5th International Workshop on Aspect-Oriented Modeling

Dominik Stein[1], Jörg Kienzle[2], Mohamed Kandé[3]

[1]University of Duisburg-Essen, Schützenbahn 70, D-45117 Essen, Germany
dstein@cs.uni-essen.de
[2]School of Computer Science, McGill University, Montreal, QC H3A 2A7, Canada
Joerg.Kienzle@mcgill.ca
[3]Condris Technologies, Switzerland
Mohamed.Kande@condris.com

Abstract. This report summarizes the outcome of the 5th Workshop on Aspect-Oriented Modeling (AOM) held in conjunction with the 7th International Conference on the Unified Modeling Language – UML 2004 – in Lisbon, Portugal. The workshop brought together researchers and practitioners from two communities: aspect-oriented software development (AOSD) and software model engineering. It provided a forum for discussing the state of the art in modeling crosscutting concerns at different stages of the software development process: requirements elicitation and analysis, software architecture, detailed design, and mapping to aspect-oriented programming constructs. This paper gives an overview of the accepted submissions, and summarizes the results of the different discussion groups.

1 Introduction

This paper summarizes the outcome of the 5th edition of the successful Aspect-Oriented Modeling Workshop series. An overview of what happened at previous editions of the workshop can be found at [12].

The workshop took place at the Vila Galé Opéra Hotel in Lisbon, Portugal, on Monday, October 11th 2004, as part of the 7th International Conference on the Unified Modeling Language – UML 2004 [1]. Participation was based on the submission of a position paper addressing aspect-oriented modeling issues. A total of 17 papers were submitted and reviewed by the program committee, 13 of which were accepted to the workshop. In order to leave enough time for discussion, only a one-and-a-half-hour session was dedicated to presentations. Four papers were chosen as representatives of the workshop submissions, and intended to stimulate and provide provocative input to the following discussion sessions. In the late morning session, the attendees split into three groups to independently discuss specific questions concerning the eligibility of concrete UML model elements to represent aspect-oriented concepts. From there, the discussions quickly led to other topics. The results of the discussion groups were collected at the end of the workshop, and presented and re-discussed with the entirety of the workshop participants. Finally, a catalogue of questions indented as an agenda for future research was established.

The rest of this report is structured as follows: Section 2 gives an overview to the accepted papers. Section 3 summarizes the results of the discussion groups. Section 4 concludes the report and presents identified future research directions.

2 Overview of Accepted Position Papers

A total of 13 papers were accepted to the workshop (see below). This section presents a brief overview of the papers, organized according to the software development life cycle phase they apply to. This structure should make it easier for an interested reader to identify the submissions that pertain to his / her research area.

Some of the papers deal with modeling of aspects at the requirements phase: *Navarro et al.* (8), for example, introduce a UML profile for ATRIUM [7], a methodology that allows the concurrent definition of requirements and software architectures using both goal-oriented and aspect-oriented techniques. The profile presented in the paper comes with a graphical notation that helps visualizing goal models. Within those goal models, candidate aspects arise implicitly from goals that participate in weaving relationships. *Spies et al.* (11) explain i* [13] ("eye-star"), an early-requirements engineering technique developed by *Yu*. They describe how the models being generated using this technique may be mapped to "concern templates" as introduced by *Brito* and *Moreira* [2], thus allowing the identification of candidate aspects.

One paper concentrated on domain models: In his paper, *Steimann* (12) claims that domain models are inherently aspect free. He picks up common usages of the term "aspect" in modeling, and investigates their meaning in software modeling: He compares aspects to roles, discusses aspects as ordering dimensions, takes a look at domain-specific aspects, and reasons on aspects of modeling. Finally, he provides a proof of his claim with help of predicate logic.

