A Survey on Aspect-Oriented Modeling Approaches

A. Schauerhuber[1]*, W. Schwinger[2],
E. Kapsammer[3], W. Retschitzegger[3], M. Wimmer[4], and G. Kappel[4]

[1] WIT Women's Postgraduate College for Internet Technologies,
Vienna University of Technology,
Favoritenstrasse 9-11/E188-4, A-1040 Vienna, Austria
schauerhuber@wit.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
[4] Business Informatics Group
Vienna University of Technology,
Favoritenstrasse 9-11/E188-4, A-1040 Vienna, Austria
{wimmer,gerti}@big.tuwien.ac.at

Abstract. Aspect-orientation provides a new way of modularization by clearly separating crosscutting concerns from non-crosscutting ones. Although originally emerged at the programming level, aspect-orientation meanwhile stretches also over other development phases. Not only due to the rise of model-driven engineering, approaches already exist for dealing with aspect-orientation at the modeling level. Nevertheless, concepts from the programming level are often simply reused without proper adaptation. Consequently, such approaches fall short in considering the full spectrum of modeling concepts not present in programming languages, like, e.g., different views on the application's structure and behavior.
In this paper we present a survey on existing aspect-oriented modeling approaches. In doing so, we first discuss a common reference architecture for aspect-oriented modeling and thus, take a step towards a consolidated and more comprehensive view on aspect-orientation. Second, we set up a framework of evaluation criteria directly derived from the common reference architecture and thus, allowing for a structured evaluation of approaches. And third, we provide a comparison of aspect-oriented modeling approaches by means of a running example making the approaches' strengths and shortcomings more explicit and report on lessons learned.

* 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 idea of *Separation of Concerns* (SoC), i.e., the identification of different *concerns* in software development and their *separation* by encapsulating them in appropriate modules or parts of the software, can be traced back to Dijkstra [19] and Parnas [51]. Aspect-oriented software development (AOSD) 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. AOSD, formerly also called Advanced Separation of Concerns (ASoC), is a fairly young but rapidly advancing research field. In particular, AOSD represents the convergence of different ASoC approaches, such as Adaptive Programming (AP) [43], Composition Filters (CF) [1], Subject-Oriented Programming (SOP) [32], Multi-Dimensional Separation of Concerns (MDSoC) [50], [49], and Aspect-Oriented Programming (AOP) [40].

Aspect-Oriented Modeling. From a software development point of view, aspect-orientation originally emerged at the programming level with AspectJ[5] as one of the most prominent protagonists. Not only due to the rise of model-driven engineering (MDE) [7], however, 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 development life cycle such as *requirements engineering*, *analysis*, and *design*. According to Pressman [55], *modeling* is the progressively detailed representation of software and encompasses parts of the *requirements engineering* phase as well as the *analysis* and *design* phase. Likewise in the field of aspect-orientation, aspect-oriented modeling (AOM) encompasses modeling activities from *requirements engineering* via *analysis* to *design*. Still, we observe that there are differences in the understanding of how the prevailing categorization of AOM approaches into *requirements*, *architecture*, and *design* approaches is mapped on the software development phases of *requirements engineering*, *analysis* and *design* [55]: In [9] *requirements* and *architecture* approaches are mapped onto the *analysis* phase[6] (e.g., [4], [36], [44]), and AO *design* approaches (e.g., [14], [20], [24], [38], [53], [58], [66]) are mapped onto the *design* phase, whereas in [55] *architecture design* is described as an early task in the *design* phase. Admittedly, the boundaries between software development phases are continuous. And thus, while *architecture* approaches certainly are found in between *requirements* and *design* approaches, some might be more aligned with the earlier software development phase and some with the later software development phase. In the context of this work, we focus on aspect-oriented modeling languages at the *design* level, only, and in the following, use aspect-oriented modeling as a synonym for those *design* level approaches.

State-of-the-Art in Aspect-Oriented Modeling. In the field of aspect-oriented modeling, developers are already facing an immense amount of AOM approaches each of them having different origins and pursuing different goals

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

[6] Please note that requirements engineering and architecture design is also the focus of the *Early Aspects Initiative* (www.early-aspects.net).

dealing with the unique characteristics of aspect-orientation. This entails not only the problem of different terminologies but also leads to a broad variety of aspect-oriented concepts. In several cases, concepts of aspect-oriented programming languages are simply incorporated unaltered into a modeling language failing to consider the different levels of abstraction. Applying aspect-orientation at the modeling level, however, is not just injecting code at a certain point within a program [70] but requires the consideration of the full spectrum of modeling concepts not present in programming languages, like, e.g., different views on the application's structure and behavior as provided by current modeling languages like UML [47].

Contribution. This paper's contribution is threefold.

1. Based on the considerations outlined above and our initial work in [63], an initial step towards a *common reference architecture* (cf. Section 3) is taken to identify the basic ingredients of aspect orientation, abstracted from certain AOP languages and AOM approaches. Since a common understanding of aspect-oriented concepts has not yet been established, our reference architecture on the one hand proposes such a common understanding and on the other hand summarizes the issues that have to be evaluated in a survey on aspect-oriented modeling.
2. A *framework of evaluation criteria* (cf. Section 4) for a structured evaluation of AOM approaches is proposed which is derived from the common reference architecture. Furthermore, we provide additional criteria that are more general and go beyond the concepts presented in our reference architecture, e.g., *maturity* and *tool support*.
3. A *comparison of existing AOM approaches* (cf. Section 5) is provided on the basis of a running example, i.e., the approaches' strengths and shortcomings are identified, illustrated using a *running example*, and summarized on a per-approach basis.

This methodology, i.e., deriving a criteria catalogue from the previously defined reference architecture and a following evaluation of AOM approaches according to those criteria, represents, in fact, an indirect mapping of the concepts identified within our reference architecture onto several AOM approaches and thus, provides for an indirect evaluation of the reference architecture itself.

The remainder of this paper is organized as follows. We motivate our work in Section 2 by reporting on related surveys. Our common reference architecture for aspect-oriented modeling and the framework of evaluation criteria are presented in Section 3 and Section 4, respectively. The results of evaluating eight aspect-oriented modeling languages are provided in Section 5. Finally, we summarize our overall findings and report on lessons learned in Section 6 and provide an outlook on future work in Section 7.

2 Related Work

In an effort to shed light on the different approaches to aspect-orientation, some surveys comparing aspect-oriented approaches at different levels in the software

development life cycle have already been published. In the following, we distinguish between *closely related* surveys particularly emphasizing on AOM, more *widely related* ones focusing on AOP, and work aiming at *unifying* the currently prevailing inconsistencies of concepts in the aspect-orientation paradigm. In particular, the latter either focus on the *conceptual level* by elaborating on a common understanding of aspect-orientation in terms of a definition of aspect-oriented concepts and reference architectures or aim at the *language level* by integrating best practises of previous AOM approaches in the definition of new AOM languages.

An extensive survey done by *Chitchyan et al.* [9], including aspect-oriented approaches for requirements engineering, analysis, and design phases, presents the evaluation results of 22 aspect-oriented design proposals. Based on this evaluation, an initial proposal for an integrated aspect-oriented analysis and design process is outlined. Although a set of criteria has been identified, a precise definition of some of the criteria used to evaluate the approaches is missing.
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.* [60], *Blair et al.* [8], and *Op de beeck et al.* [18]. While *Reina et al.* compare different AOM approaches with respect to four criteria, *Blair et al.* provide in separate sets of criteria for the phases of aspect-oriented requirements engineering, specification, and design. The major goal in *Op de beeck et al.* is to investigate 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 engineering process. In this respect, the authors have refined a set of six criteria, which partly have been presented in Chitchyan et al. [9], and additionally provide a discussion of the criteria's impact on certain software quality factors (e.g., understandability, evolvability, reusability, scalability, maintainability, and traceability).
With respect to these closely related surveys our work is different in four ways. Firstly, we have put great emphasis on the criteria's definition, whereas, e.g., in [9] not all criteria have been elaborated on in detail. Secondly, although we partly adopt and refine criteria found in those surveys, our evaluation framework comprises a different set of criteria with respect to AOM. It is derived from our common reference architecture and thus, subsumes all aspect-oriented concepts that have to be evaluated in a survey on aspect-oriented modeling. Third, our survey includes also recently published approaches such as [16], [48] not included in the other surveys. Finally, we provide a running example that is realized with each surveyed modeling approach. This further supports our 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.

Less closely related, since focusing on AOP, is the survey of *Hanenberg et al.* [31] which presents a set of criteria used to evaluate four AOP languages. *Mik Kersten* [39] also provides a comparison of four leading AOP languages having only AspectJ in common with *Hanenberg et al.* In addition, *Mik Kersten* also investigates the development environments of these AOP languages.

The evaluation criteria defined in those surveys are only partly applicable in our context, since they are defined at the programming level. Nevertheless, we have adopted their criteria where appropriate and refined them such that they can be applied at the modeling level, too.
In *Chavez et al.* [76], a conceptual framework for AOP has been proposed in terms of Entity-Relationship diagrams. Based on this conceptual framework an evaluation of four programming level approaches, namely AspectJ, Hyper/J[7], Composition Filters, and Demeter/DJ [43] is presented.
Since the definitions in the conceptual framework have been defined at the programming level they are only partly applicable in our reference architecture because ours is defined at a higher level of abstraction.

Considering the last category of related work aiming at unification in the aspect-orientation paradigm, in *van den Berg et al.* [75], an attempt towards establishing a common set of concepts for AOSD has been made, based on previous work by *Filman et al.* [22]. In particular, the concepts of two AOP languages, namely AspectJ and ComposeStar[8] have been examined and expressed in terms of separated UML class diagrams. Based on these results the initial textual definitions of concepts have been revised.
In contrast to that, our *common reference architecture for aspect-oriented modeling* [63] is complementary to the AOSD ontology of *van den Berg et al.*, in that it takes up a broader view on aspect-orientation by proposing a conceptual model of AOM concepts in terms of a UML class diagram. It is different, however, in that it specifically focuses on AOM concepts, while the AOSD ontology's textual definitions describe more generic AOSD concepts and are intended to be appropriate for all phases in the software development life cycle. The reference architecture identifies the important AOM concepts, their interrelationships and even more importantly their relationships to an arbitrary modeling language (e.g., general purpose modeling languages such as UML, or any other domain-specific modeling language). It therefore can be used as a blueprint for designing new AOM languages or for extending existing (domain-specific) modeling languages with concepts of the aspect-oriented paradigm.
Recently, *Fuentes et al.* [25] and *Krechetov et al.* [42] each have proposed a new AOM approach, both aiming at unifying the inconsistent notions of aspect-oriented concepts present in earlier AOM languages. The generic, MOF-based aspect-oriented design metamodel of *Fuentes et al.* is aimed at enabling model transformation between AOM approaches. The authors go for a least common denominator approach in identifying the important aspect-oriented concepts but allow the metamodel's extension through package merge in order to support additional aspect-oriented concepts provided in different AOM proposals. *Krechetov et al.* integrate best practices from four existing aspect-oriented architectural modeling languages focusing on a common notation for an integrated aspect-oriented architectural modeling approach.
Our reference architecture for AOM is different to both, the generic metamodel

[7] http://www.research.ibm.com/hyperspace/HyperJ/HyperJ.htm
[8] http://janus.cs.utwente.nl/twiki/bin/view/Composer/WebHome

for aspect-oriented design and the integrated aspect-oriented architectural modeling language, in that it does not primarily represent a language specification. Instead, our intention was to first identify the concepts that are necessary in specifying a new AOM language, i.e., starting from these basic ingredients of aspect-oriented modeling the design of a new AOM language is straight-forward.

Finally, by providing a running example, our survey is conducted at a detailed level and enhances our evaluation in that it first, provides an insight into each approach and second, allows to easier compare the modeling means of the approaches.

3 Basic Ingredients of Aspect-Oriented Modeling

Applying aspect-oriented concepts originally coined for the programming level (e.g., by AspectJ) to the modeling level turns out to be a challenging task. This is on the one hand due to the very specific meaning of programming level aspect-oriented concepts [70] and on the other hand due to different concepts introduced by related AOSD approaches (e.g. AP, CF, SOP, and MDSoC). An example for the first issue are AspectJ's *join points* which are defined as points in the execution of the program including field accesses, method, and constructor calls [73][9]. This definition is too restrictive for the modeling level, however, since runtime is not the only focus of modeling. An example for the second issue is the concept of *aspect* in AOP, where similar though different concepts have been introduced in other approaches, e.g., *hyperslice* in Hyper/J, *filter* in CF, and *adaptive method* in Demeter/DJ [76]. Consequently, instead of sticking with AOP concepts, it is rather advisable to find general definitions of aspect-oriented concepts that apply to any level in the software development life cycle.

In order to support the process of establishing a common terminology, we have proposed an initial version of a *reference architecture for aspect-oriented modeling* in previous work [63], which serves herein as a basis. Since the term reference architecture is used quite differently in literature [21] depending on various factors, e.g., domain and goals, we adhere to the following definition:

> A reference architecture is a *conceptual model* identifying and describing *the concepts of a domain* and the *interrelationships* thereof.

Thus in the sense of the categorization of reference architectures in [21], our AOM reference architecture represents a set of *general assertions* and *normative assertions*, which are true for AOM languages, as well as a *technique* for designing new AOM languages or for extending existing (domain-specific) modeling languages with concepts of the aspect-oriented paradigm.

[9] Admittedly, AspectJ also allows the introduction of adaptations with respect to the program's structure, but join points are defined with respect to runtime, only.

For the concepts used in our reference architecture we primarily adopt the definitions presented in [75] but refine them to be suitable for the modeling level. Additionally, based on the surveyed approaches we extend the definitions to provide a broad base of conceptualization of aspect-orientation. In Figure 1 our reference architecture is shown as a UML class diagram, which comprises the concepts of aspect-orientation at a higher level of abstraction. Thus, it represents an initial proposal for a *conceptual model for aspect-oriented modeling* in the sense it is asked for in [75].

In the following, the concepts of the reference architecture are described along with its major building blocks.

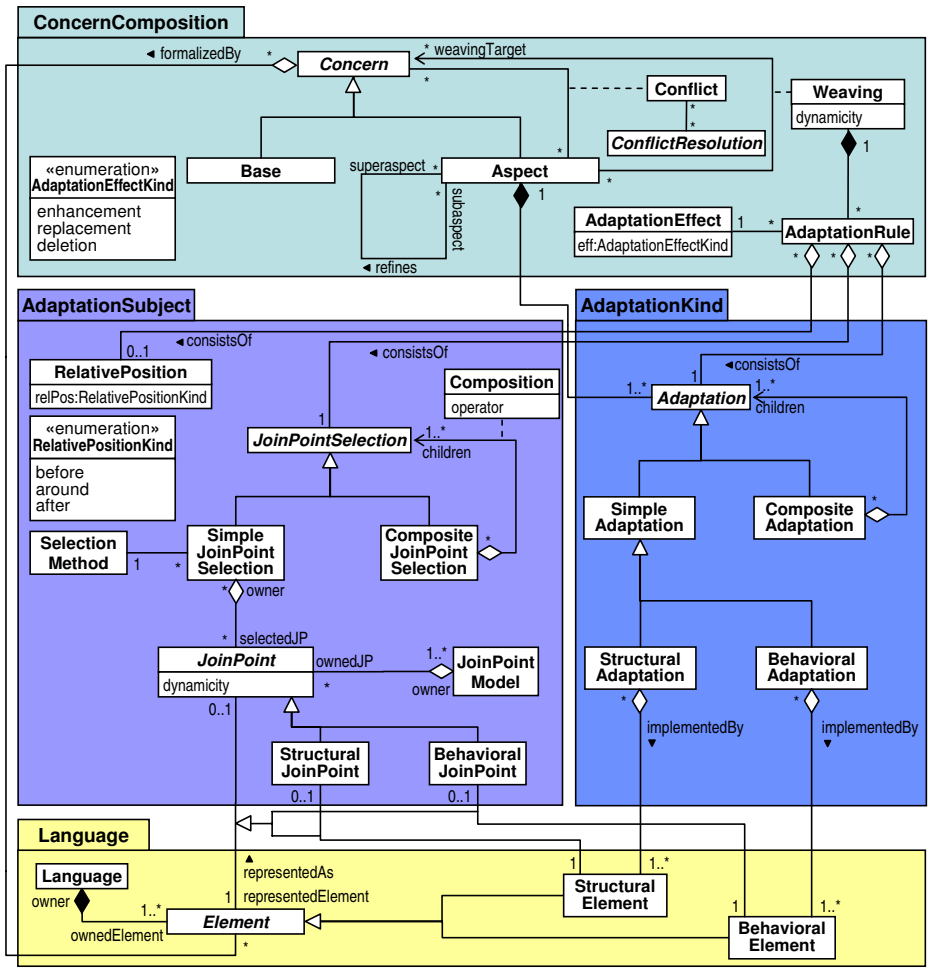

Fig. 1. The AOM Reference Architecture

3.1 ConcernComposition

Concern composition deals first, with the separation of a system's concerns into appropriate units of modularization and second, with their interrelationships, i.e., their composition by means of a weaving specification.

