XWeave: Models and Aspects in Concert

Iris Groher
Siemens AG, CT SE 2
Otto-Hahn-Ring 6
81730 Munich, Germany
+49 89 636 49477
iris.groher.ext@siemens.com

Markus Voelter
Independent Consultant
Ziegelaecker 11
89520 Heidenheim, Germany
www.voelter.de
voelter@acm.org

ABSTRACT

Model-driven software development improves the way software is developed by capturing key features of the system in models which are developed and refined as the system is created. During the system's lifecycle models are combined and transformed between different levels of abstraction and viewpoints. Aspect-oriented techniques improve software development by providing modularization constructs for the encapsulation of crosscutting concerns. Existing research has already investigated many ways of combining the two paradigms. This paper contributes by presenting XWeave, a model weaver that supports weaving of both models and meta models. XWeave supports the composition of different architectural viewpoints and eases model evolution. Furthermore, the tool plays an important role in software product line engineering, as variable parts of architectural models can be woven according to some product configuration. The concepts are illustrated with an example of a home automation system.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques. D.2.11 [**Software Engineering**]: Software Architectures.

General Terms

Design, Languages.

Keywords

Model-Driven Software Development, Aspect-Oriented Software Development, Model Weaving.

1. INTRODUCTION

Model-driven software development (MDSD) [1] improves the way software is developed by capturing key features of the system in models which are developed and refined as the system is created. During the systems lifecycle models are combined and transformed between different levels of abstraction and viewpoints. The key difference to traditional modeling is that in MDSD models do not constitute documentation but are processed by tools. Models are first class entities; they essentially play the role of source code. Model transformations are most commonly used for processing and refining models. Such transformations are similar to functions: based on one or more input models an output model is produced. Model weaving is a special case of transformation where input models are woven together based on a weaving specification to produce the desired output.

Aspect-oriented (AO) techniques [2][3] improve software development by providing constructs for the encapsulation of crosscutting concerns. Aspects encapsulate crosscutting concerns and are subsequently composed with other software artifacts using powerful composition mechanisms. A join point model captures the set of possible composition points and pointcut expressions quantify over the join point model to select the desired set of composition points for a specific aspect. Aspects are automatically composed with the rest of the system by an aspect weaver, either statically during compilation, dynamically at runtime, or at load-time. Asymmetric AO approaches such as AspectJ [4] provide constructs for the encapsulation of crosscutting concerns that are woven to some (non-AO) base system. Symmetric approaches such as CaesarJ [5] and CME [6] provide constructs for the encapsulation of all kinds of concerns which are then composed to form the final system.

While the two approaches are different in many ways – MDSD adds domain-specific abstractions and AOSD offers concerns modularization and composition mechanisms – they also have many things in common. Existing research has already investigated many ways [7][8][9][10][11][12] of combining the two paradigms. Both, MDSD and AOSD are promising technologies that improve the modularity, composability, evolvability, and reusability of software systems. The two paradigms are complementary in nature and can benefit from each other when used in combination. For example, by modularizing crosscutting concerns in models or using AO to simplify model transformations.

This paper contributes to the integration of MDSD and AOSD by presenting XWeave, a model weaver that supports weaving of both models and meta models. The weaver is based on the Eclipse Modeling Framework (EMF) [13] Ecore meta meta model. This means that the approach can weave models that are either instances of Ecore (meta models) or instances of those models. XWeave weaves crosscutting concerns encapsulated as aspect models into (non-AO) base models. This is a form of asymmetric model weaving, where there is a designated base model into which a number of aspect models are woven (as opposed to symmetric weaving, where there is no designated base model). Weaving is done based on matching names of elements in the aspect and the base model. Additionally, pointcuts based on the openArchitectureWare (oAW) expression language [14] can be defined to select sets of model elements as join points. The oAW expression language is a statically typed language based on OCL. It is part of the oAW framework [15], a framework for building MDSD tools.