Other papers focus on the impact of aspect-oriented techniques on the design phase and the design process: For example, *Li et al.* (4) elucidate the necessity of having an aspect-oriented design in order to achieve the full benefits when it comes to (aspect-oriented) programming. On the other hand, they identify limitations in prevailing aspect-oriented programming languages that impose important drawbacks on the (aspect-oriented) design. Having said that, they describe a development process that copes with these problems. *Park* and *Kang* (9) present an approach to support design phase performance simulation and analysis with help of aspect-oriented techniques. They make use of separate models to specify the core functionality and the performance requirements of a system, and map them to distinct AspectJ [6] code modules. By keeping performance specifications separate from functional specifications, they gain benefits in both the specification and the adaptation ("feedback") of the performance requirements. *Kamalakar* and *Ghosh* (3) illustrate how the aspect-oriented modeling technique developed by *France et al.* [3] can be used to encapsulate middleware-specific functionality in distinct design models. Later, these models can be woven into models specifying the business functionality of an arbitrary application, thus leading to a higher reusability of the middleware-specific design. The ideas are illustrated on a case study using CORBA.

Besides that, some papers were concerned with the verification and testing of aspect-oriented models: *Nakajima* and *Tamai* (7), for example, propose to use Alloy [5], a lightweight formal specification language and analysis tool, for the precise description of models as well as their verification. They demonstrate the use of Alloy – i.e., how to specify aspects and how to weave them – with help of two examples, logging and access control. *Tessier et al.* (13) present a formalized method, based on static model analysis, for the early detection of conflict in aspect models. To do so, they investigate the crosscutting relationships between aspects and their base classes, summarize the outcome in a table, and feed the collected data to formal expressions. That way they are able to detect possible conflicts in the ordering of aspects, in transverse specifications, or accidental recursion, etc.

Multiple papers deal with modeling notations: *Barra et al.* (1) investigate the UML 2.0 specification [10] in order to find suitable abstraction means for the representation of aspect-oriented concepts. In particular, they analyze the new UML 2.0 elements Ports and Connectors – in connection with the revised specification of Interfaces, Association Classes, and Components – and discuss their eligibility to represent join points, advice, introductions, etc. *Han et al.* (2) introduce a MOF [9] meta-model for AspectJ – capturing the syntax and semantics of each of its language constructs – in order to allow the modeling of AspectJ programs. They give a detailed description of each meta-class, its semantics, its attributes, as well as the associations it participates in. Further, they provide a corresponding visualization. *Muller* (6) presents View Components, an approach to compose generic view models (each capturing a particular functionality) to a given base model. He points out the problems encountered when using relationships to express composition directives, and explains why parameterization might overcome some of these problems. *Mahoney et al.* (5) focus on the specification of aspects using state charts. Using aspect-oriented techniques, the authors show how to define abstract statecharts that can later be woven into other statecharts, thus making behavior models reusable. Their approach bases on the reinterpretation of certain events of one statechart in the other statecharts. *Reina et al.* (10) inspect numerous existing aspect-oriented modeling approaches and observe that most of them are closely related to particular aspect-oriented programming platforms. They propose to rise the level of abstraction of aspect models to platform-independent models (PIMs) in the Model-Driven Architecture (MDA) [8]. In order to express each aspect PIM most appropriately, aspect-specific profiles or meta-model extensions should be used.

List of Position Papers

(1) Barra, E., Génova, G., Llorens, J., *An Approach to Aspect Modeling with UML 2.0*

(2) Han, Y., Kniesel, G., Cremers, A.B., *A Meta Model and Modeling Notation for AspectJ*

(3) Kamalakar, B., Ghosh, S., *A Middleware Transparent Approach for Developing CORBA-based Distributed Applications*

(4) Li, J., Houmb, S.H., Kvale, A.A., *A Process to Combine AOM and AOP: A Proposal Based on a Case Study*

(5) Mahoney, M., Bader, A., Elrad, T., Aldawud, O., *Using Aspects to Abstract and Modularize Statecharts*
(6) Muller, A., *Reusing Functional Aspects: From Composition to Parameterization*
(7) Nakajima, S., Tamai, T., *Lightweight Formal Analysis of Aspect-Oriented Models*
(8) Navarro, E., Letelier, P., Ramos, I., *UML Visualization for an Aspect and Goal-Oriented Approach*
(9) Park, D., Kang, S., *Design Phase Analysis of Software Performance Using Aspect-Oriented Programming*
(10) Reina, A.M., Torres, J., Toro, M., *Separating Concerns by Means of UML-Profiles and Metamodels in PIMs*
(11) Spies, E., Rüger, J., Moreira, A., *Using i* to Identify Candidate Aspects*
(12) Steimann, F., *Why Most Domain Models are Aspect Free*
(13) Tessier, F., Badri, L., Badri, M., *Towards a Formal Detection of Semantic Conflicts Between Aspects: A Model-Based Approach*