Concern. Along with [75] we define a *concern* 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 in this respect represents an inclusive term for *aspect* and *base*, which is depicted using generalization in Figure 1. We refrain from referring to *crosscutting* and *non-crosscutting* in our reference architecture since they represent interests with respect to a system at the level of requirements rather than the modeling level. Aspect and base, however, form a representation of concerns in a more formalized language (e.g., a modeling language or a programming language). A distinction between aspect and base concerns means supporting the *asymmetric* approach to composition [33]. An aspect might, at the same time, act as a base, i.e., as a *weavingTarget*, for other aspects and by this allow for both *symmetric* and *asymmetric* approaches to composition[10].

Base. A *base* is a unit of modularization formalizing a non-crosscutting concern. This goes in line with most programming and modeling paradigms, where the provided units of modularization allow for decomposing a system according to one dimension, only, called *dominant decomposition* [72]. The object-oriented paradigm for example provides hierarchically ordered units of modularization (i.e., classes and methods) in terms of a vertical decomposition. Thus, it does not support horizontal decomposition, i.e., crosscutting concerns, that are typically scattered across the dominant decomposition.

Aspect. An *aspect* is a unit of modularization formalizing a crosscutting concern, i.e., a set of adaptations (cf. *Adaptation*), that otherwise would be scattered across other concerns. Aspects are related to other aspects in three ways. First, aspects themselves may be acting as base for other aspects (cf. *weavingTarget* in Figure 1), i.e., an aspect may adapt another aspect. A certain aspect, however, can not be its own *weavingTarget*, which is expressed by the following OCL constraint[11]:

```
context Aspect inv:
self.weavingTarget->forAll(a:Aspect | a <> self)
```

Second, an aspect might be specialized into several sub-aspects, thus refining[12] *where* and *how* concerns might be adapted. Third, two or more aspects

[10] Please note that, the *symmetric* approach is implicitly represented in our reference architecture with the *aspect-concern relationship*, only. This is done by disallowing *base* concerns in this specific case and ignoring the weaving direction, i.e., the concerns take equal parts in the composition and none acts as the *weavingTarget*.

[11] For readability reasons we omit the specification of package names in OCL constraints.

[12] A discussion on possible refinement policies (e.g., with respect to adaptation rules, join point selections, and adaptations) is subject to future work.

might introduce adaptations to a concern in a way that causes conflicts (cf. *Conflict* in Figure 1), i.e., contradicting adaptations with respect to the same element in the model. Thus, for such aspects a conflict resolution (cf. *ConflictResolution*) may be specified defining the precedence of one aspect over another.

Weaving. In AOSD the composition of aspects with other concerns, which in turn are either bases or aspects, is called *weaving*. Weaving yields a *woven model* of the overall system, i.e., models where aspects have been woven into the base and therefore ceased to exist separately. For our purposes, we distinguish between two ways of weaving aspects into other concerns, namely *static* (i.e., at design time) and *dynamic* (i.e., at runtime). Thereby, one aspect of a system may be statically composed with other concerns, whereas another aspect may be dynamically woven, which is taken into account by an attribute (cf. *dynamicity*) of the respective association class *Weaving*. This design decision has been motivated by weaving concepts in AOP. At modeling level, it still can be argued that being able to distinguish between static and dynamic weaving of bases and aspects is advantageous for two reasons. First, if the runtime semantics of the language's metamodel has been specified, i.e., models are executable (which, considering, e.g., UML, is the case for parts of the language like state machines, only), dynamic weaving may happen while executing the models, similarly to the way it happens at code level [24]. Second, this distinction allows specifying - at the modeling level - what aspects need to be statically or dynamically woven into the base program during later stages of the development process. The weaving relationship is navigable from the aspect's side, only, meaning that the concern is *oblivious* [23] to possible adaptations by aspects[13].

AdaptationRule. *Adaptation rules* are part of a particular weaving and introduce an aspect's adaptations at certain points of other concerns, i.e., they specify *where* to adapt *how*. Consequently, an adaptation rule consists of an *adaptation* describing how to adapt the concern, as well as a *join point selection* and an optional *relative position* describing where to adapt the concern. We modeled the *consists-of* relationships using weak aggregations, since adaptation, join point selection, and relative position might be reused in other adaptation rules. The adaptations used in the adaptation rules of a certain weaving specification of an aspect and a concern have to be defined within the aspect taking part in the specific weaving. This constraint is expressed by the following OCL expression:

```
context Weaving inv:
self.aspect.adaptation->exists(
    a : Adaptation | a = self.adaptationRule.adaptation)
```

Furthermore, the *join points* selected within an adaptation rule of a certain weaving specification of an aspect and a concern have to be elements

[13] Please note that, for *symmetric* approaches to composition this weaving direction is obsolete, since both concerns represent equal participants in the composition.

formalizing the concern taking part in the specific weaving. Note that, the *getAllSimpleJPSelections()* operation collects all simple join point selections that are part of a join point selection.

```
context JoinPointSelection def:
getAllSimpleJPSelections():Bag(SimpleJoinPointSelection)=
  if self.oclIsTypeOf(SimpleJoinPointSelection)
  then Bag{self}
  else self.children->iterate(
    jps:JoinPointSelection;
    allSimpleJPSs:Bag(SimpleJoinPointSelection) = Bag{} |
    if jps.oclIsTypeOf(SimpleJoinPointSelection)
    then allSimpleJPSs.including(jps)
    else allSimpleJPSs.union(jps.getAllSimpleJPSelections())
    endif)
  endif

context Weaving inv:
self.adaptationRule.joinPointSelection.
    getallSimpleJPSelections().selectedJP->forAll(
        jp : JoinPoint |self.weavingTarget.element->exists(
            e : Element | e = jp.representedElement))
```

In AOP, the specification where to adapt how (such as the *pointcut-advice* combination in AspectJ) were specified in an intermingled way. For reusability reasons some AOM approaches [17], [28] provide an adaptation rule specification that is independent from both, base and aspect. In [17], the authors distinguish between modeling the aspect's adaptations and modeling adaptation rules by proposing a *connector metamodel* for aspect-oriented composition. Furthermore, in [28] independence of *linking technology* (e.g., AspectJ) is achieved by introducing the *connector* concept to link aspect and base concerns. Along with those approaches, we clearly separate the adaptation rule from both, base and aspect, for reasons of enhanced variability, reusability, and expressiveness.

AdaptationEffect. The *adaptation effect* specified with the adaptation rule describes what effect an aspect has on the base. One and the same adaptation may have an *enhancement* effect, a *replacement* effect, or a *deletion* effect (cf. *AdaptationEffectKind* in Figure 1) depending on the join point selection and its relative position when used in the adaptation rule. This distinction resembles a differentiation proposed in [31] in terms of constructive (cf. enhancement), and destructive (cf. replacement and deletion) adaptation effects. There exists an inherent relationship between a join point selection together with its relative position and an adaptation with respect to the adaptation effect an aspect has on the base. For example, the relative positions *before*, and *after* lead to an enhancement, whereas in case of *around* the adaptation may resemble an enhancement, a replacement (i.e., deleting the join point with the adaptation), or a deletion (i.e., deleting the join point

with an empty adaptation). This interdependency is assured by the following OCL constraint:

```
context AdaptationRule inv:
if(self.relativePosition.relPos = RelativePositionKind::before
OR self.relativePosition.relPos = RelativePositionKind::after)
then
self.adaptationEffect.eff = AdaptationEffectKind::enhancement
endif
```

Conflict. Concerns might be composed in a way that causes conflicts. According to Blair et al. [8] *aspect interaction* occurs if the behavior of one aspect is affected by the behavior of another aspect. In [41], Kienzle et al. present a classification of aspects[14] with respect to their dependency relations, distinguishing between *orthogonal* aspects which are independent from other aspects, *uni-directional preserving* aspects based on adaptations provided by other aspects without modifying the latter, and *uni-directional modifying* aspects that change services provided by other aspects (cf. definition by [8]). Thus, aspects' definitions may contradict with the specification of other non-orthogonal, or rather uni-directional modifying aspects. Additionally, aspects may also contradict with the specification of base models (e.g., introducing features that already exist in the base model) and base models themselves might contradict each other, e.g., due to multiple inheritance, which is an already known problem in the modeling field in general. Consequently, a conflict resolution mechanism (cf. *ConflictResolution*) should be provided by an approach allowing to specify the application of aspects - and allowing composition of concerns in general - in an unambiguous way.

ConflictResolution. Which kind of *conflict resolution* is applicable for a certain conflict depends on the particular domain. This fact is represented in the reference architecture by the abstract class *ConflictResolution*, which - in form of a *Strategy pattern* [26] - can embrace any concrete conflict resolution, (e.g., relative or absolute ordering) that might be applicable.

3.2 Language

The following concepts describe the *language* underlying the specification of base and aspect.

Language. Depending on the current focus in the software development life cycle, the *language* might represent for example a modeling language or a programming language. Currently, the reference architecture focuses on a single language, i.e., the same language is used, first for specifying base concerns, second for specifying the adaptation and third for specifying adaptation rules. Still, an interesting direction for future work is the consideration

[14] There have been several attempts to classify aspects according to different criteria, amongst them [15], [31], and [77].

of multiple languages. In this respect, drawing from the benefits of different domain specific languages (DSL) would by possible. This raises, however, the question to which extent these languages may be different and how much they must have in common to still allow for aspect weaving. Considering again the case of UML, it has to be investigated, if it is preferable to base these languages on the same meta-metamodel, i.e. MOF [46], or if it is beneficial to bridge the heterogeneity between the bases' and aspects' languages by means of a weaving model.

Element. Concerns are formalized using *elements* of a certain language. With respect to aspect-orientation, elements serve two purposes. First, they may represent join points and thus in the role of join points specify *where* to introduce adaptations. Second, elements of a language are used for formulating an adaptation. Such elements are either *structural elements* or *behavioral elements* as depicted in Figure 1 using generalization.

StructuralElement. *Structural elements* of a language are used to specify a system's structure.

BehavioralElement. Likewise to structural elements, *behavioral elements* of a language are used to specify a system's behavior.

3.3 Adaptation Subject

The *adaptation subject* describes the concepts required for identifying *where* to introduce an aspect's adaptations.

JoinPoint. A *join point* specifies *where* an aspect might insert adaptations. Thus, a join point is a representation of an identifiable element of the underlying language used to capture a concern. Join points can be distinguished along two orthogonal dimensions. In Hanenberg et al. [31], join points of aspect-oriented programming languages are categorized according to two dimensions[15], *feature*[16] and *dynamicity*. In contrast to that, our focus is broader in that we consider modeling level join points and in that we consider these two dimensions as being orthogonal also at the modeling level. Consequently, join points are representations of *structural elements* (cf. *StructuralJoinPoint*) or *behavioral elements* (cf. *BehavioralJoinPoint*) of a language, while *at the same time*, they are also modeling level representations of *static* or *dynamic elements* (cf. attribute *dynamicity*) in a software system. In this respect, the reference architecture supports four different kinds of join points.

StructuralJoinPoint. *Structural join points* represent structural elements of a language where an aspect's adaptations can be introduced. In addition,

[15] In literature (amongst others [5], [22], [39], [76]), we find different interpretations of what a join point is. The focus is on describing the join points' properties such as *dynamicity* and *structural and behavioral* features, sometimes mixing up terms (e.g. using static as a synonym for structural).

[16] While Hanenberg et al. [31] use the term "abstraction", we adhere to UML terminology in that we distinguish between structural and behavioral *features* [47].

structural join points can be either *static* or *dynamic* (cf. *dynamicity* attribute). *Static join points* are elements of a language that can be identified based on information available at design time (e.g., method definition). *Dynamic join points* are elements of a language that can not be identified before runtime (e.g., object and method execution). 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. Admittedly, not all languages may offer elements which allow for dynamic join points as is the case with UML.

BehavioralJoinPoint. Analogous, *behavioral join points* represent behavioral elements of a language where an aspect's adaptations can be introduced. Additionally, we distinguish between *behavioral-static join points* (e.g., activities) and *behavioral-dynamic join points* (e.g., method executions).

JoinPointModel. The *join point model* comprises as possible join points all elements of a certain language where aspects are allowed to introduce adaptations, i.e., where the *representedAs* association connects the element with *JoinPoint*.

For one join point model thereby, the associated join points may represent elements of one language, only, which is expressed by the following OCL constraint:

```
context JoinPointModel inv:
self.ownedJP->forAll(jp1, jp2 : JoinPoint |
  jp1.representedElement.owner = jp2.representedElement.owner)
```

JoinPointSelection. A *join point selection*[17] represents a subset of the join point model, i.e., the join points selected for the purpose of specifying certain adaptations. The selection of join points can be done for example by means of a query on the join point model (cf. *SimpleJoinPointSelection* and *SelectionMethod*). A join point selection specification is implemented by either a *SimpleJoinPointSelection* or a *CompositeJoinPointSelection*. We refrain from associating join points directly to an adaptation rule but instead use join point selections as a level of indirection, and thus allow for reusing join points in other adaptation rules and other join point selections.

SimpleJoinPointSelection. A simple join point selection represents a set of join points of a certain kind (e.g., structural-static), which are selected according to a certain selection method (cf. *SelectionMethod*).

CompositeJoinPointSelection. For reuse purposes, join point selections can be composed of other join point selections by means of logical *Operators*, e.g., AND, OR, NOT. All children of a *composite join point selection* refer to the same join point model, which is assured by the following OCL constraint. In particular the constraint uses the getAllSimpleJPSelections() operation, which has been introduced before, to collect all children, i.e., simple join point selections, of the composite join point selection and ensures that they all refer to the same join point model.

[17] As in [31], we refrain from using the term pointcut, since it has been coined for AspectJ.

```
context JoinPointSelection inv:
self.getAllSimpleJPSelections().selectedJP->forAll(jp1, jp2 :
JoinPoint | jp1.owner = jp2.owner)
```

SelectionMethod. The *selection method* concept describes a method for selecting from the potential join points of the join point model those that should be available for adaptation. The selection method corresponds to what is termed a primitive pointcut designator in AspectJ.

RelativePosition. A *relative position* may provide further information as to *where* adaptations have to be introduced. This is necessary since in some cases, selecting join points by join point selections, only, is not enough to specify where adaptations have to be inserted, since an adaptation can be introduced for example *before* or *after* a certain join point. Still, in some other cases 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). In AspectJ, the relative position is specified with the *advice*, i.e., an adaptation. Instead of modeling the relative position with the adaptation, in our reference architecture it is modeled separately from adaptation. Since the relative position basically represents some kind of a location specification, it perfectly fits into the *AdaptationSubject* package. Furthermore, there is an inherent relationship between the relative position and the kind of selected join points. For dynamic join points, the relative position resembles a *temporal* specification, for example before an event occurs. A typical example is AspectJs *before* advice, which is an adaptation for dynamic join points, a technique called *wrapping* in [22]. For static join points the relative position resembles rather a *local* specification and is defined with respect to the element's *structure*. For example, if a link is added, its relative position in terms of the participating object is specified. Consequently, the nature of a relative position depends on the kind of element representing the join point.

3.4 Adaptation Kind

The *adaptation kind* comprises the concepts necessary to describe an aspect's adaptation.

