

# Applying and Evaluating AOM for Platform Independent Behavioral UML Models \*

Marco Mosconi  
Technische Universität Berlin  
Berlin, Germany  
mosconi@cs.tu-berlin.de

Anis Charfi  
SAP Research  
Darmstadt, Germany

Jaroslav Svacina  
Fraunhofer FIRST  
Berlin, Germany  
jaroslav.svacina@first.fhg.de

Jan Wloka  
Rutgers University  
Piscataway, USA  
jwloka@cs.rutgers.edu

## ABSTRACT

Several approaches for aspect-oriented modeling (AOM) have been developed to modularize crosscutting concerns properly in UML models. In this position paper we present a combination of AOM approaches and show how they can be applied in a model-driven process targeting business applications. We present a UML 2 profile for platform independent AOM with advanced pointcut expressions and a corresponding model weaving mechanism for behavior models using UML 2 Actions. We show that a seamless integration of aspect-oriented concepts into an existing model-driven process can be achieved easily with state-of-the-art technology. We applied our approach in the context of an industrial case study and performed an evaluation that shows a significantly improved understandability and maintainability of platform independent models using aspects.

## Keywords

MDA, UML 2 Profiles, Aspect-Oriented Modeling, Model Weaving

## 1. INTRODUCTION

Existing model-driven development approaches are often restricted to the object-oriented modularization concepts provided by the Unified Modeling Language (UML). Similar to object-oriented programming languages, some functionalities, called crosscutting concerns, cannot be properly modularized at modeling level. The resulting models may result in diagrams with scattered model artifacts, which belong to multiple concerns. Consequently, developers have to

\*This work is supported by the European Commission 6-th Framework Programme, Project VIDE - VIsualize all moDEl drivEn programming, IST-033606-STP

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*Workshop AOM (Twelfth Edition)*, April 1, 2008, Brussels, Belgium.  
Copyright 2008 ACM 978-1-60558-146-0/00/0000 ...\$5.00.

deal with tangled and scattered structure and behavior in a diagram, which leads to decreased understandability and maintainability.

In this paper, we present an approach that extends the UML with aspect-oriented concepts to improve the modularization of crosscutting concerns in platform independent models. Aspect-oriented modularization concepts strive to improve the issues arisen from crosscutting concerns, leading to better understandability, maintenance, and evolvability [7, 8]. Our approach is inspired by programming languages with AOP support, such as AspectJ [1]. It consists of a specific UML profile that provides aspect-oriented constructs at modeling level and a model weaver for resolving pointcuts and weaving aspects within UML models.

The new constructs, such as aspect, advice, and pointcut are represented as first-class model elements, and special actions are provided for a non-invasive adaptation of modeled behavior. Our model weaver processes the extended UML models and provides concepts for passing context information into bound advice, for translating structural elements (aspect, advice, pointcut) and behavioral elements (actions). It results in a (non-AO) UML model, which can be executed with standard tools for model-driven development. Our approach was applied to a real world customer relationship management application and has shown that the quality of diagrams representing crosscutting concerns could be improved regarding size, complexity, separation of concerns, and ease of change, using aspect-oriented modularization concepts.

We therefore believe that, the use of aspect-oriented modularization concepts can improve the understandability and maintainability in executable UML models, and consider the following points as the paper's contributions:

- A UML 2 profile with aspect-oriented UML extensions to capture crosscutting concerns in platform independent UML models;
- A model weaver that resolves pointcuts and weaves aspect structure and behavior into UML models based on classes, activities, and actions;
- An industrial case study in which our approach was used and evaluated.

The remainder of the paper is structured as follows: Section 2 presents our aspect-oriented modeling approach and UML

profile for platform independent behavior and Section 3 the model weaving process and technology. Section 4 illustrates our approach with a concrete example scenario, Section 5 presents first results of an evaluation of our approach, and Section 6 compares related work to our approach. In Section 7 we conclude the paper.

## 2. PLATFORM INDEPENDENT AOM

The scope of our work is the platform independent modeling of business applications in an MDA<sup>1</sup> process. This includes system specification using standard object-oriented UML 2 notation and semantics, additionally supporting domain-specific views by using the UML 2 profile mechanism [12]. Unlike most realizations of MDA in this domain, we consider platform independent models as being complete in terms of application specific requirements. That is, the behavior of the system is specified to an extent that enables automated generation of functional implementations.