The concepts introduced in this paper are illustrated with an example of a home automation system, called Smart Home. Smart Home networks the devices installed in a house and allows inhabitants to monitor and control their status. Devices can even coordinate their behavior to fulfill complex tasks without human intervention. The example is based on real system requirements from Siemens AG and demonstrates the benefits of automated model weaving of both homogeneous and heterogeneous aspects [16] at meta model as well as at model level.

The remainder of the paper is organized as follows: Section 2 introduces our view on the relationship between MDSD and AOSD. Section 3 demonstrates the home automation example. The concepts and capabilities of XWeave are motivated, described and evaluated in Section 4. Related work is discussed in Section 5. Section 6 summarizes the paper and provides an outlook on future work.

2. MODELS AND ASPECTS

In MDSD systems are continuously described in terms of models that are subsequently developed and refined. Models do not only constitute documentation but are first class entities that are processed by tools. Every model conforms to a meta model. The meta model defines the vocabulary and grammar that can be used to build the model. Hence, models are instances of their respective meta models. A meta model also has a meta model which is called the meta meta model. There are different meta meta model formalisms such as MOF [17] or Ecore [13]. Figure 1 shows two examples of the various (meta) models for MOF and Ecore.

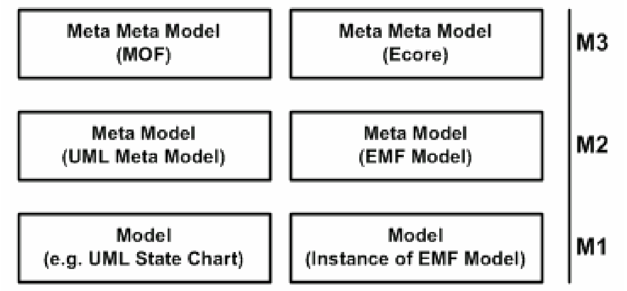
Meta Meta Model (MOF)
Meta Meta Model (Ecore)
M3
Meta Model (UML Meta Model)
Meta Model (EMF Model)
M2
Model (e.g. UML State Chart)
Model (Instance of EMF Model)
M1

Figure 1: Meta Levels

Models are combined and transformed between different levels of abstraction and viewpoints during the systems lifecycle. Usually, model transformations are used for processing and refining models. During a model transformation an output model is produced based on one or more input models. The input models are not changed during a model transformation. Model modification and model weaving are special kinds of transformations. Model modification is about changing the input model(s) in order to produce the desired output. Model weaving is about taking a base model as well as one or more aspect models and weaving them together in a user controllable way.

Extensive research has been conducted in combining MDSD and AOSD as they have many things in common [7][8][9][10][11][12]. There are many ways that these emerging paradigms may be integrated to achieve the complementary benefits of both AOSD and MDSD:

- Aspect-oriented modeling [9][10] aims at providing means for expressing aspects and their crosscutting relationships at modeling level.
- Model weaving [21] assists in the composition of different separated models into a consistent whole.
- AO templates [14] can be used when implementing a code generator. Aspect templates advice the standard code generation templates with code that is specific to some crosscutting concern.
- AO-like introductions [14] allow the contribution of additional properties to meta classes that implement a specific meta model.
- It is even possible to integrate an AOP language [4][5] into the MDSD infrastructure. Specifically, a number of pre-built advice can be defined as part of the platform and pointcuts are generated based on specifications in the model. The AOP language's standard weaver then integrates the aspects with the generated code.

3. HOME AUTOMATION EXAMPLE

The example we use to illustrate our approach is a home automation system (see also [18]), called *Smart Home*. In homes you typically find a wide range of electrical and electronic devices such as lights, thermostats, electric blinds, fire and smoke detection sensors, white goods such as washing machines, entertainment equipment such as TVs and communication devices such as phones. Smart Home connects those devices and enables inhabitants of a home to monitor and control the status of devices from a common user interface. The home network also allows the devices to coordinate their behavior in order to fulfill complex tasks without human intervention.

Sensors are devices that measure physical values of their environment and make them available to Smart Home. Controllers activate devices whose state can be monitored and changed. All installed devices are part of the Smart Home network. The status of devices can either be changed by inhabitants operating on the user interface or by Smart Home using predefined event plans. Event plans let the system act autonomously in case of certain events. For example in case of fire and smoke detection windows get closed automatically and the fire brigade is called.

