A Survey on Aspect-Oriented Modeling Approaches

A. Schauerhuber[1]*, W. Schwinger[2],
E. Kapsammer[3], W. Retschitzegger[3], M. Wimmer[1], and G. Kappel[1]

[1] Business Informatics Group
Vienna University of Technology,
Favoritenstrasse 9-11/E188-4, A-1040 Vienna, Austria
{andrea,wimmer,gerti}@big.tuwien.ac.at
[2] Department of Telecooperation
Johannes Kepler University Linz,
Altenbergerstrasse 69, A-4040 Linz, Austria
wieland.schwinger@jku.ac.at
[3] Information Systems Group
Johannes Kepler University Linz,
Altenbergerstrasse 69, A-4040 Linz, Austria
{ek,werner}@ifs.uni-linz.ac.at

Abstract. Aspect-orientation provides a new way of modularization by clearly separating crosscutting concerns from non-crosscutting ones. While aspect-orientation originally has emerged at the programming level, it now stretches also over other development phases. There are, for example, already several proposals to Aspect-Oriented Modeling (AOM), most of them pursuing distinguished goals, providing different concepts as well as notations, and showing various levels of maturity. Consequently, there is an urgent need for both, academia and practice, to provide an in-depth survey, clearly identifying commonalities and differences between current AOM approaches. Existing surveys in this area focus more on comprehensibility with respect to development phases or evaluated approaches rather than on comparability at a fine-grained level.

This paper tries to fill this gap. As a prerequisite for an in-depth evaluation, a conceptual reference model is presented, capturing the basic concepts of AOM and their interrelationships in terms of a UML class diagram. Based on this conceptual reference model, an evaluation framework has been designed by deriving a detailed and well-defined catalogue of evaluation criteria. The actual evaluation by means of this criteria catalogue and by employing a running example is done on the basis of a carefully selected set of eight AOM approaches, each of them having already reached a certain level of maturity. This per approach evaluation is complemented with an extensive report on lessons learned, summarizing the approaches' strengths and shortcomings.

* This research has been partly funded by the Austrian Federal Ministry for Education, Science, and Culture, and the European Social Fund (ESF) under grant 31.963/46-VII/9/2002.

1 Introduction

The concept of Separation of Concerns (SoC) can be traced back to Dijkstra [16] and Parnas [45]. Its key idea is the identification of different concerns in software development and their separation by encapsulating them in appropriate modules or parts of the software. Aspect-Oriented Software Development (AOSD), formerly also called Advanced Separation of Concerns (ASoC), adopts this idea and further aims at providing new ways of modularization in order to separate crosscutting concerns from traditional units of decomposition during software development. In particular, AOSD represents the convergence of different ASoC approaches, such as Adaptive Programming (AP) [39], Composition Filters (CF) [1], Subject-Oriented Programming (SOP) [25], Multi-Dimensional Separation of Concerns (MDSoC) [67], and Aspect-Oriented Programming (AOP) [33]. AOSD, nevertheless, is a fairly young but rapidly advancing research field.

Aspect-Oriented Modeling. From a software development point of view, aspect-orientation has originally emerged at the programming level with AspectJ[1] as one of the most prominent protagonists. Meanwhile, the application of the aspect-oriented paradigm is no longer restricted to the programming level but more and more stretches over phases prior to the implementation phase of the software development life cycle such as requirements engineering, analysis, and design. This development is also driven by the simultaneous rise of Model-Driven Engineering (MDE) employing models as the primary artifact in software development [56]. In the context of this, Aspect-Oriented Modeling (AOM) languages attract more and more attention.

Diversity of Approaches. As a result, there has already been a considerable number of AOM languages proposed in literature, whereof only a few in the meanwhile have come of age. Each of those AOM approaches has different origins, e.g., AOP and SOP, and pursues different goals. AOM approaches, for example, follow different school of thoughts some adhering to an asymmetric view supporting the distinction between aspect and base [26], while others have a symmetric understanding where such a distinction is not made. This entails not only the problem of different terminologies but also leads to a broad variety of aspect-oriented concepts, including composition mechanisms used [38], as well as notations.

Contributions. To cope with this rapid development, there is an urgent need for both, academia and practice to provide an in-depth survey of existing AOM approaches. Existing surveys in this area mainly focus on comprehensibility with respect to development phases or evaluated approaches, thereby getting a first impression about the state of the art. This paper tries to complement this valuable work by focusing on comparability at a fine-grained level, both concerning the language's concepts, but also their notations. Thus, a deeper understanding on commonalities and differences between existing AOM approaches can be established, especially with respect to their specific strengths and shortcomings.

[1] http://www.eclipse.org/aspectj/

Structure of the Paper. The structure of the paper is as follows. In Section 2, the contributions of this survey with respect to other existing surveys are discussed in detail. Section 3 presents a conceptual reference model in order to centrally capture the basic concepts of AOM and their interrelationships in terms of a UML class diagram. On this basis, an evaluation framework is set up by deriving a detailed and well-defined catalogue of evaluation criteria in Section 4. Section 5 is dedicated to the actual evaluation of eight selected AOM approaches by means of this criteria catalogue. To better illustrate especially the notational peculiarities of the AOM approaches this evaluation is accomplished with a running example. The findings are finally summarized and lessons learned are provided, in Section 6, before we make an outlook on future work in the final Section 7 of this paper.

2 Related Surveys

In an effort to shed light on different approaches to aspect-orientation, some surveys comparing aspect-oriented approaches at different levels in the software development life cycle have already been presented. In the following, such related surveys can be distinguished into "closely related" surveys particularly emphasizing on AOM (cf. Section 2.1) and more "widely related" ones focusing on AOP (cf. Section 2.2), which are nevertheless of interest in the context of this survey. Furthermore, there exists some work aiming at "unifying" the currently prevailing diversity of concepts in the aspect-orientation paradigm. The influence of those on this survey in terms of the CRM is discussed in detail in the Section 3.

2.1 Aspect-Oriented Modeling Surveys

With respect to closely-related surveys on AOM approaches, the most extensive work is provided by Chitchyan et al. [5]. This survey's goal is to "elicit initial insights into the roadmap for developing integrated aspect-oriented requirements engineering, architecture, and design approaches". Therefore, for each phase of the software development process a review of prominent contemporary AOSD as well as non-AOSD approaches is provided. For the design phase, the survey presents the evaluation results of 22 aspect-oriented design approaches along with UML as the only non-AOSD approach on the basis of a set of 6 evaluation criteria.

Similar, but less extensive AOM surveys with respect to both the set of criteria and the amount of surveyed approaches have been provided by Reina et al. [52], Blair et al. [4], and Op de beeck et al. [15]. Reina et al. have evaluated 13 approaches with respect to a set of 4 criteria, only. More specifically, the goal of Reina et al. has been to investigate each approach with respect to its dependency on particular platforms as well as its dependency on specific concerns, i.e., if the approach is general-purpose or not.

The major goal in Op de beeck et al. is to investigate 13 existing AOM approaches within the realm of product-line engineering of large-scale systems and to position them within the full life cycle of a software development process. In this respect, the authors have evaluated a subset of approaches already presented by Chitchyan et al., as well as refined a set of six criteria, which partly have been presented in Chitchyan et al. In addition, the authors provide a discussion of the criteria's impact on certain software quality factors.

Blair et al. provide separate sets of criteria for the phases of aspect-oriented requirements engineering, specification, and design. Concerning the design phase, the authors evaluate 5 approaches according to a set of 8 criteria.

With respect to these existing surveys the present one differs in several ways:

Evaluation Granularity. One major difference between this survey and others concerns the breadth and depth of the evaluation. In particular, the survey investigating most approaches, i.e., the survey of Chitchyan et al., aims at providing a broad overview by including all existing aspect-oriented design approaches as well as not so well-elaborated proposals for such design approaches. In contrast, this survey tries to provide an in-depth investigation of selected approaches that have already gained a certain level of maturity in terms of publications at acknowledged conferences and/or journals as well as are based on the Unified Modeling Language (UML) [44] as the prevailing standard in object-oriented modeling. For an in-depth evaluation a catalogue of criteria is provided which encompasses more than 40 criteria. In contrast, the other sets of criteria [4], [5], [15], and [52], do not include more than 8 criteria. In literature, 14 mature, UML-based AOM approaches have been identified, including the approaches already investigated in related surveys. In this survey, a representative set of 8 UML-based aspect-oriented design approaches is evaluated, which has been carefully selected from the above mentioned 14 approaches with respect to maintaining the ratio between the extension mechanisms used (metamodel vs. profile) as well as the ratio between symmetric and asymmetric approaches (cf. Section 5).

Methodology. Another important difference of this survey to the aforementioned ones lies in the applied methodology, which bases on a carefully established catalogue of criteria. Great emphasis has been put on the selection of criteria and their definition in a top-down as well as a bottom-up manner. A so-called *Conceptual Reference Model* for AOM (cf. Section 3) has been proposed, which identifies the basic AOM concepts as well as their interrelationships and thus, forms the basis for deriving the set of criteria in a top-down manner. Furthermore, for all criteria used, a clear definition along with the specification of the measurement scale is given. At the same time, this survey aims at complementing the set of criteria in a bottom-up manner by those criteria used in related AOM surveys [4], [5], [15], [52]. More specifically, criteria found in other surveys have been adopted where appropriate or they have been refined where necessary, e.g., with respect to their definition or in terms of a decomposition into sub-criteria. In the catalogue of criteria (cf. Section 4), it is indicated which criteria have been adopted

and which have been refined. Nevertheless, 6 criteria proposed in related surveys have been explicitly excluded from the evaluation framework due to methodological issues. Specifically, these criteria encompass reusability, comprehensibility, flexibility, ease of learning/use, parallel development, as well as change propagation [4], which corresponds to the evolvability criterion [5] and cannot reasonably be measured without empirical studies, e.g., user studies and extensive case studies. Thus, the catalogue of criteria subsumes the criteria derived from the CRM and the criteria provided by other surveys.

Inclusion of Recent Approaches. Furthermore, this survey also considers recently published approaches, namely [13], [37], not included in the other surveys. In this way, this survey is complementary to the aforementioned surveys by considering also very recent developments.

Running Example. Finally, in contrast to all other surveys, the evaluation is supported by a running example that is realized with each of the surveyed approaches. This further supports the evaluation in that it first, illustrates each approach and second, allows to better compare the modeling means of the approaches and understand their strengths and shortcomings. If at all, other surveys rely on diverse examples sometimes taken directly from the respective approach's publications (cf. [15]).

2.2 Aspect-Oriented Programming Surveys

Less closely related, since focusing on AOP, is the survey of Hanenberg [24] which presents a set of criteria used to evaluate four AOP languages. Kersten [32] also provides a comparison of four leading AOP languages having only AspectJ in common with Hanenberg. In addition, Kersten also investigates the development environments of these AOP languages.

Although focused on AOP, the evaluation criteria defined in those surveys are also of interest, since some of the surveyed AOM approaches are aligned to a certain aspect-oriented programming language. Nevertheless, some of them are not applicable in the context of this survey, since they are specifically related programming level issues, only. For example, Hanenberg distinguishes between "code instrumentation" and "interpretation weaving techniques" in the context of AOP weavers. In this survey, some of their criteria have been adopted and refined such that they can be applied at the modeling level, too. For example, the idea of evaluating tool support for AOM approaches has been inspired by Kersten's criteria on IDE support (e.g., editor, debugger).

3 The Conceptual Reference Model for Aspect-Oriented Modeling

The major difficulty in comparing AOM approaches is the lack of a common understanding for the basic ingredients of aspect-oriented modeling. This is on the one hand due to different concepts introduced by related AOSD approaches

(e.g., AP, CF, SOP, and MDSoC) and on the other hand due to the very specific meaning of AOP level concepts, particularly those coined by AspectJ [64]. An example for the first issue is the concept of "aspect", where similar though different concepts have been introduced by related AOSD approaches, e.g., "hyperslice" in Hyper/J, "filter" in CF, and "adaptive method" in Demeter/DJ [71]. An example for the second issue are AspectJ's join points which are defined as "points in the execution of the program" including field accesses, method, and constructor calls [68]. This definition is not comprehensive enough for the modeling level, however. First, modeling languages unify specific programming language concepts into more abstract modeling elements to be able to serve several different programming languages. And second, modeling languages typically are richer in terms of concepts, i.e., modeling elements, that could serve as join points. This is also due to different views available at modeling level, e.g., structural views and behavioral views.

In the light of different terminologies and a broad variety of aspect-oriented concepts, for an evaluation of AOM approaches it is essential to first establish such a common understanding by means of a so-called *Conceptual Reference Model* for AOM. The CRM enables to explain the basic ingredients of aspect-oriented modeling and their interrelationships both in terms of a graphical representation as a UML class diagram and in terms of a glossary comprising a textual definition for each concept introduced in the class diagram. The conceptual reference model, for which a previous version has already been proposed in [55], represents an intermediate step and forms the basis for setting up an evaluation framework, i.e., inferring concrete criteria as it is done in Section 4.

In AOSD literature, one can already find some proposals for reconciliating the currently prevailing diversity in the understanding of concepts from the aspect-orientation paradigm each pursuing a specific focus [38], [69], [71]. In this paper, they have been used as a basis for establishing the CRM for AOM. Their particular influence on the CRM is discussed as follows:

- Considering the broader research area of ASoC, one can distinguish between four composition mechanisms, namely, *pointcut-advice*, *open class*, *compositor*, and *traversal* [41]. In the CRX model of Kojarski et al. [38] pointcut-advice, open class, and compositor mechanisms are supported, only, because the traversal mechanism does not necessarily share properties with the other mechanisms that can be reasonably generalized [38]. For the CRM, in this paper, the authors' idea of first abstracting over the three aspect-oriented composition mechanisms, which later allows to compare AOM languages at a higher level, is adopted. Beyond, the CRM shall capture in detail the corresponding AOM language concepts for each composition mechanism separately. In Section 4, this allows to set up a fine-grained set of criteria for each composition mechanism and consequently allows AOM languages realizing the same composition mechanism(s) to be compared in greater detail.
- In van den Berg et al. [69] an attempt towards establishing a common set of concepts for AOSD has been made. The proposed definitions are intended to be appropriate for all phases in the software development life cycle. The

AOSD Ontology of van den Berg et al. discusses high level concepts such as "concern" and "composition", which allow abstracting over different composition mechanisms such as proposed by Kojarski et al. When looking at the concepts describing the specifics of the different composition mechanisms, however, one can see that the focus is rather on the pointcut-advice and open class mechanisms. Concepts for supporting the compositor mechanism such as "merge" and "match method" (cf. Section 3.3) are not discussed in the proposed glossary. Beyond, a visualization of the glossary and the concepts' interrelationships in terms of a conceptual model is missing. The CRM is based on the AOSD Ontology in that the proposed definitions of concepts are adopted, i.e., if such definitions are available.

– The "theory of aspects" of Chavez et al. [71] describes a "conceptual framework for AOP" in terms of Entity-Relationship diagrams and a textual description of each entity. The framework, however, explicitly supports aspect-oriented approaches that follow the pointcut-advice and open class mechanisms, only. The framework, e.g., explicitly demands the aspect-base dichotomy, meaning the clear distinction between "aspects" and "base". Consequently the "theory of aspects" does not describe concepts supporting the compositor mechanism. Nevertheless, the Entity-Relationship diagrams have served as an input for designing the CRM. The definitions of concepts proposed in the AOSD Ontology [69] have been preferred over those of Chavez et al. [71], however, since the AOSD Ontology's terminology is closer to the original terminology of the pointcut-advice mechanism (e.g., "enhancement" in [71] corresponds to "advice" in [69]).

For those concepts where no definition is available in the discussed literature, a bottom-up approach is followed, taking into consideration the surveyed approaches.

In the following, the concepts of the CRM are described along with its four major building blocks as depicted in Figure 1. The *ConcernComposition* package provides a high level view on the concepts of AOM abstracting over different composition mechanism, while the *Language* package describes the means underlying the specification of concerns. The specific composition mechanisms are specialized in separate packages, i.e., the pointcut-advice and open class mechanisms are specialized into the *AsymmetricConcernComposition* package, and the compositor mechanism is specialized in the *SymmetricConcernComposition* package. The concepts' descriptions possibly contain a reference to the source definition and an optional discussion in case the definition of a concept has been refined.

3.1 ConcernComposition

The *ConcernComposition* package abstracts over the different composition mechanisms. It deals first, with the modularization and thus with the separation of a system's concerns into appropriate units and second, with their interrelationships, and consequently their composition by means of appropriate rules.

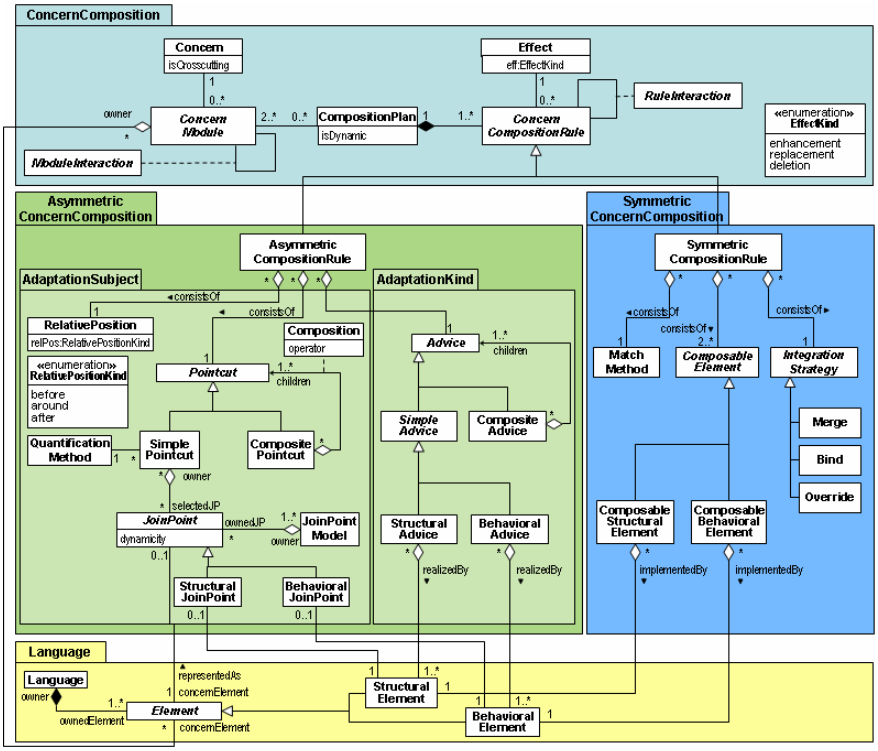

Fig. 1. The Conceptual Reference Model for Aspect-Oriented Modeling

Concern. Along with [69] a *concern* is defined as an interest which pertains to the system's development, its operation or any other matters that are critical or otherwise important to one or more stakeholders. A concern is called a *crosscutting concern* if it cannot be modularly represented within a language's decomposition technique, e.g., classes and methods in the object-oriented paradigm (cf. Figure 1 attribute *isCrosscutting*). In AOSD literature, this restriction is called the tyranny of the dominant decomposition [67]. The elements of a crosscutting concern are then said to be scattered over other concerns and tangled within other concerns of a specific system [69]. In AOSD, logging is often seen as the prime example for a crosscutting concern.

ConcernModule. One concern typically is realized by one or more *concern modules*. The term concern module is reused from Kojarski et al. [38] and encompasses a set of *concern elements* that together realize a concern or part of a concern (cf. role *concernElement* in Figure 1). Thus, it forms a representation of concerns in a formalized language (e.g., a package in UML or in Java). Some approaches have introduced the concept "aspect" [69] for modularizing otherwise crosscutting concerns, while existing units of modu-

larization formalizing non-crosscutting concerns have been called "base" [26]. This distinction has been used to categorize aspect-oriented approaches into asymmetric approaches to concern composition that support this aspect-base dichotomy and symmetric ones that do not [26]. Today, this distinction has begun to diminish and is being replaced by the more general understanding that the difference between concern modules is in how they are used during composition (cf. *ConcernCompositionRule*) [38]. In this paper, this view is adopted and consequently in the CRM only the *concern module* concept, which subsumes the notions of aspect and base, is considered.

CompositionPlan. The integration of concern modules is specified by a *composition plan* [38], which consists of a set of rules. The weaving plan concept of Kojarski et al. [38] has been renamed in favor of the more general term composition, which yields the integration of multiple modular artifacts into a coherent whole [69]. The "execution" of a composition plan results in a composed model of the overall system. During this process one distinguishes two phases, namely detection and composition. While detection is necessary to identify the concern elements that have to be integrated in the composed model, composition means the actual integration of them. For the purposes of this survey, furthermore a distinction between two ways of composing concern modules is made, namely static and dynamic (cf. attribute *isDynamic*). Thereby static indicates that the composed model is produced and thus is available to the modeler at design time analogously to compile-time weaving at programming level. Dynamic composition integrates the concern modules virtually during run-time, i.e., while executing the models. At the modeling level this requires the run-time semantics of the language's metamodel to be specified [19] (which, considering, e.g., UML, is only the case for parts of the language like state machines). This is similar to a run-time weaving that happens at programming level.

ConcernCompositionRule. The composition plan consists of a set of *concern composition rules* whereby one rule defines in detail how the various concern elements are to be composed. The general concept of concern composition rule is specialized into sub-classes according to the composition mechanism used. Following Kojarski et al. [38], the CRM foresees three composition mechanisms. Two asymmetric composition mechanisms exist in the form of pointcut-advice for introducing aspectual behavioral (e.g., intercepting method calls) and open class for introducing aspectual structure (e.g., introducing additional attributes to a class) [38]. At the modeling level, in any case augmentations or constraints need to be introduced with respect to model elements, whether they are behavioral elements or structural elements. Consequently, the *asymmetric composition rule* serves to realize both composition mechanisms. The compositor mechanism is provided by the subclass *SymmetricCompositionRule*.

Module Interaction. Concern modules might be defined in a way such that they interact with each other. Kienzle et al. [35] present a classification of interaction into "orthogonal" concern modules, "uni-directional preserving" concern modules based on other concern modules without modifying

them, and "uni-directional modifying" concern modules that change other concern modules. Another classification of Sanen et al. [54] distinguishes between "mutual exclusive" concern modules, concern modules "depending" on each other, concern modules positively "reinforcing" each other, and concern modules "conflicting" with each other. Accordingly, the abstract class *ModuleInteraction* can be specialized to represent the specific interaction types. In case of a conflict, additional resolution strategies may need to be employed.