These behavioral specifications are modeled using UML 2 *Activities*, which allow for detailed control and data flow specifications. In general, activities are used to model the behavior of operations defined on classes. Typical statements of object-oriented programming languages like object creation/destruction, read/write access to fields and method calls are supported through corresponding *Actions*. Other approaches supporting complete specification for model execution and code generation [11, 3] either address other application domains, mostly embedded or telecom systems, or use other formalisms than UML 2 actions.

Our support for aspect-oriented concepts in a model-driven engineering process consists of several components. First, the modeling language has to be extended with model elements representing aspect-oriented constructs. For this purpose we defined a UML 2 profile for platform independent aspect-oriented modeling that is presented further in this section. Usage examples can be found in Section 4. Our work was inspired by that of Fuentes and Sanchez [5], but the realization differs in several ways, some of them are mentioned in the following sections and Section 6. Even though our approach is dealing with more abstract elements of platform independent models, similar concepts as provided by mainstream AOP languages can be applied. This is because the operational semantics of UML actions are sufficiently concrete to be mapped to object-oriented implementations.

### 2.1 Aspect Structure

Our profile for platform independent aspect-oriented modeling contains the basic constructs for the definition of aspects as shown in Fig. 1. **Aspects** are defined as extension of *Class*, thus having basically the same features such as properties and *Operations*, whose behavior in turn is defined by an *Activity*. Additionally, aspects can contain **Advices**, which are stereotyped operations. Advice behavior is defined using actions similarly to normal operations, but also supporting a special **Proceed** action representing a call of the advised behavior. As this proceed action is only valid for *around* advice, the stereotype **Advice** contains a corresponding property which can be used for validation. In contrast to most other approaches advice, pointcut, and binding are separate constructs to facilitate reuse of advice and pointcuts. The advice does not directly depend on a

pointcut, but instead defines its requirements (joinpoint context) through regular operation parameters. On the other hand, pointcuts have a signature, too. Hence, a binding can be validated statically in terms of advice and pointcut signatures.

In our profile we made use of the fact that UML 2 profiles cannot only contain stereotypes, but also meta-classes. This eases the introduction of elements that do not match well existing UML elements, for example **Pointcuts** or **Bindings**. The use of meta-classes results in more precise specifications and a cleaner model because it avoids having (a) two model elements, one for the meta-class and one for the stereotype and (b) unnecessary properties in the re-used meta-class due to a conceptual mismatch. Having a cleaner interface on the model elements is beneficial for both the implementation of model transformations and the user developing the model.

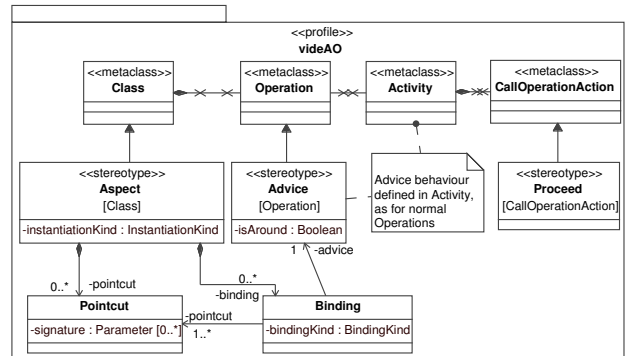


Figure 1: AO Modeling Profile – Constructs

### 2.2 Pointcut Definition

For the specification of model-level pointcuts, an expression-based approach was chosen. In contrast to [4], pointcuts here do not reference model elements directly, but instead are defined in terms of pointcut expressions (PCE). This enables the use of partial meta-information to quantify over model elements, comparable with pointcut expressions in, e.g., AspectJ. Furthermore, pointcut expressions are not specified as Strings, but using a sophisticated and fine-granular meta-model based expression language. The abstract syntax of this expression language is depicted in Fig. 2 and part of the AOM profile.