Figure 2 shows a simplified meta model of Smart Home. A house contains floors and floors contain rooms. In a room, different devices are installed and every device is controlled by a controller. Rooms also contain sensors.

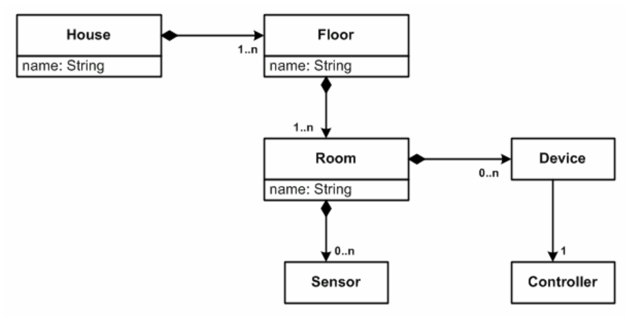

Figure 2: Smart Home Meta Model

Figure 3 shows an instance of the Smart Home meta model, i.e. a concrete home automation system. The example house only contains one floor where only one room is located. The bedroom contains one light sensor and one light device which is controlled by a light controller.

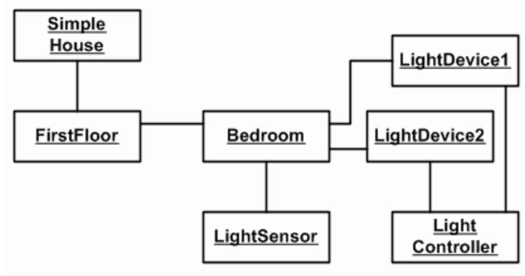

Figure 3: Smart Home Model

Various types of houses and different customer demands drive the need for different kinds of home automation systems. This kind of variability and the evolution of models over time require models to change often and quickly. The next section will show how variability within models and model evolution can be handled in an elegant way using our XWeave approach.

4. XWEAVE

4.1 The Purpose of Model Weaving

4.1.1 Model Evolution

One important application of model weaving is model evolution, the change of models over time, which can be handled easier using a model weaving approach. Changes can be localized in aspects which eases traceability and change management.

4.1.2 Product Line Engineering

Model weaving is also an important issue in software product line engineering (PLE). Product lines take advantage of the commonality within a portfolio of similar products [18]. Products usually differ by the set of features they include in order to fulfill customer requirements. Products that are part of a product line typically have a common architecture with well-defined differences among them. In PLE, these differences are formally captured, for example using feature models. In the case where MDSD is used as an implementation technique for PLE, the commonalities and variabilties between the models need to be managed. Here, model weaving can help by capturing variable parts of models in aspect models, and weaving them into a given minimal core. The core then only contains elements common to all products; the optional parts are automatically added when needed. A clear separation of optional model parts improves traceability of variability. For software product lines it is essential to know the relationship between features and the derived architectural models. The effects of variability and especially new variability brought by evolution cannot be easily modeled and managed. A model weaving approach allows the clear separation of optional parts of the model from core parts.

This approach allows variant management on model level. In PLE, two forms of variability are known: negative and positive variability. Negative variability is about removing optional parts from a given structure, whereas positive variability is about adding optional parts to a given core. Figure 4 illustrates the difference between negative and positive variability.

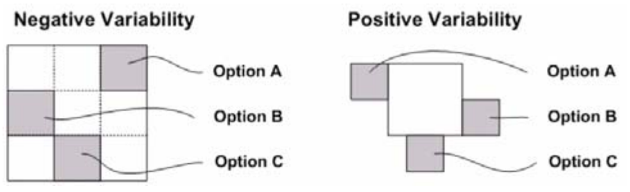

Figure 4: Negative vs. Positive Variability

An existing approach [19] combines variant specification in feature models with negative variability mechanisms in models. The basis is a complete model with all possible features included. Optional elements are associated with features in the feature model. A model element is then only present in a model if the feature it is associated with is selected in the respective configuration. By unselecting features in the feature model, the original model is “cut down” to a model that is specific to one concrete variant. This approach has the drawback that one has to start by modeling the overall, large model.