RuleInteraction. Analogously to module interaction, also concern composition rules may interact with each other. Again a *rule interaction* can be refined accordingly to support different kinds of rule interactions. For example, concern composition rules on the one hand may reinforce each other but on the other hand may also conflict with each other. Consequently, conflict resolution strategies need to be employed. In the context of UML, e.g., a relative or absolute ordering of rules could by realized with dependencies.

Effect. The *effect* specified with the concern composition rule describes what effect the integration of concern elements have. A concern composition rule may have an *enhancement* effect, a *replacement* effect, or a *deletion* effect (cf. *EffectKind* in Figure 1). This distinction resembles a differentiation proposed by Hanenberg [24] in terms of constructive (cf. enhancement), and destructive (cf. replacement and deletion) effects. There exists, however, an inherent relationship between the effect and the respective concern elements used in a particular rule. For example in case of a pointcut-advice rule, the relative positions *before*, and *after* may lead to an enhancement, whereas in case of *around* the effect may resemble an enhancement, a replacement (i.e., deleting the join point with the advice), or a deletion (i.e., deleting the join point with an empty advice).

3.2 AsymmetricConcernComposition

In the *asymmetric concern composition* the concern composition rule is specialized for covering the pointcut-advice and open class composition mechanisms [38]. The package is organized into two sub-packages, namely *AspectualSubject* and *AspectualKind*, due to the two distinct roles concern elements play in asymmetric composition.

AsymmetricCompositionRule. *Asymmetric composition rules* are part of a particular composition plan and provide support for the pointcut-advice and the open class composition mechanisms. An asymmetric composition rule consists of a pointcut (cf. *Pointcut*) together with an optional relative position (cf. *RelativePosition*) describing where to augment or constrain other concern modules as well as the advice (cf. *Advice*) describing how to augment or constrain other concern modules. The *consists-of* relationships have been modeled using weak aggregations, since advice, pointcut, and relative position might be reused in other asymmetric composition rules as well.

AspectualSubject The *aspectual subject* describes the concepts required for identifying where to augment or constrain other concern modules.

JoinPoint. According to [18] a join point is a well-defined place in the structure or execution flow of a program where additional behavior can be attached. In contrast, at modeling level the *join point* represents a well-defined place in a model represented by a concern module, which specifies where an advice (cf. *Advice*) can be introduced. Thus, a join point represents a concern element, i.e., an identifiable element of the language used to capture a concern. It has to be noted that in recent works [38], [41] the notion of join point has changed. It has been described as being a concept of the result domain, meaning it represents the composed element through which two or more concerns may be integrated. Nevertheless, the original concept of join point is essential to the pointcut-advice composition mechanism and may also be used in the open class composition mechanism [38]. Consequently, the CRM adheres to the original notion of join point for describing the concepts participating in asymmetric concern composition.
According to Hanenberg [24], join points can be distinguished along two orthogonal dimensions, namely abstraction and dynamicity. In this paper, this categorization is applied to the modeling level while adhering to UML terminology [44] through the use of the term "feature" instead of "abstraction". Consequently, join points can be structural (cf. *StructuralJoinPoint*) or behavioral (cf. *BehavioralJoinPoint*), while at the same time, they are also modeling level representations of static or dynamic elements (cf. attribute *isDynamic*) in a software system. While *static* join points are elements of a language that can be identified based on information available at design time (e.g., class and method call), *dynamic* join points are elements of a language that cannot be identified before run-time (e.g., object and method execution).

StructuralJoinPoint. *Structural join points* represent structural elements of a language where an advice can be introduced. In addition, structural join points can be either *static* or *dynamic* (cf. *isDynamic* attribute). Exemplifying those two categories by means of UML modeling elements, *structural-static join points* would be classes and *structural-dynamic join points* would be objects.

BehavioralJoinPoint. Analogous, *behavioral join points* represent behavioral elements of a language where an advice can be introduced. Additionally, a distinction is made between *behavioral-static join points* (e.g., activity) and *behavioral-dynamic join points* (e.g., method execution). Admittedly, not all languages may offer elements which allow for dynamic join points as is the case with UML together with OCL.

JoinPointModel. The *join point model* defines the kinds of join points available [69]. It comprises all elements of a certain language where it is allowed to introduce an advice (cf. *Advice*), i.e., where the *representedAs* association connects the element with *JoinPoint*. For example, some approaches might

want to restrict their join point model to a specific set of language elements, e.g., classifiers in UML.

Pointcut. A *pointcut* describes a set of join points [69], i.e., the concern elements selected for the purpose of introducing certain augmentations or constraints (cf. *Advice*). The selection of join points can be done by means of quantified statements over concern modules and their concern elements (cf. *SimplePointcut* and *QuantificationMethod*). A pointcut specification is implemented by either a *SimplePointcut* or a *CompositePointcut*.

SimplePointcut. A *simple pointcut* represents a set of join points of a certain kind (e.g., structural-static), which are selected according to a certain quantification method (cf. *QuantificationMethod*). It thus, represents a means for selecting several concern elements as join points. For this survey, the combination of simple pointcut and quantification method correspond to the definition of pointcut in [69].

CompositePointcut. For reuse purposes, pointcuts can be composed of other pointcuts by means of logical *Operators*, e.g., AND, OR, NOT, to form *composite pointcuts*. Thereby, all children of a composite pointcut, i.e., all selected join points, refer to the same join point model.

QuantificationMethod. The *quantification method* concept describes a mechanism, e.g., a predicate for selecting from the potential join points of the join point model those that should be available for introducing an advice (cf. *Advice*). The quantification method corresponds to what is termed a pointcut designator in AspectJ, i.e., its quantification mechanism according to [18].

RelativePosition. A *relative position* may provide further information as to where aspectual features (cf. *Advice*) are to be introduced. It represents some kind of location specification. This additional information is necessary in some cases when selecting join points by pointcuts only is not enough. Such aspectual features can be introduced for example *before* or *after* a certain join point. Still, in some other cases such as for the open class composition mechanism a relative positioning is not necessary, e.g., when a new attribute is introduced into a class the order of the attributes is insignificant (cf. multiplicity 0..1). While the relative position typically is specified with the advice such as in AspectJ, in the CRM it is modeled separately from the advice. The "wrapping" technique presented in [18] corresponds to the definition of relative position but in contrast is described for behavioral join points only.

AspectualKind. The *AspectualKind* package comprises the concepts necessary to describe how to augment or constrain other concern modules.

Advice. An *advice* specifies how to augment or constrain other concerns at join points matched by a pointcut [69]. An advice is realized by either a structural advice (cf. *StructuralAdvice*), a behavioral advice (cf.*BehavioralAdvice*), or both, i.e., by a composite advice (cf. *CompositeAdvice*). Historically, structural advice has been called "introduction", while behavioral advice has been

termed "advice". Recently, the advice concept is more and more used as an inclusive term for both and consequently has been employed herein.

StructuralAdvice. A *structural advice* comprises a language's structural elements for advising other concerns. For example, adding a new attribute to a class's structure represents a structural advice.

BehavioralAdvice. Likewise, a *behavioral advice* comprises a language's behavioral elements for advising other concerns. In the context of UML, adding a method call, i.e., a message in a sequence diagram represents a behavioral advice.

CompositeAdvice. For reuse purposes, an advice can be composed of a coherent set of both, structural and/or behavioral advice, to form a *composite advice*, i.e., the composite needs to be free of conflicts. For example, an attribute and an operation represent two simple advice. If composed, the composite advice includes the attribute as well as the operation. In this respect, the advice concept extends the general understanding of the advice concept described in [69].

3.3 SymmetricConcernComposition

In the *symmetric concern composition* the concern composition rule is specialized according the compositor composition mechanism [38].

SymmetricCompositionRule. A *symmetric composition rule* comprises first, a specification of the elements to be composed (cf. *ComposableElement*), second, the match method to apply upon them describing which elements to compose (cf. *MatchMethod*), and third, the integration strategy to be applied describing how to proceed on those matched elements (cf. *IntegrationStrategy*). For example in the context of UML such a symmetric composition rule could specify that classes of two packages having identical names shall be matched and their class bodies shall be combined, similarly to the UML "package-merge" operator. Again, for reuse purposes the *consists-of* relationships have been modeled using weak aggregations.

ComposableElement. *Composable elements* of a symmetric composition rule refer to the elements allowed to be composed [7]. Composable elements can be made up by any element of the underlying language. Therefore, a distinction is made also between composable structural elements (cf. *ComposableStructuralElement*) and composable behavioral elements (cf. *ComposableBehavioralElement*). In the course of a symmetric composition rule more than two of such elements can be integrated.

ComposableStructuralElement. A *composable structural element* comprises a language's structural elements (cf. *StructualElement*) and can be composed with other composable elements identified in a symmetric composition rule. Examples for composable structural elements with respect to UML are Components, Classes, but also more fine-grained concepts such as Properties.

ComposableBehavioralElement. Likewise, a *composable behavioral element* comprises a language's behavioral elements (cf. *BehavioralElement*) and can

be composed with other composable elements identified in a symmetric composition rule. With respect to UML, examples for composable behavioral elements are Activities and Actions as well as State machines and States.

MatchMethod. The *match method* applied in the detection phase of a composition identifies which concrete elements to match given as input the composable elements for the composition. It supports the specification of match criteria for composable elements and their components, e.g., a class's attributes. Examples for match methods found in literature [7], [50] comprise match-by-name, match-by-signature, no-match.

IntegrationStrategy. The *integration strategy* details how to proceed during composition with the matched elements. The general concept of integration strategy is specialized into the sub-classes *merge*, *bind*, and *override* [7], [50].

Merge. With the *merge* integration strategy two or more corresponding composable elements are merged. This set of corresponding composable elements has been identified by the applied match method.

Override. In contrast to the merge integration strategy, for applying the *override* integration strategy the overriding as well as the overridden elements have to be specified from the set of corresponding composable elements identified by the applied match method.

Bind. The *bind* integration strategy typically represents a strategy where some composable elements are treated as template parameters that need to be bound to concrete values, i.e., other composable elements. It is applied in the context of parameterizable concern modules which are often used to realize crosscutting concerns.

3.4 Language

Finally, the concepts which are part of the *Language* package describe the means underlying the specification of concerns.

Language. Concern modules are formalized using the language *elements* of a certain *language*, i.e., a modeling language like UML. Depending on the composition mechanism used, some aspect-oriented approaches have distinguished between different languages for formalizing crosscutting and non-crosscutting concerns [38].

Element. A language comprises a set of *elements*, like e.g., class, relation, package which allow the modeler to express certain concepts. Typically a language's *elements* can be distinguished into structural (cf. *StructuralElement*) and behavioral elements (cf. *BehavioralElement*). Depending on the composition mechanism, the elements of a language are used differently. With respect to asymmetric approaches, elements serve two distinct purposes. First, they may represent join points and thus in the role of join points specify where to introduce an advice. Second, elements of a language are used for formulating the advice itself. In the case of symmetric approaches such a distinction is not made.

StructuralElement. *Structural elements* of a language are used to specify a system's structure. Natural examples for such elements in the case of UML are classes, packages, and components.

BehavioralElement. Likewise to structural elements, *behavioral elements* of a language are used to specify a system's behavior. Behavior is expressed in UML through behavioral elements like actions, states, and messages.

4 Evaluation Set-Up

4.1 Methodology

Selection of Approaches. There already exists a considerable amount of proposals for AOM languages each of them having different origins and pursuing different goals dealing with the unique characteristics of aspect-orientation. Only few of them have come of age and have been presented at acknowledged conferences and journals, however. Since aspect-orientation is often considered an extension to object-orientation, it seems almost natural to use and/or extend the standard for object-oriented modeling, i.e., the Unified Modeling Language (UML), for AOM. To the best of our knowledge, there are only a few AOM proposals that do not base their concepts on UML [65], [66] compared to the amount of approaches that do. Thus, this survey focuses on UML-based approaches to aspect-oriented modeling, only.

In literature, fourteen such well-published, UML-based, design-level AOM approaches have been identified, namely: [8], [12], [13], [17], [21], [23], [27], [30], [31], [37], [47], [50], [60], and [70]. In this survey, the results of evaluating a representative set of eight AOM approaches are presented, including in the set first of all those two approaches that have not been investigated in existing surveys, namely: [13] and [37]. As indicated before, the rationale behind choosing the remaining six [8], [17], [30], [47], [50], [60] out of the identified, is to assort a representative mix of approaches. In this respect, the goal has been to maintain the ratio between approaches based on metamodel extensions and those relying on UML Profiles as well as the ratio between symmetric and asymmetric approaches.

Deriving Criteria from the Conceptual Reference Model. In the following, a catalogue of criteria for a structured evaluation of AOM approaches is proposed. The focus in designing this catalogue of criteria was to provide a fine-grained catalogue of criteria which constitutes the prerequisite for an in-depth evaluation of existing approaches and thus allows to compare different AOM approaches in greater detail than in previous surveys [4], [5], [15], [52]. The criteria of the evaluation framework have been derived in a top-down manner from the CRM (cf. Section 3) as well as in a bottom-up manner considering related AOM surveys:

The CRM presented in the previous section sketches the concepts that have been identified to be important for the AOM domain. This has been done both at an abstract level, i.e., abstracting from different composition mechanisms, and at a detailed level, i.e., looking at the specific characteristics of each composition

mechanism. Corresponding criteria in the catalogue operationalize the CRM with respect to allowing a comparison of approaches. In particular, for each concept of the CRM one or more criteria have been derived. This implies that either a concept of the CRM maps onto one-to-many criteria in the catalogue or one-to-many concepts of the CRM map onto one criterion in the catalogue. A concept that is represented as an abstract class, however, does not necessarily need a corresponding criterion in the catalogue, since it is implicitly evaluated by its sub-concepts and their criteria.

Collecting Criteria from other Surveys. Following a bottom-up approach, the goal was to complement the set of criteria by those used in related AOM surveys [4], [5], [15], [52]. More specifically, criteria definitions found in other surveys have been adopted or refined. In this respect, a refinement for instance has been the provision of a measurement scale, e.g., the UML version used for the *Language* criterion (cf. Section 4.2), or the decomposition of a criterion into several sub-criteria, e.g., the composability criterion of [5] has been refined for each composition mechanism in this survey.

Excluding Non-Measurable Criteria. From the catalogue of criteria a few criteria proposed in related surveys have been explicitly excluded, since they cannot be measured without user studies or extensive case studies. These include the following criteria of the survey of Blair et al. [4], i.e., reusability, comprehensibility, flexibility, ease of learning/use, parallel development, as well as change propagation, which corresponds to the evolvability criterion of Chitchyan et al. [5].

Establishing a Schema for Criteria Definition. Furthermore, the goal was to avoid blurred criteria by working out, as far as possible, unambiguous definitions and the criteria's values that are also measurable. Thus, each criterion is described by a set of properties:

1. a *name* along with an *abbreviation* allowing to reference the criteria during evaluation of the approaches in Section 5,
2. a *reference to the source* in case a criterion has been adopted or refined from another survey as well as an explanation of how such a refinement has been accomplished,
3. a *definition* specifying the criterion as unambiguously as possible along with an optional *discussion* on difficulties in defining the criterion,
4. an appropriate *means of measurement*, such as a list of possible values or a measurement scale, including *not applicable* as a default value for each criterion.

Categorizing the Selected Criteria. The criteria of the catalogue have been grouped into six categories (see Figure 2) with four out of them being specifically inferred from corresponding parts in the conceptual reference model (cf. Section 3) and the general categories *Maturity* and *Tool Support* providing mainly descriptive criteria.

The *Language* category provides criteria for evaluating some basic characteristics of AOM languages (e.g., the modeling language, the extension mechanism used, and traceability). Beyond, it also provides a criterion for checking the

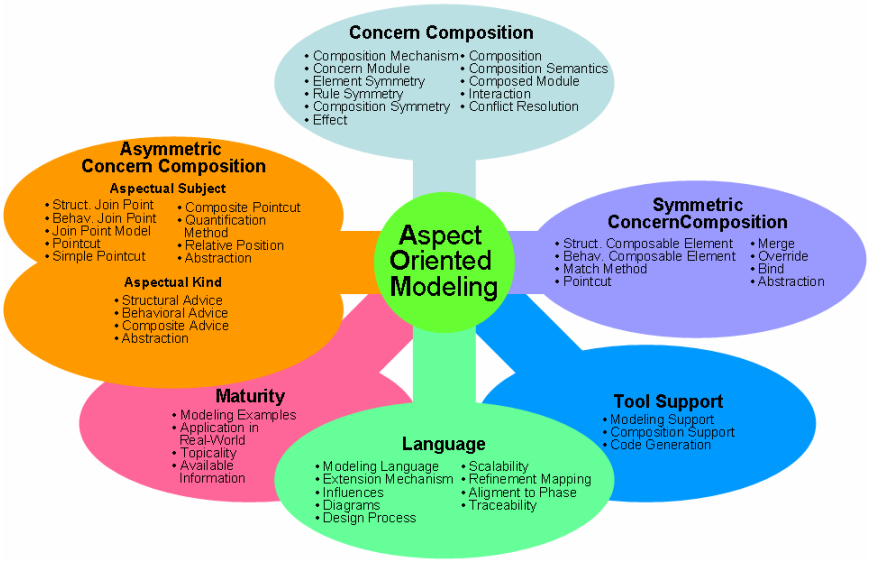

Fig. 2. Categorization of Criteria

availability of a design process. In the *ConcernComposition* category, the representation of the *concern module* concept and the composition mechanisms used is considered amongst others. With respect to symmetric concern composition in the *SymmetricConcernComposition* category, the kind of *composable elements* and provided *integration strategies* are investigated. In contrast, the *AsymmetricConcernComposition* category subsumes criteria for the *join point* and its sub-concepts (cf. *AspectualSubject* sub-category) as well as criteria evaluating the modeling support of *advice* (cf. *AspectualKind* sub-category). The *Maturity* of an approach is discussed along the criteria of provided modeling examples, real-world applications, and available information. And finally, in the *Tool Support* category the availability of tools for modeling and composing concern modules as well es for code generation is evaluated. Since a thorough evaluation of *Tool Support* for AOM would go beyond the scope of this survey, tool support is evaluated on the basis of the available literature, only. Following, each categories' criteria are presented.

4.2 Language Criteria

This category contains general criteria describing the modeling language and design process. A separate criteria for evaluating the *element* concept described in the CRM (cf. Section 3) is not considered, since it is implicitly evaluated with several other criteria that investigate the corresponding CRM's AO concepts with respect to their modeling representation.

Aspect Generality (M.AG) Besides being a general-purpose modeling language with respect to the application domain, an AOM approach also may be general-purpose with respect to aspects. The following two forms of *aspect generality* can be distinguished: A *general-purpose* AOM language supports modeling of all kinds of aspects, whereas an *aspect-specific* modeling language considers one specific aspect, only. Theoretically, there could be modeling languages that support two, three or more specific aspects. Still, these are not considered to be aspect-specific, since in that case, the definition for general-purpose modeling languages gets blurred. The aspect generality criterion has been adopted from the "purpose" criterion in *Reina et al.* [52]. In this survey, the focus is on general-purpose AOM languages, thus, also the aspect generality criterion is used for selection purposes, only.

Modeling Language (M.L) With respect to the modeling language used, UML-based AOM approaches are considered, only. Therefore, a distinction between the underlying UML version, i.e., version *1.x*[2], and version *2.0* [44] is made.

Extension Mechanism (M.E) Although UML is very expressive, its modeling mechanisms do not provide for aspect-oriented concepts. Thus, AOM proposals tend to use one out of two UML extension mechanisms to cater for the necessary modeling mechanisms. First, by what is called *heavy-weight extension*, the UML metamodel itself is extended through inheritance and redefinition of metamodel elements. Second, UML profiles, grouping user-defined extensions to metamodel elements in terms of stereotypes [53], represent UML's built-in *light-weight extension mechanism*, which permits only extensions that do not change the metamodel. This way a new dialect of UML can be defined in order to better support specific platforms or domains [44]. The light-weight extension mechanism fosters tool inter-operability [53], since they are designed in a way that tools can store and manipulate the extensions without understanding their full semantics. This criterion has been inspired by *Chitchyan et al.* [5], where this kind of information has been provided but an explicit criterion has not been defined therein.

Influences (M.I) Originally, the intention was to use "platform dependency" as a criterion for this catalogue. Still, in literature, no clear definitions of platform or platform (in)dependence, e.g., in the context of OMG's Model Driven Architecture (MDA) [43], have been available. For example, there may be many abstraction levels between MDA's Platform Independent Models (PIM) and Platform Specific Models (PSM). Consequently, what defines platform and platform-independence is a matter of objectives and has to be determined in the context of one's own work. In this survey, a common definition of platform for the evaluated approaches is not attempted. Instead, the "inspired by" criterion of *Reina et al.* [52] is resumed, according to which many of the AOM approaches have been inspired by concepts expressed in a specific aspect-oriented programming language. In contrast to [52], this criterion is not restricted to AOP platforms but *lists research areas* (e.g., SOP,

[2] http://www.omg.org/technology/documents/vault.htm#modeling

MDSoC, and CF) *and platforms* in general that have "influenced" a particular approach. In addition, platforms are also listed if models can be mapped onto them, provided that proof is given through a mapping definition or at least appropriate examples.

Diagrams (M.D) The emphasis in modeling concern modules can be its structure and/or its behavior. In this respect, the kinds of supported structural and/or behavioral diagrams to specify aspect-orientation are evaluated. Hence, this property *lists all UML diagram types and possibly proprietary diagram types* that have been used to support on the one hand *structural* and on the other hand *behavioral* modeling of concern modules. This criterion also has been inspired by *Chitchyan et al.* [5], where this kind of information has been provided but an explicit criterion has not been defined.

Design Process (M.DP) A design process describes a well-defined, step-wise approach to modeling. This criterion has been adopted from *Op de beeck et al.* [15] and evaluates if the surveyed AOM approach provides *explicit* support for a design process or if some *implicit* design process support is available, e.g., in terms of guidelines, only.

Traceability (M.T) The traceability criterion is defined as a property of a relationship between two models where one model is a refinement of another, and has been adopted from the work of *Chitchyan et al.* [5]. More specifically, the criterion distinguishes between external and internal traceability. The external traceability measure focuses at aspect-oriented design models in relation to the full software development life cycle, i.e., requirements (R), analysis (A), design (D), and implementation (I). Possible values are combinations such as $R \rightarrow D \rightarrow I$, which means traceability from a requirements specification over design to the implementation level. The internal traceability measure deals with traceability between models belonging to one phase in the software development life cycle. In this survey, AOM approaches are investigated if during design more abstract design models are refined into more detailed design models. This sub-criterion evaluates to *supported* or *not supported*, respectively.