3 Results of Discussion Groups

A primary goal of the workshop was to provide a platform for researchers to discuss the impact of aspect-oriented software development on software model engineering, and vice versa. Therefore, a major part of the workshop was spent debating on important aspect-oriented modeling issues. In order to maximize productivity, the participants split into three discussion groups, each of six to eight persons. The results of the discussion are summarized in the following subsections.

3.1 Reasons for Aspect-Oriented Modeling

During the workshop, we observed that different participants had different motivations and expectations regarding the new area of aspect-oriented modeling. One of the contributions of this workshop was to identify and discuss the benefits of expressing aspects at software modeling level. Many participants agreed that aspect-oriented modeling is important because it expresses crosscutting structures and behavior at a higher level of abstraction than aspect-oriented code. However, several other interesting opinions were expressed. In particular, people expect aspect-oriented modeling to provide means for:

- *Resolving conflicts in software models*. The idea is to use aspects at modeling level to allow designers detect and resolve conflicts at early stages of the development process. Typically, code-level conflicts that result from weaving processes or aspect compositions should be detected and resolved at early stages, not at execution time.
- *Modeling reusable business rules*. The idea is to express business rules as aspects in software models that can be reused for various systems, and at different levels of abstraction.
- *Model evolution and maintenance*. Similar to programming level aspects, modeling aspects need to provide mechanisms for modularizing crosscutting concerns in software models in order to facilitate both the evolution and maintenance of models.

- *Expressing reusable functions*. The idea is that aspects can be used to express reusable functions, such as annotation diagrams for performance measurements.
- *Managing requirements*. The idea is that requirements captured from different stakeholders are naturally entangled; they should be expressed as separate aspects of the system at hand. People hope that advancing aspect-oriented modeling can help separate, combine and/or manage requirements.

To achieve these expectations much research needs to be done. We hope that submissions to future workshops will address some of the above issues.

3.2 Aspect-Oriented Modeling and Terminology

At the workshop, one discussion dealt with the terminology in aspect-oriented modeling and its relation to the terminology in aspect-oriented programming. Some of the participants thought that aspect-oriented software development in general and aspect-oriented modeling in particular could benefit from the definition of an aspect-oriented vocabulary. Terms such as "aspect", "join point" and "weaving" might have similar, yet slightly different meanings at different levels of software development – similar to what happens in object-orientation: High-level analysis objects are not identical to programming language-level objects.

Firstly, the definition of "aspect" from the programming language-level was looked at: an aspect at the programming language-level is a modularized implementation of a concern that otherwise (in a non-aspect-oriented implementation) would crosscut the program structure (or its behavior). The two key elements in this sentence are "modularize" and "crosscut". Most participants agreed that an aspect at the model-level is a modularized model of a concern that otherwise (in a non-aspect-oriented model) would crosscut the main model structure. Some participants, however, interpret the word "aspect" more like what others call "concern", and hence believe that especially at the early stages of software development there are no such things as "crosscutting aspects". At that level, every concern is a first-class citizen, so to speak. It was discovered that the particular conception of "aspects" often differs considerably depending on the abstraction level that researchers are working on (cf. section 3.3, as well).

Then the discussion moved on to try and define the term "join point", which was initially suggested to designate at modeling-level all those points at which models can be merged / composed / woven together. Unfortunately, the term "join point" was deemed problematic, as it is coined by AspectJ, coming with a well-defined meaning that may cause considerable misconception when used in the modeling domain. "Join points" in the modeling domain commonly refer to a more generalized concept, such as "some points where aspects can hook onto" (for example). The question arose whether or not "join points" should be named differently in the modeling domain to highlight this distinction.