A **Pointcut** consists of one or more pointcut expressions (PCE) and a signature which is used for passing context information to an advice (see Section 2.1). Multiple expressions in one pointcut and multi-valued properties of expressions are implicitly interpreted disjunctively. For example, to define a pointcut expression that matches all calls to methods named `set*`, `add*` or `put*`, one only has to specify one **OperationPCE** of type `call` with three `namePattern` values. In AspectJ three separate call expressions would have to be manually combined. This allows for very flexible and fine-granular quantification. To combine pointcut expressions conjunctively, an **IntersectionPCE** can be used. Even though the use of operators for conjunction and disjunction of every part of an expression would be even more flexible, we believe that our implicit approach supports the most typical scenarios best.

<sup>1</sup>Model Driven Architecture (<http://www.omg.org/mda>)

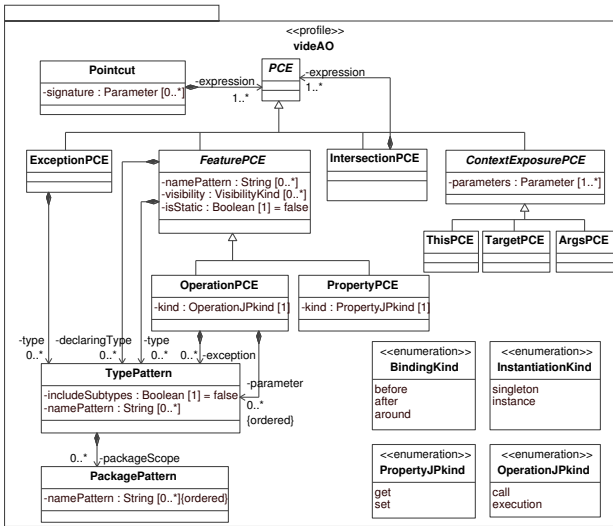


Figure 2: AO Modeling Profile – Pointcut Expressions

### 3. ASPECT MODEL COMPOSITION

To integrate aspect-oriented composition in the given model-driven process, we implemented an approach where the aspect composition is performed incrementally on the source models. The aspect weaving is done completely at model level, resulting again in a platform independent UML model. This enables an execution of the woven model with tools that are not aware of aspects at all.

Our approach is supported by an integrated tool chain that is based on open-source Eclipse technology. Topcased<sup>2</sup> is used for modeling, models are handled by the UML2 plugin<sup>3</sup>, and ATL<sup>4</sup> serves as model transformation engine. We implemented the model weaving as a 2-step process using Two ATL model transformations respectively for pointcut resolving and advice weaving. The transformation rules in ATL are defined in terms of the meta-models of source and target models and thus are not dependent on a specific serialization format as required by other approaches [5].

#### 3.1 Pointcut Resolving

The first transformation step involves querying the input model for model elements that match pointcut expressions. The transformation adds a stereotype application to each matched model element, identifying it as a joinpoint shadow. Thus, an intermediate model is generated as the result of the pointcut resolving process. The implementation of pointcut resolving consists of declarative transformation rules for each joinpoint kind and ATL helper functions that implement the matching of pointcut expressions.

#### 3.2 Joinpoint Shadow Profile

The joinpoint shadow profile defines the joinpoint meta-model, i.e., the different kinds of joinpoints to which advice can be bound. Currently, our approach supports the joinpoint kinds that are most common in AOP, being method

call/execution and property get/set. Unlike a black-box joinpoint model [5] that basically uses generic message send and receive, we use specific actions and allow for potentially every UML action to be used as a joinpoint. Of course, profiles and transformations have to be extended to support new joinpoint kinds.

Joinpoint shadow stereotypes include references to all aspect bindings for which at least one pointcut matched a given action. These references are used in the next step to get the information needed for weaving, i.e., the advice and if it should be woven before, after, or around the joinpoint shadow.

### 3.3 Advice Weaving

In a second transformation step the actual aspect weaving is performed. It first locates all joinpoint shadows in the intermediate model and then processes their aspect bindings. Depending on the joinpoint and binding kind, different weaving strategies are applied.