Adaptation. An *adaptation* specifies in what way the concern's structure or behavior is adapted, i.e., enhanced, replaced or deleted (cf. *AdaptationEffect*). Historically, structural adaptations (cf. *StructuralAdaptation*) have been called *introduction*, while behavioral adaptations (cf. *BehavioralAdaptation*) have been termed *advice*. Recently, the advice concept is more and more used as an inclusive term for structural and behavioral adaptations [75]. Thus, our *adaptation* concept is similar to the commonly found definition of an advice which represents an artifact that augments or constraints concerns (cf. [75]). An adaptation specification is implemented by *structural adaptations*, *behavioral adaptations*, or both, i.e., by *composite adaptations*.

StructuralAdaptation. A *structural adaptation* comprises a language's structural elements for adapting concerns (e.g., adding a new attribute to a class's structure).

BehavioralAdaptation. Likewise, a *behavioral adaptation* comprises a language's behavioral elements for adapting concerns (e.g., adding a method call).

CompositeAdaptation. For reuse purposes, adaptations can be composed of a coherent set of both, structural and behavioral adaptations (cf. *Composite pattern* [26]). In this respect, the adaptation concept extends the general understanding of the advice concept described in [75].

4 Evaluation Framework

4.1 Methodology

In the following, we propose a criteria catalogue for the structured evaluation of aspect-oriented modeling languages. First, the criteria are the result of a *top-down* approach considering the reference architecture presented in Section 3. The reference architecture subsumes the concepts that have been identified as important for aspect-oriented modeling and thus, corresponding criteria in the evaluation framework operationalize the reference architecture. Our goal has been to provide criteria for each concept of the reference architecture. This implies that either a concept of the reference architecture maps onto one-to-many criteria in the evaluation framework or one-to-many concepts of the reference architecture map onto one criterion in the evaluation framework. A concept that is represented as an abstract class, however, does not need a corresponding criterion in the evaluation framework, since it is implicitly evaluated by its sub-concepts and their criteria.
Second, we provide additional criteria, e.g., criteria describing *maturity* and *tool support*, which do not necessarily have corresponding concepts in the reference architecture. Those criteria partly have been identified following a *bottom-up* approach, taking into consideration interesting issues from related surveys [8], [9], [18], [31], [39], [60], [76].
The overall emphasis of our evaluation framework focuses on *functional* criteria. The inclusion of non-functional criteria in terms of several "ilities" such as evolvability, scalability, traceability, reusability, understandability, maintainability, or flexibility is subject to future work. This is due to the fact that a serious evaluation according to these criteria would require testing the approaches in real-world projects with real users.

Concerning the selected set of criteria, it turned out that some of them still were difficult to evaluate, which was due to ambiguous definitions found in literature. In fact, for some prominent criteria such as *platform dependency* and *level of abstraction*, articulating good definitions is a highly challenging task, and we have not come across any that would allow the inference of proper metrics as to measure them. Consequently, we have tried to avoid blurred criteria by working out, if possible, un-ambiguous definitions and the criteria's values that are also

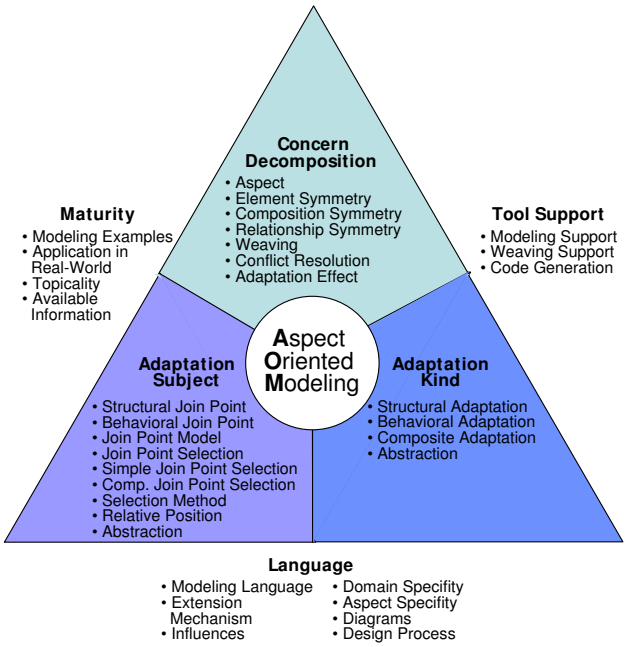

Fig. 2. Categorization of Criteria

measurable. Thus, each criterion is described by a set of properties: the *name* along with an *abbreviation* allowing to reference the criteria during evaluation of the approaches in Section 5, an optional *discussion* on difficulties in defining the criterion and a *definition* specifying the criterion as unambiguously as possible by providing appropriate *means of measurement*, such as a list of possible values or a measurement scale, including *unknown* as a default value for each criterion. 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., 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, such as [71], [78]. Therefore, in this work we consider UML-based approaches, only, which in turn influences some criteria and their possible values.

We have categorized the criteria of our evaluation framework into six groups (see Figure 2) with four out of them being specifically inferred from corresponding parts in the reference architecture (cf. Section 3) and *Maturity* and *Tool Support* providing general criteria.

Some basic characteristics of AOM languages, amongst them the modeling language, extension mechanism used and kinds of behavioral and structural diagrams employed, are evaluated by criteria within the *Language* category. In the *ConcernComposition* category, we consider amongst others the representation of the aspect concept and the weaving mechanism used. We also investigate

which model elements represent join points and how model elements for a specific aspect are selected in the *AdaptationSubject* category. The *AdaptationKind* category focuses on modeling support of aspect-oriented concepts for adaptation purposes. The *Maturity* of an approach is discussed along the criteria of provided modeling examples, real-world application, and available information in terms of manuals, papers, and books. And finally, an AOM approach should be supported by appropriate tools to improve developer productivity and ensure syntactical correctness of the model [29]. Since a thorough evaluation of *Tool Support* for AOM would go beyond the scope of this work, tool support is evaluated on the basis of the available literature, only.
Following, each categories' criteria are presented.

4.2 Language

This category contains criteria describing the modeling language philosophy. We do not provide separate criteria for evaluating the *element* concept described in the reference architecture (cf. Section 3), since it is implicitly evaluated with several other criteria that investigate the corresponding reference architecture's AO concepts with respect to their modeling representation.
Modeling Language (L.L) Since we consider UML-based AOM approaches, only, with respect to the modeling language[18] used, a distinction between the underlying UML version, i.e., version *1.x*[19], version *2.0* [47] , and *unknown* is made.
Extension Mechanism (L.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, in what is called *heavy-weight extension*, the UML metamodel itself is extended through inheritance and redefinition of metamodel elements. Second, *profiles*, grouping user-defined extensions to metamodel elements in terms of stereotypes [61], 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 [47]. Designed in a way that tools can store and manipulate the extensions without understanding their full semantics, the light-weight extension mechanism fosters tool inter-operability [61]. Besides the above mentioned possibilities, an AOM approach can also rely on *non-UML-conform* (or *UML-like*) extensions. Finally, the fourth value for this criterion is *unknown*, which is used in those cases where it is not clear what kind of UML extension mechanism has been chosen.
Influences (L.I) Originally, we intended to use *platform dependency* as a criterion for this evaluation framework. We have found, however, no clear definitions

[18] Please note, that the reference architecture's *language* concept in principal comprises both, programming and modeling languages. Still, this work's focus is the modeling level and thus, we consider modeling languages in the language category, only.
[19] See http://www.omg.org/technology/documents/vault.htm#modeling

in literature of what a platform or what platform (in)dependence is such as in the context of OMG's Model Driven Architecture (MDA) [45]. Since there may be many abstraction levels between MDA's Platform Independent Models (PIM) and Platform Specific Models (PSM), what defines platform and platform-independence is a matter of objectives and has to be determined in the context of one's own work. We refrain, however, from trying to find a common platform definition for the evaluated approaches. Thus, for want of a clear definition, we resume the *inspired by* criterion of Reina et al. [60], according to which, many of the aspect-oriented modeling approaches have been inspired by concepts expressed in a specific aspect-oriented programming language. In contrast to [60], this criterion is not restricted to AOP platforms but *lists research areas* (e.g., SOP [32], MDSoC [50], [49], 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.

Domain Generality (L.DG) Our focus of interest is if an aspect-oriented modeling language provides general support for different kinds of domains, or if it is defined for a particular class of problems. This criterion therefore has two possible values, namely *general-purpose* or *domain-specific*, including a declaration of the supported domain.

Aspect Generality (L.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. We distinguish between the following two forms of *aspect generality*[20]: A *general-purpose* aspect-oriented modeling language supports modeling of all kinds of aspects, whereas an *aspect-specific* modeling language considers one[21] specific aspect, only.

Diagrams (L.D) The emphasis in specifying an aspect can be *structural* and/or *behavioral*. In this respect, we evaluate which kind of structural and/or behavioral diagrams are supported by the approach to specify aspect-orientation. 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 *behavioural* modeling of aspects.

Design Process (L.DP) A design process describes a well-defined, step-wise approach to modeling[22]. This criterion 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.

---

[20] This criterion has been termed "purpose" in [60].

[21] Theoretically, there could be modeling languages that support two, three or more specific aspects. We do not consider these as aspect-specific, since in that case, the definition for general-purpose modeling languages gets blurred.

[22] This criterion has been adopted from [18].

4.3 ConcernComposition

Aspect (C.A) This criterion investigates the aspect'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 (C.ES) Considering the broader research area of ASoC, we are able to distinguish between two possible ways of concern decomposition, *symmetric* and *asymmetric*. While, in the asymmetric paradigm one distinguishes between concerns of different structure, i.e. between aspects and all other core concerns of a base model, 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" [33].
Composition Symmetry (C.CS) While in the *asymmetric* paradigm composition happens between aspects and base models only, i.e. aspects are woven *into* base models, in the *symmetric* paradigm all concerns are orthogonal, which results in three composition possibilities: aspects-base, base-base, aspects-aspects. A detailed discussion on the ramifications of symmetric and asymmetric composition approaches can be found in [33].
Relationship Symmetry (C.RS) The relationship between concerns can be specified in a *symmetric* or in an *asymmetric* way [33]. In particular, the symmetry is determined by the placement of the weaving specification or rather the adaptation rules. Relationship asymmetry defines the adaptation rules within one of the concerns that are to be composed, whereas relationship symmetry defines them in neither of the concerns.
Weaving (C.W) This criterion evaluates the weaving mechanism provided by an AOM approach, taking into consideration if it is possible to specify the dynamicity of how aspects are woven into a base, i.e., *statically* or *dynamically*. Nonetheless, a particular approach might just provide aspect and base models and *defer weaving to later phases* in the software development process by separately generating aspect and base code from models, which are finally woven by a dedicated mechanism of the underlying aspect-oriented programming language. The advantages of approaches that support model weaving are first, at code level non aspect-oriented platforms can be used and second, the weaving results can be validated prior to implementation. However, once the aspects have been eliminated, they cannot be recovered at later stages thus causing traceability problems.
Adaptation Effect (C.AE) This criterion evaluates if the approaches provide means for visualizing the adaptation effect of aspects or rather their adaptations in models. The criterion's possible values are *supported* or *not supported*.
Conflict Resolution (C.CR) In accordance with [8], conflict resolution may be based on a mechanism to *avoid* conflicts in advance or to *detect* conflicts[23] and then *resolve* them manually by using *graphical* or *textual* means. While conflict

[23] A mechanism for identifying conflicts in models is implicitly provided if a conflict resolution mechanism is available. Thus, we do not provide a separate criterion for evaluating the *conflict* concept of the reference architecture (cf. Section 3).

avoidance might be a possible solution to cope with conflicting aspects, we still might need ways to detect and resolve conflicts that could not be captured by conflict avoidance in advance. Finally, in case no conflict resolution has been specified, this criterion evaluates to *not supported*.

4.4 AdaptationSubject

Structural Join Point (S.SJp) This criterion evaluates if structural join points are supported. On one hand, we are interested what kind - with respect to dynamicity - of structural join point are considered in the approaches, i.e., *structural-static* or *structural-dynamic*. And on the other hand we are interested in their realization by concepts of the modeling language or extensions thereof (i.e., in terms of the UML *meta-class* used or a *stereotype* definition) and how they have been depicted in the *notation*, if provided.
Behavioral Join Point (S.BJp) This criterion evaluates if *behavioral-static* or *behavioral-dynamic* join points are supported by the approaches. Furthermore, this criterion provides information on their representation based on concepts of the modeling language or extensions thereof and their *notational* representation, if provided by the approach.
Join Point Model (S.JpM) We distinguish 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 done for example by enhancing the language's metamodel in a way that certain model elements inherit from a join point metaclass such as in [74] or by at least identifying and declaring the join points of a language in "natural language" such as in [68] or [73]. 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.
Join Point Selection (S.JpS) Although the join point selection concept is represented as an abstract class in the reference architecture (cf. Section 3), a separate criterion is required for evaluating the commonalities of the concrete join point selection sub-classes. In particular, we evaluate if the join point selection mechanism has been realized based on a *standard* (e.g., AspectJ code, UML, OCL) or on a *proprietary* language.
Simple Join Point Selection (S.SJpS) This criterion evaluates the realization of the join point selection concept by concepts of the modeling language or extensions thereof and particularly distinguishes between *graphical* and *textual* representations of simple join point selections.
Composite Join Point Selection (S.CJpS) Furthermore, we evaluate if at all and how composite join point selections are represented in the modeling approach. Again, we distinguish between *graphical* and *textual* representations of composite join point selections.
Selection Method (S.SM) This criterion evaluates in which ways join points are selected in a certain approach. The selection of join points can be specified *declaratively*, *imperatively*, or simply by *enumeration*.

Relative Position(S.RP) This criterion evaluates 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 (S.A) The importance of the appropriate level of detail to model an aspect has already been acknowledged before [9], [34], [56], [60]. However, we feel that there is need for a more precise definition of what *abstraction* is in the context of AOM, than what is given in the above mentioned literature. The definition that a "low-level design of a system can also be viewed as the high-level coding of its implementation" [34], comes closest to our understanding of abstraction. However, we consider two dimensions of abstraction in aspect-oriented modeling, namely *abstraction with respect to the subjects of adaptation* and *abstraction concerning the kind of adaptation* (cf. Section 4.5). In the following, we distinguish between *high* and *low* levels of abstraction. Thus, models on a high level of abstraction are incomplete with respect to providing a specification for code generation. A high level of abstraction with respect to the subjects of adaptation means that the weaving points might not be identified yet, i.e., the model only specifies the fact that a certain aspect affects other concerns, but not exactly where. On the contrary, modeling languages providing a low-level of abstraction allow specifying the exact points where adaptations take effect.

4.5 AdaptationKind

Structural Adaptation (K.SA) This criterion evaluates if AOM approaches provide ways of specifying structural adaptations and in particular what *concepts or extensions* of the modeling language and what *notational elements* have been used for representation.

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

Composite Adaptation (K.CA) In addition to evaluating structural and behavioral adaptation support, we are particularly interested, how the approaches provide ways of composing adaptations to more complex adaptations in terms of *concepts or extensions* of the modeling language and appropriate *notational elements*.

Abstraction (K.A) Since models on a high level of abstraction are incomplete with respect to providing a specification for code generation, a *high* level of abstraction with respect to the adaptation kind means that it is not yet clear *how* the base model should be adapted, i.e., the model only specifies that a certain aspect exists, but not the actual adaptations it provides. Consequently, *low*-level models provide detailed information on how the aspect's internals (i.e. the actual adaptations and auxiliary functionality) look like.

4.6 Maturity

Modeling Examples (M.E) One indication for maturity of an approach is the *number of different modeling examples* discussed. Thus, the values for this criterion are different *numbers* of provided modeling examples by each approach.
Application in Real-World Projects (M.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[24]. Possible values are *yes*, *no*, and *partially*.
Topicality (M.T) The topicality criterion checks for each approach when, the *most recent piece of work* has been published to indicate whether the approach is still under development or not. The possible values, thus, provide the *the year of publication* of the most recent piece of work.
Available Information (M.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

Tool support improves developer productivity and ensures syntactical correctness of the model [29]. While we distinguish between modeling support, weaving support and support for code generation, support for both weaving and code generation are dependent on modeling support.
Modeling Support (T.M) Modeling support is defined as the support of the modeling language's notation and furthermore the support of validating the created aspect-oriented models for syntactical and semantical correctness [29]. 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*.
Weaving Support (T.W) If modeling concepts for weaving are provided by the AOM approach, this criterion specifies if weaving of aspects into base models is also *supported* or *not supported* by a tool.
Code Generation (T.G) In line with the concepts of model-driven engineering, code generation facilities should be provided, thus requiring a mapping between the notation and the supported implementation language. With this criterion, we evaluate if code generation, in principle, is possible and beyond, if there is a more sophisticated mechanism to code generation such as the OMG's MDA [45] (i.e. existence of platform-independent models, platform definition models and their transformation into platform-specific models by using a mapping mechanism).