Scalability (M.S) Scalability, which is defined as the ability to cope with small as well as large modeling projects, is investigated with respect to first, which *high-level modeling elements* of an approach support scalability, e.g., UML packages, and/or *high-level diagram types*, and second, if scalability has been *proven or not proven* in real-world projects or by modeling examples that go beyond the composition of two concern modules. This definition of scalability has been refined from *Chitchyan et al.* [5] with respect to its measurement scales.

Refinement Mapping (M.R) The refinement mapping criterion is adopted from *Op de beeck et al.* [15]. It describes how the refinement of an initial abstract design model into a more detailed one is achieved. One can distinguish the *extending* step-wise refinement from the *creating* step-wise refinement. The difference between these two possibilities is that for the latter a new instance of the model is created with every step in the refinement process.

Alignment to Phase (M.A) Design is just a phase embedded in the overall software development life cycle. An AOM approach therefore may be more aligned to certain phases in the software development than to others. Ideally, an approach is balanced between the abstraction available from the requirements phase and the abstraction needed for the implementation phase. An AOM approach can thus be aligned to *requirements* and/or the *implementation* phases but also to none of the phases. This criterion has been adopted from *Op de beeck et al.* [15].

4.3 ConcernComposition Criteria

This category considers criteria derived from the corresponding package in the CRM, amongst others, the representation of the *concern module* concept, the composition mechanisms used as well as the approaches symmetry.

Composition Mechanism (CC.M) The concepts described in the CRM support the pointcut-advice (*PA*), open class (*OC*), and compositor (*CMP*) composition mechanisms. This criterion therefore allows to evaluate which of the three composition mechanism is realized by the AOM approaches. It is also possible to support more than one composition mechanism.

Concern Module (CC.CM) This criterion investigates the concern modules's representation in the modeling language in terms of a UML *meta-class* or a *stereotype* definition and, if provided, the notational element used.

Element Symmetry (CC.ES) Two possible ways of concern decomposition can be distinguished, namely, *symmetric* and *asymmetric* concern decomposition. In the asymmetric paradigm one distinguishes between concern modules of different structure, i.e., between "aspects" and "base". As an example some AOM approaches, introduce a new stereotype ≪aspect≫ derived from UML meta-class *Class* to distinguish "aspects" from normal "base" classes. In the symmetric paradigm no such distinction is made. In fact, the symmetric paradigm treats all concerns, both crosscutting and non-crosscutting, as "first-class, co-equal building-blocks of identical structure" [26].

Rule Symmetry (CC.RS) The rules for composing concern modules can be specified in a *symmetric* or in an *asymmetric* way [26]. In particular, the symmetry is determined by the placement of the concern composition rules. Rule asymmetry defines the concern composition rules within one of the concern modules that are to be composed (e.g., in AspectJ the rules are captured within the aspect in terms of pointcut-advice combinations), whereas rule symmetry defines them in neither of the concern modules. Please note, that rule symmetry corresponds to relationship symmetry in [26].

Composition Symmetry (CC.CS) This criterion has been adopted from the work of *Op de beeck et al.* [15] and investigates which concern modules are allowed to be composed with each other. While in the *asymmetric* case composition happens between "aspects" and "bases" only, i.e., "aspects" are woven into "bases", in the *symmetric* case all concern modules can be composed with each other. For those approaches supporting element asymmetry

and thus distinguishing between "aspects" and "bases", symmetric composition is only supported if the following combinations are allowed: aspect-base, aspect-aspect, base-base. Approaches supporting element symmetry accordingly also support composition symmetry.

Effect (CC.E) This criterion evaluates if the approaches provide means for modeling the effect of the integration of concern elements via concern composition rules. The criterion's possible values are *supported* or *not supported*.

Composition Semantics (CC.S) The composition semantics criterion has partly been inspired by the survey of *Chitchyan et al.* [5], though not explicitly defined therein. This criterion evaluates if the composition semantics have been *defined* or *not defined* for both the detection of the elements to be composed as well as for their actual composition into a composed element.

Composition (CC.C) A distinction between composing concern modules *statically* or *dynamically*, i.e., by executing the models, is made. Nonetheless, a specific approach might neither support static nor dynamic composition at modeling level but defer weaving to later phases in the software development process, e.g., by separately generating code from concern modules, which are finally composed by a dedicated mechanism of the underlying AOP language. The advantages of approaches that support model composition are first, at code level non aspect-oriented platforms can be used and second, the composite results can be validated prior to implementation. However, once composed, the concern modules cannot be recovered at later stages thus causing traceability problems.

Composed Module (CC.CP) This criterion evaluates the resulting composed module in terms of its modeling representation. In particular, this criterion distinguishes between composed modules represented with *standard UML* and composed modules represented based on the *extensions made to the UML*. The composed module criterion has been adopted from the "composability" criterion of *Chitchyan et al.* [5].

Interaction (CC.I) An AOM approach may offer ways to specify *interactions* between *concern modules* on the one hand but also between *concern composition rules* on the other hand. This criterion evaluates for both concepts, what *kind of interactions* can be modeled and the *modeling representations* thereof, e.g., UML meta-class or stereotype.

Conflict Resolution (CC.CR) In accordance with [4], conflict resolution may be based on a mechanism to *avoid* conflicts in advance or to *detect* conflicts and then *resolve* them manually. While conflict avoidance might be a possible solution to cope with conflicting aspects, one still might need ways to detect and resolve conflicts that could not be captured by conflict avoidance in advance. In case no conflict resolution has been specified, this criterion evaluates to *not supported*.

4.4 AsymmetricConcernComposition Criteria

This category subsumes criteria for evaluating approaches following an asymmetric way to concern composition which are categorized into two sub-categories *AspectualSubject* and *AspectualKind*.

AspectualSubject Criteria. The *AspectualSubject* sub-category provides criteria for evaluating concepts used to describe where to augment or constrain other concern modules, e.g., the *join point* and its sub-concepts.

Structural Join Point (AS.SJP) This criterion evaluates if structural join points are *supported*. More specifically, the focus is on what kind - with respect to dynamicity - of structural join point are considered in the approaches, i.e., structural-static join points like classes or structural-dynamic join points like objects.

Behavioral Join Point (AS.BJP) Likewise, the behavioral join point criterion evaluates if behavioral join points are supported by the surveyed AOM approaches. In this respect, examples for a behavioral-static join point are UML activities and messages. Behavioral-dynamic join points typically depend on certain conditions evaluated at run-time. Specifying such conditions can be done for example with OCL.

Join Point Model (AS.JPM) This criterion distinguishes between two possible ways of specifying a join point model. First, the join point model can be made *explicit* by identifying a language's model elements as join points. This can be achieved for example by enhancing the language's metamodel in a way that certain model elements inherit from a join point meta-class or by at least identifying and declaring the join points of a language in "natural language" such as in [62] or [68]. Second, the join point model can be defined *implicitly* by the AOM language's join point selection mechanism, thus, comprising all join points that the join point selection mechanism is able to select.

Pointcut (AS.P) Although the pointcut concept is represented as an abstract class in the CRM (cf. Section 3), a separate criterion is foreseen for evaluating the commonalities of the concrete pointcut sub-classes. In particular, the criterion evaluates if the pointcut mechanism has been realized based on a *standard* (e.g., AspectJ code, UML, OCL) or on a *proprietary* language.

Simple Pointcut (AS.SP) This criterion evaluates how simple pointcuts are represented by concepts of the modeling language or extensions thereof and particularly distinguishes between *graphical* and *textual* representations of simple pointcuts.

Composite Pointcut (AS.CP) Furthermore, the composite pointcut criterion evaluates if at all and how composite pointcuts are represented in the modeling approach. Again, a distinction is made between *graphical* and *textual* representations of composite pointcuts.

Quantification Method (AS.SM) This criterion evaluates which quantification methods are employed to select join points in a certain approach. The selection of join points can be specified *declaratively*, *imperatively*, or simply by *enumeration*.

Relative Position(AS.RP) This criterion investigates the general support of specifying a relative position with respect to join points and, if provided, lists the different possibilities of relative position specification, i.e., *after*, *before*, and *around*, supported by the approaches.

Abstraction (AS.A) This criterion is refined from the definition given in *Chitchyan et al.* [5]. In contrast, in the context of asymmetric concern composition, two dimensions of abstraction are considered, namely *abstraction with respect to the aspectual subjects (AS.A)* and *abstraction concerning the aspectual kind (AK.A)* (cf. Section 4.4). With respect to the aspectual subjects, a *high* level of abstraction means that the join points might not have been identified yet, i.e., the model only specifies the fact that a certain concern module affects others, but not exactly where. On the contrary, modeling languages providing a *low* level of abstraction allow specifying the exact points where advice take effect.

AspectualKind Criteria. The *AspectualKind* sub-category subsumes criteria for evaluating concepts used to describe how to augment or constrain other concern modules, e.g., the *advice*, as well as the abstraction level at which modeling of the *advice* is possible.

Structural Advice (AK.SA) This criterion evaluates if AOM approaches provide ways of specifying structural augmentations and/or constraints. Furthermore, the *concepts or extensions* of the modeling language as well as the *notational elements* used for representation are investigated.

Behavioral Advice (AK.BA) Likewise to structural advice, this criterion evaluates if AOM approaches provide ways of specifying behavioral advice and in particular what *concepts or extensions* of the modeling language and what *notational elements* have been used for representation.

Composite Advice (AK.CA) In addition to evaluating structural and behavioral advice support, the focus is on how the approaches provide ways of composing multiple pieces of advice to form a more complex advice in terms of *concepts or extensions* of the modeling language and appropriate *notational elements*.

Abstraction (AK.A) This criterion has been refined from the definition given in [5] for the asymmetric concern composition. It is decomposed into the criteria *abstraction with respect to the aspectual subjects (AS.A)* (cf. Section 4.4) and *abstraction concerning the aspectual kind (AK.A)*. Analogously to (AS.A), it specializes the criterion given in [5] for the aspectual kind and, as already mentioned, contributes to the overall evaluation of the abstraction of an approach. Since models at a high level of abstraction might be incomplete with respect to providing a specification for code generation, a *high* level of abstraction with respect to the aspectual kind means that it might not yet be clear *how* the specific concern module(s) should be advised, i.e., the model only specifies that a certain concern module exists, but not the actual advice it provides. In contrast, *low*-level models of aspectual kind refer to models that provide detailed information on how the concern module's internals (i.e., the actual advice and auxiliary functionality) look like.

4.5 SymmetricConcernComposition Criteria

This category subsumes criteria for evaluating approaches following a symmetric way to concern composition, i.e., the necessary concepts identified in the CRM as well as the level of abstraction at which modeling is supported.

Structural Composable Element (S.SCE) This criterion evaluates if and what structural composable elements are supported by an AOM approach. It *lists the UML meta-classes representing structural concepts* that in a symmetric concern composition approach can be composed.

Behavioral Composable Element (S.BCE) Likewise, the behavioral composable element criterion evaluates if and what behavioral composable elements are supported by an AOM approach. This criterion therefore *lists the UML meta-classes representing behavioral concepts* that in a symmetric concern composition approach can be composed.

Match Method (S.MM) This criterion evaluates which method(s) to identify the matching elements out of the set of composable elements are foreseen by an approach. It distinguishes between three possible methods, namely *match-by-name*, *match-by-signature*, and *no-match*.

Merge (S.M) This criterion investigates if AOM approaches supporting the symmetric concern composition provide ways of defining the specific integration strategy *merge*. In particular, it investigates what *concepts or extensions* of the modeling language as well as what *notational elements* have been used for representation.

Override (S.O) Similarly, this criterion checks if an AOM approach allows for modeling symmetric concern composition rules with an *override* integration strategy. Again, it also investigates what *concepts or extensions* of the modeling language as well as what *notational elements* have been used for representation.

Bind (S.B) Like the previous two, the *bind* criterion evaluates possible *extensions* of the modeling language to support such a binding and provides information on the *notational elements* used.

Abstraction (S.A) Analogous to the abstraction criteria for the asymmetric concern composition, this criterion has been refined from the definition given in [5]. In this context, however, the level of abstraction is defined with respect to the composable elements used in a symmetric composition rule. A *high* level of abstraction is supported if the symmetric composition rule is used to compose two or more higher-level or composite modeling elements, such as UML packages, of which their internals have not been specified. A *low* level of abstraction is provided, if these composite modeling elements can be detailed, e.g., a class diagram for a UML package, and if symmetric composition rules can also be specified for more fine-grained modeling elements such as UML attributes.

4.6 Maturity Criteria

The criteria in this section intend to evaluate the approaches' maturity in general. It has to be noted that in [4] the criterion maturity was used to evaluate

whether an approach has been used in real world examples, only, whereas in this survey maturity is evaluated with a set of sub-criteria described in the following.

Modeling Examples (Ma.E) Besides evaluating the breadth of modeling examples, it is also interesting to investigate the modeling examples' depth in terms of how many different concern modules are integrated within the examples. Thus, the criterion is supported by two values, namely, the *number of provided modeling examples by each* approach as well as the *maximum number of concern modules* integrated in one example.

Application in Real-World Projects (Ma.A) The successful deployment of the AOM approach in the *design of a real-world application* proves its applicability and consequently indicates a high level of maturity of the modeling concepts. Possible values are *yes*, and *no*.

Topicality (Ma.T) The topicality criterion presents for each approach when the *most recent piece of work* in terms of the year of publication has been published to indicate the approach's topicality and thus, gives an indication whether the approach might still evolve or not. This criterion has been refined from the "year" criterion of *Reina et al.* [52]

Available Information (Ma.I) Another measure of the approaches' maturity is the available *amount of manuals, papers* and *books*. Although, admittedly, the amount of publications does not necessarily correlate with an approach's quality. The values for this criterion provide the *number* of different resources of information.

4.7 Tool Support Criteria

Tool Support improves the adoption of an approach an developer productivity as well as ensures syntactical correctness of the model. While the criterion distinguishes between support for modeling, composition and code generation, the latter are both dependent on modeling support.

Modeling Support (T.M) Modeling support is defined as providing the means to use the modeling language's notation and furthermore of validating the created aspect-oriented models for syntactical and semantical correctness. If the modeling language is realized in terms of a UML profile, modeling support should be portable to any UML modeling tool. This criterion evaluates to *supported*, possibly providing further information on modeling support, or *not supported*.

Composition Support (T.C) This criterion specifies if composition of concern modules is also *supported* or *not supported* by a tool an thus allows to view and/or simulate the composed model.

Code Generation (T.G) In line with the concepts of MDE, code generation facilities should be provided, thus requiring a mapping between the notation and the supported implementation language. This criterion evaluates if code generation, in principle, is possible. Beyond, this criterion also evaluates if there is a more sophisticated mechanism to code generation such as the

OMG's MDA [43] (i.e., existence of platform-independent models, platform definition models and their transformation into platform-specific models by using a mapping mechanism). Thus, possible values for this criterion are *supported* or *not supported*. Additional information is provided in case of a more sophisticated code generation mechanism.

4.8 Modeling Example: The Observer Pattern Applied to a Library Management System

Motivation. As an appropriate running example of a crosscutting concern to be applied to a system, in this evaluation, the well-known observer pattern [22] is adopted, a prominent example not only in AOSD literature (cf. [11], [48], [60]) but also in software engineering literature. In the running example, the observer pattern is applied as a crosscutting concern to a library management system, of which an overview along with the underlying model is given in the following. It has to be emphasized that on the basis of this rather simple example it is not (and cannot be) the intention to illustrate each and every concept of the approaches but rather to foster their overall understandability and comparability.

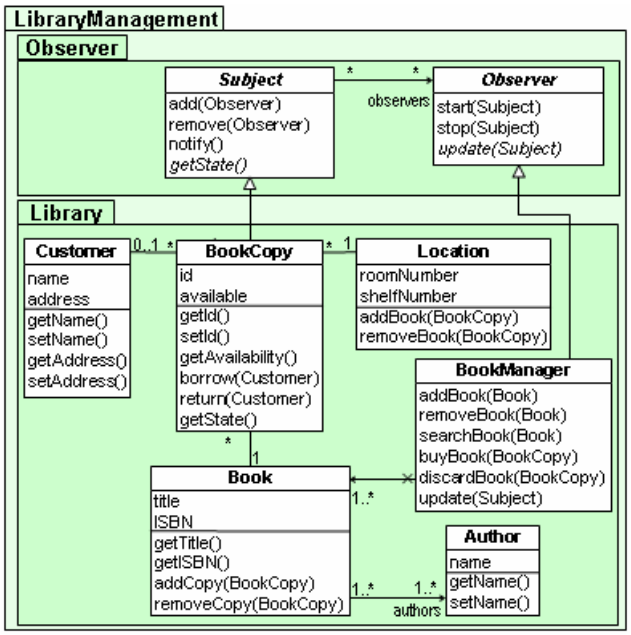

Fig. 3. The Library Management System with Observer Aspect.

An Example Library Management System. In Figure 3, the *Library* package models the structure for managing books of a library in a library man-

agement system based on [11]. Of course, it only depicts a small excerpt of such a system, primarily containing those parts of the system that are crosscut by the observer concern.

A *BookManager* manages a list of *Books* (cf. *addBook(Book)* and *removeBook(Book)*) allowing users to search (cf. *searchBook(Book)*) the list and access provided information for each book (e.g., *authors*). A library may offer several copies of each *Book*, i.e., the physical entities (cf. *BookCopy*), which need to be managed accordingly. *BookCopies* might get lost or be stolen. Still, a *Book* does not have to be removed from the *BookManager*'s list until new *BookCopies* are obtained. The *BookManager* associates *BookCopies* with their *Books* as they are bought (cf. *buyBook(BookCopy)* and *addCopy(BookCopy)*) and likewise, disassociates them as they are discarded (cf. *discardBook(BookCopy)* and *removeCopy(BookCopy)*). *Books*, in particular their copies, have a *Location* on a certain shelf in a certain room of the library. The status of each *BookCopy*, i.e., its availability, should be kept up-to-date. Thus, each time a *BookCopy* is borrowed or returned by a *Customer* (cf. *borrow(Customer)* and *return(Customer)*), the *BookManager* has to be notified. This notification functionality is not provided by the library management system, but is applied using the observer pattern as depicted in Figure 3.

The Observer Pattern. The observer pattern [22] as depicted in the *Observer* package in Figure 3 defines a one-to-many dependency between objects in a way that whenever a *Subject* (i.e., a *BookCopy*) changes its state, all its dependent *Observers* (i.e., instances of *BookManager*) are notified (cf. *notify()*) by using their provided update interface (cf. *update(Subject)*). While *Observers* can register and unregister with their *Subjects* of interest using the methods *start(Subject)* and *stop(Subject)*, a *Subject* keeps a list of *Observers* (cf. *add(Observer)* and *remove(Observer)*), which are interested in changes of the *Subject*'s state.

In Figure 3, thus, the *Subject* and *Observer* roles are adopted by *BookCopy* and *BookManager*, respectively. Applying the observer pattern, however, affects the library management system's modularity. In particular, the abstract methods *getState()* and *update(Subject)* have to be implemented by *BookCopy* and *BookManager*, respectively. Additional code modifications are necessary to call *start(Subject)*/*stop(Subject)* whenever a *BookCopy* is bought/discarded and to call *notify()* whenever a *BookCopy* is borrowed or returned. Therefore, the observer functionality can be regarded as crosscutting concern and, thus, be realized with the concepts of various AOM approaches.

Limitations of the Running Example. The observer pattern is a well-known example for a crosscutting concern and actually has been used in three of the surveyed approaches (cf. [7], [20], [60]). One might argue that the use of an example which has already been used by some of the analyzed approaches might lead to a bias in the evaluation. Since the running example is used to only visualize the respective approach to the reader and to have a side by side comparison of the approaches, any biased influence on the survey itself is negligible. Still, some approaches do not allow for fully operationalizing the running example,

which is due to their particular focus. For instance, the approach of Klein et al. [37] does not allow to model crosscutting structure, since the approach's focus is rather on a weaving algorithm for (non-)crosscutting behavior. Nevertheless, the application of one running example for all approaches generated some insight into the differences of each individual approach. Of course all AOM approaches should be tested in a real world setting or at least in a non-trivial example, which encompasses more than two concerns as well as all concepts described in the conceptual reference model, e.g., the *AO Challenge* [34]. Such an example would allow for testing the approaches' means for capturing interactions and resolving conflicts, which in this survey, can only be described textually. Still, the obvious advantages of a small and easy to understand running example would be lost.

5 Comparison of Approaches

This survey is based on a literature study, including modeling examples, provided by the individual AOM approaches. For each surveyed approach, additional information and discussion is provided in the following. The evaluation of each approach follows the order of categories of the criteria catalogue presented in Section 4. Moreover, a running example (cf. Section 4.8) that is modeled by means of the concepts of each AOM approach is provided. This further enhances the evaluation in that it first, provides an insight into each approach and second, allows to easier compare the modeling means of the approaches.

In the following, the modeling means of each surveyed AOM approach is presented by means of this running example. Basically, the approaches realizing the *pointcut-advice* and *open class* composition mechanisms are presented first and then those realizing the *compositor* composition mechanism are elaborate on. In particular, the first two approaches of Stein et al. (cf. Section 5.1), Pawlak et al. (cf. Section 5.2), are similar, since they have been specifically designed as modeling languages for two aspect-oriented programming platforms, i.e., AspectJ and the JAC Framework[3] respectively. The commonalities of the third approach of Jacobson et al. (cf. Section 5.3) and the approach of Pawlak et al. are that they do not envisage composition of concerns at modeling level but defer composition to the implementation phase. The next two approaches, are both very recent proposals to AOM focusing on composing behavioral diagrams. In this respect, the approach of Klein et al. (cf. Section 5.4) presents an algorithm for first, detecting the model elements to be composed and second, composing them. The approach of Cottenier et al. (cf. Section 5.5) also supports composition of models and, in contrast to all others, already comes with tool support for modeling, composition and code generation. The last group of three approaches supports the *compositor* composition mechanism. While the approach of Aldawud et al. (cf. Section 5.6) focuses on the composition of state machines, the approaches of Clarke et al. (cf. Section 5.7) and France et al. (cf. Section 5.8) originally have considered the composition of class diagrams. Lately, the approach of France et

[3] http://jac.objectweb.org/

al. also realizes the pointcut-advice composition mechanism through the composition of sequence diagrams. The results of the comparison are discussed and illustrated in Section 6 *Lessons Learned* at a glance (cf. Table 1 to 7).