Likewise, the term "weaving" was put to question, and opposed to "composition". "Composition" of models has been known in software modeling for a long time, and the knowledge gained in this area of research helps the aspect-oriented modeling community to specify and assess the effects of model-"weaving". However, it remains unclear if both terms may be used as true synonyms, or whether "weaving" is a special kind of "composition".

Unfortunately, no general consensus was reached for any of the terms "aspect", "join point", and "weaving". While one part of the participants believed that using an aspect-oriented vocabulary throughout the software development life cycle would benefit the aspect-oriented software development community, the other participants deemed it too early to try and define these terms in a concise way.

3.3 Aspects in the Modeling Process

During the discussion it became manifest that in the conventional software development process – i.e., in requirements engineering, analysis, design, and implementation – different aspects appear in different phases and on different levels of abstraction. Requirement level aspects such as maintainability or reusability are specified during early stages of software development, for example, while other issues such as caching or synchronization seem to be rather implementation level aspects that cannot easily be traced back to particular requirements in all cases. Finally, there are concerns that are present throughout the entire software lifecycle, e.g., security, persistence, or auditing, which are most likely to take different forms during development. At one level, they might be modeled as aspects, i.e. their model crosscuts the main model structure, but on other levels they do not.

It turns out that expectations for, and problems of, aspect-oriented modeling often differ considerably between development phases and abstraction levels. Differences exist, for example, in what should be modeled as an aspect, what should be regarded as a join point, how and when aspects should be composed with the primary model, etc. Therefore, authors were asked to clearly indicate at which level of abstraction, or phase of software lifecycle, their work is situated in order to avoid misunderstandings during the discussion.

3.4 Aspect-Oriented Modeling and UML Model Elements

One major issue in the discussion groups was to elucidate to what extent existing UML model elements are capable of expressing aspect-oriented concepts, and/or why they fail to do so. The UML elements of interest were: UML classes, UML packages, UML collaborations, UML use cases, UML templates, UML (2.0) components, ports and connectors, OCL statements, as well as *sets* of UML diagrams (e.g., class diagrams, collaboration diagrams, state charts, etc.). These model elements were collected from previous work on aspect-oriented modeling and current workshop submissions, supplemented by brain storming in the morning session of the workshop, and then discussed by each individual group.

Everyone agreed that current UML model elements were deemed to lack important characteristics of aspects:

The UML class construct provides a module for encapsulating state and behavior, and therefore seems suitable to model an aspect. Abstract classes, for instance, could be used to encapsulate partial behavior, which can then be completed and reused in subclasses. However the class construct alone is not sufficient to model the encapsulated behavior; additional UML diagrams such as state diagrams or sequence diagrams are needed. Also, a class cannot expose required interfaces, or join points, to the rest of the system.

The new UML 2.0 component element offers that possibility to expose required interfaces. This feature could be used to explicitly declare a component's join points as part of its interface. This means, however, that aspects can only hook onto such declared join points, which makes it hard to handle unforeseen extensions.

UML sequence diagrams were briefly discussed regarding the new fragments feature introduced in UML 2.0, which might make it possible to declare potential extension points.

UML templates allow a modeler to expose interfaces, and to reuse partial model elements and configure them to his / her needs. Unfortunately, standard UML templates are not powerful enough. All currently known approaches that use templates to model aspects had to extend the template mechanism and step outside the UML.

Finally, UML packages were looked at. They are the most general UML model element, since they can contain any other diagram. They provide a nice means for separating and grouping together all elements related to a certain concern, and therefore do a great job in modularizing an aspect. Their capabilities, however, are limited when trying to model the weaving, i.e. showing how elements of an aspect affect external entities.

To summarize, aspects in aspect-orientation were identified to meet much of the characteristics of the previously mentioned UML elements, such as:

- being the encapsulation of (some structural and/or behavioral) properties,
- being first-class entities that can exist on their own right,
- being instantiatable classifiers that can have multiple instances, each having its own state, etc.