Model weaving supports positive variability within models. Based on whether a feature is selected in the variant specification, a certain model aspect is woven or not.

4.1.3 Architectural Viewpoints

Another important application of model weaving is the combination of different architectural viewpoints. When creating the final system the different viewpoint models have to be combined into a consistent whole. Using a model weaving approach different viewpoints can be modeled in separation and later composed to form the final system.

4.2 Concepts and Capabilities of XWeave

XWeave is a model weaver based on EMF’s Ecore meta meta model [13]. This means that the tool can weave models that are either instances of Ecore (these are called meta models) or instances of those models (we call these models). Ecore is Eclipse EMF’s implementation of the Essential MOF (EMOF), a simplified version of the original OMG MOF [17] standard. We have selected Ecore as the meta meta modelling formalism because it integrates with a large number of tools such as Eclipse GMF [20] for graphical modelling and oAW [15] for model-to-model transformations and code generation. Figure 5 (highlighted part) shows the relevant core of the Ecore meta meta model.

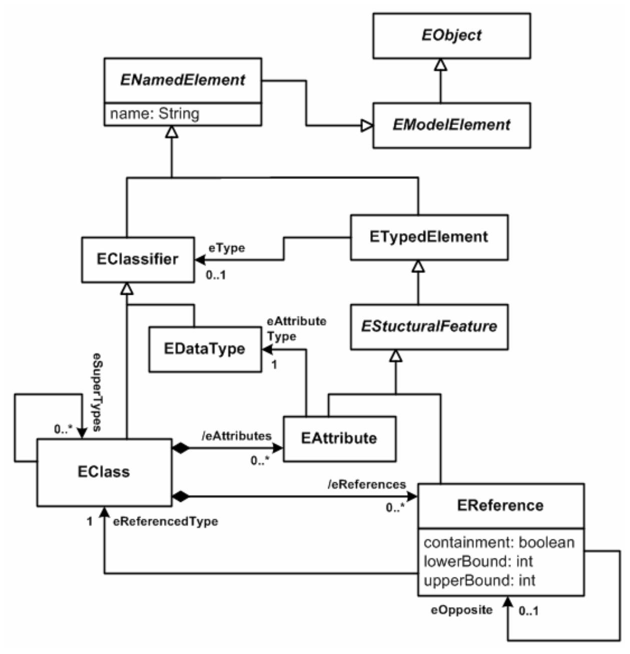

Figure 5: The Ecore Meta Meta Model

XWeave takes a base model as well as one or more aspect models as input and weaves the content of the aspect model into the base model. This is a form of asymmetric AO. There are two ways of specifying pointcuts: name matching and explicit pointcut expressions (note that we will provide examples for both of these mechanisms below):

- Name matching means that if a model element in the aspect model has a corresponding element in the base model (corresponding means that both name and type are equal) the element is woven.
- Pointcuts can be defined with a dedicated expression language. Expressions can select one or more elements of the base model and are defined external to both models. Every expression has a name and can be referenced by this name. The expression language used in XWeave is the oAW expression language [14] which will be introduced in Section 4.2.3. The named expressions (pointcuts) can be used in the aspect model. If an aspect element's name starts with % followed by the name of a defined expression, the expression will be evaluated for this element.

Weaving an element means that all properties of the element including its child elements are woven into the base model.

Using the capabilities of XWeave, both heterogeneous as well as homogeneous aspects can be woven. To illustrate the concepts of XWeave, let's look at the following examples.

4.2.1 A Homogeneous Aspect Model

Homogeneous aspects apply the same piece of advice to several places [16].

Consider the Smart Home example introduced Section 3. You might want to provide an optional feature *Fire Detection*. This means that, for all the rooms in the house, you have to add a *fire detection sensor* (and many other things we don't discuss here). This example requires a homogeneous aspect model to be woven into the model as the same aspect element (*FireSensor*) is applied to several places (*rooms*). Figure 6 illustrates the resulting model. The base model is grey, the elements added by the aspect are highlighted in black.