[24] It has to be noted that, in [8] "maturity" was used to evaluate if aspect-oriented requirements engineering approaches had been used in real projects.

Thus, possible values for this criterion are *supported* or *not supported*. Additional information is provided in case of a more sophisticated code generation mechanism.

5 Comparison of Approaches

Developers are already facing an immense amount of AOM approaches each of them having different origins and pursuing different goals dealing with the unique characteristics of aspect-orientation. A comparison of all of them obviously can not be covered in this work. Following, we present the results of evaluating eight approaches to AOM. The rationale behind choosing these eight is to assort a representative mix of approaches having different origins and goals, thus, supporting different concepts and different application areas, as well as approaches having a different level of maturity and topicality.

Our findings are based on a literature study, including modeling examples, provided by the individual AOM approaches. While we make our findings available in the Appendix at a glance (cf. Table 1 to 6), in the following, we provide additional information and discussion on a *per-approach* basis. The evaluation of each approach follows the ordering of categories of our evaluation framework presented in Section 4. Moreover, we provide a *running example* (cf. Section 5.1) that is modeled by means of the concepts of each AOM approach. This further enhances our evaluation in that it first, provides an insight into each approach and second, allows to easier compare the modeling means of the approaches[25].

5.1 The Observer Pattern applied to a Library Management System

As an appropriate example for an aspect to be applied to a system we decided to adopt the *observer pattern* [26], which is already popular in AOSD literature [14], [25], [28], [35], [66], [54]. In our running example, we apply the *observer pattern* as an aspect to a *library management system*, a prominent example not only in AOSD literature [14], [28], [35] but also in software engineering literature.
A Library Management System. In Figure 3, the *Library* package models the structure of a library management system based on [14] for managing books in a library. It depicts a small excerpt of a such a system, primarily containing those parts of the system that are crosscut by the observer aspect.
The *BookManager* manages a list of *Books* (cf. *addBook(Book)* and *remove(Book)*) allowing users to search (cf. *searchBook(Book)*) the list and access provided information for each book (e.g., *authors*). The library may offer several copies of each *Book*, i.e., the physical entities (cf. *BookCopy*). 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

[25] Due to space limitations, we have included only the core parts of the modeling examples in this work. The fully modeled examples and implementations of the running example in several AOP languages are provided at a dedicated web site (http://wit.tuwien.ac.at/schauerhuber/aomSurvey/).

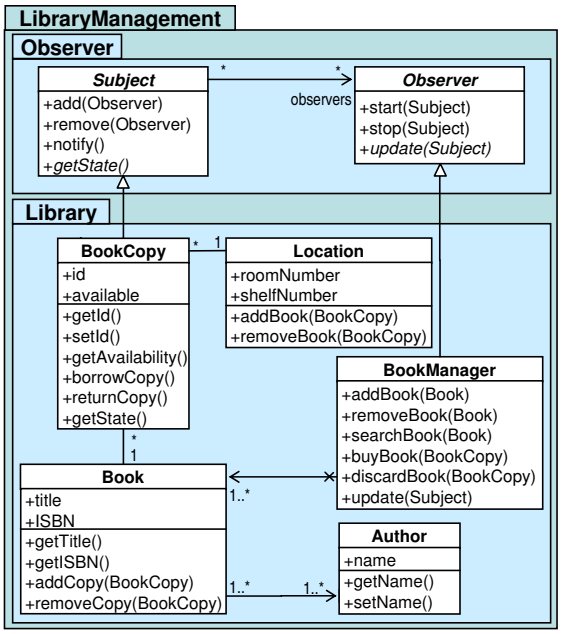

Fig. 3. The Library Management System with Observer Aspect.

(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 (cf. *borrowCopy()* and *returnCopy()*), the *BookManager* has to be notified. This notification functionality is not provided by the base system, but is applied using the observer pattern as depicted in Figure 3.

Observer Pattern. The observer pattern [26] 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.

In the following, the modeling means of each surveyed AOM approach is evaluated by means of this running example. Basically, we present the approaches in an age-based ordering, with the first three ones being approaches that have been introduced very recently. While the approaches of *Ortiz et al.* and of *Conejero et al.*, i.e, the first two approaches, focus on a specific domain and aspect, respectively, the approaches of *Groher et al.*, *Pawlak et al.*, and *Stein et al.* have been influenced by specific AOP models or languages and thus, represent AOM approaches that are more platform-specific. The last three approaches of *Aldawud et al.*, *Clarke et al.*, and *France et al.*, provide support for weaving aspect models and base models. In this respect, France et al. represent the most sophisticated approach, since a prototype tool for weaving models has already been developed.

5.2 Modeling Aspect-Oriented Web Services by Ortiz et al.

Language. In [48], *Ortiz et al.* propose an MDA approach to modeling Web Services, thus restricting the modeling language to the domain of Web Services (L.DG). They propose a general-purpose AOM language with respect to aspect generality, however (L.AG). *Ortiz et al.* give no information on the UML version used (L.L), but present a UML profile (L.E) devised to model aspects at a platform-independent level (L.I). As means for structural modeling, *class diagrams* have been taken into consideration by the proposed language, only (L.D). The authors provide neither guidelines nor outline a process for designing models based on their approach (L.DP).

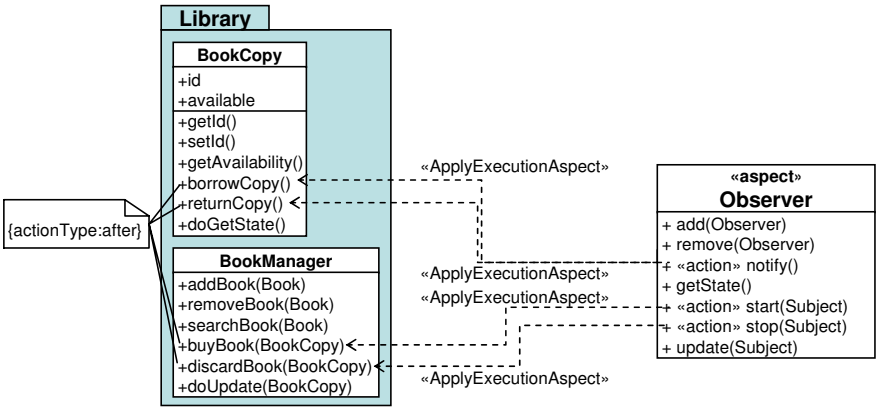

Fig. 4. The Observer Aspect modeled using the UML Profile of *Ortiz et al.*

ConcernComposition. Aspect behavior is encapsulated in the ≪aspect≫ stereotype derived from the UML meta-class *Class* and for notation purposes the UML class symbol is reused (C.A), (C.ES). The modeling approach follows the asymmetric paradigm of composition (C.CS), (C.RS). While three possible ways of weaving are discussed, the authors have not yet decided if weaving should be performed first, at the platform-independent level, thus, producing platform-specific woven models or second, maintain aspects in the platform-specific models and provide a weaving mechanism for platform-specific models, or third, defer weaving to implementation level (C.W). Similarly, no mechanisms for conflict resolution (C.CR) or specifying adaptation effects have been presented (C.AE).

AdaptationSubject. *Ortiz et al.* have explicitly identified two types of join points comprising the join point model for the Web Service domain (S.JpM). However, the join point concept has neither entered the UML profile nor does there exist a graphical representation of join points in the models. Both join point types represent behavioral-dynamic join points (S.BJp). On the one hand there are *service execution join points* (at the service provider's side) called *Execution-Aspects* and on the other hand there are *service call join points* (at the service consumer's side) called *CallAspects*. Join point selections are represented in the form of two stereotypes derived from the UML meta-class *Association* (S.JpS), (S.SJpS), (S.SM). Depending on the join point, the associations are stereotyped with ≪ApplyExecutionAspects≫ or with ≪ApplyCallExecutionAspect≫. They bind an aspect's adaptation either directly to the operation of a Web Service or to the class with all its operations. Hence, on the one hand composite join point selections are possible (S.CJpS) and on the other hand the modeling language allows modeling at a low level as well as at a high level of abstraction with respect to the subjects of adaptation (S.A). The enumeration *ActionType* allows three relative positions with respect to join points where adaptations can be inserted, namely before, after, and around. In models, the relative position is attached to adaptations as a *tagged value* using UML comments (S.RP).

AdaptationKind. Since Web Services are black boxes, structural adaptations are not possible in this domain and consequently not supported within the approach (K.SA). This restriction and the restriction of the subjects of adaptation to behavioral join points, however, means that the approach does not offer appropriate means to model the observer pattern within our library management system as presented in Section 5.1. The library management system would have to provide appropriate interfaces for retrieving the availability of a *BookCopy* and for updating the *BookManager* (cf. *doGetState()* and *doUpdate(BookCopy)* in Figure 4) in advance that could then be called by the Observer Aspect using *getState()* and *update(Subject)*, respectively. *Behavioral adaptations* of an aspect are represented by the ≪Action≫ stereotype that is derived from meta-class *Operation* with a tagged value of type *ActionType* that indicates the applied relative position (K.BA). Thus, adaptations are described at a low level of abstraction (K.A). No information has been given on composite adaptations(K.CA).

Maturity. The approach has recently been introduced (M.T), (M.I) and has not yet been tested in real-world projects (M.A). The applicability of the proposed

UML profile, however, has been illustrated by modeling security, logging, and service composition aspects in a travel agency case study (M.E).
Tool Support. While modeling support for UML profiles can be provided by any UML tool (T.M), further support for weaving and code generation is not offered (T.W), (T.C).

5.3 The Notification Aspect Profile of Conejero et al.

Language. In [16], *Conejero et al.* propose the aspect-specific UML profile AspectNotification (L.AG), (L.E) for modeling the event notification aspect, i.e., the notification functionality of the observer pattern, in Corba distributed applications (L.DG). The authors base their extension on the UML 2.0 meta-model (L.L) and claim independence from any aspect-oriented platform or the Corba version chosen, although the provided example has been described using AspectJ at the programming level (L.I). Besides class diagrams for modeling the structural features of aspects, sequence diagrams are used to model aspectual behavior (L.D). A design process, however, has not been described for the approach (L.DP).

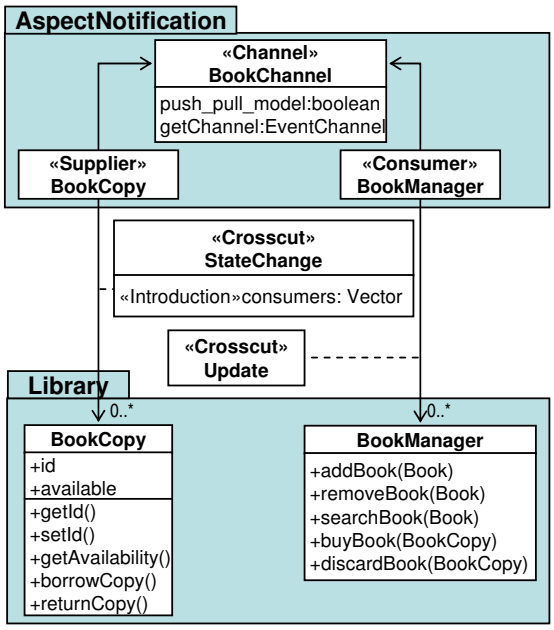

Fig. 5. The Observer Aspect using the Notation of *Conejero et al.*

ConcernComposition. A distinct representation for aspects does not exist, since the profile actually provides modeling concepts for one specific aspect,

i.e., the event notification aspect, only. However, special stereotypes derived from meta-class *Class* are introduced to represent the participating roles in the notification aspect, namely ≪Supplier≫, i.e., modeling the aspect of producing events, ≪Consumer≫, i.e., modeling the aspect of consuming events, and ≪Channel≫, i.e., modeling the aspect of transporting events (C.A). Still, the modeling approach follows the asymmetric school of thought with respect to elements (C.ES) and composition (C.CS), since there is a distinction between the aspect's roles, i.e., ≪Supplier≫ and ≪Consumer≫, and the composition is directed (cf. ≪Supplier_crosscut≫ and ≪Consumer_crosscut≫ in Figure 5), respectively. With respect to relationship symmetry (C.RS), however, the approach is symmetric (cf. AdaptationSubject, stereotype ≪Crosscut≫). Neither a weaving mechanism (C.W), nor means of conflict resolution (C.CR), nor ways to specify adaptation effects (C.AE) are provided by the modeling approach.

AdaptationSubject. In *Conejero et al.*, a join point model is implicitly defined, only (S.JpM). The join point selection mechanism consist of several parts (S.JpS). First, the subjects of adaptation are identified at a high level of abstraction (S.A) using the stereotyped UML *Associations* ≪Supplier_crosscut≫ and ≪Consumer_crosscut≫ between roles of the aspect (S.SM), i.e., ≪Supplier≫ and ≪Consumer≫, respectively, and classes (S.SJp) in the base model. Second, a stereotyped UML *AssociationClass* ≪Crosscut≫ can be added to these associations. The sets of points where to apply behavioral adaptations can then be specified by stereotyped *Operations*, i.e., ≪Pointcut≫, of the association class (S.A). Since, no further information is given on how these join point selections are realized, we have not been able to apply the ≪Pointcut≫ concept in the modeling example (S.SJpS), (S.CJpS). Furthermore, specifying a relative position is not possible (S.RP).

AdaptationKind. The AOM language proposed by *Conejero et al.* allows modeling adaptations at a high level of abstraction, only, since information on how to use the ≪Crosscut≫ stereotype to detail adaptations is not provided (K.A). Structural adaptations are specified in the attributes compartment of the ≪Crosscut≫ stereotype. Specifically, a stereotype ≪Introduction≫ has been defined for modeling structural adaptations, i.e., introducing new attributes (K.SA). While the authors introduced the ≪Pointcut≫ stereotype as a means for identifying where to apply behavioral adaptations, no further information has been given on how behavioral adaptations have been reified in the modeling language (K.BA). Finally, composite adaptations are not considered by the approach (K.CA).

Maturity. The approach of *Conejero et al.* has recently been published in [16] (M.T),(M.I). Nevertheless, the profile has been applied in a real-world project, a distributed control application of European Installation Bus (EIB) installations (e.g., Facility Management, Temperature Control) (M.A). The aspect-specific approach has been illustrated using one modeling example, only (M.E).

Tool Support. While modeling support is guaranteed due to the use of UML's light-weight extension mechanisms (T.M), weaving support is not considered at all (T.W). A code generation tool is planned for future work, however (T.C)

5.4 The sUFA approach of Groher et al.

Language. sUFA (Standard UML for Aspects) proposed by *Groher et al.* [28] supports the *aspectual collaborations* programming model, i.e., a collaboration of classes that collectively realize an aspect (L.I). The approach is based on the UML for Aspects (UFA) approach [35], i.e., a graphical design notation for aspectual collaborations. In contrast to the non-UML-conform UFA approach, sUFA extends UML 1.4 (L.L) using the UML profile extension mechanism (L.E). The approach is neither restricted to a specific aspect (L.AG) nor to a specific domain (L.DG). sUFA uses package and class diagrams, only (L.D), and does neither provide a design process nor guidelines (L.DP).

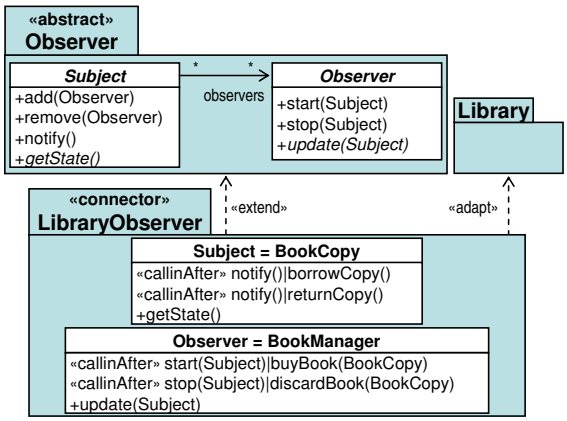

Fig. 6. The Observer Aspect modeled with *sUFA*.