To weave the additional advice behavior *around* a joinpoint shadow into the base model, first the captured joinpoint shadow is copied into a public accessible activity. This allows the invocation of private features and generalize further steps. Furthermore, the aspect infrastructure containing aspect classes with the selected instantiation mechanism (e.g., singleton) is created. Advices are represented as methods with extended signatures and are located in the corresponding aspect classes. The invocation of the captured joinpoint shadows from within an advice method is enabled by providing *Closure* classes and interfaces, using the same mechanism as described by Hilsdale and Hugunin [6]. *Proceed* actions are converted into invocations of appropriate methods enclosing by the closure objects, which are passed into the advices using the extended advice signature. Finally, the original joinpoint shadow is replaced by additional behavior like collecting context information, creating a closure object, instantiating the aspect and invoking the responsible advice method. The resulting behavioral model does not make any use of aspect-oriented constructs introduced in Section 2 and can be processed by standard MDA tools.

### 4. ILLUSTRATING EXAMPLE

Customer Relationship Management (CRM) involves three core processes: marketing, sales, and service. The sales process includes activities such as opportunity management, quotations, sales orders, and invoice processing. Opportunity management is about handling potential selling possibilities and turning them into sales orders.

In the *opportunity management* part of an object-oriented SAP CRM application, we identified several crosscutting concerns such as consistency checks, partner determination, compliance, access rights management, etc. To improve the modularization of these concerns in PIM models, we apply the approach we presented in Sections 2 and 3. For space limitations we concentrate on consistency checks.

Many consistency checks have to be performed when the state of the opportunity business object or some of the associated objects changes to verify that certain constraints are still satisfied. Some of these constraints are crosscutting, i.e., they involve attributes defined at least in two separate classes. For example, the constraints C1 and C2 are crosscutting constraints, which specify that an opportunity is inconsistent if they are true:

<sup>2</sup><http://www.topcased.org/>

<sup>3</sup><http://www.eclipse.org/modeling/mdt/?project=uml2>

<sup>4</sup><http://www.eclipse.org/m2m/at1/>

(C1) *Opportunity.processStatusValidSinceDate* < *SalesForecast.expectedProcessingDatePeriod.StartDate*  
(C2) *SalesCycle.phaseProcessingDatePeriod.StartDate* < *SalesForecast.expectedProcessingDatePeriod.StartDate*

To verify such consistency constraints, appropriate logic is needed. As more than one class is involved in these constraints, the logic that checks them is spread across several classes. This logic has to be executed when the fields corresponding to some constraints are modified and/or the respective setter methods are called. For example, assuming that the attributes that appear in C1 are private, enforcing that constraint would require logic in the method body of *setProcessStatusValidSinceDate* in the class *Opportunity* to verify that the date parameter is smaller than the value of the *expectedProcessingDatePeriod.StartDate* in the associated *SalesForecast* object. In addition, enforcing C1 also requires similar logic in the method body of *setExpectedProcessingDatePeriod* to verify that the *StartDate* of the period parameter is greater than the value of the field *processStatusValidSinceDate* in the associated *Opportunity* object.

#### 4.1 Modeling C1 without Aspects

We modeled the behavior for selected opportunity management classes with activity diagrams and UML actions using Topcased. For example, Fig. 3 shows the method *setProcessStatusValidSinceDate*, which contains behavior for enforcing C1.

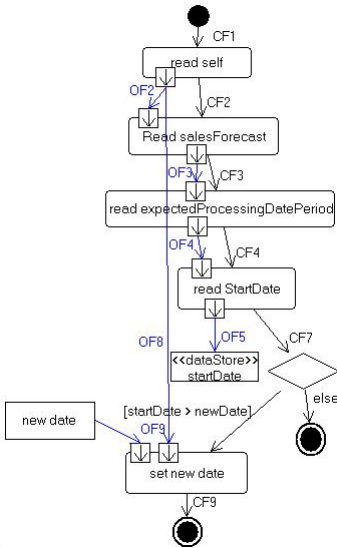


Figure 3: Method body of *setProcessStatusValidSinceDate*

#### 4.2 Modeling C1 with an Aspect

Next, we show how our approach can be applied to modularize the enforcement of C1 using the introduced aspect-oriented constructs. For that purpose, we will extract the check that is performed before the attribute *processStatusValidSinceDate* of class *Opportunity* is set into a separate module and we move the respective behavior into the activity depicted in Fig. 4. This activity is associated to an operation annotated with the stereotype *Advice*. The information, expected by the advice, such as the date to be set

