow weaver* can produce the workflow artifact corresponding to a product configuration. As can be seen from the figure, the overall concept does not imply any model-driven technology and, thus, could also be implemented by classical, non-model-driven aspect-oriented means.

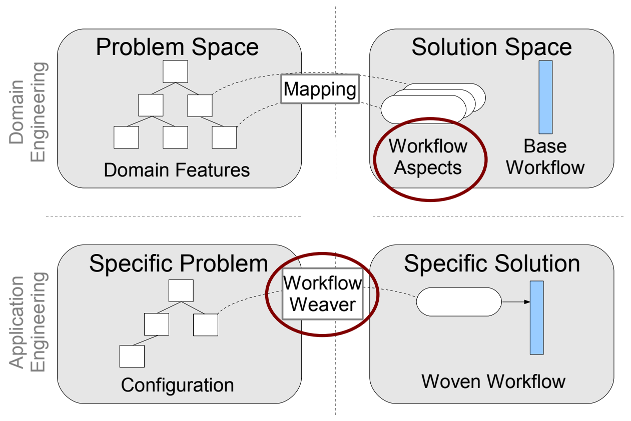

Figure 1. Outline of Workflow Weaving

The figure already indicates where design decisions are necessary: for workflow aspects and for the workflow weaver. As we work in the context of MDD, both workflow aspects and the base workflow have a model representation consisting of model elements.

Workflow Aspect. The *workflow aspect* must be able to specify the affected join points and the advice implementation.

- A **join point** is a point in the control flow during workflow execution. We assume that we can specify this point by referencing a certain model element of the base workflow. This element will usually be a simple or compound workflow task. In some cases, it might be useful to reference groups of model elements (e.g., to introduce parallel execution with respect to several consecutive workflow tasks).
- A **pointcut** references a set of join points. Although the sequential structure of workflows shows no urgent demand for quantification, in some cases (like the inevitable logging) it may render useful. Pointcut operations should allow checking for equality and pattern matching for attributes of a join point model element itself as well as for subordinated model elements.
- An **advice** specification must be able to formulate the advice type (e.g., before, after, around). It will usually introduce new model elements and thus implement positive variability [8]. To implement negative variability, around advice can be specified to substitute or remove model elements.

As the aspect language is a DSML, it can reference the workflow-specific model elements of the workflow DSML. We expect that, except these references, the workflow aspect DSML actually can have a generic, reusable AO metamodel that can be processed by a generic, workflow-agnostic weaver. The AO metamodel could then also be used for other languages based on the same metamodeling technology.

In the first instance, we plan to specify aspects as models by using a simple model editor based on the AO metamodel. In a further step, we then will analyze ways for rapidly specifying the AO language's concrete syntax, which will usually depend on the language to extend.

Workflow Weaver. The workflow weaving takes place in two stages.

- The product-line-aware **weaving initiator** evaluates the mappings from features to workflow aspects and generates an intermediate weaving model according to the current feature selection.

- The **generic aspect weaver** then performs the actual weaving in the second stage. It only relies on the AO metamodel to perform weaving and is thus workflow-agnostic. The output of the generic aspect weaver is an executable woven workflow.

4. Integration Into the openArchitectureWare Framework

openArchitectureWare (oAW) [19] is a comprehensive framework for model-driven software development with an already large number of users in industry and research. We will describe the most important technologies of the current (at the time of writing) version 4.3 and we will show how our approach will complement them.

4.1. openArchitectureWare

The oAW framework comprises of a language for model-to-model transformations, XTend, and for model-to-text transformations, XPand. It describes modeling workflows by the oAW workflow language (OAWWL, as described in Section 2). Hence, it can easily integrate external transformation technologies and other workflow tasks. Both XTend and XPand have full aspect orientation support [27] and are, thus, very well suited to formulate separation of concerns on transformation and generator level, respectively. As described, the workflow language also has some aspect-oriented capabilities to express variability (see Section 2.3). However, we identified that it reaches its limits when more complex and unforeseen changes to the workflow become necessary.

4.2. Modeling Workflow Engine

The prospective version 5.0 of the oAW framework will contain various changes and improvements. Most notably in the context of this paper, the workflow language will become autonomous as *Modeling Workflow Engine (MWE)* within the Eclipse Modeling Framework Technology (EMFT) project [18]. The developers of MWE kindly provided us with insights into the plans as well as into the source code of the upcoming version. It will be DSML-based and thus satisfy the needs we identified for flexible workflow modularization. Furthermore, there are plans to support the specification of parallel execution and more complex conditional expressions, what we also identified as shortcomings in Section 2.

4.3. XWeave

With XWeave [9], the oAW framework also contains facilities for generic, aspect-oriented model weaving. It currently only provides positive variability. More complex pointcuts can be formulated external to the aspect model in a dedicated expression language.

4.4. Future Plans

As the MWE is still under heavy development, it is too early to provide a concrete implementation of our concept for feature-based workflow modularization. The developers gave us the opportunity to follow the current early development stage. As soon as there will be a stable version of the workflow DSML, we will begin to analyze the capabilities of XWeave to support different workflow weaving scenarios with positive and negative variability and pointcuts of different complexity. So, we will be able to draw conclusions regarding the further development of XWeave to extend it to a fully-fledged generic modeling aspect weaver as well as for giving feedback on MWE for the purpose of weaving workflows.