ConcernComposition. sUFA distinguishes between *base* and *aspects* (i.e., aspectual collaborations), where aspects are woven into bases, only. Hence, the approach follows element and composition asymmetry (C.ES), (C.CS). Aspectual collaborations form patterns, i.e., collaborations of classes which together implement aspect functionality independently of the context in which they will be used. Since they contain abstract methods, which are bound to concrete methods when applying the pattern onto a certain application, they are represented using the ≪abstract≫ stereotype which is derived from the UML meta-class *Package* (C.A) (cf. *Observer* Figure 6). The binding of aspects to the base, i.e., the specification of the subjects of adaptation, is realized by a separate, i.e., symmetric, ≪connector≫ package (cf. *LibraryObserver* Figure 6) (C.RS). In particular, while a *dependency relationship* stereotyped with ≪extend≫ defines the relationship between the ≪connector≫ and the aspect, a *dependency relationship* stereotyped with ≪adapt≫ defines the relationship between the ≪connector≫ and base. A woven representation of aspects and bases is not

supported by sUFA (C.W), a conflict resolution mechanism has not been mentioned either (C.CR), and the approach offers no modeling means for specifying the adaptation effect (C.AE).

AdaptationSubject. An explicit join point model does not exist (S.JpM), neither are join points expressed as a modeling language concept. In the ≪connector≫ package each nested class of the aspectual collaboration is bound to a class of the base, e.g., in Figure 6 *Subject* is bound to class *BookCopy*. Each join point is bound separately to the adaptation using stereotyped methods (S.SM), e.g., *notify()* is bound to *borrowCopy()* and *returnCopy()*, respectively. Although the approach provides no explicit join point selection concept, the binding mechanism uses, e.g., UML class names for specifying the subjects of adaptation (S.JpS) and is thus restricted to simple join point selections (S.SJpS), (S.CJpS). The binding mechanism further allows selecting structural-static (S.SJp) and behavioral-static join points, only (S.BJp). The relative positions before, after, and around are defined in association with behavioral adaptations, only, i.e., they are part of the name used for stereotypes describing adaptations (e.g., ≪callinBefore≫) (S.RP). The sUFA notation in general can be seen as both low-level and high-level (S.A). This is since, the binding between aspects and the base is done in such a way, that the subjects of adaptation are visible and code can be generated in an automated way. But it is also possible to hide the internals of the ≪connector≫ to obtain a high-level view of the system.

AdaptationKind. sUFA provides means to represent structural and behavioral adaptations (K.SA), (K.BA). Composite adaptations, however, are not mentioned (K.CA). The stereotypes ≪callinAfter≫, ≪callinBefore≫, and ≪callinReplace≫[26], allow to specify method calls from the base application (e.g., *borrowCopy()*) to aspect method calls (e.g., *notify()*). The ≪callout≫ stereotype is used to express that the aspect's classes call the external behavior of base classes. They define how the base class methods should behaviorally be adapted. As an example, *update()* might call existing behavior of *BookManager*. In our modeling example, however, we do not bind *update()* to an existing method of the base, but directly "implement" it in the binding specification. Again with respect to adaptation, the approach supports both a high and a low level of abstraction depending if the internals of the aspectual collaboration are made explicit such as in Figure 6 or not (K.A).

Maturity. The sUFA approach has been recently developed (M.T), (M.I). So far, one modeling example is available, only, realizing the observer aspect for a library management system similarly to the example presented in Figure 6 (M.E). There is no evidence for applications of sUFA in real-world projects (M.A).

Tool Support. While tools for modeling support (T.M) and code generation support are available (T.G) within sUFA, weaving is deferred to the programming level. Developers only need to graphically specify how components are

[26] It has to be noted that, with respect to the relative position the authors introduced the ≪callinReplace≫ stereotype with the semantics of the relative position kind *around*.

linked together using Enterprise Architect[27] and the bindings are generated automatically using openArchitectureWare[28].

5.5 The JAC Design Notation of Pawlak et al.

Language. The UML notation proposed by *Pawlak et al.* [52], [53] is mainly designed for the JAC Framework[29] (L.I). The notation is based on light-weight UML extensions (L.E). It has been developed out of a pragmatic need to express crosscutting concerns in the JAC Framework and the authors do not claim full compliance with UML but aim at keeping it intuitive and simple. The notation is based on UML version 1.x (L.L)[30]. It can be seen as general-purpose (L.DS), (L.AS), since no restriction on a domain or aspect support exists. The approach relies on class diagrams (L.D) and provides no description of a design process or guidelines (L.DP).

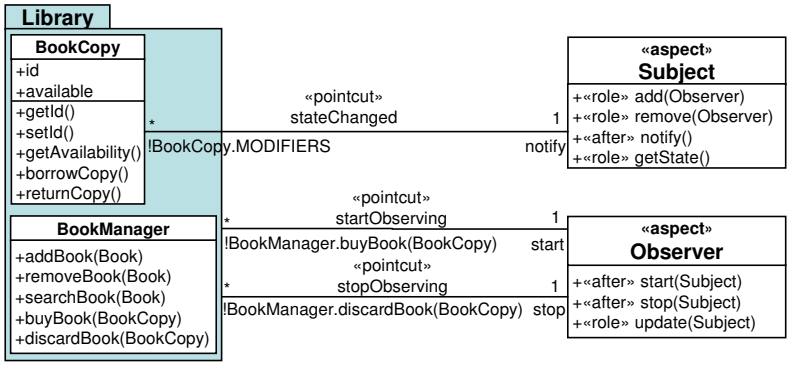

Fig. 7. The Observer Aspect Depicted using the *JAC Design Notation.*

ConcernComposition. The design notation for JAC follows the asymmetric approach for both elements (C.ES) and composition (C.CS). With respect to relationships (C.RS), the design notation represents a symmetric approach using a UML *Association* stereotyped with ≪pointcut≫ (cf. AdaptationSubject). The stereotype ≪aspect≫ which is derived from the UML meta-class *Class* is

[27] http://www.sparxsystems.com.au/

[28] http://www.openarchitectureware.org/

[29] JAC (http://jac.objectweb.org/) is an open source framework which includes a complete IDE with modeling support and serves as a middleware layer for aspect components.

[30] The authors provide no information on the UML version. The extended UML metamodel in [53], however, suggests that a UML version prior to version 2.0 is used, since the metamodel contains the meta-class "Link" which is discarded as of UML version 2.0

used to represent aspects (C.A) (cf. *Observer* in Figure 7). While a weaving mechanism is not available at modeling level but deferred to runtime (C.W), conflict resolution (C.CR) is not addressed at all. In the JAC design notation there are five stereotypes derived from the UML meta-class *Operation* (cf. AdaptationKind) which specify adaptations. The specification of adaptation effects is partly considered by the stereotype ≪replace≫ which provides for either a replacement or a deletion. All other stereotypes are considered to enhance the base (C.AE).

AdaptationSubject. A join point model is implicitly defined in [53] (S.JpM) and join points are limited to method calls, only (S.BJp), (S.SJp). The additional concept of *pointcut relation* corresponds to the adaptation rule concept defined in our reference architecture (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 7). The adaptation rule (i.e., the *pointcut relation*) connects the actual join point selection definition with the adaptation, i.e., the association ends contain information about the join point selection definition and the adaptation, respectively. Join point selections are defined using a *proprietary*, textual language based on regular expressions and/or keywords (S.JpS), (S.SM), e.g., *!BookCopy.MODIFIERS* in Figure 7 selects as join points all *method invocations* ('!') of methods from class *BookCopy* that modify the object state (S.SJpS). Thus, the notation provides a low level of abstraction (S.A). The provided join point selection mechanism also allows composing simple join point selections using operators, e.g., AND, OR, etc (S.CJpS). Concerning the relative position, this is specified for behavioral adaptations, only, by three out of the five stereotypes for adaptations, i.e., ≪before≫, ≪after≫, and ≪around≫ (S.RP).

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 base types that might need the same set of adaptations. It is, thus, part of the join point selection mechanism. For example in the observer aspect, subjects, i.e., arbitrary base types 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.

AdaptationKind. Both behavioral and structural adaptations are represented as methods of aspect classes. The kind of adaptation is indicated by the stereotype of the adaptation operation (e.g., ≪after≫ *update()* in Figure 7). The stereotypes ≪before≫, ≪after≫, ≪around≫, and ≪replace≫ indicate behavioral adaptations (K.BA), whereas ≪role≫, i.e., the fifth stereotype for adaptations, represents a structural one (K.SA). In the JAC design notation, structural adaptations which are implemented by ≪role≫ methods[31] are not really added to the structure of the base but can be invoked on the objects that are extended by the aspect, e.g., ≪role≫ *addObserver(BookManager)* can be invoked on *Book-*

[31] Role methods are similar to the *introduction* concept of AspectJ.

Copy (cf. Figure 7). Moreover, they can access the extended class attributes and the attributes of the aspect. The notation of *Pawlak et al.* is predominantly a low level modeling approach, also with respect to adaptation, i.e., it shows aspect internals (K.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. 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.W) but not at design time.

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

Language. The *Aspect-Oriented Design Model* (AODM) approach of *Stein et al.* [67], [68], [66] is a general-purpose, i.e., domain- and aspect-independent (L.DG), (L.AG), design notation for AspectJ (L.I). For this approach, both AspectJ and UML have been studied in order to first, find corresponding parts for AspectJ's concepts in UML, second, extend UML to support AspectJ's concepts if necessary, and third, 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 [66]. AODM is basically specified using the UML 1.x light-weight extension mechanism (L.L), (L.E), though extensions of the metamodel have also been necessary, e.g., ≪crosscut≫ relationship. Structural and behavioral modeling is done using class diagrams, collaborations, and sequence diagrams. In addition, sequence diagrams are used for visualizing join points, e.g., messages, while use case diagrams and collaborations realize AspectJ's weaving mechanism (L.D). The AODM approach does not outline a design process or provide guidelines (L.DP).
ConcernComposition. AODM represents a notation designed for AspectJ and consequently follows the *asymmetric* school of thought (C.ES), (C.CS), (C.RS). An aspect in AODM is represented by a stereotype ≪aspect≫ (cf. *SubjectObserverProtcolImpl* in Figure 8[32]), which is derived from the UML meta-class *Class*. In addition, several attributes capture the peculiarities of AspectJ's aspects, e.g., the instantiation clause (C.A). The weaving mechanism in AODM is specified in terms of UML *use case diagrams* and *collaborations* (C.W) (see [66]). A graphical conflict resolution mechanism, which determines the weaving order for aspects, is provided in terms of a stereotyped dependency relationship

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

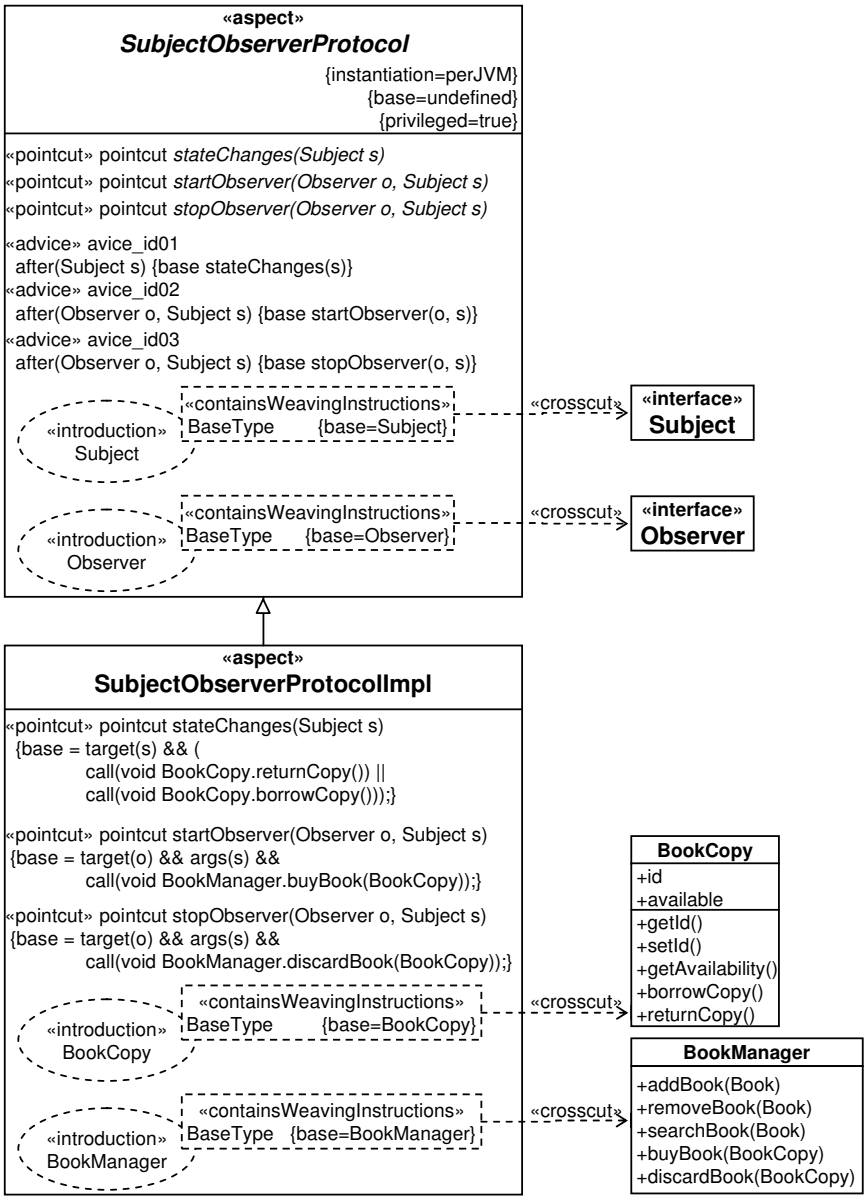

Fig. 8. The Observer Aspect modeled using the *AODM Notation for AspectJ*.

between aspects, i.e., ≪dominates≫ (C.CR) (see [67]). A means for explicitly specifying adaptation effects in models, however, is not addressed in AODM (C.AE).

AdaptationSubject. Though AODM has been specifically designed as a modeling language for AspectJ, *Stein et al.* [68] extend their notion of a join point model (S.JpM): UML *Classifiers* are identified as structural-static adaptation hooks (S.SJp), besides, UML *Links* represent behavioral-dynamic join points (S.BJp). Behavioral dynamic join points are depicted by highlighted messages in sequence diagrams (see [66]). 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 adaptation are specified at a high level of abstraction (S.A). The join point selection in AODM is similar to AspectJ's *pointcuts*. Selections of behavioral-dynamic join points are represented by ≪pointcut≫ stereotyped UML *Operations* that are implemented by special ≪ContainsWeavingInstructions≫ stereotyped *Methods*. An attribute *base* introduced for this ≪ContainsWeavingInstructions≫ stereotype then holds the join point selection in the form of AspectJ code (S.JpS), (S.SM), thus allowing first, the specification of composite join point selections (S.SJpS), (S.CJpS), and second, the specification of the subjects of adaptation at a low level of abstraction (S.A). In addition, a stereotype ≪ContainsWeavingInstructions≫ derived from the UML meta-class *TemplateParameter*[33] is used to specify the join point selections for structural adaptations (e.g., ≪introduction≫ *Subject*). The attribute *base* introduced for this ≪ContainsWeavingInstructions≫ stereotype then specifies the join point selection in the form of AspectJ's *type patterns*. AspectJ's - and consequently AODM's - means for join point selection is following a specific conceptual model. Recently, the authors haven been working on a more expressive join point selection mechanism supporting different conceptual models [69], which is independent from the AODM approach, however (cf. below).

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 as a keyword in the signature of behavioral adaptations (S.RP).