However, essential differences between aspects in aspect-orientation and existing model elements in the UML were identified as well. For example, aspects were identified to:

- provide introspection capabilities (e.g. pointcuts) and intercession capabilities (e.g. pieces of advice or introductions)
- provide a mechanism to define (extrinsic) properties of other elements
- break encapsulation of other elements, etc.

Concerning the introspection capabilities of aspects, possible solutions were sketched during the discussion: Pointcuts, such as in AspectJ for example, were identified as declarative expressions of introspection. These expressions could be looked at as *patterns* that are matched against the elements of the base system. Looking for ways to express patterns in the UML, the participants identified UML templates and the Object Constraint Language (OCL) as possible solutions. Adaptations to these approaches were deemed necessary, though, as UML templates are designed to generate – rather than match – model elements, while the OCL is not capable of expressing timely patterns (such as denoted by AspectJ's *cflow* construct), for example. Possible help on how to deal with introspection capabilities on the modeling level could also be found in the MOF specification, as it already contains reflective capabilities.

In conclusion, the discussed ideas were deemed to be still very immature, and therefore need to be further investigated. Finding appropriate modeling solutions that can address all issues mentioned above was considered subject for future research.

3.5 Aspect-Oriented Modeling and Weaving of Models

Another important discussion topic dealt with the question how to weave models, and what model elements should serve as join points. In general, weaving on the modeling level was deemed to be more powerful than weaving on the implementation level: Firstly, weaving on the modeling level can use any model element as join point (rather than being restricted to some Join Point Model as in AspectJ, for example). Secondly, weaving on the modeling-level brings aspect-orientation to just any platform, and hence makes aspect-oriented development independent of the availability of an aspect-oriented compiler for the programming language used for implementation.

A general problem of weaving pertains to the correctness of models. Since aspects break up encapsulation of the primary model elements, weaving is most likely to change their semantics. Therefore, means must be found to identify and resolve weaving conflicts, and research in validation and testing of woven models is essential.

Besides that, parallels between weaving of models in aspect-oriented modeling and the transformation of models in the Model-Driven Architecture (MDA) have been pointed out. In a MDA context, weaving was considered to be a horizontal model transformation – rather than a vertical model transformation (which was considered to represent a refinement). The Query View Transformation (QVT) language was deemed to bring much benefit to the specification of weaving.

Another matter of interest was to identify the point in the software development process when an aspect is woven with the rest of the model (e.g., at the requirements elicitation phase, during the design phase, or not until the implementation phase). Currently there seems to be no general rule that determines the ideal time for weaving. Some aspects might be woven at an early stage, some aspects only at implementation-time, depending on the kind of aspect and the specific application is applies to. The workshop participants expressed the desire to find more general criteria, or heuristics, that would provide a guidance that helps developers to determine when weaving of which aspects should be accomplished, at what point in time, and in what order.

Finally, the issue of symmetric models vs. asymmetric models was raised [4]. In an asymmetric model, there exists a base model of the system under development that captures its main functionality. At weave time, aspects get woven into the base model, and hence the weaving transformation is asymmetric. In symmetric models, no distinction is made among different models because of the concern they address. There is no base, and hence the weaving transformation is symmetric.

Not much time was spent on discussing these two models, but it seems like the differences are similar to the ones identified at the programming language level, for example when comparing AspectJ [6] to Hyper/J [11]. The asymmetric model is easier to work with on a conceptual level. It simplifies identifying aspects since they crosscut the base model. Also, the asymmetric model can be seen as an add-on to standard object-oriented modeling. The symmetric model is appealing because of its simplicity, but might require a complete rethinking of the way we do modeling.

4 Concluding Remarks and Outlook

The purpose of this paper is *not* to provide a complete and widely accepted opinion on aspect-oriented modeling of all the authors, organizers, and workshop participants. Our intention is to give an essential input for future research on aspect-oriented modeling, pointing thus researchers to current problems and possible matters of interest. To do so, the authors' goal was to draw a full picture of all topics that have been discussed at the workshop.