As identified in Section 3.2, besides a generic aspect weaver, a weaving initiator is necessary, which generates a weaving model depending on the selected features. We plan to integrate it and the other described concepts as a case study into the home automation product line SmartHome [28], which already has integrated various state-of-the-art aspect-oriented and model-driven technologies and development techniques. Eventually, we will be able to validate our approach by refactoring the modeling workflow of SmartHome based on its features, which comprise, for example, security and various comfort functions.

5. Related Work

As the most promising approaches for modeling workflow specification languages have already been dealt with in Section 2, we will now only discuss related approaches that do not claim to have tool support or which do not have adequate means for integrating external workflow tasks. Furthermore, we will address approaches that are not directly related to modeling workflows, but nevertheless comprise similar concepts.

The Transformation Composition Modelling Framework of OLDEVIK describes black-box model transformations hierarchically with an UML2 class diagram and composes them by means of activity diagrams. No tool support is provided.

WAGELAAR [29] proposes a DSML with a textual concrete syntax for model transformation composition. Based on the DSML, an Ant [1] build script is generated, which performs the actual transformation. In his approach, every model transformation needs a dedicated metamodel element in the DSML. This makes his textual syntax very concise. The author does not address the scalability problem and the additional implementation overhead introduced when every transformation integrated into the transformation chain needs its own metamodel representation.

Several model transformation languages, like the XTend language from openArchitectureWare or the Query/View/Transformation (QVT) language [21] of the MDA approach, facilitate the execution of external program code (e.g., Java). This, in principle, allows the integration of external transformations by using program code adaptation layers. With their rich feature sets, they can evaluate conditions and perform model-checking right in place. However, integrating other model checkers, as well as loading and storing models into files, is out of their scope.

Stragego/XT [26] is a program transformation language and toolset that transforms abstract syntax trees (ASTs) by applying pattern-based rewrite rules. So called *strategies* control which rules shall be applied in which order. Stratego offers the strategy operators *choice* and *sequence* according to our comparison in Section 2.2, but no *parallel* operator, as it rather intended for working on one AST at a time. It comprises interesting additional operators specifically for program transformations, like non-deterministic and recursive application of rules. Furthermore, strategies also define the AST traversal direction (bottom-up, top-down, etc.) of rules. Depending on the application domain, these operators might also be useful for MDD. None of the in Section 2 compared approaches supports these operators. However, a model transformation is usually much coarser-grained than a rewrite rule for an AST, so that, for example, recursion and traversal direction are rather features of model transformation technologies than of modeling workflow languages.

There also exists an aspect-oriented extension of Stratego [12] offering fine-grained join points, pointcuts, and advice constructs on strategy and rule level. In comparison to our approach, it is a specific language extension and does not aim to support aspect orientation for several DSML using the same metamodeling technology.

AO4BPEL [3], as already mentioned in Section 3.1, is as well a specific aspect-oriented language extension. Its implementation is based on a modified BPEL engine, which checks at all potential join points, if an aspect has specified it in its pointcut. This allows easy and dynamic weaving of BPEL aspects with the drawback of less performance. As we plan to implement generic aspect-oriented mechanisms, we will not change the workflow engine but perform weaving on model level prior to workflow execution.

6. Conclusion and Outlook

In this paper, we introduced our concept of model-driven workflow modularization by means of aspectual model weaving techniques. First, we analyzed current modeling workflow specification languages and showed the lack of concepts for specifying feature-based modularization. Then, we presented a concept for aspect-oriented workflow model weaving aimed at model-driven software product lines. We embedded our concept into the upcoming Modeling Workflow Engine (MWE) and highlighted our future steps towards an implementation and its evaluation by means of a case study.

The strict separation of concerns, which we foster throughout this paper, has a particular reason: we especially are interested in the terms and conditions for the decomposition and composition of whole product lines. Product generation of a compound product originated from several product lines is a potentially highly complex task. Several product generation processes, one of each product line, may have to interact to create the final product. Thus, a clear interface for interaction is necessary, and, for model-driven product lines, model transformation workflows result to be a promising integration point for that purpose.

7. Acknowledgements

We would like to thank Markus Völter, Christa Schwanninger and the anonymous reviewers for their valuable comments on earlier drafts of this paper, as well as Sven Efftinge and Patrick Schönbach for providing insights into the development of the Modeling Workflow Engine.

References

[1] The Apache Ant Project. http://ant.apache.org.

[2] D. Beuche. Variant management with pure::variants. Technical report, pure-systems GmbH, 2003. http://www.pure-systems.com/.

[3] A. Charfi and M. Mezini. Aspect-Oriented Web Service Composition with AO4BPEL. In *Proceedings of the 2004 European Conference on Web Services(ECOWS '04)*. Springer-Verlag, 2004.

[4] R. S. et al. Model Driven Architecture. *OMG white paper*, 2000.