AdaptationKind. Adaptations in AODM are modeled at a low level of abstraction (K.A). Behavioral adaptations in AODM are represented by stereotyped UML operations, i.e. ≪advice≫. These are implemented by special ≪ContainsWeavingInstructions≫ *Methods*, which contain the actual adaptation in the method's *body* attribute and reference a join point selection (i.e., a *pointcut declaration*) in the introduced *base* attribute. In a class diagram, behavioral adaptations are depicted in the operation compartment of a class consisting of the operation's signature and a *base* tag containing the join point selection (K.BA). Additionally, behavioral adaptations are specified in terms of sequence diagrams. Structural adaptations are realized as parameterized collaboration templates with stereotype ≪introduction≫. The parameters are of type ≪ContainsWeavingInstructions≫, which specify the subjects of adaptation in

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

the form of AspectJ's *type patterns* (K.SA). The details of the collaboration templates are shown in Figure 9. Composite adaptations, since not a concept available in AspectJ, are not addressed by AODM (K.CA).

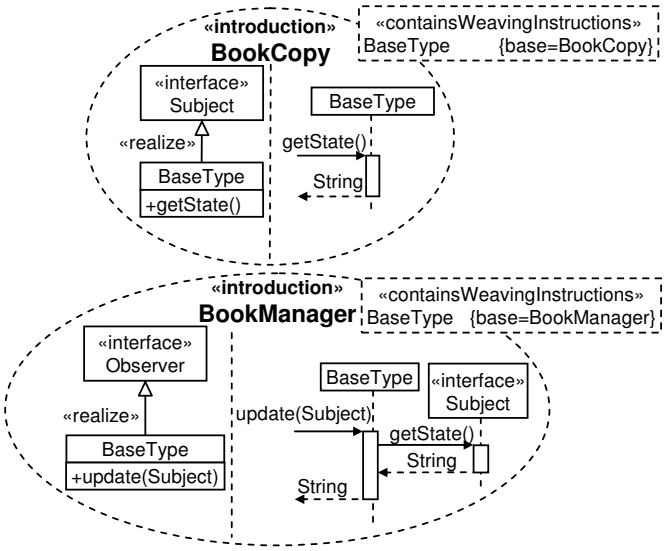

Fig. 9. Structural Adaptations in *Stein et al.*

Maturity. AODM has been described in several publications (M.I). While the approach has not been tested in a real-world application, several modeling examples have been provided, e.g., timing and billing aspects for a system in the area of telecommunication [73] and the realization of the observer pattern (M.E). Today, the authors have moved on and specifically focus on research towards a graphical way to select join points in UML . For this they have introduced Join point designation diagrams (JPDD) [69], which basically are UML diagram (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 join point selection mechanism that can be applied to any UML-based AOM language, allow to select all kinds of join points (i.e., structural-static, structural-dynamic, behavioral-static, behavioral-dynamic) and support composite join point selections (M.T).

Tool Support. The approach claims rapid modeling support by a wide variety of CASE tools [67], 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 weaving support and code generation support are currently not considered (T.W), (T.C).

5.7 The AOSD Profile of Aldawud et al.

Language. The AOSD Profile (L.E) of *Aldawud et al.* [3], [20] is based on UML version 1.x (L.L) and is aimed at being independent of any particular AOP language (L.I). The influence of AspectJ in earlier work [2], [79] has to be mentioned, however. The approach is not restricted to a specific domain (L.DG) or aspect (L.AG). While *class diagrams* are used to express the structural dependencies, *state charts* model the behavioral dependencies of base and aspects. Furthermore, state charts also implicitly express the weaving mechanism (L.D). The approach also offers a set of *guidelines* for using the concepts provided by the AOSD Profile (L.DP).

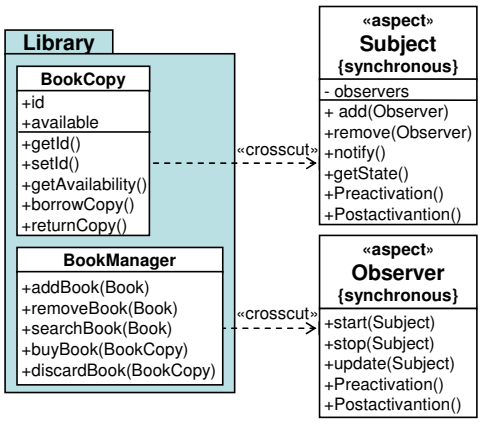

Fig. 10. The Observer Aspect modeled using the *AOSD Profile*.

ConcernComposition. Aspects are represented by the stereotype ≪aspect≫ (C.A), (C.ES), which is derived from the UML meta-class *Class* (cf. Figure 10). In addition, the approach allows to tag aspects and thus distinguish between *synchronous* aspects, i.e., aspects that impact and control the functionality of the base such as the observer aspect, and *asynchronous* aspects which have no impact on the base, e.g., a logging aspect. If synchronous, then he profile requires that the aspect must define the *Preactivation* and *Postactivation* operations in order to synchronize the aspect's behavior with the base's behavior. The synchronize tag, thus, allows to specify possible conflicts and mechanisms to resolve them (C.RS). 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 (C.CS). An integrated model view where aspects would already be woven into the base classes is not provided (C.W). The approach, however, uses an event mechanism (C.RS) to indicate the flow of crosscutting behavior in state charts (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 (C.CR). The effect of adaptations, however, can not be modeled.

AdaptationSubject. The approach of *Aldawud et al.* neither provides a join point model (S.JpM), nor a representation of join points in the profile (S.Jp). Join point selections are represented by stereotyped associations between aspects as well as aspects and concerns (S.JpS). This ≪crosscut≫ stereotype (cf. Figure 10) indicates where the aspect crosscuts a concern in the approach[34] (S.SM), thus, supporting structural-static join points, only (S.SJp), (S.BJp). The absence of a join point model and a more fine grained join point selection mechanism (S.SJpS), (S.CJpS) is traced back to the fact that the approach of *Aldawud et al.* aims at describing the concerns at a *high* level of abstraction (S.A) and avoids using properties which might re-contextualize the ones defined in specific AOP languages. The relative position concept is not part of the profile but has been regarded in an earlier work [79] (S.RP). The relative positions of adaptations could be retrieved from the state charts, however, where the flow of (non-)crosscutting behavior implicitly depicts the relative position of new crosscutting behavior.

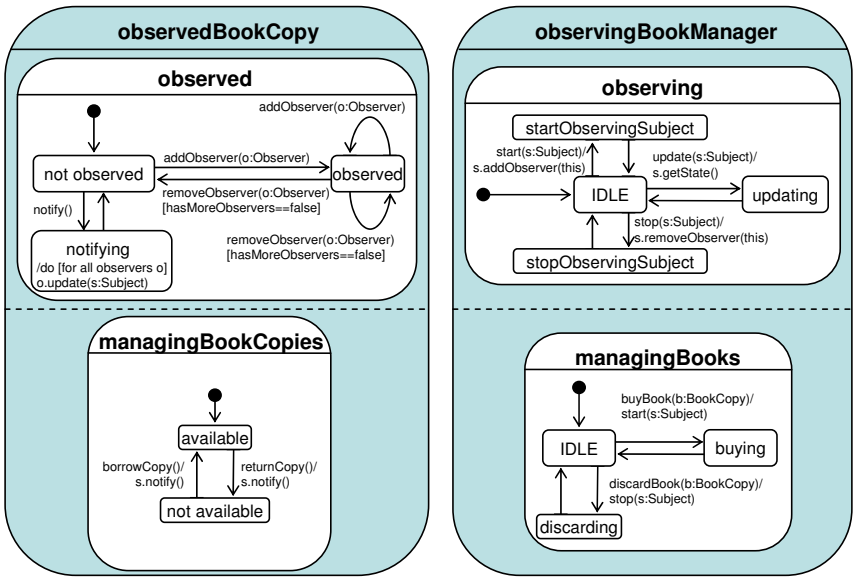

Fig. 11. The Observer's Crosscutting Behavior in the Approach of *Aldawud et al.*

AdaptationKind. While an aspect's methods supposedly can be used to specify adaptations in the class diagram, no information on how this is archived has been

[34] 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*.

given on this account in any of the examples. Still, behavioral adaptations are implicitly indicated in the state charts (K.BA) The examples, however suggest, that the aspects' attributes can be introduced as new features to the base, thus allowing structural adaptations (K.SA). The approach provides no means for indicating complex adaptations, however (K.CA). With respect to abstraction, the AOSD profile allows modeling adaptations at a low level of abstraction (K.A).
Maturity. Although the approach is described in several publications (M.I), (M.T), it is illustrated using a single example, the *bounded buffer system*, only. Still, it covers many different aspects, namely, synchronization, scheduling, and error handling (M.E). The example shows the structural and behavioral relationship between base and aspect classes and also indicates possibilities of extending the model with additional aspects. 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 guaranteed through existing UML modeling tools. In addition, Zakaria et al. [79] partly realized the profile as an add-in to Rational Rose[35] (T.M). Code generation using this approach was proven to be successful by using Rhapsody[36] (T.G). Weaving support, however, has not yet been addressed (T.W).

5.8 The Theme/UML Approach of Clarke et al.

Language. The *Theme* approach of *Clarke et al.* [11], [14], [13] provides means for AOSD in the analysis phase with *Theme/Doc*[37], and in the design phase with *Theme/UML*. We focus on *Theme/UML* which is used in producing separate design models for each *theme*, i.e., an encapsulation of a concern, which represents some kind of functionality in a system [11]. *Theme/UML* is based on an extension of the UML meta-model version 1.3 (L.L), and represents a general-purpose AOM language with respect to both domain generality and aspect generality (L.DG), (L.AG). It is designed as a platform-independent AOM approach, which originated from SOP [12], and evolved from the composition patterns approach of *Clarke* [10], and provides mappings to AspectJ and Hyper/J (L.I). Basically, *Theme/UML* poses no restrictions on what UML diagrams might be used for modeling [11]. Nevertheless, particularly package and class diagrams are used for modeling structure and sequence diagrams are used for visualizing where behavioral adaptations have to be inserted (L.D). In addition, the authors outline a design process for their modeling approach (L.DP).
ConcernComposition. *Theme/UML* represents a symmetric approach[38] (C.ES), (C.CS), (C.RS). Concerns in *Theme/UML* are encapsulated in UML packages denoted with a stereotype ≪theme≫ (cf. *Observer* and *Library* in Figure 12).

[35] http://www.ibm.com/software/rational
[36] http://www.ilogix.com/
[37] Theme/Doc assists in identifying crosscutting concerns in requirements documents [11].
[38] For a detailed discussion see [11].

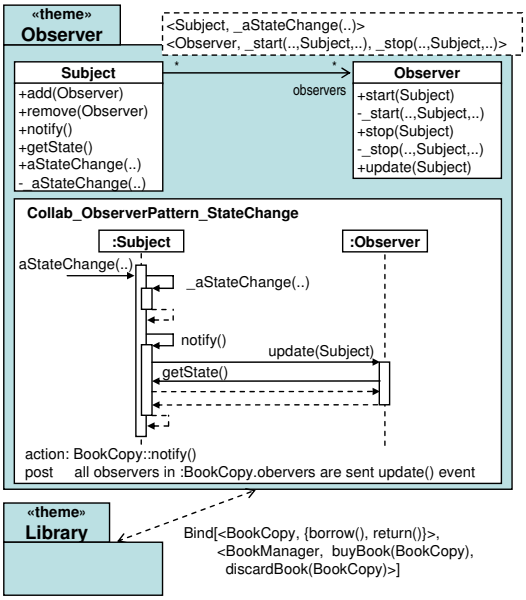

Fig. 12. The Observer Aspect depicted using *Theme/UML*.

While base themes are modeled using standard UML notation, crosscutting themes, i.e., aspects, are realized using UML's modified template mechanism, which allows instantiating template parameters more than once (C.A), i.e., the binding of a template is allowed more than once. Weaving is catered for through three different *composition relationships* (specialization from UML meta-class *Relationship*), namely *merge* and *override* (i.e., base-base composition), and *bind* (i.e., aspect-base composition), which is a specialization of merge (C.W). A woven representation of the *Observer* and *Library themes* is shown in Figure 13. Special attachments or *tags* to the 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 weaving output values. *Theme/UML* wants developers to first compose all base themes and then weave crosscutting themes one after the other into the woven base model, thus forcing the developer to consider the ordering of crosscutting themes (C.CR). The approach, however, does not provide modeling means for specifying the effects of adaptations (C.AE).

AdaptationSubject. The join point model comprises all subclasses of the new meta-class *ComposableElements* [10], i.e., the UML meta-classes Attribute, Operation, Association, Classifier, Collaboration, Interaction, and Package (S.JpM), (S.SJp), (S.BJp). An explicit representation for join points does not exist, how-

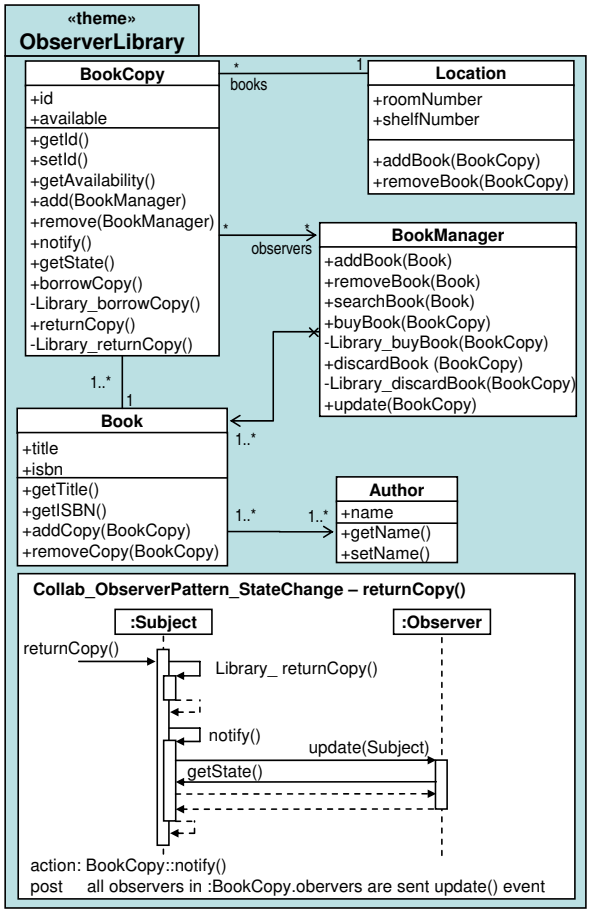

Fig. 13. The Woven Model Using the Notation of *Clarke et al.*

ever. The join point selection mechanism in *Theme/UML* turns out to be more complex (S.SJp): First, the sequence diagram templates[39] in a theme (cf. *ObserverPattern_StateChange* in Figure 12) provide a graphical means for selecting behavioral-dynamic join points, such as within the control flow of other operations. Additionally, the parameter specification of the signature of operations is used for selection. At the same time, the sequence diagram templates model what behavioral adaptation has to be inserted and thus, implicitly define the relative position with respect to the selected join point, only (S.RP). Second, the template parameters (i.e., classes, operations, and attributes) in a dotted box placed on the theme package template represent sort of the aspect's *abstract join point selection interface*. Thus, crosscutting themes, i.e., the package template can be seen as *abstract aspects* in the sense of AspectJ. Third, the concrete join point selections are specified by *bind relationships* between the crosscutting theme and other themes, which binds the template parameters to actual classes, operations, and attributes possibly using wildcards (S.SJpS). Composite join point selections, however, are not supported by *Theme/UML* (S.CJpS). Thus concerning the subjects of adaptation, *Theme/UML* models provide information at a detailed level. If omitting the bind tag the subjects of adaptation are identified at a high level of abstraction, i.e. at theme level[40] (S.A).

AdaptationKind. In a crosscutting theme, *class diagrams* are used to model the aspect's structure. Classes, attributes, operations, interfaces, and associations that are no templates represent structural adaptations to base themes (K.SA). Behavioral adaptations are defined within *sequence diagram templates* (K.BA). There are neither explicit modeling concepts for behavioral adaptations nor for composite adaptations (K.CA). Adaptations in Theme can be modeled at a *high* level of abstraction by using packages and composition relationships, only. The adaptations can then be refined by several structural and behavioral UML diagrams (K.A).

Maturity. The Theme/UML approach represents one of the most mature and well-engineered approaches to AOM (M.I), (M.T). It comes with a plethora of literature, modeling examples such as the synchronization and observer aspects in the *digital library example* [14], the logging aspect in the *expression evaluation system example* (M.E), and several other aspects in the *crystal game example application* [11] (M.A). In [11], two case studies (i.e. *phone features* and *usage licensing*) are presented.