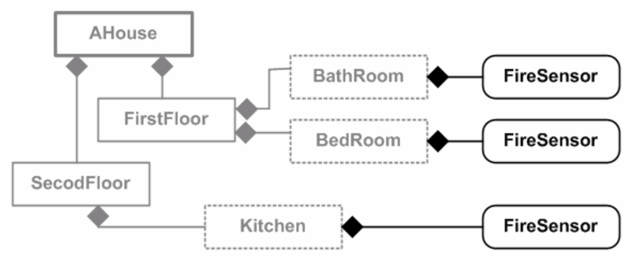

Figure 6: Resulting Model after Homogeneous Weaving

Figure 7 shows a graphical representation of the aspect woven into the base model. Note how we use the different line styles to distinguish the various meta classes.

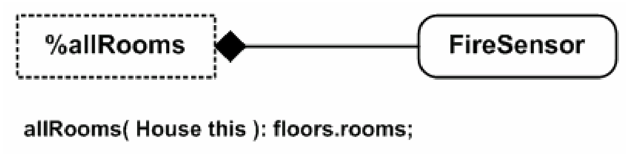

Figure 7: A Homogeneous Aspect

The pointcut expression *allRooms* collects all the rooms of all the floors in a given house. As we explained above, all attributes and references of the selected elements are woven, so each of the rooms is supplied with a new *FireSensor*.

This example shows a homogeneous aspect, since the same advice (the *FireSensor*) is woven "into many locations" in the model in the same way. We used a pointcut expression to identify those locations.

4.2.2 A Heterogeneous Aspect Model

Heterogeneous aspects add different pieces of advice to different places [16]. In this example we show a heterogeneous aspect that uses name matching to identify the target join points. To illustrate XWeave's capability to also work at the meta model level, we show how to vary the meta model. Consider the following scenario: You are the vendor of smart home systems. As part of your product, you also provide a tool to plan (i.e. model) smart home systems. As part of your product line of smart home systems, you have several levels of sophistication: for example, your customers can optionally buy a *Presence Management* feature. This is a feature that tracks who is in which room. In order to support this feature, the meta model needs to be varied as shown in Figure 8.

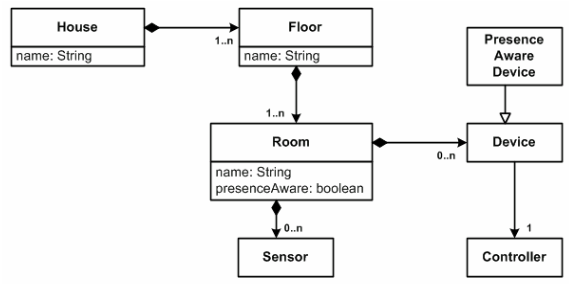

Figure 8: Meta Model with Heterogeneous Aspect Woven

The aspect that can be used to effect this variant looks as illustrated in Figure 9.

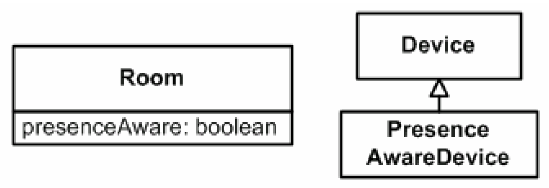

Figure 9: A Heterogeneous Aspect

Here we weave two additional elements into the meta model at well defined locations – a heterogeneous aspect. We use name matching to identify the join points, hence, no pointcut expression is necessary.

4.2.3 The Expression Language

Like all the other components of the openArchitectureWare toolkit, XWeave also uses the oAW expression language [14]. This language is statically typed and based on OCL [26]. It contains a number of additional features, mainly convenience functions and syntactic sugar. oAW provides a powerful syntax highlighting and code-completing editor for expressions.