5.1 The Aspect-Oriented Design Model of Stein et al.

Language. The *Aspect-Oriented Design Model (AODM)* of Stein et al. [60], [61], [62] has been developed as a design notation for AspectJ (L.I) and thus is aligned to implementation (L.A) as well as allows for external traceability from design to implementation (L.T). Internal traceability is not applicable (L.T), since a refinement of models models is not forseen in *AODM* (L.R). For this approach, both AspectJ and UML have been studied in order to find corresponding parts for AspectJ's concepts in UML and extend it to support AspectJ's concepts if necessary as well as identify where UML's concepts used in *AODM* are more expressive than actually necessary, e.g., the destruction of an instance is not part of AspectJ's join point model [60]. *AODM* is basically specified using the UML 1.x light-weight extension mechanism (L.L), (L.E), though extensions of the meta-model have also been necessary. For example, the UML extend relationship from which the ≪crosscut≫ stereotype has been derived originally can be specified between use cases, only [61]. Structural and behavioral modeling is achieved by employing class diagrams, UML 1.x collaborations, and sequence diagrams. In addition, sequence diagrams are used for visualizing join points, e.g., messages, while use case diagrams and collaborations demonstrate AspectJ's composition semantics (L.D). In the *AODM* thus, UML is used such that scalability in terms of high-level modeling elements is not supported and no other proof in terms of non-trivial modeling examples is available (L.S). The approach furthermore does not outline a design process or provide guidelines (L.DP).

ConcernComposition. *AODM* represents a notation designed for AspectJ and consequently supports the pointcut-advice and open class composition mechanisms (CC.M) as well as follows the asymmetric school of thought (CC.ES), (CC.CS), (CC.RS). A distinct concern module for crosscutting concerns has been introduced in *AODM* and is represented by a stereotype ≪aspect≫ (cf. *Subject-ObserverProtcolImpl* in Figure 4[4]), which is derived from the UML meta-class *Class* (CC.CM). In addition, several meta-attributes capture the peculiarities of AspectJ's aspects, e.g., the instantiation clause. The composition actually is deferred until the implementation phase (CC.C). Nevertheless the composition semantics of AspectJ have been redefined for the modeling level to a limited extent, e.g., in terms of UML use case diagrams and collaborations (CC.S), (CC.CP) [60]. The only way for modeling interactions (CC.I) is to manually specify the order for composing ≪aspects≫ in terms of a stereotyped dependency relationship between ≪aspects≫, i.e., ≪dominates≫ for conflict resolution (CC.CR) [61]. A means for explicitly specifying the effects of the concern composition rules or rather of the advice in models, however, is not addressed in *AODM* (CC.E).

[4] Please note that, in AspectJ the Observer functionality is realized using interfaces instead of abstract classes.

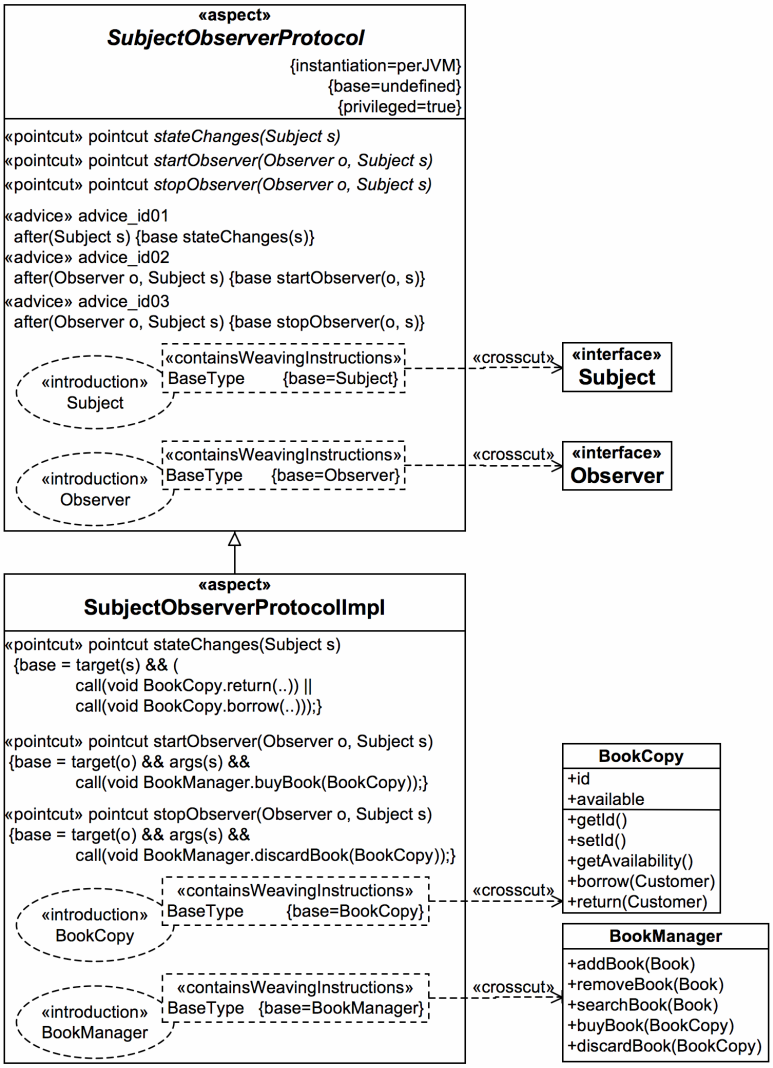

Fig. 4. The Observer Aspect Modeled Using the Aspect-oriented Design Model.

AsymmetricConcernComposition.

AspectualSubject. Though *AODM* has been specifically designed as a modeling language for AspectJ, Stein et al. [62] extend their notion of a join point model (AS.JPM): UML Classifiers are identified as structural-static hooks (AS.SJP). Besides, UML 1.x Links represent behavioral-static join points. Behavioral-static join points are depicted by highlighted messages in sequence diagrams (see [60]) (AS.BJP). For those join points where no such messages exist

(e.g., field reference, field assignment, initialization, execution) pseudo operations and special stereotypes have been provided. Using a ≪crosscut≫ dependency relationship, the subjects of structural advice are specified at a high level of abstraction (AS.A). The pointcuts in *AODM* are similar to AspectJ's pointcuts (AS.P). Selections of behavioral-static join points and behavioral-dynamic join points (AS.BJP) are represented by ≪pointcut≫ stereotyped UML Operations that are implemented by special ≪ContainsWeavingInstructions≫ stereotyped UML Methods. A meta-attribute "base" introduced for this ≪ContainsWeavingInstructions≫ stereotype then holds the pointcut in the form of AspectJ code (AS.P), (AS.QM). This allows first, the specification of composite pointcuts (AS.SP), (AS.CP), and second, the specification of the aspectual subjects at a low level of abstraction (AS.A). In addition, a second stereotype ≪ContainsWeavingInstructions≫ at this time derived from the UML meta-class TemplateParameter[5] is used to specify the pointcuts for structural advice (e.g., ≪introduction≫ *Subject* in Figure 4). The new meta-attribute "base" introduced for the ≪ContainsWeavingInstructions≫ stereotype specifies the pointcut in the form of AspectJ's type patterns. AspectJ's - and consequently *AODM*'s - means for specifying a pointcut is following a specific conceptual model. Recently, the authors have been working on a more expressive pointcut mechanism supporting different conceptual models [63], which is independent from the *AODM* approach, however. Concerning the declaration of a relative position, *AODM* supports the relative positions before, after, and around for behavioral-dynamic join points, only, and depicts them in an AspectJ-like manner as a keyword in the signature of behavioral advice (AS.RP).

AspectualKind. In a class diagram, behavioral advice are depicted in the operation compartment of a class consisting of the operation's signature as well as a base tag containing the pointcut's signature. Behavioral advice in *AODM* are represented by stereotyped UML Operations, i.e., ≪advice≫. These are implemented by special ≪ContainsWeavingInstructions≫ Methods, which contain the actual behavior in the method's "body" meta-attribute and reference a pointcut in the introduced "base" meta-attribute (AK.BA). Additionally, behavioral advice are specified in terms of sequence diagrams. Thus, behavioral advice are modeled at a high as well as a low level of abstraction (AK.A) likewise structural advice are modeled at a high and low level of abstraction: Structural advice are realized as parameterized collaboration templates with the stereotype ≪introduction≫. The parameters are of type ≪ContainsWeavingInstructions≫, which specify the subjects of advice in the form of AspectJ's type patterns (AK.SA). The details of the collaboration templates are shown in Figure 5. Composite advice, since not a concept available in AspectJ, are not addressed by *AODM* (AK.CA).

Maturity. *AODM* has been described in some publications (M.I). While the approach has not been tested in a real-world application (M.A), some modeling examples have been provided, e.g., timing and billing aspects for a system in the area of telecommunication [68] and the realization of the observer pattern (M.E).

[5] Stein et al. apparently have used the same name for two different stereotypes.

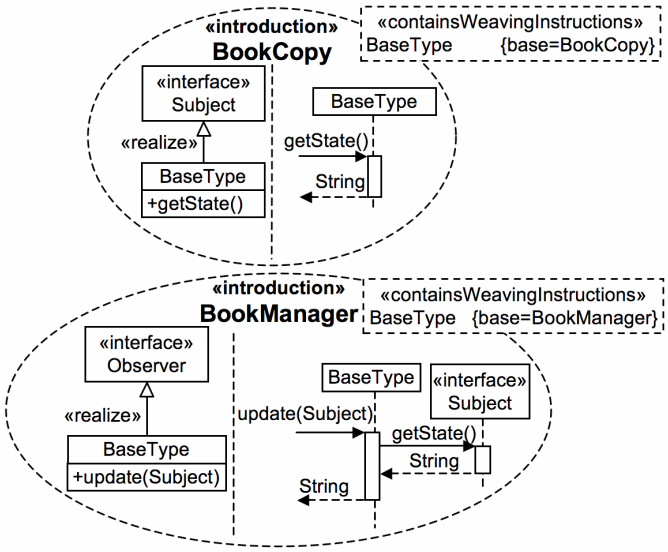

Fig. 5. Structural Advice in the Aspect-Oriented Design Model Notation

Today, the authors have moved on and specifically focus on research towards graphical ways to select join points in UML . For this they have introduced join point designation diagrams (JPDD) [63], which basically are UML diagrams (i.e., class and object diagrams, as well as, state charts, sequence, and activity diagrams) enriched with e.g., name and signature patterns, and wildcards. They represent an independent pointcut mechanism that can be applied to any UML-based AOM language, allows to select all kinds of join points (i.e., structural-static, structural-dynamic, behavioral-static, and behavioral-dynamic) as well as supports composite pointcuts (M.T).

Tool Support. The approach claims rapid modeling support by a wide variety of CASE tools [61], which is due to using UML's light-weight extension mechanism. This is, however, questionable, since the authors also extended UML's metamodel (T.M). Both composition support and code generation support are currently not considered (T.C), (T.G).

5.2 The JAC Design Notation of Pawlak et al.

Language. The *JAC Design Notation* proposed by Pawlak et al. [46], [47] is mainly designed for the JAC Framework, i.e., an open source framework which includes a complete IDE with modeling support and serves as a middleware layer for aspectual components (L.I). Thus similar to the AODM approach of Stein et al., the *JAC Design Notation* represents an approach aligned to implementation as well as supporting external traceability from design to implementation (L.A), (L.T). Internal traceability is not applicable, since models typically are

not refined in the approach (L.R), (L.T). The approach is based on light-weight UML extensions. Since it has been developed out of a pragmatic need to express crosscutting concerns in the JAC Framework, the authors do not claim full compliance with UML but aim at keeping it intuitive and simple (L.E). The authors provide no information on the UML version used. The extended UML metamodel in [47], however, indicates the usage of a UML version prior to version 2.0 (L.L). The approach relies on class diagrams, only (L.D). Consequently, scalability is not supported by the *JAC Design Notation* (L.S). Beyond, the approach provides neither a description of a design process nor guidelines (L.DP).

ConcernComposition. The *JAC Design Notation* realizes the pointcut-advice composition mechanism (CC.M). The stereotype ≪aspect≫ which is derived from the UML meta-class Class is used to represent crosscutting concern modules (CC.CM) (cf. *Observer* in Figure 6). Consequently, the approach follows the asymmetric approach with respect to elements (CC.ES). Since ≪aspects≫ are composed with normal classes only, the *JAC Design Notation* also supports composition asymmetry (CC.CS). With respect to concern composition rules (CC.RS), the design notation represents a symmetric approach using a UML Association stereotyped with ≪pointcut≫ (cf. AspectualSubject). Composition is not available at modeling level but deferred until implementation (CC.C) and therefore no composed model is available either (CC.CP). Consequently, the composition semantics are those of the JAC framework (CC.S). Both modeling of interactions (CC.I) as well as conflict resolution (CC.CR) are not addressed at all by the approach. There are five stereotypes derived from the UML meta-class Operation (cf. AspectualKind) which specify advice. The specification of effects is partly considered by one of them, i.e., the stereotype ≪replace≫ which provides for either a replacement or a deletion. All other stereotypes are considered to have an enhancement effect (CC.E).

AsymmetricConcernComposition.

AspectualSubject. A join point model is explicitly defined in natural language, only [47] (AS.JPM) and join points are limited to method calls thus supporting behavioral-static join points, only (AS.BJP). Nevertheless, structural-dynamic join points are also supported via the ≪role≫ stereotype (cf. AspectualKind) (AS.SJP). The additional concept of "pointcut relation" corresponds to the asymmetric composition rule concept defined in the CRM (cf. Section 3). It is an association stereotyped with ≪pointcut≫. The association has a name and an optional tag to allow for adding extra semantics (cf. *stateChanged* in Figure 6). The rule connects the actual pointcut definition with the advice, i.e., the association ends contain information about the pointcut definition and the advice, respectively. Pointcuts are defined using a proprietary, textual language based on regular expressions and/or keywords (AS.P), (AS.QM), e.g., *!BookCopy.MODIFIERS* in Figure 6 selects as join points all method invocations (denoted with '!') of methods from class *BookCopy* that modify the object state (AS.BJP). Thus, the notation provides a low level of abstraction, while a high level of abstraction is not addressed (AS.A). The provided pointcut mechanism also allows composing simple pointcuts using operators, e.g., AND,

OR, etc. (AS.SP), (AS.CP). Furthermore, the approach introduces the "group" concept supporting the design of distributed applications. ≪group≫ is depicted as a stereotyped class but is derived from UML meta-class ModelElement and subsumes arbitrary and probably distributed classes that might need the same set of advice. It is, thus, part of the pointcut mechanism. For example in the observer aspect, subjects, i.e., arbitrary "base" classes that have to be observed, might be distributed and can be abstracted within a ≪group≫ named Subject. In the library management system such subjects might represent other resources than books such as journals, CDs, etc. The relative position is specified for behavioral advice, only, by three out of the five stereotypes for advice, i.e., ≪before≫, ≪after≫, and ≪around≫ (S.RP).

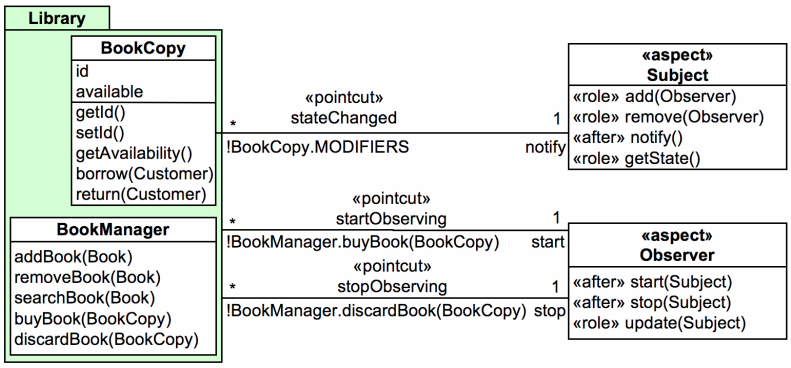

Fig. 6. The Observer Aspect Depicted Using the JAC Design Notation.

AspectualKind. Both behavioral and structural advice are represented as methods of ≪aspect≫ classes. The kind of advice is indicated by the stereotype of the advice operation. The stereotypes ≪before≫, ≪after≫, ≪around≫, and ≪replace≫ indicate behavioral advice, e.g., ≪after≫ *update()* in Figure 6 (AK.BA), whereas ≪role≫, i.e., the fifth stereotype for advice, represents a structural one (AK.SA). In the *JAC Design Notation*, structural advice which are implemented by ≪role≫ methods are not really added to the structure of the base class but can be invoked on the objects that are extended by the aspect, e.g., ≪role≫ *addObserver(BookManager)* can be invoked on *BookCopy* (cf. Figure 6). Moreover, these methods can access the extended class attributes and the attributes of the ≪aspect≫. Role methods therefore are similar to the "introduction" concept of AspectJ. Composite advice, in principle are possible within the JAC Framework through method composition. The *JAC Design Notation*, however, provides no means for explicitly modeling such composite advice (AK.CA). With respect to abstraction, the notation of Pawlak et al. represents predominantly a low level modeling approach, also with respect to advice, i.e., it shows aspect internals (AK.A).

Maturity. The *JAC Design Notation* has already been well described (M.T), (M.I) and has been applied to several well-known aspects like caching, authentication, tracing, and session in the context of a simple client-server application but not in combination with each other. These examples generally do not greatly differ from each other and follow the same simple principals but show the applicability of the notation to any aspect in general (M.E). It has been tested in real industrial projects like an online courses intranet site, an incident reporting web site, and a business management intranet tool (M.A).

Tool Support. The JAC Framework includes a complete IDE with modeling support. The provided modeling tools allow for designing base and aspect classes as well as their relations using the proposed UML notation (T.M). The IDE also supports code generation (i.e., Java) for the JAC framework (T.G). Weaving is supported at runtime (T.C) but not at design time.

5.3 Aspect-Oriented Software Development with Use Cases, Jacobson et al.

Language. The approach of Jacobson et al. [30] represents a use case driven software development method that has been realized by extending the UML 2.0 metamodel (L.L), (L.E). *Aspect-Oriented Software Development with Use Cases (AOSD/UC)* comes with a systematic process that focuses on the separation of concerns throughout the software development life cycle, i.e., from requirements engineering with use cases down to the implementation phase (L.DP). Since the approach covers the whole software development life cycle, it is aligned to both the requirements and the implementation phase (L.A). Furthermore, the approach fosters external traceability between all phases through explicit ≪trace≫ dependencies between models (L.T). During the whole software development life cycle the approach makes use of different UML diagrams including use case diagrams in the requirements phase as well as class diagrams and communication diagrams in the analysis phase. For the design phase, component diagrams can be refined into class diagrams (L.T), (L.R), while sequence diagrams are used to model behavioral features (L.D). The language extensions reflect the influence by the Hyper/J and AspectJ languages (L.I). Scalability of the approach is supported with high-level modeling elements (i.e., ≪use case slice≫) and has been demonstrated with a non-trivial example (L.S).

ConcernComposition. Concerns are modeled with the ≪use case slice≫ stereotype, which is derived from the UML meta-class Package (CC.CM). At this level, the approach of Jacobson et al. supports element symmetry (CC.ES). Taking a closer look, however, the ≪use case slice≫ - inspired by the Hyper/J language - encapsulates modeling artifacts of one phase in the software development life cycle, i.e., concerns are kept separately until the implementation phase. In this evaluation, the focus is on slices produced during design, where the artifacts include classes, sequence diagrams and ≪aspect≫ classifiers as depicted in Figure 7. Consequently, at this level of abstraction the approach follows element asymmetry (CC.ES). Although, at first sight it seems that *AOSD/UC* supports the compositor composition mechanism on the basis of UML package

merge, the internals of ≪use case slice≫ are such that they actually support the pointcut-advice and open class mechanisms (CC.M). In fact, composition is deferred until implementation (CC.C) thus, a composed module is not available at modeling level (CC.CP). Furthermore, the composition semantics seem to be those of AspectJ (CC.S), (CC.RS), (CC.CS). Since concerns are modeled as ≪use case slice≫, which represent use cases in the design phase, they inherit the relationships of use cases, i.e., inheritance, ≪extend≫ and ≪include≫ (CC.I). With respect to conflicting interactions, the approach follows a strategy that avoids conflicts through refactoring actions performed on models (CC.CR). The effects of composition are not modeled in *AOSD/UC* (CC.E).

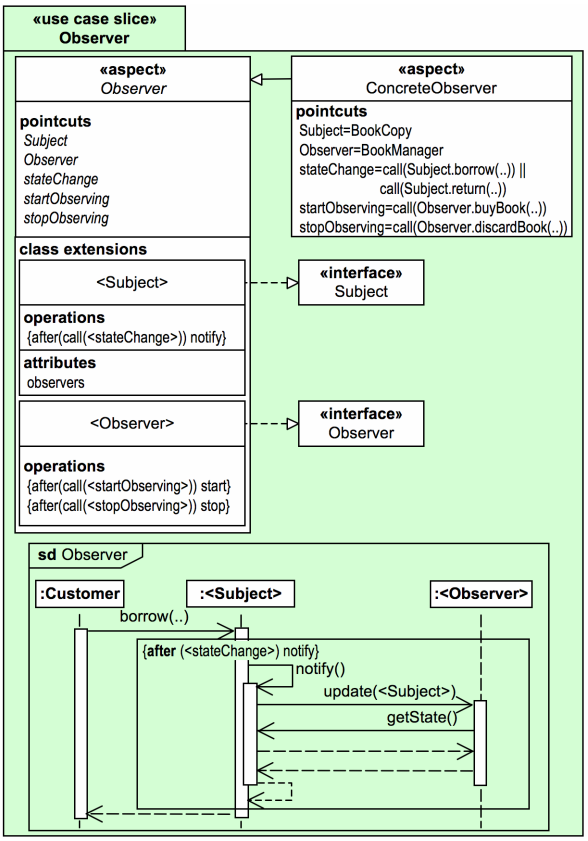

Fig. 7. The Observer Aspect Modeled Using the Notation of Jacobson et al.

AsymmetricConcernComposition.

AspectualSubject. The join point model of the approach is similar to that of AspectJ and is implicitly defined by the pointcut mechanism used (AS.JPM).