Tool Support. Information on a tool for *Theme/UML* has not been provided (T.M), (T.W), (T.C).

---

[39] Please note, that templates for *Observer.start()* and *Observer.stop()* have been omitted. The complete modeling example can be found at http://wit.tuwien.ac.at/schauerhuber/aomSurvey/ .

[40] Note that Theme/Doc, which is not our focus, also allows identifying the subjects of adaptation at a *high* level of abstraction.

5.9 Aspect-Oriented Architecture Models of France et al.

Language. The Aspect-Oriented Architecture Models (AAM) approach of *France et al.* [24] is based on UML 2.0 (L.L) and recently, has been further developed [59] with respect to its composition mechanism. The language is designed as a platform-independent approach (L.I) and represents a general-purpose approach regarding both, domain generality (L.DG) and aspect generality (L.AG), although the authors particulary address *dependability concerns* such as security and fault-tolerance. Similar to the Theme/UML approach of *Clarke et al.*, aspect models are modeled using template diagrams, i.e., class diagram templates and communication diagram templates describing the aspects' structural and behavioral features, respectively (L.D). For readability purposes, however, the authors prefer to provide a notation different to standard UML templates. It is unclear if and how the authors have extended the UML. A design process is briefly outlined in terms of guidelines (L.DP).

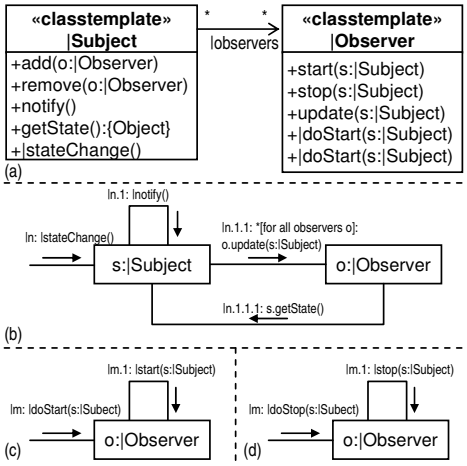

Fig. 14. The Observer Aspect Model using the Notation of *France et al.*

ConcernComposition. France et al. distinguish between aspect models and base models (i.e., *primary* models) (C.ES). Aspect models are modeled using template diagrams, which are described by parameterized packages (C.A) (cf. Figure 14). A textual *binding* to a certain application instantiates *context-specific* aspect models (cf. Figure 15) from the UML templates (C.RS). The following binding instantiates the aspect model for the observer pattern shown in Figure 14:

```
(|Subject,BookCopy);              (|Observer, BookManager);
(|stateChange(),borrowCopy()); (|doStart(s:|Subject),buyBook());
```

```
(|stateChange(),returnCopy()); (|doStop(s:|Subject),discardBook());
(|observer, bookManagers);
```

The context-specific aspect models are finally used for composition with the base model (C.CS). In [59], the authors introduce a *signature-based*[41] weaving mechanism[42] supported by *composition directives* of currently structural diagrams, only (C.W). This weaving mechanism is defined in terms of a composition metamodel, which is implemented in KerMeta[43]. Since KerMeta allows specifying operations with its action language, dynamic weaving of models is subject to future work. The composition directives, at the same time, serve as a conflict resolution mechanism. On the one hand *element* composition directives amongst others allow to add, remove, and override model elements. On the other hand, the *model* composition directives *precedes* and *follows*, depicted as stereotypes ≪precedes≫ and ≪follows≫ derived from UML meta-class *Dependency*, allow to specify the weaving order of aspects into the base model (D.CR). The approach does not describe ways to specify adaptation effects (C.AE).

AdaptationSubject. The join point model is defined with AAM's composition metamodel and comprises all subclasses of the meta-class *Mergeable* [59] amongst them the (UML) meta-classes Operation, Classifier, and Model (S.JpM). Join points, i.e., template parameters for classes, operations and their parameters as well as attributes, in aspect models are denoted using '|' (cf. e.g., |*Subject* and +|*aStateChange()* in Figure 14 (a)) instead of using UML's notation, i.e., a dashed box. While template parameters in class diagrams represent structural-static join points (S.SJp), template parameters in communication diagrams represent behavioral join points, i.e., both static and dynamic ones (S.BJp). Thus, template diagrams represent a (simple) join point selection mechanism in AAM (S.JpS), (S.SJpS), (S.CJpS). At the same time, they model what adaptations have to be inserted into the base model. While the class diagram templates model what structural adaptations have to be inserted (cf. Figure 14 (a)), the communication diagram templates model what behavioral adaptations have to be inserted and implicitly define the relative position to the selected join point (cf. Figure 14 (b)-(c)) (S.RP). Thus, concerning the subjects of adaptation aspect models provide information at a detailed level (S.A).

AdaptationKind. There are no explicit modeling concepts for adaptations in general (K.CA). In AAM class diagram templates, however, are used to model

[41] A model element's signature is defined in terms of its syntactic properties, where a syntactic property is either an attribute or an association end defined in the element's UML metamodel class [59].

[42] Please note that, the woven model is not shown due to space limitations. The woven model is different to the context-specific aspect model in Figure 15, however, in that it adds the classes, operations, and associations of the base model currently no present in the aspect-specific model. The woven model can be found at http://wit.tuwien.ac.at/schauerhuber/aomSurvey/ .

[43] The KerMeta metamodeling language (http://www.kermeta.org) extends the Essential Meta-Object Facility (EMOF 2.0) with an action language that allows to describe the behavior of operations in a metamodel.

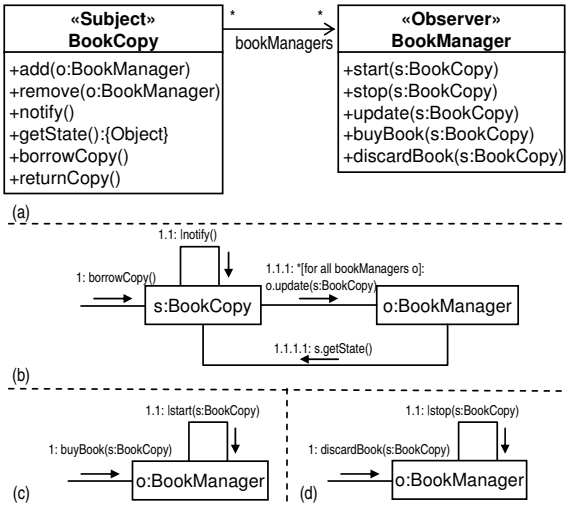

Fig. 15. The Context-Specific Aspect Model using the Notation of *France et al.*

the aspects structure. All structural elements, e.g., classes, attributes, and associations, that do not represent template parameters are structural adaptations to the base model (K.SA). Behavioral adaptations, i.e., elements that do not represent template parameters, are defined within communication diagram templates (K.BA). Adaptations in AAM are modeled at a low level of abstraction, only (K.A).

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

Tool Support. Currently, a prototype implementation of an integrated toolset provides modeling support (T.M), i.e., for modeling aspect model diagram templates (built on top of the Eclipse Modeling Framework[44] and for instantiating context-specific aspect models from these templates (built on top of Rational Rose) as well as weaving support (T.W), i.e., for composing base and context-specific aspect models (built on top of KerMeta). A tool for code generation currently is not planned (T.C) [57].

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

6 Lessons Learned

The results of our evaluation have revealed interesting peculiarities of current AOM approaches. We sum up our findings in the following.

6.1 Language

Omnipresence of UML. Currently, we know of only two approaches [71], [78] that did not base their aspect-oriented modeling proposal on UML. Furthermore, the majority of the evaluated approaches base their work on UML versions prior to the new UML 2.0 specification. For extending UML with aspect-oriented concepts, UML's inherent extension mechanism, i.e., profiles, is popular.
Popularity of General-Purpose Languages. The evaluation has shown that aspect-oriented modeling approaches usually are general with respect to both the application domain and the modeled aspects. While there already exist some domain specific modeling approaches (*Ortiz et al.*, [80]), we know of one approach offering modeling means for one specific aspect, only (*Conejero et al.*).
Predominance of Structural Diagrams. While all approaches make extensive use of structural diagrams (e.g., class diagrams and package diagrams), only few make use of behavioral diagrams in order to demonstrate behavioral features of aspects or to specify when crosscutting behavior should occur relative to the base behavior.
Missing Guidance in the Design Process. Since aspect-oriented modeling is still in its infancy, it is not surprising that only two approaches (*France et al.* and *Aldawud et al.*) provide at least some guidance in designing aspect-oriented models in terms of guidelines. Only one approach (*Clarke et al.*) provides a complete design process description.

6.2 ConcernComposition

Symmetry of Elements, Composition and Relationship Adopted from AOP Languages. Since aspect-oriented software development has clearly been driven by the emergence of aspect-oriented languages such as AspectJ, most AOM approaches rely on the asymmetric paradigm. While those AOM proposals that have emerged out of a symmetric programming paradigm such as SOP and MDSoC naturally tend to follow a symmetric paradigm at the modeling level.
Adaptation Effect Not Considered. Modeling the effect of aspects' adaptations is not considered at all. Only the JAC design notation of *Pawlak et al.* provides a stereotype ≪replace≫ for adaptations which - possibly not intended - indicates an adaptation effect, i.e., either a replacement or a deletion.
Almost No Weaving at Modeling Level. Weaving at modeling level is only supported by few approaches (*Stein et al.*, *Clarke et al.*, and *France et al.*), only. The majority defers weaving to the programming level.
Graphical Conflict Resolution Preferred. A conflict resolution is provided by half of the approaches with most of the proposed mechanisms being graphical ones. Only one approach's weaving mechanism (*France et al.*) allows to detect

conflicts and some provide guidelines to avoid conflicts. The conflict resolution mechanisms in most cases comprise means for specifying an ordering of aspects, some provide further means, e.g., to resolve naming conflicts.

6.3 AdaptationSubject

Missing Formal Definition of Join Point Models. While half of the approaches provide an explicit join point model and most of them in terms of a natural language description, only, the other half only implicitly provides a join point model, which is defined via the join point selection mechanism.
No Support for Structural-Dynamic Join Points. While almost all approaches support structural-static join points and almost all approaches consider either behavioral-static or behavioral-dynamic join points or both, structural-dynamic join points are not supported by any of the approaches.
Graphical, Standards-Based Join Point Selection Mechanism Preferred. The majority of the approaches uses a graphical join point selection mechanism based either on UML associations or the binding of UML templates. While *Pawlak et al.* proposes the only proprietary join point selection mechanism based on regular expressions and/or keywords, *Stein et al.* reuse AspectJ's pointcut language.
Rare Support of Composite Join Point Selection. While *Ortiz et al.* propose the only graphical join point selection mechanism supporting composite join point selections to a limited extend, two further approaches supporting composite join point selections (*Pawlak et al.* and *Stein et al.*) provide textual mechanisms that allow composing simple join points using logical operators.
No Imperative Join Point Selection. The approaches exclusively allow to select join points declaratively and/or by enumeration.
Good Support of Relative Position. Half of the approaches explicitly provide modeling concepts for specifying where, i.e., relative to the selected join points, adaptations have to be introduced. The majority of the others implicitly define the relative position of adaptations in the control flow modeled via behavioral diagrams (e.g., *Clark et al.*, *France et al.*, and *Aldawud et al.*).
Modeling Adaptation Subjects at a High & Low Level of Abstraction. All approaches allow to relate aspects to bases at a high level of abstraction but the majority of approaches also allows modeling the subjects of adaptation at an even lower level of abstraction. For the applicability of AOM, a high level of abstraction is beneficial, whereas for an automated execution of the model a detailed specification at a low level of abstraction is necessary.

6.4 AdaptationKind

Composite Adaptations Not Considered. While most approaches provide modeling means for both behavioral and structural adaptations, composing adaptations to form more complex ones and to foster reuse is not considered by any of the approaches.

Modeling Adaptation Kinds at a High & Low Level of Abstraction. While those approaches that support modeling at a low level of abstraction also support modeling at a high level of abstraction simply by omitting the adaptation details, the approach of *Conejero et al.* allows modeling at a high level of abstraction, only.

6.5 Maturity

Rarely Application in Real-World Projects. The applicability of aspect-oriented modeling languages has rarely been tested in real-world projects but is demonstrated using rather simple modeling examples, only. It is, thus, not possible to obtain reasonable evaluation results on issues such as traceability, usability, etc.

6.6 Tool Support

Missing Tool Support for Weaving 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 is rare and support for weaving is provided by only one approach (*France et al.*).

7 Conclusion and Outlook

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 *common reference architecture for aspect-oriented modeling*. In a second step, on the basis of the common reference architecture we established a *set of criteria*, which have been specifically derived from the reference architecture in order to evaluate the important concepts identified before in each aspect-oriented modeling approach. The actual *evaluation of eight aspect-oriented modeling approaches* according to our evaluation framework is furthermore supported by a *running example*, which has proven to be very helpful on one hand, to explore the applicability of each individual approach and on the other hand, to allow for a direct comparison of the approaches.

To establish an elaborate overview on the AOM field, we are extending our survey and include further aspect-oriented modeling approaches. Besides, future work heads into two different directions:
One crucial activity we are currently focusing on is to further justify the appropriateness of our reference architecture in terms of its unification ability. For this, despite having already defined implicit mappings via our evaluation framework to eight AOM approaches, we intend to specify direct mappings to existing AOM and AOP approaches. Such mappings could be, e.g., defined on basis of OMG's QVT [30] proposal. On the basis of such mapping definitions our reference architecture could also act as a pivot model translating between different

aspect-oriented languages.
With respect to application domains for AOM, we concentrate on ubiquitous web applications (UWA), similar to [6]. Modeling of customization in UWAs, i.e., the adaptation of their services according to the context of use, is a complex task, which we have already tackled in several projects [27], [37], [65]. Since customization modeling affects all levels of a UWA, i.e., the content, the hypertext, and the presentation, customization represents a crosscutting concern. We propose to use aspect-orientation as driving paradigm for capturing customization of UWAs at the modeling level [62]. In particular, we plan the extension of an existing Web modeling language, i.e., a refined version of the WebML metamodel [64], with concepts from the aspect-orientation paradigm.

8 Acknowledgements

We would like to thank the authors of the surveyed approaches for providing valuable feedback for the survey.

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*, LNCS 615, pages 372–395. Springer-Verlag, June/July 1992.
2. Omar Aldawud, Tzilla Elrad, and Atef Bader. A UML profile for aspect oriented modeling. In *Proc. of Workshop on Advanced Separation of Concerns in Object-Oriented Systems (OOPSLA'01), Tampa, Florida*, October 2001.
3. Omar Aldawud, Tzilla Elrad, and Atef Bader. UML Profile for Aspect-Oriented Software Development. In *Proc. of the 3rd International Workshop on Aspect Oriented Modeling, Boston, Massachusetts*, March 2003.
4. Elisa Baniassad and Siobhán Clarke. Finding Aspects in Requirements with Theme/Doc. In *Proc. of Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, Lancaster, England*, pages 15–22, March 2004.
5. Eduardo Barra, Gonzalo Génova, and Juan Llorens. An approach to Aspect Modelling with UML 2.0. In *Proc. of 5th Aspect-Oriented Modeling Workshop (UML'04), Lisbon, Portugal*, October 2004.
6. Hubert Baumeister, Alexander Knapp, Nora Koch, and Gefei Zhang. Modelling Adaptivity with Aspects. In *Proc. of the 5th International Conference on Web Engineering (ICWE'05), Sydney, Australia*, LNCS 3579, pages 406–416. Springer-Verlag, July 2005.
7. Jean Bézivin. On the Unification Power of Models. *Journal on Software and Systems Modeling*, 4(2):171–188, May 2005.
8. Gordon S. Blair, Lynne Blair, Awais Rashid, Ana Moreira, Jo ao 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.