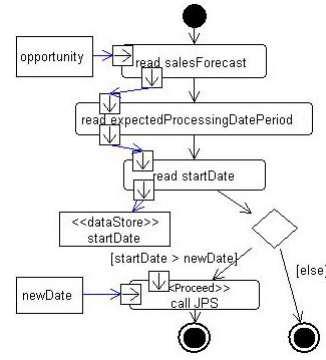


Figure 4: Advice for Enforcing C1 in *Opportunity*

and the class of business object (*Opportunity*), are passed into the advice using activity parameters. If the condition succeeds, the model elements representing intercepted joinpoints have to be processed. This is expressed by the stereotyped *CallOperationAction*. Since the target object of the *Proceed* action is not known at design time, it is not set in the model.

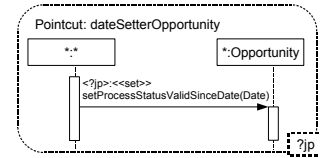


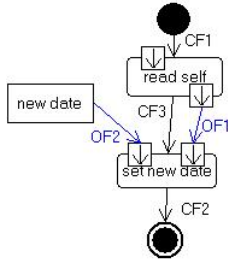
Figure 5: Pointcut *dateSetterOpportunity*

The extracted advice behavior can be used for intercepting all write accesses to the attribute *processStatusValidSinceDate* in class *Opportunity*. To this end, the pointcut *dateSetterOpportunity*, depicted in Fig. 5, is modeled using the constructs introduced in Section 2. The modeled pointcut can be visualized in an understandable way using *Join Point Designation Diagrams* [14]. The context information can be passed to the advice by associating the respective context exposure pointcut expressions (*argsPCE*, *ThisPCE*) with the parameters of the pointcut signature. During the weaving process, the required information are connected with the call to the advice operation using object flow connections. An advice can be bound to a pointcut using a *Binding*, which defines the binding kind. In our case an *around* binding is used to call the advice behavior instead of the joinpoint.

All created elements, such as advice, pointcuts, and bindings, are modularized within an aspect. Our approach also supports different aspect instantiation strategies. In the aspect model *ConsistencyCheck* the instantiation kind *singleton* indicates that all aspect-related advice invocations are done on a single aspect instance.

After the consistency check was modularized in a separate aspect, the behavior model of the method *setProcessStatusValidSince* (and all affected methods) contains only elements that are related to business logic, as depicted in Fig. 6.

To make the *ConsistencyCheck* model executable by common engines without support of AO semantics, advice weaving has to be processed. Our model weaver transforms the model according to the description in 3.3.



**Figure 6: Method body of setProcessStatusValidSinceDate without Consistency Check logic**

## 5. EVALUATION

In the following, we evaluate our aspect-oriented modeling approach where we focus on the software properties *understandability* and *maintainability*. For each property we choose some factors and respective metrics: *size* and *complexity* are two factors that affect understandability, whereas *separation of concerns* and *ease of change* are two factors that affect maintainability.

For *size*, we introduce two new metrics for the size of a behavioral model based on UML actions: *Number of Actions (NoA)* counts the total number of UML actions in a method or constructor body and *Number of Model Elements (NoME)* counts the number of control flows and object flows in addition to the actions. For *complexity*, we use the *Cyclomatic Method Complexity (CC)* metric, which measures the flow complexity of a method [10] by counting the number of places in its body where the flow changes from a linear flow. For *ease of change*, we adapt two metrics from [9] to our approach to measure the level of difficulty for changing the model elements that belong to a crosscutting concern: *Number of Impacted Components (NIC)* counts the number of classes and aspects affected by a change and *Number of Impacted Members (NIM)* counts the number of affected operations and attributes. For *separation of concerns*, we use *Concern Diffusion over Modules (CDM)* [13], which counts the number of classes and aspects contributing to the implementation of a concern and *Concern Diffusion over Operations (CDO)* [13], which counts the number of methods and advice contributing to the implementation of a concern.