UML Classifiers are used as structural-static join points (AS.SJP), while behavioral-static and behavioral-dynamic join points are identified with the AspectJ pointcut language (AS.BJP), (AS.P). Consequently, the approach supports simple as well as complex pointcuts (AS.SP), (AS.CP). Pointcuts are specified in a separate "pointcuts" compartment of an ≪aspect≫ classifier. If specified as abstract, pointcuts need to be defined in a concrete aspect such as depicted in Figure 7 with ≪aspect≫ *ConcreteAspect*. The pointcut then is specified with AspectJ code allowing for usage of name patterns, type patterns and wildcards (AS.QM), e.g., pointcut *stateChange* represents a complex pointcut quantifying behavioral-static join points and the pointcut *Subject* represents a simple pointcut quantifying a structural-static join point. The aspectual subject is thus modeled at a detailed level but also can be modeled at a higher level of abstraction, i.e., with component interfaces in component diagrams (AS.A). As an alternative, placeholders such as *<Subject>* and *<stateChange>* can be used for parameterizing the use case slice, similarly to the template-based approaches of Clarke et al. (cf. Section 5.7) and France et al. (cf. Section 5.8). As it is done in AspectJ, the relative position (i.e., *before*, *after*, or *around*) is specified with the advice. In Figure 7 the *<Subject>* classifier is extended with an operation which is to be executed *after* the pointcut *<stateChange>* matches a join point (AS.RP).

AspectualKind. Advice are modeled at a low level of abstraction, only (AK.A) and are detailed as "class extensions" in a separate compartment of the ≪aspect≫ stereotype (cf. Figure 7 ≪aspect≫ *Observer*). As an example, for structural advice, the *<Subject>* classifier is extended with an attribute *observers* and an operation declaring the aspectual advice (cf. Figure 7). The fact, that the *<Subject>* classifier needs to implement the ≪interface≫ *Subject* is represented with a UML realization dependency (AK.SA). The behavioral advice is further detailed within sequence diagrams. So called "frames" are used to insert aspectual behavior and are labelled with the signature of corresponding operations, such as {*after (<stateChange>) notify*} in Figure 7 (AK.BA). There is no way for modeling composite aspectual features (AK.CA).

Maturity. The approach of Jacobson et al. has been recently elaborated in detail in three publications [30], although some of the ideas can be traced back to earlier works of the authors (M.I), (M.T). A hotel management system has been used as a comprehensive example encompassing several different concerns. The example is used to illustrate each phase in the software development life cycle (M.E). Still, no information of applications in real-world projects could be identified (M.A).

Tool Support. The *AOSD/UC* approach does not come with tool support. Since composition is deferred to the implementation phase, composition support within a tool is not the authors' focus (T.C). Nevertheless, modeling the extensions made to the UML metamodel currently are not supported within a tool (T.M), neither is it possible to generate code for a specific AO platform (T.G).

5.4 Behavioral Aspect Weaving with the Approach of Klein et al.

Language. The approach of Klein et al. [37] is originally based on Message Sequence Charts (MSC) a scenario language standardized by the ITU [57]. UML 2.0 sequence diagrams have been largely inspired by MSCs. Thus, the approach can be applied to sequence diagrams as is shown in the KerTheme proposal [29] (L.L). No extensions to the UML sequence diagrams (or MSCs) have been made in this respect (L.E). Klein et al. have designed a "weaving algorithm" for composing behaviors, i.e., scenarios modeled with UML sequence diagrams (or MSCs) (L.D). The composition is specified at modeling level regardless of any implementation platform (L.I). The approach does neither outline a design process nor guidelines (L.DP), since the goal is rather on complementing existing AOM approaches with a weaving mechanism for aspect behavior. The approach is thus not aligned to other phases in the software development life cycle either (L.A), nor are sequence diagrams further refined in the process of modeling (L.R). Consequently, the approach does neither provide means for supporting traceability (L.T) nor scalability (L.S)

ConcernComposition. The approach of Klein et al. supports the pointcut-advice composition mechanism (CC.M). Modeling all behavior is achieved by means of sequence diagrams (CC.ES). Nevertheless, an aspect consists of two scenarios having distinct roles when used in a composition (CC.CM): one defines a part of behavior, i.e., the pointcut, that should be replaced by another, i.e., the advice (cf. Figure 8). This replacement is done every time the behavior defined by the pointcut appears in the semantics of the base scenario. In this respect the approach follows rule asymmetry (CC.RS). Concerning composition symmetry, however, the approach is symmetric, since behavior that once has served as advice or pointcut could serve as base behavior some other time or vice versa (CC.CS). The composition semantics are clearly defined by the two-phase weaving algorithm. In the first phase, join points are detected in the base behavior according to the pattern specified in the pointcut. In the second phase, the base behavior then is composed with the behavior specified in the advice (CC.S). Currently, models are composed statically, the implementation of the algorithm is, however, subject to future work (CC.C) The composition results again in a sequence diagram (CC.CP) as depicted in Figure 9. The effect in the approach is always replacement by definition, since the behavior detected via pointcuts is replaced by the advice behavior. Consequently, with the current weaving algorithm there is no need for specifying an effect (CC.E). A means for specifying interactions (CC.I) and/or handling conflicts currently is not addressed but stated to be subject to future work (CC.CR).

AsymmetricConcernComposition.

AspectualSubject. Join points in the approach of Klein et al. are sub-MSCs or a sequence of messages in a sequence diagram (AS.JPM) that match the sub-MSC or sequence diagram defined by the pointcut as is depicted in Figure 8 (AS.P), (AS.QM). Consequently, the approach's join point model supports behavioral-static join points, only (AS.BJP), (AS.SJP). The pointcuts are modeled at a detailed level, only (AS.A) and in principle can be composed using

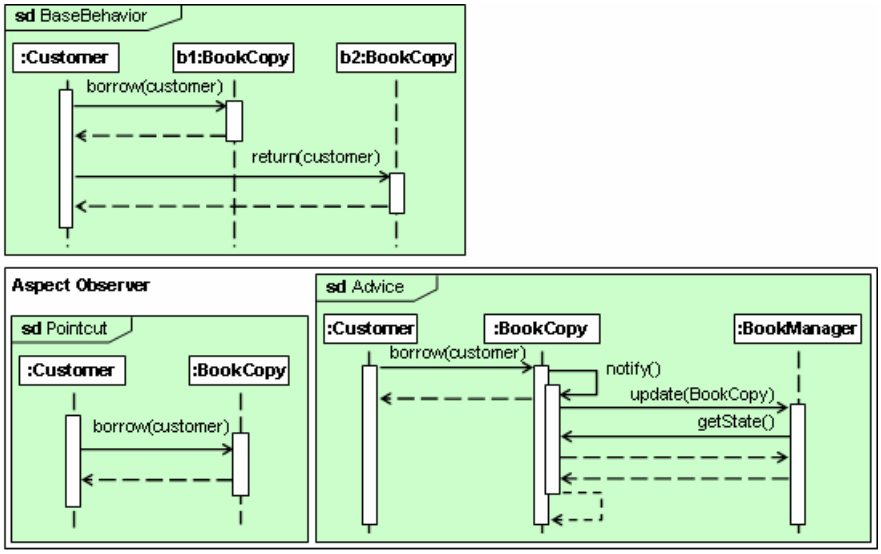

Fig. 8. The Observer Aspect Depicted Using the Approach of Klein et al.

the sequential composition operator of basic MSCs [37], although no explicit concept for composite pointcut is available (AS.SP), (AS.CP). There is no need for specifying a relative position, since the behavior detected via pointcuts is always replaced by the advice behavior and this basically simulates the *around* relative position kind (AS.RP). It has to be noted that the pointcut used in the running example cannot fully illustrate the expressiveness of the weaving algorithm's join point detection mechanism. The algorithm allows to detect much more complex patterns [37]. For instance, it is easy to express a pointcut as a sequence of messages.

AspectualKind. Like pointcuts, aspectual behavior is modeled as MSCs thus only supporting behavioral advice (AK.BA), (AK.SA). In principle, they also can be composed using the sequential composition operator of basic MSCs, although no explicit concept for composite advice is available (AK.CA). Aspectual features in the approach are modeled at a detailed level, only (AK.A). The advice illustrated in Figure 8, shows the observer behavior being inserted after the *BookCopy b1* is borrowed by the *Customer*. As already pointed out above, the behavior specified by the pointcut, i.e., the *borrow(customer)* message, has to be modeled within the advice if it is to appear in the composed behavior (cf. Figure 9).

Maturity. The approach of Klein et al. has been described in several applications (M.I), where similar examples have been used, i.e., a login process. The focus has been on demonstrating the weaving algorithm's join point detection mechanism on the basis of complex behaviors, i.e., pointcuts, to be detected in the base behavior (M.E). In order to allow for testing aspect-oriented models, the

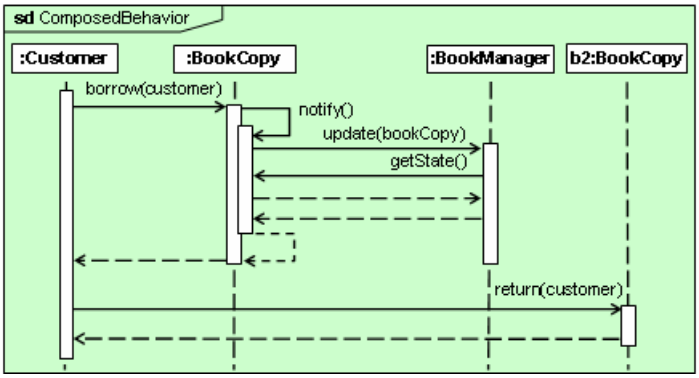

Fig. 9. The Observer Aspect Depicted Using the Approach of Klein et al.

approach recently has been combined with the Theme/UML approach of *Clark et al.* [29] (M.T). So far, the approach has not been employed in a real-world application (M.A).

Tool Support. Modeling of scenarios is possible with tools either supporting MSC or UML 2.0 sequence diagrams (T.M). The authors are currently working on an implementation of the weaving algorithm within KerMeta [42] (T.C). Still, code generation currently seems not to be the authors' focus. Since weaving is supported at modeling level, existing code generation facilities could be reused (T.G).

5.5 The Motorola Weavr Approach of Cottenier et al.

Language. The *Motorola Weavr* approach [13], [14] and tool has been developed in an industrial setting, i.e., the telecom infrastructure software industry. The modeling language of choice in this domain is the Specification and Decription Language (SDL) ITU recommendation [58] of which features such as composite structure diagrams and transition-oriented state machines have been adopted in UML 2.0. The *Motorola Weavr* approach consequently is based on UML 2.0 and a light-weight profile that completes the UML specification towards the SDL and AOM concepts (L.L), (L.E). Besides class diagrams the approach makes heavily use of composite structure diagrams as a refinement of class diagrams (L.R), (L.T). The behavioral features of concerns are modeled using transition-oriented state machines (cf. Figure 10 (c), (e), (f)) and the SDL action language as well as sequence diagrams which are used as test cases. A special "deployment diagram" is used to direct the composition of concern modules (L.D), (L.S). Although targeted at telecommunications domain, the approach is platform-independent. Indeed, platform-specific details are encapsulated within several code generators and code optimizers that prohibit round-trip engineering (L.T). Consequently, the approach rather aims at composing concerns at

the modeling level than drawing mappings onto platform-specific models (L.I). Nevertheless, the approach is aligned to the implementation phase (L.A). Furthermore, since Motorola uses the approach in production, one can infer that the approach supports scalability, which has already been proven with appropriate modeling examples (L.S). A design process for the approach has not yet been described, however. Likewise no guidelines are given (L.DP).

ConcernComposition. The *Motorola Weavr* approach supports the pointcut-advice composition mechanism (CC.M). Aspects are represented by the stereotype ≪Aspect≫ which is derived from the UML meta-class Class (CC.CM), (CC.ES). The approach puts forward rule asymmetry (CC.RS), since pointcuts and the binding to advices are modeled as parts of the aspect (cf. Figure 10 (d)). Furthermore, the approach also supports composition asymmetry, i.e., aspects can be woven into the base but not the other way round (CC.CS). The deployment of aspects to multiple base models as well as aspects can be modeled using the ≪crosscuts≫ stereotype derived from the UML meta-class Dependency. In order to resolve possible conflicts, the approach allows to define precedence relationships between aspects using a stereotype ≪follows≫, also derived from the UML meta-class Dependency. It has to be further noted, that precedence can also be defined at the level of concern composition rules again using the stereotype ≪follows≫ (CC.CR). Beyond, the approach defines two further dependency stereotypes for specifying interactions at concern module level, namely ≪hidden_by≫ and ≪dependent_on≫ [72] (CC.I). These relationships are usually depicted in a separate diagram called "deployment diagram" (cf. Figure 10 (a)). The approach, however, does not offer a way to specify effects (CC.E). The Motorola Weavr tool supports the static weaving of aspects into base models. The composition semantics consequently is clearly defined. More specifically, the approach distinguishes between a phase of detection (i.e., "connector instantiation") and of composition (i.e., "connector instance binding") (CC.S). The approach/tool, however, does not show the results of composition which are internally available in standard UML in a composed model (CC.CP). Instead, the modeler can simulate the composed model and view the specific base or aspect parts during execution (CC.C).

AsymmetricConcernComposition.

AspectualSubject. The join point model consists of action join points including call expression actions, output actions, create expression actions, and timer (re)set actions as well as transition join points including start transitions, initialization transitions, termination transitions, and triggered transitions (AS.JPM). Consequently, the approach's join point model supports behavioral-static join points (AS.BJP). Object instances represent structural-dynamic join points (AS.SJP). Pointcuts are defined using a stereotype which is derived from the UML meta-class Operation (AS.P), e.g., the pointcut ≪operation,Pointcut≫ *stateChange* in Figure 10 (d). The implementation of a pointcut is modeled as a transition-oriented state machine, which can be specified using wildcards (AS.QM) (cf. Figure 10 (b)). Consequently, the approach's pointcuts are modeled at a high level of abstraction where only the pointcuts' parameters are

known and at a detailed level using state machines (AS.A). Beyond, pointcuts can be composed to form more complex pointcuts by means of AND and OR logical composition operators [13] (AS.SP), (AS.CP). The relative position kind cannot be modeled but the approach supports the *around* relative position kind: Join points are always replaced by the advice behavior but can be called using an AspectJ-like *proceed()* action such as is depicted in Figure 10 (c) (AS.RP).

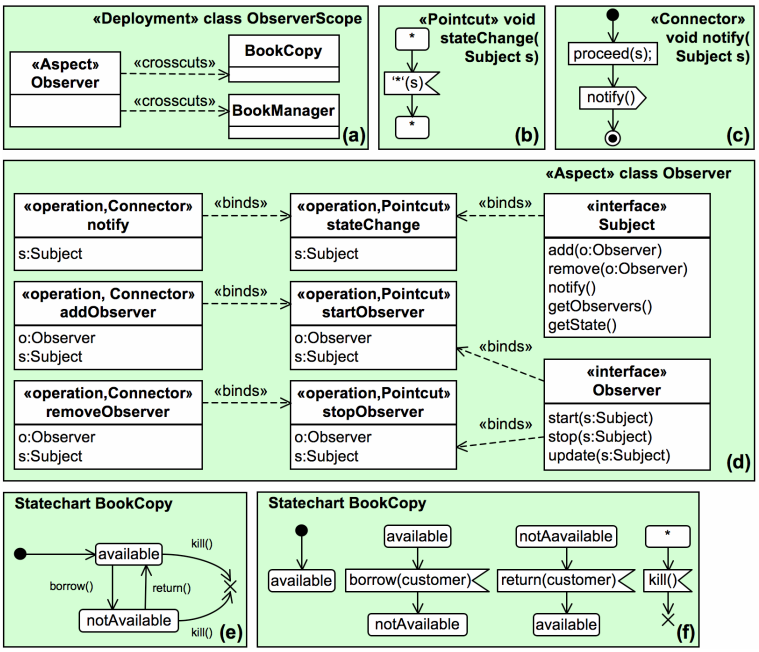

Fig. 10. The Observer Aspect Depicted Using the Approach of Cottenier et al.

AspectualKind. Like pointcuts, behavioral advice are modeled using a stereotype derived from the UML meta-class Operation, e.g., the ≪Connector≫ *notify()* in Figure 10(d). A connector corresponds to AspectJ's advice and like pointcuts in the *Motorola Weavr* approach is implemented as a transition-oriented state machine (cf. Figure 10 (c))(AK.BA). Thus, advice in the approach are modeled both at a high and a low level of abstraction (AK.A). The connectors are bound to the pointcuts to which they shall be applied using a ≪bind≫ stereotype which is derived from the UML meta-class Dependency (cf. Figure 10 (d)). It has to be further noted that precedence can also be defined for connectors again using the stereotype ≪follows≫. Structural advice are modeled via interfaces, only (AK.SA). In the running example, the ≪Aspect≫ Observer introduces two interfaces *Subject* and *Observer*, which are bound to pointcuts of the aspect using the ≪bind≫ dependency (cf. Figure 10 (d)). The seman-

tics of this relationship is that the interfaces are bound to the object instances that contain joinpoints for the specified pointcuts [13], which is similar to the approach of Pawlak et al. (cf. Section 5.2). Consequently, the *Subject* interface is bound to the *stateChange* pointcut, while the *Observer* interface is bound to the other pointcuts. To the best of our knowledge, there exists no means for modeling composite advice (AK.CA).

Maturity. The approach of Cottenier et al. represents one of the most recent approaches to AOM and has already been illustrated in several publications (M.I), (M.T). Besides a set of simple modeling examples, aspects covering exception handling, recovery, atomicity, and a two-phase commit protocol have been applied to a server-based communication system [13] (M.E). The *Motorola Weavr* approach and tool is already being used in production (M.A) and is made available to academia under a free of charge license.

Tool Support. The *Motorola Weavr* is designed as an add-in for the Telelogic TAU MDA tool[6] (T.M) and allows composing aspect models with base models as well as verification of the composed model via simulation (T.C). Starting from the composed model, existing code generation facilities such as the Motorola Mousetrap code generator [3] can be used (T.G). The Motorola Weavr tool is already being deployed in production at Motorola, in the network infrastructure business unit [13].

5.6 The AOSD Profile of Aldawud et al.

Language. The *AOSD Profile* (L.E) of Aldawud et al. [2], [17] is based on UML version 1.x (L.L) and is aimed at being independent of any particular AOP language (L.I). While class diagrams are used to express the structural dependencies, state machines model the behavioral dependencies of concerns (L.D). The models are continuously refined from class diagrams to state machines (L.R). In order to do so, a set of guidelines for using the concepts provided by the *AOSD Profile* is offered (L.DP), which allows for external traceability from the requirements phase but not specifically for internal traceability (L.T), (L.A). The specific usage of state machines and their event propagation mechanism indicates that the approach does not support scalability and we are not aware of a modeling example proving the opposite (L.S).

ConcernComposition. In the *AOSD Profile*, crosscutting concerns have a separate representation in the form of the stereotype ≪aspect≫ (CC.CM), (CC.ES), which is derived from the UML meta-class Class (cf. Figure 11). Although, it is allowed to relate aspects to other aspects, each aspect has to be woven into at least one base class and, hence, this actually constitutes an asymmetric view of composing concerns (CC.CS). An integrated model view where aspects would already be woven into the base classes is not provided (CC.CP). Composition is rather deferred to implementation [40] (CC.C). At a more detailed level, one can see that the approach supports the compositor composition mechanism (CC.M): in the *AOSD Profile* approach concurrent state machines

[6] http://www.telelogic.com/products/tau/g2/

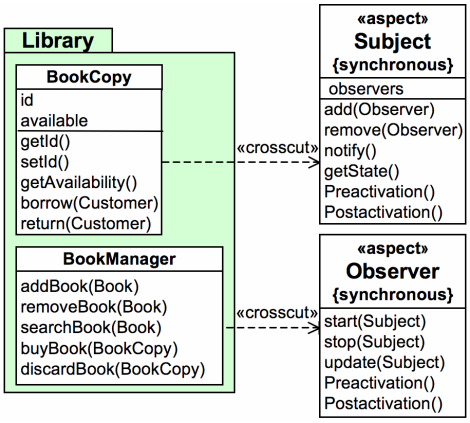

Fig. 11. The Observer Aspect Modeled Using the AOSD Profile.

are used to model both non-crosscutting and crosscutting behavior in orthogonal regions, meaning element symmetry at the level of state machines (cf. Figure 12). More specifically, the composition semantics (CC.S) are "specified" by the event mechanism used to indicate the flow of crosscutting behavior in state charts (CC.RS). The state machines thus implicitly express the composition semantics. The ≪crosscut≫ dependencies[7] between aspects and base classes as well as aspects and aspects dictate the ordering of events propagated in the orthogonal regions of statecharts (CC.I) (cf. Figure 11). Since the state charts allow for specifying the temporal sequence in the control flow of crosscutting behavior, i.e., an ordering of aspects, further conflict resolution is implicitly available (CC.CR). The effect of adaptations, however, cannot be modeled (CC.E).

SymmetricConcernComposition. Composable elements in the approach of Aldawud et al. are elements of UML state machine diagrams, in particular events (S.BCE), whereas structural composable elements are not supported (S.SCE). Events trigger transitions from one state to another. The approach of Aldawud et al. makes use of broadcasting events to cause transitions in orthogonal regions of the same or other state machines, i.e., to activate other concerns. For example, the *observingBookManager* state machine in Figure 12 describes the behavior of the *BookManager* class. If a new *BookCopy* is bought (cf. *buyBook()*) the transition from state IDLE to state observing is triggered. This transition, however, triggers the transition from *IDLE* to the *startObservingSubject* state in the *observing* region. For the *observedBookCopy* state machine of the *BookCopy* class, this means a transition from state *notObserved* to state *observed*, given that the *BookCopy* has been in the state *notObserved*. The event mechanism of state machines allow to "compose" the behavior of dif-

[7] Please note, that the reading direction of the ≪crosscut≫ dependencies is different to the other approaches, e.g., *BookCopy* is *crosscut by* the aspect *Subject*.

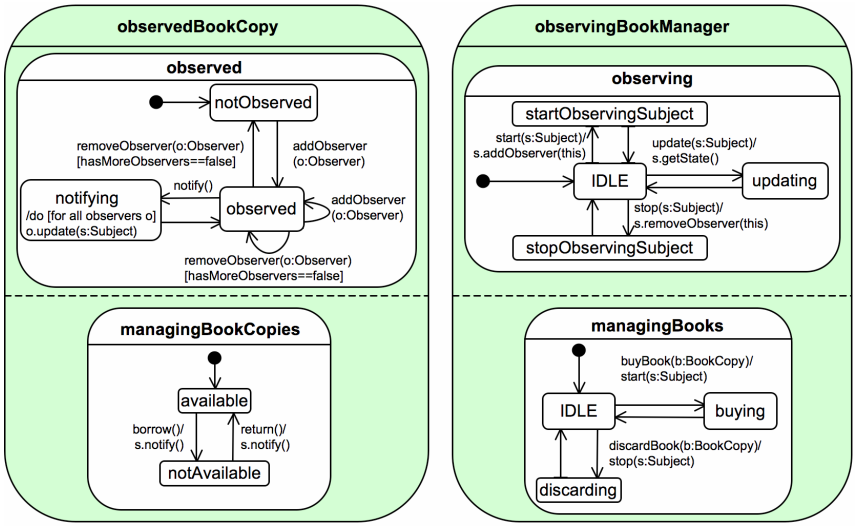

Fig. 12. The Observer's Crosscutting Behavior, Aldawud et al.

ferent concerns represented in orthogonal regions of state machines following a merge integration strategy (S.M). The corresponding elements, or rather events, are explicitly defined by using naming patterns. The approach thus supports a name-based match method, only (S.MM). With respect to the level of abstraction, the details captured by the state machines suggest a low level but not a high level of abstraction. Indeed, recently the use of the State pattern [22] has been used to translate the behavior captured within state machines to code [40] (S.A).