Like OCL, the oAW expression language provides the following features (only the features that are relevant for XWeave are listed here):

- *Path expressions* allow the navigation over several steps (using the familiar dot-notations). This navigation also works for multi-value properties, in which case the expression returns the leaves of the tree created by the expression.
- *Working with collections:* The expression language provides primitives to work with sets: *union, difference, without*, etc. *aCollection.forAll(predicate)* checks whether *predicate* is true for all elements of a collection. a*Collection.exisits(predicate)* checks whether there is at least one element in the collection for which the predicate is true.
- *Selection/Filtering:* A given set of elements can be filtered based on a boolean predicate. *aCollection.select(e| e.someProp == someValue)* picks all elements from *aCollection* whose property *someProp* has the value *someValue*.

4.3 Evaluation

The current state of XWeave supports the purposes of model weaving that we defined in Section 4.1. We are able to weave one or more homogeneous and heterogeneous aspects into a given base model. The join point model is based on the base models' meta model and is thus very generic. Weaving can be based on name equivalence, or using pointcut expressions.

However, the advice is currently limited. We cannot remove change, or override existing base model elements using aspects. XWeave thus currently supports essentially only additive weaving, where additional elements are added to the base model.

For our short term purpose this is sufficient but we will address these limitations in the future.

5. RELATED WORK

5.1 AMW

AMW, the Atlas Model Weaver [21], is a tool created by INRIA as part of the ATLAS Model Management Architecture. It's primary goal is to establish links between models. In the first phase of working with AMW, a number of links are established between two or more models. This process can be manual or semi-automatic. The result is called the weaving model. Based on that model, you can then generate model transformations that merge models.

AMW is similar to XWeave as you can merge or weave models. It is, however, also different in several ways. For example, AMW contains an interactive tool to build the weaving model, whereas XWeave uses name correspondence or pointcut expressions. An important reason for building XWeave is its integration with the rest of the openArchitectureWare tools, e.g. being compatible with oAW's workflow engine and using the oAW expression language.

5.2 C-SAW

The C-SAW project [22] is developed by the University of Alabama at Birmingham. It is a general transformation engine for manipulating models and is a plug-in for GME. C-SAW modifies complex models based on aspect specifications using ECL (a variant of the Object Constraint Language, OCL). The weaver traverses the model and selects a set of elements to which the aspect should be applied – essentially, a procedurally implemented pointcut. The advice then modifies the selected element in some way, for example by adding a precondidition or changing the element structure somehow.

C-SAW has been developed to tackle the challenge of evolving potentially very large models in consistent ways. Instead of applying a set of changes manually, you merely write an aspect that applies the changes to all selected elements in the model.

Comparing it to XWeave reveals that C-SAW doesn't weave models (in the sense of merging them) as XWeave does. Rather, it efficiently applies (crosscutting) changes to a collection of elements in a large model.

5.3 Others

Theme/UML [23][10] is a design modeling language that provides modeling constructs for separating aspects during design. It is suitable for both symmetric as well as asymmetric AO. The separated design models can be composed using defined composition operators. In contrast to XWeave, Theme/UML does not provide any tool support for the automatic composition of models.

Join point designation diagrams (JPDD) [24] provide new means for modeling pointcuts, i.e. the places where crosscutting occurs. UML classifiers are used to represent join points in structural models and UML messages to represent join points in behavioral models. XWeave uses name matching and pointcut expressions to represent join points at modeling level.

6. SUMMARY AND FUTURE WORK

AOSD and MDSD are both emerging new paradigms that improve software development and provide even more benefits when used in combination. One possible way of integrating the two approaches is model weaving. In this paper we have presented XWeave, a model weaver that supports weaving of both meta models and models. XWeave is based on the EMF Eore meta meta model. The tool takes a base model as well as one or more aspect models and weaves them together in a user controllable way. Pointcuts can be defined based on matching names of model elements or expressions. We have demonstrated the weaving of both homogeneous and heterogeneous aspect models based on examples of a home automation system. XWeave improves model evolution as changes can be localized in aspect models which eases traceability and change management. Furthermore optional parts of the model can be separated as aspect models and only woven into the base model when needed. This is especially helpful in software product line development. In cases where different architectural viewpoints have to be combined XWeave can be used as well. The viewpoints can be developed in isolation and the tool composes them to form a final system.