Table 1 presents the measured data for both the object-oriented and the aspect-oriented design of the behavioral models of the two methods involved in enforcing the constraint C1. For short, we refer in that table to `setExpectedProcessingStartDate` by *sEPSD* and to `setProcessStatusValidSinceDate` by *sPSVS*. *Advice OPP* refers to the advice that enforces C1 on Opportunity objects whereas *advice SF* refers to the advice that enforces C1 on SalesForecast objects.

We observe that the size of the methods `setExpectedProcessingStartDate` and `setProcessStatusValidSinceDate` has been reduced when the logic for enforcing the constraint C1 is externalized into an aspect. In fact, the number of actions in each method decreased from 5 to 2 actions as well as the number of model elements, which went down from 18 and 19 to 7 and 6 respectively. Although this complexity is moved to the advice, the overall complexity in the aspect-oriented design (i.e., sum of actions and model elements in the base model and in advice) is less than in the

object-oriented design. Moreover, the cyclomatic complexity of these two methods decreased as the logic for enforcing C1 is no longer part of them. However, there is some complexity, which is hidden in the pointcut expressions as users should understand when the advice behavior joins the core behavior. This complexity can be reduced by developing similar tools to AJDT<sup>5</sup> for aspect-oriented modelling.

We also observe that the CDM value went down from 2 to 1 for the specific consistency check C1. When other consistency checks are considered the CDM value goes down from higher values (i.e., the number of classes with logic for consistency checks) to 1. As shown by the NIC metric, which went down from 2 at least to 1, the scope of changes to crosscutting concerns is reduced to the aspects. As a result, the time and cost overhead for identifying places where a crosscutting constraint should be checked and verifying the correctness of such modifications is reduced.

As we got similar evaluation data also for other examples of crosscutting concerns, the evaluation confirms our position statement and shows that aspect-oriented modularization concepts improve the understandability and maintainability in executable UML models.

## 6. RELATED WORK

Various approaches have been proposed, to introduce aspect-oriented semantics at the level of UML models. In [2] a survey of the most recent approaches is provided. This section presents approaches that are most relevant to our work.

Evermann [4] presents an approach to support AspectJ modeling in UML. The meta-model, realized as UML profile, offers constructs and concepts for designing AspectJ applications. In the context of MDA, this approach allows for platform specific models only, which are intended to be transformed into AspectJ code. In contrast to our approach, it provides no means for modeling advice-specific behavior. Consequently, aspect weaving and respective behavioral adaptations at model level cannot be supported by this approach. Moreover, pointcuts reference joinpoints directly and have to be assigned "manually" by the designer. Our expression-based approach for modeling pointcuts allows developers to declare a set of joinpoints by quantifying over their properties. This approach provides more flexibility, however, requires a pointcut resolving mechanism to select targeted model elements.

Since our approach was partially inspired by the idea of Fuentes and Sanchez, there exist some similarities. Nevertheless, the approach as presented in the paper [5], differs conceptually regarding the aspect-oriented modeling and weaving approach, and in the used technology. Fuentes and Sanchez use a generic joinpoint model that focuses on message communication between objects. The context exposure for advice constructs is realized by using reflective actions. We use explicit advice and pointcut signatures that allow for typed context information. Therefore, the information provided by selected joinpoints can be statically validated regarding the parameters expected by advice. In the approach of Fuentes and Sanchez, the binding information are modeled in pointcuts, which prevents the reuse of pointcuts. Similar to our approach, Fuentes and Sanchez use 2-step aspect composition (pointcut resolving and aspect weaving). However, their transformations are based on XSLT and con-

<sup>5</sup><http://www.eclipse.org/ajdt/>

Metric	Value in OO Design	Value in AO Design
Num. of Actions	5 per method	2 per method, 4 advice OPP, 3 advice SF
Num. of Model Elements	18 sEPSD, 19 sPSVS	7 sEPSD, 6 sPSVS, 16 advice OPP, 13 advice SF
Cyclomatic Complexity	2 per method	1 per method, 2 per advice
Concern Diff. o. Operations	2 for C1	2 for C1 (the 2 advice)
Concern Diff. o. Modules	2 for C1	1 for C1 (the aspect)
Num. of Impacted Components	2 at least	1 (the aspect)
Num. of Impacted Members	2 at least (the 2 setter methods)	2 advice

**Table 1: Evaluation data for both designs**

sume models in the XMI format. In contrast, our approach uses a model-driven weaving mechanism (ATL transformations on meta-model, which works directly on EMF model repositories) and can be integrated into an existing MDA tool chain more easily. The weaving process as described in [5] inlines the advice behavior as Structured Activities into the base model. There is no appropriate aspect representation in woven models. Hence, aspects with the need of context information and helper operations are difficult to realize.