Maturity. Although the approach is described in several recent publications (M.I), (M.T), it is illustrated using a single example, the bounded buffer system, only. Still, it covers various aspects, namely, synchronization, scheduling, and error handling (M.E). A real-world application of the approach, however, is not available (M.A).

Tool Support. Due to using the UML profile extension mechanism, modeling support within the approach is available through existing UML modeling tools (T.M). Nevertheless, neither composition (T.C) nor code generation support have yet been addressed (T.G).

5.7 The Theme/UML Approach of Clarke et al.

Language. The *Theme* approach of Clarke et al. [8] provides means for AOSD in the analysis phase with *Theme/Doc*, which assists in identifying crosscutting concerns in requirements documents, and in the design phase with *Theme/UML*. In this survey, the focus is on *Theme/UML*, which is used in producing separate design models for each "theme" from the requirements phase (L.T), (L.A),

i.e., the encapsulation of a concern representing some kind of functionality in a system [8]. *Theme/UML* is based on a heavy-weight extension of the UML metamodel version 1.3 (L.L), (L.E). It is designed as a platform-independent AOM approach, which originated from SOP [9], and evolved from the composition patterns approach of Clarke [6], [7] as well as provides mappings to AspectJ, AspectWerkz, and Hyper/J (L.I), (L.T), (L.A). Basically, *Theme/UML* poses no restrictions on what UML diagrams might be used for modeling. Nevertheless, particularly package and class diagrams are used for modeling structure and sequence diagrams are used for behavioral modeling (L.D). Theme/UML allows every concern to be refined separately and then to be composed into a new model (L.R). Scalability of the approach is supported by using UML packages for modeling concerns and has been demonstrated with non-trivial examples [8] (L.S). Beyond, the authors outline a design process for their modeling approach (L.DP).

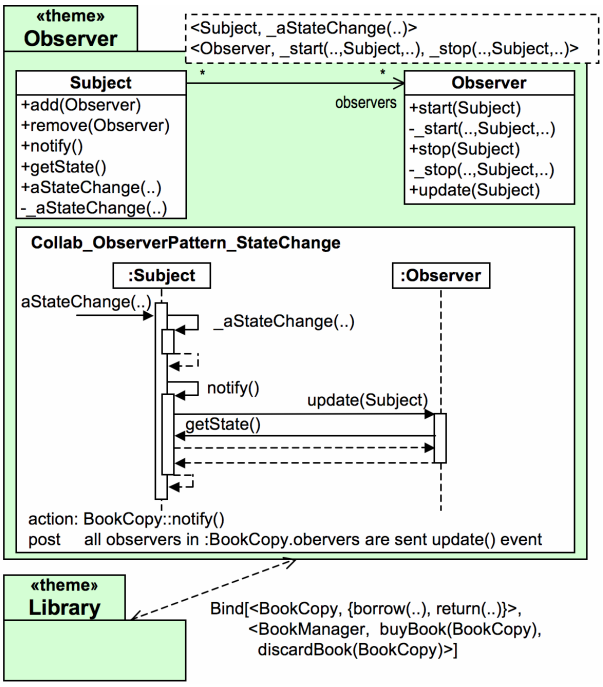

Fig. 13. The Observer Aspect Depicted Using Theme/UML.

ConcernComposition. *Theme/UML* realizes the compositor composition mechanism (CC.M). Concerns are encapsulated in UML packages denoted with a stereotype ≪theme≫ (cf. *Observer* and *Library* in Figure 13). Concern modules generally are modeled using standard UML notation. Crosscutting concerns,

however, are realized using UML's modified template mechanism, which allows instantiating template parameters more than once, thus supporting multiple bindings (CC.CM), (CC.ES), (CC.CS). The composition semantics are clearly stated in [6] for both how to detect the corresponding elements to be composed as well as for the actual composition itself (CC.S). A set of themes is composed statically into a composed ≪theme≫ (CC.C) as is shown in Figure 14, i.e., the *ObserverLibrary theme* is composed of the *Observer* and *Library themes* from Figure 13 (CC.CP). Besides relating two or more themes through *Theme/UML*'s "composition relationships", i.e., specialization from UML meta-class Relationship (CC.RS), there is no other way to model interactions between concern modules (CC.I). The composition relationships can also be used at a more fine-grained level, e.g., for specifying composition at the level of classes and attributes. Special attachments or "tags" to the *Theme/UML* composition relationships represent the conflict resolution mechanism. First, the "prec" tags define an ordering for theme precedence, with 1 indicating the highest precedence. Second, in case of a conflict the "resolve" tag allows specifying default values for elements of a certain type (e.g., visibility of attributes is private). And third, for a specific conflict the "resolve" tag allows defining explicit composition output values. *Theme/UML* wants developers to first compose all non-crosscutting themes and then weave crosscutting themes one after the other into the composed model, thus forcing the developer to consider the ordering of crosscutting themes (CC.CR). The approach, however, does not provide modeling means for specifying the effects of composition (CC.E).

SymmetricConcernComposition. Composable elements in Theme/UML are identified with the introduction of the new meta-class ComposableElements [7]. More specifically, the UML meta-classes Attribute, Interaction, Collaboration (S.BCE), Operation, Association, Classifier, and Package (S.SCE) all inherit from the new meta-class and thus are allowed to be composed. For identifying the corresponding elements of two or more themes, the approach allows to tag the composition relationships with the "match" tag. *Theme/UML*, currently supports two ways of matching composable elements, namely match-by-name and no-match. The latter states that the composeable elements participating in the composition relationship do not match (S.MM). Composition is catered for through three different integration strategies (specialization of UML meta-class Relationship), "merge" (S.M) and "override" (S.O), and "bind" (S.B), which is a specialization of merge and allows composing crosscutting themes with non-crosscutting ones. This binding can be done for several themes. In Figure 13, the crosscutting ≪theme≫ *Observer* is composed with the *Library* ≪theme≫ using the bind integration strategy. The template parameters of the crosscutting theme (i.e., classes, operations, and attributes) placed on the theme package template within a dotted box need to be bound to concrete modeling elements of a non-crosscutting theme. The sequence diagram templates in a crosscutting theme (cf. *ObserverPattern_StateChange* in Figure 13) allow modeling crosscutting behavior and when it shall be triggered, e.g., within the control flow of other operations. In contrast, the class diagram templates of a crosscutting

theme allow modeling crosscutting structure. The concrete binding is specified by the "bind" tag placed on the composition relationship between the crosscutting theme and other themes. It binds the template parameters to actual classes, operations, and attributes possibly using wildcards. This way, the *Subject* class is bound to *BookCopy*, and the *stateChange()* operation is bound to *borrow()* and *return()* (S.B). Since, *Theme/UML*'s composition relationships can relate composite elements such as package as well as fine-grained ones such as attributes, the approach supports both modeling at a high level of abstraction as well as at a low level (S.A).

Maturity. The Theme/UML approach represents one of the most mature, and still evolving approaches to AOM (M.I). Lately, first results on the approach's extension with the join point designation diagrams of Stein et al. [63] has been presented [28] as well as an extension with the weaving algorithm of Klein et al. [37] has been published [29] (M.T). Theme/UML comes with a plethora of literature and modeling examples such as the synchronization and observer aspects in a digital library example [11], the logging aspect in an expression evaluation system example and a course management system example. The crystal game application presented in [8] consists of more than 15 concern modules amongst them two crosscutting ones. The composition of some of them is demonstrated. Furthermore, two similarly sized case studies are presented, i.e., phone features and usage licensing (M.E). It is not clear, however, if the approach has been applied in a real world project (M.A).

Tool Support. Besides first proposals in [29] with respect to composition, no information on a tool for *Theme/UML* supporting either modeling, composition or code generation has been provided (T.M), (T.C), (T.G).

5.8 Aspect-Oriented Architecture Models of France et al.

Language. The *Aspect-Oriented Architecture Models (AAM)* approach of France et al. [19], [50] is based on UML 2.0 (L.L). The language is designed as a platform-independent approach with no particular platform in mind (L.I). Concerns are modeled using template diagrams, i.e., package diagram templates, class diagram templates and communication diagram templates [19] as well as recently sequence diagram templates [51], [59] (L.D). With respect to using UML templates, the approach is similar to *Theme/UML* (cf. Section 5.7). For readability purposes, however, the authors prefer to provide a notation different to standard UML templates and in contrast denote template model elements using '|'. This notation is based on the Role-Based Metamodeling Language [20], [36], which is a UML-based pattern language designed as an extension to the UML (L.E). The use of packages for capturing concerns caters for scalability, although this has not yet been demonstrated within an example encompassing several concerns (L.S). The approach is not specifically aligned to the requirements or implementation phases (L.A) and does not support external traceability (L.T). Nevertheless, similar to *Theme/UML*, the different models are continuously refined and at some point composed (L.T), (L.R). A design process is briefly outlined in terms of guidelines (L.DP).

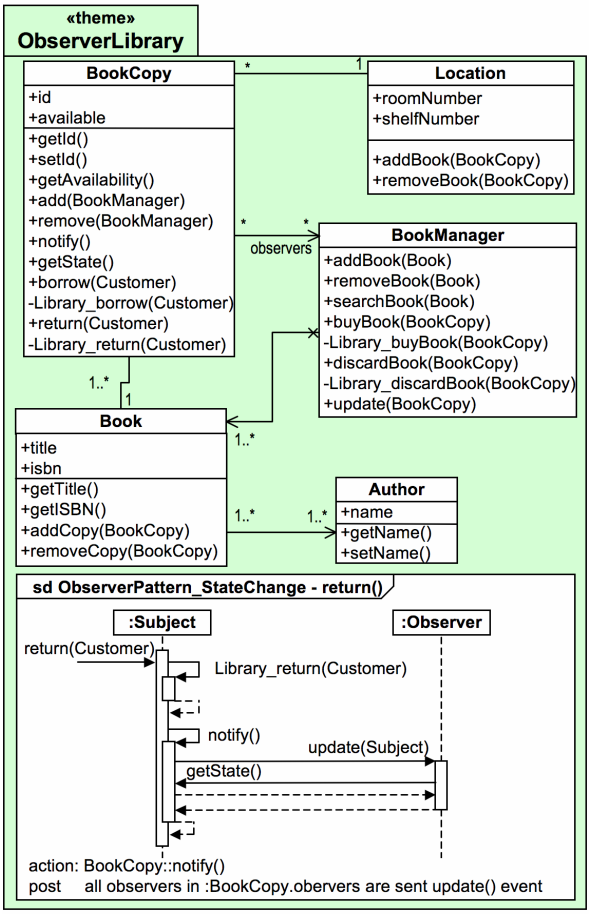

Fig. 14. The Composed Model, Clarke et al.

ConcernComposition. The approach of France et al. originally is based on the compositor composition mechanism similar to the *Theme/UML* approach. Recently, specific attention has been paid, however, to the composition of sequence diagrams [51], [59], which realizes the pointcut-advice composition mechanism (CC.M). France et al. support element symmetry in that all concerns are modeled as UML packages (CC.CM), (CC.ES). The authors distinguish, however, between "primary models" and "aspect models", which model crosscutting concerns. Aspect models are based on template diagrams, which are described by parameterized packages. These packages include class diagram templates as in Figure 15 (a), communication diagram templates as in Figure 15 (b)-(d), and recently sequence diagram templates (cf. Figure 18 (a)). A textual "binding" to a certain application instantiates a "context-specific" aspect model from the

UML template. In the context of the library management system, the following binding instantiates the aspect model for the observer pattern from Figure 15 and results in the context-specific aspect shown in Figure 16:

```
(|Subject,BookCopy);            (|Observer, BookManager);
(|stateChange(),borrowCopy()); (|doStart(s:|Subject),buyBook());
(|stateChange(),returnCopy()); (|doStop(s:|Subject),discardBook());
(|observers, bookManagers);
```

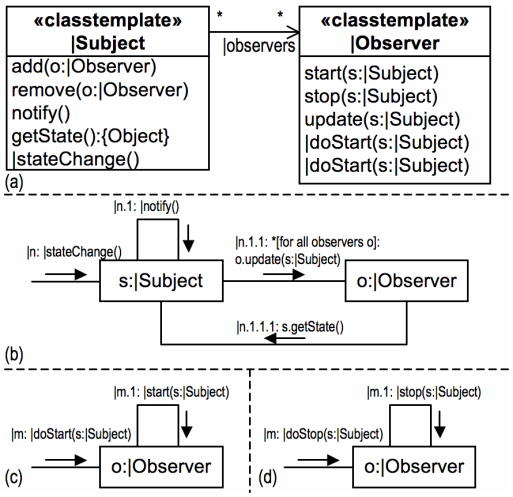

Fig. 15. The Observer Aspect Model, France et al.

The context-specific aspect models are finally used for composition with the base model, suggesting rule and composition symmetry (CC.RS), (CC.CS). However, the composition of sequence diagrams is somewhat different. The locations of where to introduce a behavioral advice defined within an aspect sequence diagram template (cf. Figure 18 (a)) are specified using "tags" in the primary model (cf. Figure 18 (b)). The aspect sequence diagram template can be composed with the primary model, only (CC.CS), and the rule information is placed within the primary model (CC.RS), meaning asymmetric composition and asymmetric placement of rules. Recently, the approaches composition semantics (CC.S) in terms of a composition metamodel have been operationalized in KerMeta, a metamodeling language that extends the Essential Meta-Object Facility (EMOF 2.0) with an action language. Thereby, the semantics of detection have also been operationalized (i.e., the getMatchingElements() operation), which allows for detecting (syntactical) conflicts (CC.CR). The composition is done statically yielding standard UML diagrams, i.e., class diagrams, communication diagrams and sequence diagrams (CC.CP). The composed model is shown in Figure 17 and

Figure 18 (c), respectively (CC.CP) in terms of standard UML. Since KerMeta allows specifying operations with its action language, dynamic composition of models is subject to future work (CC.C). The approach also proposes so called "composition directives" which are intended to refine the concern composition rules used to compose models. The use of so called "model composition directives", allows specifying the order in which aspect models are composed with the primary model. These "precedes" and "follows" model composition directives are depicted as stereotyped UML dependencies between aspect models and represent a conflict resolution mechanism. Other forms of interactions between modules, however, cannot be modeled (CC.I). So called "element composition directives" amongst others allow to add, remove, and replace model elements. The element composition directives, consequently, also serve as a conflict resolution mechanism (CC.CR). The approach does not describe ways to specify effects (CC.E).

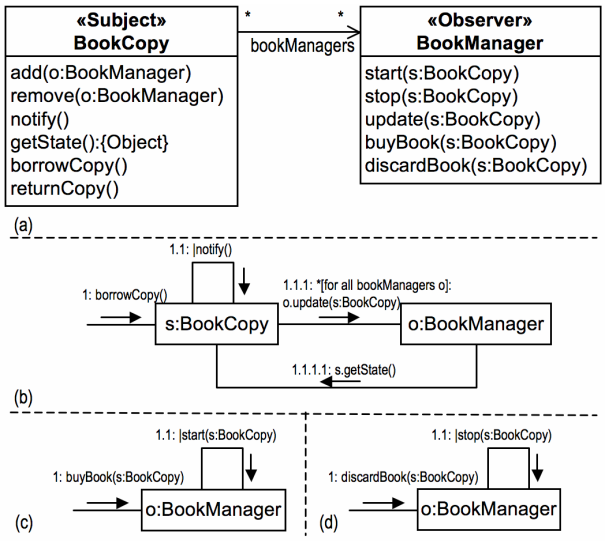

Fig. 16. The Context-Specific Aspect Model, France et al.

SymmetricConcernComposition. The composition of class diagrams is specified with the composition metamodel defined in [50]. Composable elements in the composition metamodel are realized with the meta-class Mergeable. Currently, the composition metamodel has been operationalized for class diagrams only. In this respect, mergeable elements are Operation, Association, Classifier, and Model (S.SCE), (S.BCE). Originally, the approach used only name-based matching method in order to identify the corresponding elements in different models. In [50], this mechanism has been extended. The authors introduce a

signature-based method, which means that elements are matched according to their syntactic properties, i.e., an attribute or an association end defined in the element's meta-class. This match method is realized with the getMatchingElements() operation of the composition metamodel (S.MM). The approach basically, supports a merge integration strategy, only (S.M). Support for the bind integration strategy is realized through the instantiation of aspect models to context-specific aspect models. The template parameters of the aspect models denoted using '|' need to be bound to concrete modeling elements of the primary model. For example, the class |*Subject* (cf. Figure 15 (a)) is bound to *BookCopy* and the operation |*stateChange()* is bound to *borrow()* and *return()* (cf. Figure 15 (b)-(d)). This is done with the textual binding as specified before and the resulting context-specific aspect model is shown in Figure 16 . While the class diagram templates model crosscutting structure, the communication diagram templates model crosscutting behavior. The context-specific aspect models then can be composed with the primary model using the original merge integration strategy (S.B). Lately, the possibility of overriding model elements has been introduced with the introduction of composition directives. The "override" element composition directive defines an override relationship between two potentially conflicting model elements [50] (S.O). Consequently, with respect to symmetric concern composition the approach supports both modeling at a high level of abstraction (e.g., with a high-level model view and model composition directives) as well as at a low level (e.g., with detailed class and communication diagrams as well as element composition directives) (S.A)

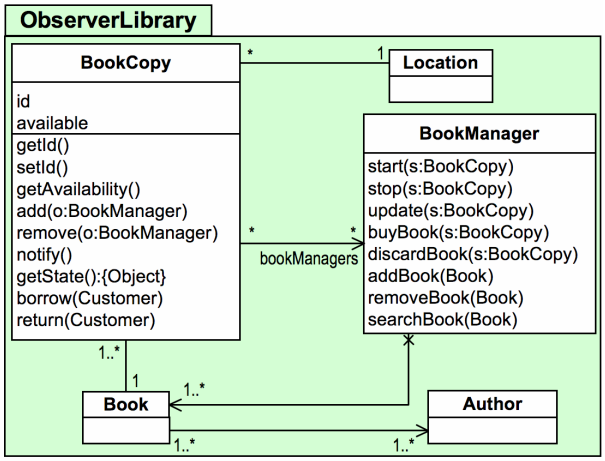

Fig. 17. The Composed Model, France et al.

AsymmetricConcernComposition. The composition of sequence diagrams realizes the pointcut-advice composition mechanism.

AspectualSubject. The join point model is implicitly defined (AS.JPM) as the set of primary sequence model elements, e.g., lifelines and messages to which the aspect sequence model elements need to be composed, thus supporting solely behavioral-static join points. (AS.SJP), (AS.BJP) [51]. The locations of where to apply the advice in the primary sequence diagram are "tagged" with special stereotypes (AS.P). Thereby, two stereotypes can be distinguished: A ≪simpleAspect≫ is a stereotyped UML message originating from and targeting the same lifeline, which is used when the aspectual behavior just needs to be inserted. A ≪compositeAspect≫ is a stereotype for UML's Combined fragment, that captures a message or a sequence of messages in the primary model as join point (AS.SP), (AS.QM). In the running example, the ≪compositeAspect≫ *Observer* is used to tag the primary sequence diagram *Borrow* (cf Figure 18 (b)). The ≪compositeAspect≫ also includes the binding of the *BookCopy* and *BookManager* classes to the corresponding template lifelines of the aspect sequence diagram template. This pointcut mechanism in terms of tagging a model does not allow for composed pointcuts (AS.CP). Concerning the aspectual subjects, thus, models provide information at a detailed level (AS.A). The relative position is modeled within the aspect sequence diagram template using stereotyped combined fragments. Besides the typical before, after, and around relative position kinds, the approach provides two special stereotypes, namely, ≪begin≫ and ≪end≫. The begin/end combined fragment captures the aspectual behavior that should preced/follow the messages encompassed in the ≪compositeAspect≫ of the primary model. In contrast, the ≪after≫ combined fragment shown in Figure 18 (a) defines the aspectual behavior that will appear after each message encompassed in the ≪compositeAspect≫ of the primary model.

AspectualKind. On the basis of sequence diagram templates, the approach of France et al. provides for behavioral advice, only (AK.BA), (AK.SA). The approach does not forsee possibilities of combining two or more behavioral advice to form a more complex one (AK.CA). The running example shows that behavioral advice are modeled at a detailed level within the approach (AK.A).

Maturity. The approach of France et al. is among the most mature AOM approaches and has been elaborated on in numerous publications (M.I) providing examples such as a simple banking system including the authorization aspect, the replicated repository aspect, the redundant controller aspect, and the transaction aspect for controlling money transfers, as well as the buffer aspect which decouples output producers from the writing device in a system (M.E) Currently, the approach is further developed with respect to its composition mechanism and tool support thereof (M.T). Yet, it has not been applied in a real-world project (M.A).