There are several active research groups in the aspect-oriented community and the software model engineering community working on theoretical and practical aspect-oriented modeling issues. However, as the workshop discussions have shown, there is still lots of interesting work to be done to make aspect-oriented modeling cover the whole software development lifecycle. In particular, we need to make use of a widely accepted vocabulary; to provide well-defined modeling elements for "aspects"; to define a standardized way of identifying "join points", and supporting "weaving" mechanisms, while allowing modelers to evaluate and validate alternative aspect-oriented designs. Workshops such as this one can play a major role in addressing the above modeling issues.

At the end of the workshop, the participants were asked to provide a list of important questions to be looked at in a near future. They will be considered when establishing the agenda for envisioned successor workshops. The identified questions were:

- What is the benefit of using aspect-oriented modeling? What are the reasons for using it in the context of each software development phase?
- What prior art applies to aspect-oriented modeling? What can be learned from its origins, e.g. object-oriented abstraction mechanisms, model composition and transformation, and techniques using reflection?
- How can modeling notations visualize aspect-specific peculiarities? For example, how can we depict aspect-oriented introspection and intercession capabilities?

Acknowledgements

We'd like to thank everybody who helped to make this workshop a success: In particular, the members of the organization committee who could not make it to the workshop, i.e., Faisal Akkawi, Omar Aldawud, Grady Booch, Tzilla Elrad and Jeff Gray; the members of the program committee, Mehmet Aksit, Atef Bader, Siobhán

Clarke, Bill Harrison, Hoda Hosny, Karl Lieberherr, Peri Tarr and Aida Zakaria. Finally, we would like to thank all the submitters and participants.

References

[1] Baar, Th., Strohmeier, A., Moreira, A., Mellor, St., Proc. of 7th International Conference on the Unified Modeling Language 2004, Lisbon, Portugal, October 10-15, 2004

[2] Brito, I., Moreira, A., *Integrating the NFR Framework in a RE Model*, Early-Aspects Workshop at 3rd International Conference on Aspect-Oriented Software Development 2004, Lancaster, UK, March 22, 2004

[3] France, R., Kim, D.K., Georg, G., Ghosh, S., *An Aspect-Oriented Approach to Design Modeling*, in: IEE Proc. – Software, Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, to appear in 2004

[4] Harrison, W., Ossher, H., Tarr, P., *Asymmetrically vs. Symmetrically Organized Paradigmes for Software Composition*, TR RC22685, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA, December 2002

[5] Jackson, D., Shlyakhter, I., Sridharan, M., *A Micromodularity Mechanism*, in: Proc. of 9th International Symposium on Foundations of Software Engineering 2001, Vienna, Austria, September 10-14, 2001, pp. 62-73

[6] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G., *An Overview of AspectJ*, in: Proc. of the 15th European Conference on Object-Oriented Programming 2001, Budapest, Hungary, June 18-22, 2001, pp. 327-353

[7] Navarro, E., Ramos, I., Pérez, J., *Software Requirements for Architectured Systems*, in: Proc. of 11th International Conference on Requirements Engineering 2003, Monterey, CA, September 8-12, 2003, pp. 356-366

[8] OMG, *MDA Guide*, Version 1.0, OMG Document omg/2003-05-01, May 2003

[9] OMG, *MOF 2.0 Core Final Adopted Specification*, OMG Document ptc/03-10-04

[10] OMG, *UML 2.0 Infrastructure Specification*, *UML 2.0 Superstructure Specification*, OMG Documents pct/03-09-15 and ptc/03-08-02

[11] Tarr, P., Ossher, H., Sutton, S., *Hyper/J: Multi-Dimensional Separation of Concerns for Java*, in: Proc. of the 24th International Conference on Software Engineering 2002, Orlando, Florida, May 19-25, 2002, pp. 689-690

[12] *The 5th International Workshop on Aspect-Oriented Modeling*, Homepage, List of Position Papers, and Schedule, http://www.cs.iit.edu/~oaldawud/AOM/index.htm

[13] Yu, E., *Modeling Strategic Relationships for Process Reengineering*, PhD Thesis, DKBS-TR-94-6, Department of Computer Science, University of Toronto, 1995