## 7. CONCLUSIONS

In this paper we have presented an approach for extending the UML with aspect-oriented modularization concepts and effectively shown (i) that aspect-oriented modularization concepts can be introduced on the abstraction level of UML models with a similar semantics as known from AOP, (ii) crosscutting concerns in executable UML models can be modularized in separate aspects, and (iii) an UML compliant extension using profiles and meta-classes allows for first-class aspect-oriented model elements, small transformations in the model weaver, and the use of standard tooling for the implementation.

Furthermore, we presented a case study in which we have evaluated our approach in a real world CRM application. In this study we increased the quality of the models regarding size, complexity, separation of concerns, and ease of change, using aspect-oriented modularization concepts. Hence, the understandability and maintainability of platform independent models representing crosscutting concerns could be improved in industrial relevant application scenarios. Finally, we have shown that it is possible to implement such an approach in a reasonable way, with no need for special tools, but using standard, meta-model based open-source technology.

## 8. REFERENCES

- [1] Homepage of the AspectJ Programming Language. <http://www.eclipse.org/aspectj>.
- [2] R. Chitchyan, A. Rashid, P. Sawyer, A. Garcia, M. P. Alarcon, J. Bakker, B. Tekinerdogan, S. Clarke, and A. Jackson. Report synthesizing state-of-the-art in aspect-oriented requirements engineering, architectures and design. Technical report, Lancaster University, AOSD-Europe Deliverable D11, AOSD-Europe-ULANC-9, May 2005.
- [3] T. Cottenier, A. van den Berg, and T. Elrad. Motorola WEAVR: Aspect Orientation and Model-Driven Engineering. In *Journal of Object Technology*, vol. 6, no. 7, *Special Issue: Aspect-Oriented Modeling*, pages 51–88, August 2007.
- [4] J. Evermann. A Meta-Level Specification and Profile for AspectJ in UML. In *Journal of Object Technology*, vol. 6, no. 7, *Special Issue: Aspect-Oriented Modeling*, pages 27–49, August 2007.
- [5] L. Fuentes and P. Sánchez. Designing and Weaving Aspect-Oriented Executable UML models. In *Journal of Object Technology*, vol. 6, no. 7, *Special Issue: Aspect-Oriented Modeling*, pages 109–136, August 2007.
- [6] E. Hilsdale and J. Hugunin. Advice weaving in aspectj. In *AOSD '04: Proceedings of the 3rd international conference on Aspect-oriented software development*, pages 26–35, New York, NY, USA, 2004. ACM.
- [7] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An Overview of AspectJ. In J. L. Knudsen, editor, *Proc. ECOOP 2001, LNCS 2072*, pages 327–353, Berlin, June 2001. Springer-Verlag.
- [8] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. Technical Report SPL97-008 P9710042, Xerox PARC, Feb. 1997.
- [9] M. L. Lee. *Change Impact Analysis of Object-Oriented Software*. PhD thesis, George Mason University, Virginia, USA, 2000.
- [10] McCabe and Associates. Object-Oriented Tool User's Instructions, 1994.
- [11] S. Mellor and M. Balcer. *Executable UML: A Foundation for Model Driven Architecture*. Addison-Wesley Professional, 2002.
- [12] Object Management Group. UML Superstructure Specification, v2.1.1, 2007. OMG Document formal/2007-02-05.
- [13] C. Sant'anna, A. Garcia, C. Chavez, C. Lucena, and A. v. von Staa. On the reuse and maintenance of aspect-oriented software: An assessment framework. In *Proceedings of the XVII Brazilian Symposium on Software Engineering*, 2003.
- [14] D. Stein, S. Hanenberg, and R. Unland. Expressing different conceptual models of join point selections in aspect-oriented design. In *AOSD '06: Proceedings of the 5th international conference on Aspect-oriented software development*, pages 15–26, New York, NY, USA, 2006. ACM.