Tool Support. The implementation of an integrated toolset has been proposed in [50]. This toolset shall provide modeling support (T.M), i.e., for modeling aspect model diagram templates built on top of the Eclipse Modeling Framework[8] and for instantiating context-specific aspect models from these templates

[8] http://www.eclipse.org/emf/

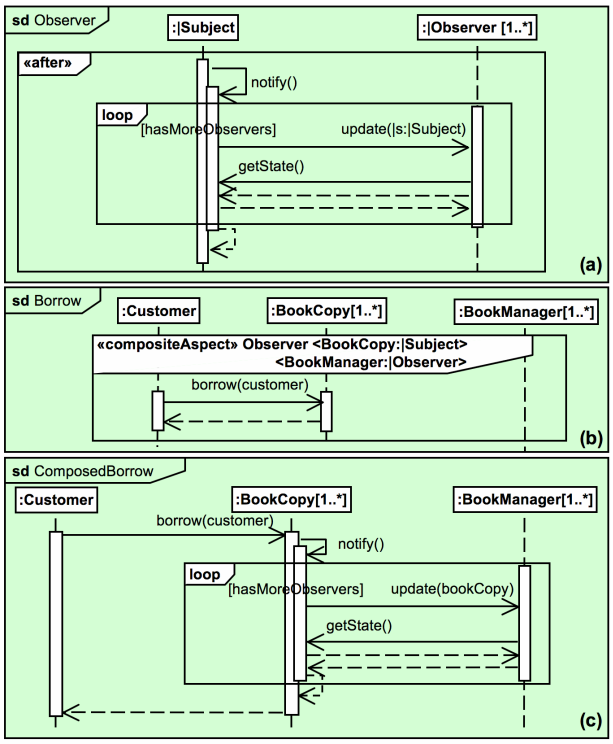

Fig. 18. Weaving Aspectual Behavior With Sequence Diagrams, France et al.

built on top of Rational Rose. In [**?**], the implementation of the Kompose tool[9] for composing structural base and context-specific aspect models which is built on top of KerMeta is discussed (T.C). A tool supporting composition of behavioral models, i.e., sequence diagrams is currently under development [51], while a tool for code generation currently is not planned (T.G).

6 Lessons Learned

The results of the evaluation have revealed interesting peculiarities of current AOM approaches. In the following, the findings are summarized and the results are illustrated at a glance with tables according to the six categories of criteria from the criteria catalogue within Sections 6.1 to 6.6. Finally, Section 6.7 presents the general findings and conclusions concerning the AOM research field and specifically points out what needs to be done in terms of further development of AOM approaches.

[9] http://www.cs.colostate.edu/puml/kompose2.html

6.1 Language

The summary of the evaluation with respect to the *Language* category can be found in Table 1.

Popularity of UML Profiles and UML 2.0. For the design phase, one can observe that UML is the choice for designing an aspect-oriented design language with two exceptions [65], [66], only. With respect to the UML version used, there is quite a balance between UML 1.x and UML 2.0. Typically, recent approaches already are based on the new UML 2.0 specification. Furthermore, it is advisable that existing UML 1.x approaches are updated to also support UML 2.0. For extending UML with aspect-oriented concepts, UML's inherent extension mechanism, i.e., profiles, is popular. In terms of tool support this is beneficial, since UML profiles are supported by almost any UML tool. Nevertheless, it has to be noted, that the profile-based approaches of Stein et al. and Aldawud et al. do not provide a reference implementation of their profiles within a UML tool. Consequently, designers first are required to manually redefine the profile specification in their UML tool of choice. With respect to those approaches that are based on metamodel extensions, i.e., Clark et al., Jacobson et al., and France et al., almost no modeling support is available. Only France et al. are currently developing an integrated toolset providing modeling support upon the EMF.

Behavioral Diagrams are Catching up. Except for the approach of Klein et al. all approaches make extensive use of structural diagrams, i.e., class diagrams and package diagrams as well as component diagrams and composite structure diagrams. The use of behavioral diagrams in order to describe crosscutting behavior is more and more emphasized by also composing behavioral diagrams such as in the approaches of Klein et al., Cottenier et al., and France et al.

Missing Guidance in the Design Process. Since AOM is still a quite young research field, the support of design processes has often been disregarded and thus is rather underdeveloped. Only two surveyed approaches, i.e., Clarke et al. and Jacobson et al., provide a detailed design process description. Additionally, two approaches have some guidance in designing aspect-oriented models in terms of guidelines, namely France et al. and Aldawud et al. It is not surprising, that the two approaches offering a design process have been chosen by the AOSD Europe Network of Excellence to form the basis for defining a generic aspect-oriented design process [10].

Missing Full External Traceability. The majority of approaches does not support external traceability at all or with respect to either prior or later development phases, only. The approaches of Jacobson et al. and Clarke et al. are the only ones that do support external traceability from the requirements engineering phase until implementation. Since traceability is a crucial issue in any kind of software development, AOM approaches need to take care to support for traceability.

Moderate Scalability. Half of the approaches provides for scalability by supporting modeling concepts that allow hiding details from the modeler and thus, modeling at a high level of abstraction. The modeling examples used,

	Modeling Language (L.L)		Extension Mechanism (L.E)		Platform Influences(L.I)	Diagrams (L.D)		Design Process (L.DP)		Scalability (L.S)		Refinement Mapping (L.R)	Alignment to Phase (L.A)	Traceability (L.T)	
	UML 1.x	UML 2.0	Metamodel	UML Profile		Structural Diagrams (L.D)	Structural Diagrams (L.D)	process description	guidlines	high-level modeling elements	proven with examples			internal	external
Stein	✓		✓	✓	AspectJ	CD, ColD	SD, UCD						I	n/a	D->I
Pawlak	✓		✓	✓	JAC	CD							I	n/a	D->I
Jacobson		✓	✓		AspectJ, Hyper/J	CD, CompD	UCD, SD, ComD	✓		✓	✓	c,e	R,I	✓	R->A->D->I
Klein		✓					SD						n/a	n/a	
Cottenier		✓		✓		CD, CSD, DD	SD, SMD			✓	✓	c,e	I	✓	
Aldawud	✓			✓		CD	SMD		✓			e	R	n/a	R->D
Clarke	✓		✓		SOP, AspectJ, Hyper/J	PD, CD	SD	✓		✓	✓	c,e	R,I	✓	R->D->I
France		✓	✓			PD, CD	ComD, SD		✓	✓		c,e	n/a	✓	

Legend:

✓	supported
	not supported
n/a	not applicable
c	creating
e	extending

CD	Class Diagram
CompD	Component Diagram
CSD	Composite Structure Diagram
PD	Package Diagram
ColD	Collaboration Diagram (UML 1.x)

SD	Sequence Diagram
UCD	Use Case Diagram
ComD	Communication Diagram
SMD	State Machine Diagram
DD	Deployment Diagram

R	Requirements
A	Analysis
D	Design
I	Implementation

Table 1. Language

however, are seldom of a size that justifies scalability. Only, the approaches of Jacobson et al., Cottenier et al., and Clarke et al. have provided proof that their approaches can cope with the composition of three or more concerns. Due to this limited interest in proving the applicability of the AOM approaches in large-scale applications, it will be required in the future to evaluate how scalable those approaches are in real large-scale applications by means of quantitative studies.

6.2 ConcernComposition

For the *ConcernComposition* category, the lessons learned are drawn from Table 2.

Popularity of Asymmetric Concern Composition. The majority of AOM approaches follows the pointcut-advice mechanism or a combination of the pointcut-advice and open class mechanisms. The approach of France et al. seems to be the first that combines the compositor mechanism and the pointcut-advice mechanism. It is however interesting to note that up to now little interest has been shown in evaluating when asymmetric and symmetric approaches have prevailing advantages and shall be employed.

Influence of Composition Mechanism on Element, Composition, and Rule Symmetry. Generally, one can observe that asymmetric composition mechanisms usually imply element asymmetry, composition asymmetry as well as rule asymmetry, although this is not an inherent characteristic of asymmetric approaches. On the opposite, a symmetric composition continuously achieves symmetric values. The approach of Klein et al. is one exception to the rule supporting element and composition symmetry, since the modeling concepts used for modeling the base behavior, the crosscutting behavior as well as the pointcuts is done using sequence diagrams, which in different contexts can play different roles. Other examples are the approaches of Jacobson et al. and Aldawud et al. which at a higher level do support element symmetry (≪use case slice≫) and element asymmetry (≪aspect≫), respectively. At a lower level, however, one can observe the reverse: Jacobson et al. model and compose ≪aspect≫ classes with normal classes while Aldawud et al. uses state machine regions to model (non-)crosscutting concerns. The approach of France et al. that combines the compositor mechanism and the pointcut-advice mechanism results in rule symmetry for the compositor mechanism and rule asymmetry for the pointcut-advice mechanism.

Composition often Deferred to Implementation. Composition at modeling level is only supported by half of the surveyed approaches. The composition always yields a composed model conforming to standard UML except for the approach of Clarke et al., where the outcome is represented by a composite ≪theme≫. The composition semantics of Cottenier et al. and France et al. have already been implemented within (prototype) tools, while Klein et al. have defined a weaving algorithm. In contrast to Clarke et al., this operationalization enables dynamic composition at modeling level. Well-defined semantics already at the modeling level is a necessary prerequisite for achieving more than models-as-blue-print. If, as intended in MDE, models shall replace code appropriate semantics along with composed models to assist the designer will be required.

Moderate Support for Modeling Interactions. Modeling of interactions both at the level of modules and the level of rules is still considered rather moderately. Typically the means for specifying interactions at the same time are modeling concepts for resolving conflicts, e.g., an ordering for composing concern modules or concern composition rules. The approach of Cottenier et al. represents an exception by proposing ≪hidden_by≫ and ≪dependent_on≫ dependencies between aspects. Since it is natural to expect that large-scale systems might put forward interaction of modules, for an unambiguous specification of the system it will be necessary to make module interaction explicit.

Conflict Resolution Based on an Ordering for Composition, Only. A conflict resolution is provided by half of the approaches focusing on resolving conflicts. The conflict resolution mechanisms in most cases comprise means for specifying an ordering of how concern modules are composed, some provide further means, e.g., to resolve naming conflicts. Only one approach's composition semantics (France et al.) allows to detect (syntactical) conflicts. The approach of Jacobson et al. is the only one that explicitly avoids conflicts by continuously

	ConcernModule (CC.CM)	Composition Mechanism (CC.M)	Element Symmetry (CC.ES)	Composition Symmetry (CC.CS)	Rule Symmetry (CC.RS)	Effect (CC.E)	Composition Semantics (CC.S)		Composition (CC.C)		Composed Module (CC.CP)	Interaction (CC.I)		Conflict Resolution (CC.CR)		
							detection	composition	static	dynamic		Module	Rule	avoid	detect	resolve
Stein	<<aspect>> Class	PA, OC					✓	✓	✓		n/a	✓				✓
Pawlak	<<aspect>> Class	PA		✓		✓					n/a					
Jacobson	<<use case slice>> Package; <<aspect>> Classifier	PA, OC	~								n/a	✓		✓		
Klein	Pair of basic SDs representing Pointcut & Advice	PA	✓	✓			✓	✓	✓	~	Stand. UML					
Cottenier	<<Aspect>> Class	PA					✓	✓	✓	✓	Stand. UML	✓	✓			✓
Aldawud	<<aspect>> Class; State machine regions	CMP	~				✓	✓			n/a	✓				✓
Clarke	<<theme>> Package (Template)	CMP	✓	✓	✓		✓	✓	✓		Themes	✓	✓			✓
France	Package (Template)	CMP, PA	✓	✓	~		✓	✓	✓	~	Stand. UML	✓	✓		✓	✓

Legend:

✓	supported
	not supported
~	partly supported
n/a	not applicable

PA	Pointcut-Advice
OC	Open Class
CMP	Compositor

Table 2. ConcernComposition

refactoring models. Consequently, the provision of more sophisticated conflict resolution mechanisms including the detection of conflicts should be focussed in future.

Effect Not Considered. Modeling the effect of composition is not considered at all. Only the JAC design notation of Pawlak et al. provides a stereotype ≪replace≫ for advice which indicates an effect of either a replacement or a deletion. The possibility of modeling the effect, however, would enhance the explicitness of models and thus allow for providing better conflict identification.

6.3 AsymmetricConcernComposition

This part of the lessons learned specifically summarizes the results for the approaches adhering to the pointcut-advice and/or open class composition mechanism, i.e., Stein et al., Pawlak et al., Jacobson et al., Klein et al., Cottenier et al., and France et al.

AspectualSubject The results of evaluating the approaches according to the criteria encompassed by the *AspectualSubject* sub-category are shown in Table 3.

Missing Formal Definition of Join Point Models. Half of the surveyed approaches made the Join Point Model not explicit but defined it "implicitly" via their pointcut mechanism. The remaining did provide a Join Point Model but mostly in terms of a natural language description, only. Consequently, formal definitions of join point models are missing and shall be considered in future development of AOM approaches.

	Structural Join Point (AS.SJP)		Behavioral Join Point (AS.BJP)		Explicite JoinPointModel (AS.JPM)	Standardized Pointcut (AS.P)	SimplePointcut (AS.SP)		CompositePointcut (AS.CP)		Quantification Method (AS.QM)			Relative Position (AS.RP)			Abstraction (AS.A)	
	static	dynamic	static	dynamic			graphical	textual	graphical	textual	declarative	imperative	enumeration	before	around	after	high	low
Stein	✓		✓	✓	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	✓
Pawlak		✓	✓		✓			✓		✓	✓		✓	✓	✓	✓		✓
Jacobson	✓		✓	✓		✓		✓		✓	✓		✓	✓	✓	✓	✓	✓
Klein			✓		✓	✓	✓		~		✓		✓		✓			✓
Cottenier		✓	✓		✓	✓	✓			✓	✓		✓		✓		✓	✓
France			✓			✓	✓				✓		✓	✓	✓	✓		✓

Legend:

✓	supported
	not supported
~	partly supported

Table 3. AspectualSubject

Limited Support for Structural Join Points. Supporting the full spectrum of join points in AOM approaches would be beneficial with respect to possible enhancement, replacements, and deletions made in the form of advice. The approaches realizing the open class composition mechanism support structural-static join points. The approaches of Pawlak et al. and Cottenier et al. allow for structural advice which are introduced at run-time, however, thus requiring structural-dynamic join points (e.g., object instances). The approaches of Klein et al. and France et al. do not provide structural join points at all. Nevertheless, it has to be noted, that the approach of France et al. compensates the lack of at least structural-static join points due to also supporting the compositor composition mechanism. All approaches consider either behavioral-static or behavioral-dynamic join points or both.

Standards-Based Pointcut Mechanism Preferred. All but one of the approaches provide a standards-based pointcut mechanism. Pawlak et al. proposes the only proprietary pointcut language based on regular expressions and/or keywords as well as on UML associations. The approaches of Jacobson et al. and

Stein et al. reuse AspectJ's pointcut language. The rest relies on UML behavioral diagrams to specify pointcuts, e.g., sequence diagrams (Klein et al.) and combined fragments in sequence diagrams (France et al.), as well as state machines (Cottenier et al.)

Good Support of Composite Pointcuts. The reuse of simple pointcuts is fostered by the use of composite pointcuts, for which good support is provided in almost all the approaches. The textual pointcut mechanism of Stein et al., Pawlak et al., and Jacobson et al. provide textual mechanisms that allow for composing simple pointcuts using logical operators. Cottenier et al. allow composing pointcuts with special logical composition operators. While France et al. provides no support for modeling composite pointcuts, the approach of Klein et al. in principle could support composition of sequence diagrams on the basis of the sequential composition operator.

No Imperative Pointcuts. The approaches exclusively allow to select join points declaratively and/or by enumeration but not in the form of imperative pointcuts, which could serve as a more verbose pointcut definition.

Good Support of Relative Position Kinds. Except for the approaches of Klein et al. and Cottenier et al., which only cater for the *around* relative position kind and therefore subsume the *before* and *after* kinds, the other approaches provide full support of relative position kinds. Interestingly, France et al. support two uncommon positions, namely "begin" and "end". Furthermore, it might be interesting to discuss how the relative positions shall be interpreted in the light of model elements, since finally the composition in any case will do concrete insertions and deletions of metamodel instances.

Modeling Aspectual Subjects at Different Levels of Abstraction. All approaches allow to apply advices to the base at a low level of abstraction, but half of the approaches also allows modeling the subjects of adaptation at a higher level of abstraction. For the applicability of AOM, a high level of abstraction is beneficial, whereas for code generation purposes as well as an automated execution of the model a detailed specification at a low level of abstraction is necessary.

AspectualKind In Table 4, the results for the *AspectualKind* sub-category are provided.

Composite Advice Not Considered. While most approaches provide modeling means for both behavioral and structural advice, composing advice to form more complex ones and to foster reuse is not considered by any of the approaches. Nevertheless, the approach of Klein et al. in principle could support composition of sequence diagrams, i.e., behavioral advice, on the basis of the sequential composition operator.

Modeling Aspectual Kinds at a Low Level of Abstraction. Again, as for abstraction with respect to the aspectual subjects all approaches allow modeling advice at a low level of abstraction. The approach of Cottenier et al. is the only one supporting modeling also at a high level of abstraction. It would

	Behavioral Advice (AK.BA)	Structural Advice (AK.SB)	Composite Advice (AK.CA)	Abstraction (AK.A)	
				high	low
Stein	✓	✓			✓
Pawlak	✓	✓			✓
Jacobson	✓	✓			✓
Klein	✓		~		✓
Cottenier	✓	✓		✓	✓
France	✓				✓

Legend:

✓	supported
	not supported
~	partly supported

Table 4. AspectualKind

be beneficial, however, if approaches would provide for high as well as low level of abstraction.

6.4 SymmetricConcernComposition

This part of the lessons learned specifically summarizes the results for those approaches supporting the compositor composition mechanism, i.e., Aldawud et al., Theme et al., and France et al (cf. Table 5).

Equal Support of Structural and Behavioral Composable Elements. While the approach of Aldawud et al. is based on state machines, the approach of France et al. allows composing class diagrams. Thus, the supported composable elements are for the first case behavioral and in the second case structural. Ideally, composition is possible for both kinds, such as in the composition metamodel of Clarke et al. Nevertheless, for the approach of France et al., it has to be noted that the lacking support for behavioral composable elements can be seen as compensated due to supporting the pointcut-advice composition mechanism for composing sequence diagrams.

Predominance of Matching with Names. Finding corresponding elements on the basis of a name-based match method represents an easy-to-implement method and in many cases quite an effective way, which is consequently supported by all three approaches. Clarke et al. additionally allow to explicitly model which elements shall not match by supporting a *no-match* method. Recently, France et al. have proposed a more expressive match method based on the elements signatures. The advantage of such a matching method lies in the possibility of finding corresponding elements with more fine-grained matching criteria other than the element name, e.g., the values of meta-class properties such as the "isAbstract" meta-attribute of classes, as well as in the possibility of detecting and resolving conflicts.

Merge Integration as a Default Strategy. With respect to supporting different integration strategies, merge is supported by all surveyed approaches.

	Comp. Elements		Match Method (S.MM)			Integration Strategy			Abstraction (S.A)	
	Structural Composeable Elements (S.SCE)	Behavioral Composeable Elements (S.BCE)	match-by-name	match-by-signature	no-match	Merge (S.M)	Override (S.O)	Bind (S.B)	high	low
Aldawud		✓	✓			✓				✓
Clarke	✓	✓	✓		✓	✓	✓	✓	✓	✓
France	✓		✓	✓		✓	✓	✓	✓	✓

Legend:	
✓	supported
	not supported

Table 5. SymmetricConcernComposition

The override and bind strategies are also supported by all approaches but Aldawud et al. In terms of expressivity, ideally all integration strategies are supported by an approach.

Modeling at Different Abstraction Levels. The approaches generally provide good support for modeling at a high and a low level of abstraction. In particular, the approaches of Clarke et al. and France et al. offer high level views on the concern modules to be composed in terms of UML packages, while Aldawud et al. model at a detailed level by means of state machines. A high level of abstraction is beneficial in terms of an approach's scalability, whereas for code generation purposes as well as an automated execution of the model a low level view such as the one of Aldawud et al. is required. In this respect, state machines probably represent the most elaborate mechanism for describing an objects life cycle, and are supported by code generation tools such as Raphsody and StateMate[10].

6.5 Maturity

In Table 6, the measures for the *Maturity* category are depicted for all surveyed approaches.

Missing Complex Examples. The majority of the approaches, have demonstrated their approaches on the basis of rather trivial examples in which not more than two concerns are composed. In this respect, Jacobson et al., Cottenier et al., and Clarke et al. set a good example by demonstrating their approaches with non-trivial modeling problems. It would therefore be beneficial if all AOM approaches would document their capabilities on bases of more complex examples.

Lack of Application in Real-World Projects. The applicability of AOM languages has rarely been tested in real-world projects. An exception is the approach of Pawlak et al., which has already been applied to real industrial projects

[10] http://www.ilogix.com/

	Modeling Examples (M.E)		Applications (M.A)	Publications (M.I)	Topicality (M.T)
	Concerns	Examples			
Stein	2	2		4	`02
Pawlak	1	1	✓	3	`05
Jacobson	>3	1		3	`05
Klein	1	1		9	`06
Cottenier	>4	2	✓	9	`06
Aldawud	>2	1		5	`05
Clarke	>15	>5		>15	`06
France	1	3		>10	`06

Legend:

✓	supported
	not supported
n	number of concerns, examples, publications
`YY	year of most recent publication

Table 6. Maturity

like an online courses intranet site, an incident reporting web site, and a business management intranet tool. Another exception is the approach of Cottenier et al., since their Motorola Weavr tool is already being deployed in production at Motorola.

6.6 Tool Support

Finally, the results concerning the approaches' *Tool Support* are summarized in Table 7.

Missing Tool Support for Composition and Code Generation. While modeling support in many approaches is implicitly available due to the use of UML's profile mechanism, support for code generation and composition is rare. The approach of Cottenier et al. is the only one that allows for modeling, composition, and code generation. On the one hand, for those approaches that defer composition to the implementation phase, code generation facilities that produce code for the target AOP platform would be beneficial. For example, the approach of Pawlak et al. allows for code generation for the JAC Framework. On the other hand, composition tool support is essential for those approaches, that have specified the composition semantics for model composition. The acceptance of an AOM approach will be minimal, if it requires the designers to first model each concern separately and then to manually compose them. In this respect, the approaches of France et al. and Cottenier et al. provide appropriate tool support, while the "weaving algorithm" of Klein et al. is currently being implemented.

6.7 General Findings

This section sums up the most important conclusions that are valid for the academic community as well as for practitioners.

	Modeling Support (T.M)	Composition Support (T.C)	Code Generation (T.G)
Stein	✓		
Pawlak	✓		✓
Jacobson			
Klein	✓	~	
Cottenier	✓	✓	✓
Aldawud	✓		
Clarke			
France	✓	✓	

Legend:

✓	supported
	not supported
~	partly supported

Table 7. Tool Support

No Explicit Winner. From the results obtained in the evaluation, it is not possible to nominate a winner. The selection of an AOM approach thus has to be made in the context of a specific project. For instance, if the requirement is enabling documentation and communication between partners of an AspectJ-based project, a design notation for AspectJ programs is needed and the approach of Stein et al. would be a good solution. On the other hand, one might wish for separating concerns in different class diagrams at design time and then before implementation compose the different views. In this respect, both approaches of Clark et al. as well as France et al. would be possible options. Depending on when the current prototype implementation of their integrated toolset is made available, the approach of France et al. might even be preferred for its tool support. Staying with tool support, the approaches of Pawlak et al. and Cottenier et al. might be of interest. In contrast to the approach of Stein et al., the *JAC Design Notation* of Pawlak et al. has been specifically designed for the JAC Framework but comes with modeling support as well as a code generation facility. What might argue in favor of the *AODM* of Stein et al. is AspectJ's maturity. The unique selling point of the Motorola Weavr of Cottenier et al. is its comprehensive tool support and in particular its composition mechanism for state machines, for which an academic license can be obtained. Nevertheless, the approach does not allow composing structural diagrams.