In the future we will combine XWeave with a variant management tool such as pure::variants [25]. Users can then model optional parts of the model as aspect models, relate them to features in the feature model and let the tool weave the relevant aspect models according to some configuration (i.e. selection of features).

We will also address the limitations stated in Section 4.3, specifically, the fact that we currently only support additive weaving. In the future we plan to extend XWeave in order to support removing, changing, or overriding of existing base model elements using aspects.

Another possible extension of XWeave is support for symmetric model weaving. This kind of weaving does not distinguish between aspect and base models. Models are woven together according to defined rules to form the final system.

7. ACKNOWLEDGMENTS

This work is supported by AMPLE Grant IST-033710. The authors would like to thank Christa Schwanninger and Andrew Jackson for their valuable comments on earlier drafts of this paper.

8. REFERENCES

[1] Stahl, T., and Völter, M. Model-Driven Software Development. Wiley & Sons, 2006.

[2] AOSD website, http://www.aosd.net

[3] Filman, R., Elrad, T., Clarke, S., and Aksit M. Aspect-Oriented Software Development. Addison-Wesley, 2004.

[4] AspectJ website, http://www.eclipse.org/aspectj/

[5] CaesarJ website, http://www.caesarj.org/

[6] Concern Manipulation Environment (CME) website, http://www.research.ibm.com/cme/

[7] First Workshop on Models and Aspects – Handling Crosscutting Concerns in MDSD, Glasgow, UK, July, 2005. http://www.st.informatik.tu-darmstadt.de:8080/ecoop2005/maw/

[8] Second Workshop on Models and Aspects – Handling Crosscutting Concerns in MDSD, Nantes, France, July, 2006. http://www.kircher-schwanninger.de/workshops/MDD&AOSD/

[9] Aspect-oriented Modelling Workshops, http://www.aspect-modeling.org/

[10] Clarke, S., and Baniassad, E. Aspect-Oriented Analysis and Design. The Theme Approach. Addison-Wesley, 2005.

[11] Simmonds, D., Solberg, A., Reddy, R., France, R., and Ghosh, R. "An Aspect Oriented Model Driven Framework". In Proceedings of the Ninth IEEE "The Enterprise Computing Conference" (EDOC), Enschede, Netherlands, September, 2005.

[12] Sánchez, P., Magno, J., Fuentes, L., Moreira, A., and Araújo, J. "Towards MDD Transformations from AO Requirements into AO Architecture". In Proceedings of the Third European Workshop on Software Architecture (EWSA), Nantes, France, September, 2006.

[13] Eclipse Modeling Framework website, http://www.eclipse.org/emf

[14] openArchitectureWare Documentation website, http://www.eclipse.org/gmt/oaw/doc/

[15] openArchitectureWare website, http://www.eclipse.org/gmt/oaw

[16] Lopez-Herrejon, R., E. "Towards Crosscutting Metrics for Aspect-Based Features". In Proceedings of the First Workshop on Aspect-Oriented Product Line Engineering (AOPLE), Portland, Oregon, October, 2006.

[17] OMG MetaObject Facility website, http://www.omg.org/mof/

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

[19] Czarnecki, K., and Antkiewicz, M. "Mapping Features to Models: A Template Approach Based on Superimposed Variants". In Proceedings of the Fourth International Conference on Generative Programming and Component Engineering (GPCE), Tallinn, Estonia, September, 2005.

[20] Eclipse Graphical Modeling Framework website, http://www.eclipse.org/gmf

[21] Atlas Model Weaver website, http://www.eclipse.org/gmt/amw

[22] C-SAW website, http://www.cis.uab.edu/gray/Research/C-SAW/

[23] Clarke, S. Composition of Object-Oriented Design Models. PhD thesis, Dublin City University, 2001.

[24] Stein, D., Hanenberg, S., and Unland, R. "Modeling Pointcuts". In Proceedings of the Early Aspects Workshop, Lancaster, UK, March, 2004.

[25] pure::variants Variant Management Tool website, http://www.pure-systems.com/3.0.html

[26] OMG UML 2.0 Object Constraint Language website, http://www.uml.org