[5] M. D. D. Fabro, J. Bzivin, F. Jouault, E. Breton, and G. Gueltas. AMW: a generic model weaver. In *Proceedings of the 1re Journe sur l'Ingnierie Dirige par les Modles (IDM05)*, 2005.

[6] R. E. Filman and D. P. Friedman. Aspect-oriented programming is quantification and obliviousness. In *Workshop on Advanced SoC (OOPSLA '00)*, Oct. 2000.

[7] J. Gray, Y. Lin, and J. Zhang. Automating Change Evolution in Model-Driven Engineering. *IEEE Computer*, 39(2):51–58, 2006.

[8] I. Groher and M. Voelter. Expressing Feature-Based Variability in Structural Models. *Proceedings of the Workshop on Managing Variability for Software Product Lines (SPLC ,07)*, 2007.

[9] I. Groher and M. Voelter. XWeave: models and aspects in concert. *Proceedings of the 10th International Workshop on Aspect-oriented modeling*, pages 35–40, 2007.

[10] J.-B. Guillois and F. Jaouen. D3.3-2a Orchestration Tool - Tutorial. http://www.modelware-ist.org/, 2006.

[11] F. Jouault and I. Kurtev. Transforming models with ATL. In *Satellite Events at the MoDELS 2005 Conference*, volume 3844/2006 of *Lecture Notes in Computer Science*, pages 128–138, Heidelberg, BW, Germany, 2006. Springer-Verlag.

[12] K. T. Kalleberg and E. Visser. Combining Aspect-Oriented and Strategic Programming. In *Proceedings of the 6th International Workshop on Rule-Based Programming (RULE), ENTCS*, pages 168–177. Elsevier, 2005.

[13] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (ODA) Feasibility Study. Technical report, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, PA, Nov. 1990.

[14] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. In J. L. Knudsen, editor, *Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP '01)*, volume 2072 of *Lecture Notes in Computer Science*, pages 327–353. Springer-Verlag, June 2001.

[15] A. Kleppe. MCC: A Model Transformation Environment. In *Proceedings of the Second European Conference on Model Driven Architecture – Foundations and Applications (ECMDA-FA '06)*, volume 4066/2006 of *Lecture Notes in Computer Science*, pages 173–187. Springer-Verlag, 2006.

[16] A. Kleppe, editor. *Proceedings of the First European Workshop on Composition of Model Transformations (ECMDA-CMT '06)*, 2006.

[17] C. W. Krueger. Biglever software gears and the 3-tiered spl methodology. In *OOPSLA '07: Companion to the 22nd ACM SIGPLAN conference on Object oriented programming systems and applications companion*, pages 844–845, New York, NY, USA, 2007. ACM.

[18] Eclipse Modeling Framework Technology (EMFT) - Modeling Workflow Engine (MWE). http://www.eclipse.org/modeling/emft/?project=mwe.

[19] OpenArchitectureWare Homepage. http://www.openarchitectureware.org.

[20] Object Management Group (OMG). Meta Object Facility (MOF) Specification. formal/2002-04-03, April 2002.

[21] Object Management Group (OMG). Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.0. formal/08-04-03, April 2008.

[22] H. Ossher and P. Tarr. Multi-dimensional separation of concerns and the hyperspace approach. In *Symposium on Software Architectures and Component Technology: The State of the Art in Software Development*. Kluwer Academic Publishers, 2000.

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

[24] B. Vanhooff, D. Ayed, S. V. Baelen, W. Joosen, and Y. Berbers. UniTI: A Unified Transformation Infrastructure. In *Proceedings of the 10th International Conference on Model Driven Engineering Languages and Systems 2007*, pages 31–45, 2007.

[25] B. Vanhooff, S. V. Baelen, A. Hovsepyan, W. Joosen, and Y. Berbers. Towards a Transformation Chain Modeling Language. In *Proceedings of the 6th International Workshop (SAMOS '06) at Embedded Computer Systems: Architectures, Modeling, and Simulation*. Springer-Verlag, 2006.

[26] E. Visser. Program transformation with Stratego/XT. In *Domain-Specific Program Generation*, volume 3016/2004 of *Lecture Notes in Computer Science*, pages 216–238, Heidelberg, BW, Germany, 2004. Springer-Verlag.

[27] M. Voelter and I. Groher. Handling Variability in Model Transformations and Generators. *Proceedings of the 22st ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '07)*, 2007.

[28] M. Voelter and I. Groher. Product Line Implementation using Aspect-Oriented and Model-Driven Software Development. *Proceedings of the 11th Software Product Line Conference (SPLC '07)*, pages 233–242, 2007.

[29] D. Wagelaar. Blackbox Composition of Model Transformations using Domain-Specific Modelling Languages. In *Proceedings of the First European Workshop on Composition of Model Transformations (ECMDA-CMT '06)*, 2006.

[30] A. Yie, R. Casallas, D. Deridder, and R. V. D. Straeten. Multi-Step Concern Refinement. In *Proceedings of the Workshop on Early Aspects (AOSD-EA '08)*, 2008.