Full Spectrum of UML not Exploited. Interestingly enough, apart of the approaches of Palawak et al. and Klein et al. the surveyed approaches support structural as well as behavioral diagrams. Thus, in principle, the approaches allow the modeler to consider both structure and behavior through their approach. Nevertheless, currently no approach addresses the full spectrum of UML in terms of UML's structural and behavioral diagrams as well as their composition. It is comforting that the presumably most often employed UML diagrams have been addressed by AOM approaches. As can be seen in Table 1, the most important structural diagram, i.e., class diagram, is supported by all approaches addressing

structural modeling some of them also allowing their composition. Likewise for modeling behavior the sequence diagram is covered by all approaches addressing behavior, some of them also supporting their composition. It would therefore be interesting to investigate how to compose diagram types for which composition has not yet been specified, e.g., composite structure diagrams. As a consequence, it would also be interesting how the approaches can be combined in order to gain from best practises in AOM. In this respect, a first promising attempt has recently been conducted by Clarke et al. and Klein et al. by proposing *KerTheme* [29], a combination of their approaches. Furthermore, the *Theme/UML* approach of Clarke et al. has also been extended with join point designation [28] diagrams of Stein et al.

Missing Guidance on When to use Asymmetric vs. Symmetric Approaches. It might be a natural pre-assumption that approaches either follow the asymmetric school of thought or the symmetric school of thought. France et al. is interesting in this respect since it provides for both. The recent extension made to *Theme/UML* in the *KerTheme* proposal also follow this direction of combining different composition mechanisms. In terms of expressivity, the advantages of using different composition mechanisms are obvious. Nevertheless, the question when to best apply an asymmetric or a symmetric approach has not yet been answered sufficiently.

Missing Tool Support. Certainly, one of the most vital factors for the adoption of AOM in practise but also in academia is the provision of appropriate tools. Basic modeling support is provided for some approaches, i.e., for those approaches which rely on UML profiles and consequently can rely on existing UML modeling tools. AOM, however, is also about the composition of various concerns that have been carefully separated beforehand. This is a complex task to be understood by the modeler hence, support for model composition is vital. Still, this is not commonly provided by the AOM approaches and is also hampered by the fact that not all AOM approaches provide for a well-defined composition semantics. Finally, code generation, which is an important requirement for MDE, is least supported by tools. Only the approaches of Pawlak et al. and Cottenier et al. provide such facilities.

Adoption of Approaches Requires Scalability. For the adoption of AOM, it would be beneficial if its applicability would be better evaluated with respect to large scale applications and real-world scenarios. This is currently only sufficiently addressed by very few approaches, namely, Clarke et al., Jacobson et al., and Cottenier et al. Nevertheless, scalability is a feature important to practitioners and has a great impact on the chances of AOM approaches to be adopted.

7 Conclusion and Outlook

This paper presents the evaluation of eight aspect-oriented modeling approaches. Since the research field of aspect-oriented modeling is quite young and a common understanding of concepts has not yet been established, we identified prior to

our survey the important concepts of aspect-oriented modeling in the form of a conceptual reference model for aspect-oriented modeling. This conceptual reference model is an intermediate but essential step towards defining an evaluation framework, i.e., it forms the basis for inferring a set of concrete criteria. The actual evaluation according to our evaluation framework is furthermore supported by a running example, which has proven to be very helpful on the one hand, to explore the applicability of each individual approach and on the other hand, to allow for a direct comparison of the approaches. The evaluation results reveal that currently there is no decidedly superior aspect-oriented modeling approach but that each individual approach has it specific strengths and shortcomings. For applying aspect-oriented modeling in a project this means selecting an aspect-oriented modeling approach by matching the projects requirements with the approach's features. In this respect, the evaluation results and lessons learned represent the basis for a well-founded decision.

We are currently working on completing our survey with the evaluation results of those approaches that could not be presented in this survey due to the space restrictions, in particular [12], [21], [23], [27], [31], and [70]. Furthermore, we also plan to evaluate domain-specific approaches to aspect-oriented modeling.

Another research question we would like to investigate concerns the kind of examples that are needed to fully evaluate aspect-oriented modeling approaches. While our running example's obvious advantages are its conciseness and understandability, it does not allow to investigate each and every concept of the aspect-oriented modeling approaches, e.g., modeling interactions between concern modules. In this respect, we propose a catalogue of aspect-oriented modeling examples, an initiative that is not new but has successfully been applied in some disciplines[11]. A set of common, well-defined problems would provide a way of comparing approaches and results. Such a catalogue of standard example problems could serve as a kind of benchmark for the aspect-oriented modeling domain possibly posing some form of compliance levels for new proposals to aspect-oriented modeling but also provide for teaching materials.

As a consequence of such a catalogue of aspect-oriented modeling examples, in a further extension of this survey, each approach could be applied to more complex problems or within extensive case studies. Ideally, the approaches could be applied simultaneously in a real world project. On the one hand, this would provide more insight into already supported criteria of our evaluation framework such as scalability. On the other hand, such an empirical study is necessary for otherwise non-measurable criteria such as reusability, evolvability, flexibility, and ease of learning.

8 Acknowledgements

We would like to thank the authors of the surveyed approaches for providing us with their feedback on an earlier version of this work.

[11] http://www.cs.cmu.edu/M̃odProb/index.html

References

1. Mehmet Akşit, Lodewijk Bergmans, and Sinan Vural. An Object-Oriented Language-Database Integration Model: The Composition-Filters Approach. In *Proc. of the 6th European Conference on Object-Oriented Programming (ECOOP'92), Utrecht, The Netherlands*, June/July 1992.
2. Omar Aldawud, Tzilla Elrad, and Atef Bader. UML Profile for Aspect-Oriented Software Development. In *Proc. of the 3rd Int. Workshop on Aspect Oriented Modeling, Boston, Massachusetts*, March 2003.
3. Paul Baker, Shiou Loh, and Frank Weil. Model-Driven Engineering in a Large Industrial Context - Motorola Case Study. In *Proc. of the 8th Int. Conference on Model Driven Engineering Languages and Systems (MoDELS'05), Montego Bay, Jamaica*, October 2005.
4. Gordon S. Blair, Lynne Blair, Awais Rashid, Ana Moreira, João Araújo, and Ruzanna Chitchyan. Engineering Aspect-Oriented Systems. In R.E. Filman, T. Elrad, S. Clarke, and M. Akşit, editors, *Aspect-Oriented Software Development*, pages 379–406. Addison-Wesley, Boston, 2005.
5. Ruzanna Chitchyan, Awais Rashid, Pete Sawyer, Alessandro Garcia, Mónica Pinto Alarcon, Jethro Bakker, Bedir Tekinerdoğan, Siobhán Clarke, and Andrew Jackson. Survey of Aspect-Oriented Analysis and Design Approaches. Technical Report D11 AOSD-Europe-ULANC-9, AOSD-Europe, May 2005.
6. Siobhán Clarke. *Composition of Object-Oriented Software Design Models*. PhD thesis, Dublin City University, January 2001.
7. Siobhán Clarke. Extending Standard UML with Model Composition Semantics. *Science of Computer Programming*, 44(1):71–100, July 2002.
8. Siobhán Clarke and Elisa Banaissad. *Aspect-Oriented Analysis and Design The Theme Approach*. Addison-Wesley, Upper Saddle River, March 2005.
9. Siobhán Clarke, William Harrison, Harold Ossher, and Peri Tarr. Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code. In *Proc. of the 14th Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'99), Denver, Colorado*, pages 325–339, November 1999.
10. Siobhán Clarke and Andrew Jackson. Refined AOD Process. Technical Report D57 AOSD-Europe-TCD-D57, AOSD-Europe, August 2006.
11. Siobhán Clarke and Robert J. Walker. Generic Aspect-Oriented Design with Theme/UML. In R.E. Filman, T. Elrad, S. Clarke, and M. Akşit, editors, *Aspect-Oriented Software Development*, pages 425–458. Addison-Wesley, Boston, 2005.
12. Wesley Coelho and Gail C. Murphy. Presenting Crosscutting Structure with Active Models. In *Proc. of the 5th Int. Conference on Aspect-Oriented Software Development (AOSD'06), Bonn, Germany*, March 2006.
13. Thomas Cottenier, Aswin van den Berg, and Tzilla Elrad. Joinpoint Inference from Behavioral Specification to Implementation. In *Proc. of the 21st Europ. Conf. on Object-Oriented Programming, Berlin, Germany*, to appear July 2007.
14. Thomas Cottenier, Aswin van den Berg, and Tzilla Elrad. The Motorola WEAVR: Model Weaving in a Large Industrial Context. In *Proc. of the 6th Int. Conf. on Aspect-Oriented Software Development (AOSD'07), Vancouver, Canada*, March 2007.
15. Steven Op de beeck, Eddy Truyen, Nelis Boucké, Frans Sanen, Maarten Bynens, and Wouter Joosen. A Study of Aspect-Oriented Design Approaches. Technical Report CW435, Department of Computer Science, Katholieke Universiteit Leuven, 2006.

16. Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, Englewood Cliffs, New Jersey, 1976.
17. Tzilla Elrad, Omar Aldawud, and Atef Bader. Expressing Aspects Using UML Behavioral and Structural Diagrams. In R.E. Filman, T. Elrad, S. Clarke, and M. Akşit, editors, *Aspect-Oriented Software Development*, pages 459–478. Addison-Wesley, Boston, 2005.
18. Robert E. Filman, Tzilla Elrad, Siobhán Clarke, and Mehmet Akşit, editors. *Aspect-Oriented Software Development*. Addison-Wesley, Boston, 2005.
19. Robert France, Indrakshi Ray, Geri Georg, and Sudipto Ghosh. Aspect-oriented Approach to Early Design Modelling. *IEE Proceedings Software*, 151(4):173– 185, August 2004.
20. Robert B. France, Dae-Kyoo Kim, Sudipto Ghosh, and Eunjee Song. A UML-Based Pattern Specification Technique. *IEEE Trans. Software Eng.*, 30(3):193–206, 2004.
21. Lidia Fuentes, Mónica Pinto, and José M. Troya. Supporting the Development of CAM/DAOP Applications: An Integrated Development Process. *Software - Practice and Experience*, 37(1):21–64, 2007.
22. Erich Gamma, Richard Helm, Ralph Hohnson, and John Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesely, 2004.
23. John Grundy. Multi-Perspective Specification, Design and Implementation of Software Components Using Aspects. *Int. Journal of Software Engineering and Knowledge Engineering*, 20(6), 2000.
24. Stefan Hanenberg. *Design Dimensions of Aspect-Oriented Systems*. PhD thesis, University Duisburg-Essen, October 2005.
25. William H. Harrison and Harold L. Ossher. Subject-Oriented Programming - A Critique of Pure Objects. In *Proc. of the 8th Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'93), Washington, DC, USA*, September 1993.
26. William H. Harrison, Harold L. Ossher, and Peri L. Tarr. Asymmetrically vs. Symmetrically Organized Paradigms for Software Composition. Technical report, IBM Research Division, Thomas J. Watson Research Center, December 2002.
27. Wai-Ming Ho, Jean-Marc Jézéquel, François Pennaneac'h, and Noël Plouzeau. A Toolkit for Weaving Aspect Oriented UML Designs. In *Proc. of the 1st Int. Conference on Aspect-oriented Software Development (AOSD'02), Enschede, The Netherlands*, 2002.
28. Andrew Jackson and Siobhán Clarke. Towards the Integration of Theme/UML and JPDDs. In *Proc. of the 8th Int. Workshop on Aspect-Oriented Modeling at AOSD'06, Bonn, Germany*, March 2006.
29. Andrew Jackson, Jacques Klein, Benoit Baudry, and Siobhán Clarke. KerTheme: Testing Aspect Oriented Models. In *Proc. Workshop on Integration of Model Driven Development and Model Driven Testing (ECMDA'06), Bilbao, Spain*, July 2006.
30. Ivar Jacobson and Pan-Wei Ng. *Aspect-Oriented Software Development with Use Cases*. Addison-Wesley, 2005.
31. Mika Katara and Shmuel Katz. A Concern Architecture View for Aspect-Oriented Software Design. *Software and System Modeling*, 2006.
32. Mik Kersten. AOP Tools Comparison (Part 1 & 2). http://www-128.ibm.com/developerworks/java/library/j-aopwork1/, March 2005.
33. Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In *Proc. of the 11th Europeen Conference on Object-Oriented Programming, Jyväskylä, Finland*, 1997.

34. Jörg Kienzle and Samuel Gélineau. AO Challenge - Implementing the ACID Properties for Transactional Objects. In *Proc. of the 5th Int. Conference on Aspect-Oriented Software Development (AOSD'06), Bonn, Germany*, March.
35. Jörg Kienzle, Yang Yu, and Jie Xiong. On Composition and Reuse of Aspects. In *Proc. of FOAL: Foundations of Aspect-Oriented Languages, Boston, Massachusetts*, March 2003.
36. Dae-Kyoo Kim, Robert B. France, and Sudipto Ghosh. A UML-based Language for Specifying Domain-Specific Patterns. *Journal of Visual Languages and Computing*, 15(3-4):265–289, 2004.
37. Jacques Klein, Loïc Hélouët, and Jean-Marc Jézéquel. Semantic-Based Weaving of Scenarios. In *Proc. of the 5th Int. Conference on Aspect-Oriented Software Development (AOSD'06), Bonn, Germany*, March 2006.
38. Sergei Kojarski and David H. Lorenz. Modeling Aspect Mechanisms: A Top-Down Approach. In *Proc. of the 28th Int. Conference on Software Engineering (ICSE'06), Shanghai, China*, May 2006.
39. Karl J. Lieberherr. *Adaptive Object-Oriented Software: the Demeter Method with Propagation Patterns*. PWS Publishing Company, Boston, 1996.
40. Mark Mahoney, Atef Bader, Omar Aldawud, and Tzilla Elrad. Using Aspects to Abstract and Modularize Statecharts. In *Proc. of the 5th Aspect-Oriented Modeling Workshop (UML'04), Lisbon, Portugal*, October 2004.
41. Hidehiko Masuhara and Gregor Kiczales. Modeling Crosscutting in Aspect-Oriented Mechanisms. In *Proc. of the 17th European Conference on Object-Oriented Programming (ECOOP'03), Darmstadt, Germany*, July 2003.
42. Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. Weaving Executability into Object-Oriented Meta-languages. In *Proc. of the 8th Int. Conference on Model Driven Engineering Languages and Systems (MoDELS'05), Montego Bay, Jamaica*, 2005.
43. Object Management Group (OMG). MDA Guide Version 1.0.1. http://www.omg.org/docs/omg/03-06-01.pdf, June 2003.
44. Object Management Group (OMG). UML Specification: Superstructure Version 2.0. http://www.omg.org/docs/formal/05-07-04.pdf, August 2005.
45. David L. Parnas. On the Criteria To Be Used in Decomposing Systems into Modules. *Comm. ACM*, 15(12):1053–1058, December 1972.
46. Renaud Pawlak, Laurence Duchien, Gerard Florin, Fabrice Legond-Aubry, Lionel Seinturier, and Laurent Martelli. A UML Notation for Aspect-Oriented Software Design. In *Proc. of the 1st Workshop on Aspect-Oriented Modeling with UML (AOSD'02), Enschede, The Netherlands*, March 2002.
47. Renaud Pawlak, Lionel Seinturier, Laurence Duchien, Laurent Martelli, Fabrice Legond-Aubry, and Gérard Florin. Aspect-Oriented Software Development with Java Aspect Components. In R.E. Filman, T. Elrad, S. Clarke, and M. Akşit, editors, *Aspect-Oriented Software Development*, pages 343–369. Addison-Wesley, Boston, 2005.
48. Eduardo Kessler Piveta and Luiz Carlos Zancanella. Observer Pattern using Aspect-Oriented Programming. In *Proc. of the 3rd Latin American Conference on Pattern Languages of Programming, Porto de Galinhas, PE, Brazil*, August 2003.
49. Raghu Reddy, Robert France, and Geri Georg. An Aspect Oriented Approach to Analyzing Dependability Features. In *Proc. of the 6th Int. Workshop on Aspect-Oriented Modeling (AOSD'05), Chicago, Illinois*, March 2005.

50. Raghu Reddy, Sudipto Ghosh, Robert B. Rance, Greg Straw, James M. Bieman, Eunjee Song, and Geri Georg. Directives for Composing Aspect-Oriented Design Class Models. In *Transactions on Aspect-Oriented Software Development I*, LNCS 3880, pages 75 – 105. Springer-Verlag, 2006.
51. Raghu Reddy, Arnor Solberg, Robert France, and Sudipto Ghosh. Composing Sequence Models using Tags. In *Proc. of the 9th Int. Workshop on Aspect-Oriented Modeling at MoDELS'06, Genova, Italy*, October 2006.
52. Antonia M. Reina, Jesus Torres, and Miguel Toro. Separating Concerns by Means of UML-profiles and Metamodels in PIMs. In *Proc. of the 5th Aspect-Oriented Modeling Workshop (UML'04), Lisbon, Portugal*, October 2004.
53. James Rumbaugh, Ivar Jacobson, and Grady Booch, editors. *The Unified Modeling Language Reference Guide*. Addison-Wesley, Boston, 2005.
54. Frans Sanen, Eddy Truyen, Bart De Win, Wouter Joosen, Neil Loughran, Geoff Coulson, Awais Rashid, Andronikos Nedos, Andrew Jackson, and Siobhán Clarke. Study on interaction issues. Technical Report D44 AOSD-Europe-KUL-7, AOSD-Europe, February 2006.
55. Andrea Schauerhuber, Wieland Schwinger, Elisabeth Kapsammer, Werner Retschitzegger, and Manuel Wimmer. Towards a Common Reference Architecture for Aspect-Oriented Modeling. In *Proc. of the 8th International Workshop on Aspect-Oriented Modeling at AOSD'06, Bonn, Germany*, March 2006.
56. Douglas C. Schmidt. Guest editor's introduction: Model-driven engineering. *IEEE Computer*, 39(2):25–31, 2006.
57. ITU Telecommunication Standardization Sector. ITU-T Recommendation Z.120: Message Sequence Chart (MSC), Geneva, Switzerland. http://www.itu.int/rec/T-REC-Z/en.
58. ITU Telecommunication Standardization Sector. ITU-T Recommendation Z.100: Specification and Description Language (SDL), Geneva, Switzerland. http://www.itu.int/rec/T-REC-Z/en, August 2002.
59. Arnor Solberg, Devon Simmonds, Raghu Reddy, Sudipto Ghosh, and Robert B. France. Using Aspect Oriented Techniques to Support Separation of Concerns in Model Driven Development. In *Proc. of the 29th Annual Int. Computer Software and Applications Conference (COMPSAC 2005), Edinburgh, Scotland, UK*, July 2005.
60. Dominik Stein, Stefan Hanenberg, and Rainer Unland. An UML-based Aspect-Oriented Design Notation. In *Proc. of the 1st Int. Conference on Aspect-Oriented Software Development (AOSD'02), Enschede, The Netherlands*, April 2002.
61. Dominik Stein, Stefan Hanenberg, and Rainer Unland. Designing Aspect-Oriented Crosscutting in UML. In *Proc. of the 1st Workshop on Aspect-Oriented Modeling with UML (AOSD'02), Enschede, The Netherlands*, March 2002.
62. Dominik Stein, Stefan Hanenberg, and Rainer Unland. On Representing Join Points in the UML. In *Proc. of the 2nd Int. Workshop on Aspect-Oriented Modeling with UML (UML'02)*, September 2002.
63. Dominik Stein, Stefan Hanenberg, and Rainer Unland. Expressing Different Conceptual Models of Join Point Selections in Aspect-Oriented Design. In *Proc. of the 5th Int. Conference on Aspect-Oriented Software Development (AOSD'06), Bonn, Germany*, March 2006.
64. Dominik Stein, Jörg Kienzle, and Mohamed Kandé. Report of the 5th Int. Workshop on Aspect-Oriented Modeling. In *UML Modeling Languages and Applications: 2004 Satellite Activities, Lisbon, Portugal*, pages 13–22. Springer-Verlag, October 2004.

65. Stanley M. Sutton, Jr. and Isabelle Rouvellou. Concern Modeling for Aspect-Oriented Software Development. In R.E. Filman, T. Elrad, S. Clarke, and M. Akşit, editors, *Aspect-Oriented Software Development*, pages 479–505. Addison-Wesley, Boston, 2005.
66. Davy Suvée, Wim Vanderperren, Dennis Wagelaar, and Viviane Jonckers. There are no Aspects. *Electronic Notes in Theoretical Computer Science*, 114, 2005.
67. Peri L. Tarr, Harold L. Ossher, William H. Harrison, and Stanley M. Sutton, Jr. N Degrees of Separation: Multi-Dimensional Separation of Concerns. In *Proc. of the 21st Int. Conference on Software Engineering (ICSE'99), Los Angeles, California*, May 1999.
68. The AspectJ Team. The AspectJ (TM) Programming Guide. http://eclipse.org/aspectj/doc/released/progguide/index.html, October 2005.
69. Klaas van den Berg, José M. Conejero, and Ruzanna Chitchyan. AOSD Ontology 1.0 - Public Ontology of Aspect-Orientation. Technical Report D9 AOSD-Europe-UT-01, AOSD-Europe, May 2005.
70. Christina von Flach Garcia Chavez. *A Model-Driven Approach for Aspect-Oriented Design*. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro, April 2004.
71. Christina von Flach Garcia Chavez and Carlos J. P. de Lucena. A Theory of Aspects for Aspect-Oriented Software Development. In *Proc. of the 7th Brazilian Symposium on Software Engineering (SBES'2003)*, 2003.
72. Jing Zhang, Thomas Cottenier, Aswin Van Den Berg, and Jeff Gray. Aspect Interference and Composition in the Motorola Aspect-Oriented Modeling Weavr. In *Proc. of the 9th Int. Workshop on Aspect-Oriented Modeling (MODELS'06), Genova, Italy*, October 2006.