9. 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 AOSD-Europe-ULANC-9, AOSD-Europe, May 2005.
10. Siobhán Clarke. Extending standard UML with model composition semantics. *Science of Computer Programming*, 44(1):71–100, July 2002.
11. Siobhán Clarke and Elisa Banaissad. *Aspect-Oriented Analysis and Design The Theme Approach*. Addison-Wesley, Upper Saddle River, March 2005.
12. 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.
13. Siobhán Clarke and Robert Walker. Towards a Standard Design Language for AOSD. In *Proc. of the 1st International Conference on Aspect-Oriented Software Development (AOSD'02), Enschede, The Netherlands*, pages 113–119. ACM Press, April 2002.
14. 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.
15. Adrian Colyer. AspectJ. In R.E. Filman, T. Elrad, S. Clarke, and M. Akşit, editors, *Aspect-Oriented Software Development*, pages 123–143. Addison-Wesley, Boston, 2005.
16. José Conejero, Juan Hernández, and Roberto Rodríguez. UML Profile Definition for Dealing with the Notification Aspect in Distributed Environments. In *Proc. of the 6th International Workshop on Aspect-Oriented Modeling (AOSD'05), Chicago, Illinois*, March 2005.
17. Thomas Cottenier, Aswin Van Den Berg, and Tzilla Elrad. Modeling Aspect-Oriented Compositions. In *Proc. of the 7th International Workshop on Aspect-Oriented Modeling (MODELS'05), Montego Bay, Jamaica*, October 2005.
18. 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.
19. Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, Englewood Cliffs, New Jersey, 1976.
20. 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.
21. Peter Fettke and Peter Loos. Referenzmodellierungsforschung. *Wirtschaftsinformatik*, 46(5):331–340, August 2004.
22. Robert E. Filman, Tzilla Elrad, Siobhán Clarke, and Mehmet Akşit, editors. *Aspect-Oriented Software Development*. Addison-Wesley, Boston, 2005.
23. Robert E. Filman and Daniel P. Friedman. Aspect-Oriented Programming Is Quantification and Obliviousness. In R.E. Filman, T. Elrad, S. Clarke, and M. Akşit, editors, *Aspect-Oriented Software Development*, pages 21–35. Addison-Wesley, Boston, 2005.
24. 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.

25. Lidia Fuentes and Pablo Sánchez. Elaborating UML 2.0 Profiles for AO Design. In *Proc. of the 8th International Workshop on Aspect-Oriented Modeling (AOSD'06), Bonn, Germany*, March 2006.
26. Erich Gamma, Richard Helm, Ralph Hohnson, and John Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesely, 2004.
27. Franca Garzotto, Paolo Paolini, Marco Speroni, Birgit Pröll, Werner Retschitzegger, and Wieland Schwinger. Ubiquitous Access to Cultural Tourism Portals. In *Proc. of the 3rd International Workshop on Presenting and Exploring Heritage on the Web (DEXA'04)*, Aug./Sep. 2006.
28. Iris Groher, Stephan Bleicher, and Christa Schwanninger. Model-Driven Development for Pluggable Collaborations. In *Proc. of the 7th International Workshop on Aspect-Oriented Modeling (MODELS'05), Montego Bay, Jamaica*, October 2005.
29. Iris Groher and Stefan Schulze. Generating aspect code from UML models. In *Proc. of the 4th Aspect-Oriented Modeling Workshop with UML, San Francisco, California*, October 2003.
30. QVT-Merge Group. Revised Submission for MOF 2.0. OMG Query/Views/Transformations RFP(ad/2002-04-10), Version 2.0, ad/2005-03-02, March 2005.
31. Stefan Hanenberg. *Design Dimensions of Aspect-Oriented Systems*. PhD thesis, University Duisburg-Essen, October 2005.
32. 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)*, pages 411–428, September 1993.
33. 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.
34. William H. Harrison, Peri L. Tarr, and Harold L. Ossher. A Position On Considerations In UML Design of Aspects. In *Proc. of the 1st Workshop on Aspect-Oriented Modeling with UML (AOSD'02), Enschede, The Netherlands*, March 2002.
35. Stephan Herrmann. Composable Designs with UFA. In *Proc. of the 1st Workshop on Aspect-Oriented Modeling with UML (AOSD'02), Enschede, The Netherlands*, March 2002.
36. Ivar Jacobson and Pan-Wei Ng. *Aspect-Oriented Software Development with Use Cases*. Addison-Wesley, 2005.
37. Gerti Kappel, Birgit Pröll, Werner Retschitzegger, and Wieland Schwinger. Customisation for Ubiquitous Web Applications - A Comparison of Approaches. *Int. Journal of Web Engineering and Technology*, 1(1), 2003.
38. Mika Katara and Shmuel Katz. Architectural Views of Aspects. In *Proc. of the 2nd International Conference on Aspect-Oriented Software Development (AOSD'03), Boston, Massachusetts*. ACM Press, March 2003.
39. Mik Kersten. AOP tools comparison (Part 1 & 2). http://www-128.ibm.com/developerworks/java/library/j-aopwork1/, March 2005.
40. 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*, LNCS 1241, pages 220–242. Springer-Verlag, 1997.
41. 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.

42. Ivan Krechetov, Bedir Tekinerdoğan, Alessandro Garcia, Christina Chavez, and Uirá Kulesza. Towards an Integrated Aspect-Oriented Modeling Approach for Software Architecture Design. In *Proc. of the 8th International Workshop on Aspect-Oriented Modeling (AOSD'06), Bonn, Germany*, March 2006.
43. Karl J. Lieberherr. *Adaptive Object-Oriented Software: the Demeter Method with Propagation Patterns*. PWS Publishing Company, Boston, 1996.
44. Ana Moreira, Jo ao Araújo, and Awais Rashid. A Concern-Oriented Requirements Engineering Models. In *Proc. of the 17th International Conference on Advanced Information Systems Engineering (CAISE'05)*, June 2005.
45. Object Management Group (OMG). MDA Guide Version 1.0.1. http://www.omg.org/docs/omg/03-06-01.pdf, June 2003.
46. Object Management Group (OMG). Meta Object Facility (MOF) 2.0 Core Specification Version 2.0. http://www.omg.org/docs/ptc/04-10-15.pdf, October 2004.
47. Object Management Group (OMG). UML Specification: Superstructure Version 2.0. http://www.omg.org/docs/formal/05-07-04.pdf, August 2005.
48. Guadalupe Ortiz, Juan Hernández, Pedro J. Clemente, and Pablo A. Amaya. How to Model Aspect-Oriented Web Services. In *Proc. of Workshop on Model-driven Web Engineering (ICWE'05), Sydney, Australia*, July 2005.
49. Harold L. Ossher and Peri L. Tarr. Multi-dimensional Separation of Concerns in Hyperspace. In *Proc. of International Workshop on Aspect-Oriented Programming (ECOOP'99), Lisbon, Portugal*, June 1999.
50. Harold L. Ossher and Peri L. Tarr. Multi-Dimensional Separation of Concerns using Hyperspaces. Technical Report 21452, IBM Research Report, April 1999.
51. David L. Parnas. On the Criteria To Be Used in Decomposing Systems into Modules. *Comm. ACM*, 15(12):1053–1058, December 1972.
52. 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.
53. 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.
54. 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*, August 2003.
55. Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, New York, 2005.
56. Andreas Rausch, Bernhard Rumpe, and Lucien Hoogendoorn. Aspect-Oriented Framework Modeling. In *Proc. of the 4th Aspect-Oriented Modeling Workshop with UML, San Francisco, California*, October 2003.
57. Raghu Reddy, Robert France, and Geri Georg. An Aspect Oriented Approach to Analyzing Dependability Features. In *Proc. of the 6th International Workshop on Aspect-Oriented Modeling (AOSD'05), Chicago, Illinois*, March 2005.
58. Raghu Reddy, Robert France, Sudipto Ghosh, Franck Fleurey, and Benoit Baudry. Model Composition - A Signature-Based Approach. In *Proc. of the 7th International Workshop on Aspect-Oriented Modeling (MODELS'05), Montego Bay, Jamaica*, October 2005.

59. 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.
60. 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.
61. James Rumbaugh, Ivar Jacobson, and Grady Booch, editors. *The Unified Modeling Language Reference Guide*. Addison-Wesley, Boston, 2005.
62. Andrea Schauerhuber. PhD Proposal. aspectUWA: Applying Aspect-Orientation to the Model-Driven Development of Ubiquitous Web Applications. http://www.wit.at/people/schauerhuber/AOSD06_SpringSchool_Schauerhuber-0302.pdf, March 2006.
63. 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 (AOSD'06), Bonn, Germany*, March 2006.
64. Andrea Schauerhuber, Manuel Wimmer, and Elisabeth Kapsammer. Bridging existing Web Modeling Languages to Model-Driven Engineering: A Metamodel for WebML. In *Proc. of Workshop on Model-Driven Web Engineering (ICWE'06), Standford Linear Accelerator Center, Palo Alto, California*, July 2006.
65. Wieland Schwinger, Christoph Grün, Birgit Pröll, Werner Retschitzegger, and Hannes Werthner. Pinpointing Tourism Information onto Mobile Maps A Light-Weight Approach. In *Proc. of ENTER 2006 - International Conference on Information Technology and Travel & Tourism*. Springer-Verlag, January 2006.
66. Dominik Stein, Stefan Hanenberg, and Rainer Unland. An UML-based Aspect-Oriented Design Notation. In *Proc. of the 1st International Conference on Aspect-Oriented Software Development (AOSD'02), Enschede, The Netherlands*, pages 106–112. ACM Press, April 2002.
67. 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.
68. Dominik Stein, Stefan Hanenberg, and Rainer Unland. On Representing Join Points in the UML. In *Proc. of the 2nd International Workshop on Aspect-Oriented Modeling with UML (UML'02)*, September 2002.
69. Dominik Stein, Stefan Hanenberg, and Rainer Unland. Expressing Different Conceptual Models of Join Point Selections in Aspect-Oriented Design. In *Proc. of the 5th International Conference on Aspect-Oriented Software Development (AOSD'06), Bonn, Germany*, pages 15–26. ACM Press, March 2006.
70. Dominik Stein, Jörg Kienzle, and Mohamed Kandé. Report of the 5th International Workshop on Aspect-Oriented Modeling. In *UML Modeling Languages and Applications: 2004 Satellite Activities, Lisbon, Portugal*, LNCS 2397, pages 13–22. Springer-Verlag, October 2004.
71. 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.
72. 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 International Conference on Software Engineering (ICSE'99), Los Angeles, California*, pages 107–119. IEEE Computer Society Press, May 1999.

73. The AspectJ Team. The AspectJ (TM) Programming Guide. http://eclipse.org/aspectj/doc/released/progguide/index.html, October 2005.
74. Maria Tkatchenko and Gregor Kiczales. Uniform Support for Modeling Crosscutting Structure. In *Proc. of the 6th International Workshop on Aspect-Oriented Modeling (AOSD'05), Chicago, Illinois*, March 2005.
75. Klaas van den Berg, José M. Conejero, and Ruzanna Chitchyan. AOSD Ontology 1.0 - Public Ontology of Aspect-Orientation. Technical Report AOSD-Europe-UT-01, AOSD-Europe, May 2005.
76. 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.
77. Christina von Flach Garcia Chavez, Alessandro Garcia, Uriá Kulesza, Cláudio San'anna, and Carlos Lucena. Taming Heterogeneous Aspects with Crosscutting Interfaces. In *Proc. of the 9th Brazilian Symposium on Software Engineering (SBES'2005)*, October 2005.
78. Dennis Wagelaar. A Concept-Based Approach for Early Aspect Modelling. In *Proc. of Early Aspects 2003: Aspect-Oriented Requirements Engineering and Architecture Design, Boston, Massachusetts*, March 2003.
79. Aida A. Zakaria, Hoda Hosny, and Amir Zeid. A UML Extension for Modeling Aspect-Oriented Systems. In *Proc. of the 2nd International Workshop on Aspect-Oriented Modeling with UML (UML'02)*, September 2002.
80. Gefei Zhang, Hubert Baumeister, Nora Koch, and Alexander Knapp. Aspect-Oriented Modeling of Access Control in Web Applications. In *Proc. of the 6th International Workshop on Aspect-Oriented Modeling (AOSD'05), Chicago, Illinois*, March 2005.

Appendix

In the following, we make the findings of our evaluation available at a glance. Table 1 to 6 summarize the evaluation results according to the six categories of criteria from our evaluation framework presented in Section 4.

	UML 2.0 (L.L)	UML Profile (L.E)	Platform-independent (L.I)	Domain Generality (L.DG)	Aspect Generality (L.AS)	Structural Diagrams (L.D)	Behavioral Diagrams (L.D)	Design Process (L.DP)	Design Guidelines (L.DP)
Ortiz et al.	~	✓	✓		✓	CD			
Conejero et al.	✓	✓				CD	SD		
Groher et al.		✓		✓	✓	PD, CD			
Pawlak et al.		✓		✓	✓	CD			
Stein et al.		✓		✓	✓	CD, ColD	SD, UCD		
Aldawud et al.		✓	✓	✓	✓	CD	SMD		✓
Clarke et al.				✓	✓	CD	SD	✓	
France et al.	✓	~	✓	✓	✓	CD	ComD		✓

Legend:

✓	supported
	not supported
~	not applicable / not decideable

CD	class diagram
PD	package diagram
ColD	collaboration diagram
SD	sequence diagram
UCD	use case diagram
ComD	communication diagram
SMD	state machine diagram

Table 1. Language

	Aspect (C.A)	Element Symmetry (C.ES)	Composition Symmetry (C.CS)	Relationship Symmetry (C.RS)	Adaptation Effect (C.AE)	Weaving (C.W)		Conflict Resolution (C.CR)				
						static	dynamic	avoid	detect	resolve	graphical	textual
Ortiz et al.	«aspect» from Class											
Conejero et al.	UML profile			✓								
Groher et al.	«abstract» from Package			✓								
Pawlak et al.	«aspect» from Class				✓							
Stein et al.	«aspect» from Class					✓				✓	✓	
Aldawud et al.	«aspect» from Class							✓		✓	✓	
Clarke et al.	«theme» Package (Template)	✓	✓	✓		✓		✓		✓	✓	✓
France et al.	Package (Template)			✓		✓	✓	✓	✓	✓	✓	✓

Legend:

✓	supported
	not supported

Table 2. ConcernComposition

	Struct. Join Point (S.SJp)		Behav. Join Point (S.BJp)		Explicite JoinPointModel (S.JpM)	Standardized JoinPoint Selection (S.JpS)	SimpleJoinPointSelection (S.SJpS)	CompositeJoinPointSelection (S.CJpS)	Selection Method (S.SM)			Relative Position (S.RP)			Abstr-action (S.A)	
	static	dynamic	static	dynamic					declarative	imperative	enumeration	before	around	after	high	low
Ortiz et al.			✓		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	
Conejero et al.	✓					✓	✓		✓		✓				✓	
Groher et al.	✓		✓			✓	✓		✓		✓	✓	✓	✓	✓	✓
Pawlak et al.			✓				✓	✓	✓		✓	✓	✓	✓		✓
Stein et al.	✓			✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
Aldawud et al.	✓					✓	✓		✓		✓				✓	
Clarke et al.	✓		✓	✓	✓	✓	✓		✓		✓					✓
France et al.	✓		✓	✓	✓	✓	✓		✓		✓					✓

Legend: ✓ supported; not supported

Table 3. AdaptationSubject

	Behavioral Adaptation (K.BA)	Structural Adaptation (K.SA)	Composite Adaptation (K.CA)	Abstraction (K.A)	
				high	low
Ortiz et al.	✓				✓
Conejero et al.		✓		✓	
Groher et al.	✓	✓		✓	✓
Pawlak et al.	✓	✓			✓
Stein et al.	✓	✓			✓
Aldawud et al.				✓	✓
Clarke et al.	✓	✓		✓	✓
France et al.		✓			✓

Legend: ✓ supported; not supported

Table 4. AdaptationKind

	Modeling Examples (M.E)	Applications (M.A)	Publications (M.P)	Topicality (M.T)
Ortiz et al.	✓		1	'05
Conejero et al.	✓	✓	1	'05
Groher et al.	✓		2	'05
Pawlak et al.	✓	✓	3	'05
Stein et al.	✓		4	'02
Aldawud et al.	✓		5	'05
Clarke et al.	✓		>5	'05
France et al.	✓		>5	'06

Legend:

✓	supported
	not supported
n	number of publications
'YY	year of most recent publication

Table 5. Maturity

	Modeling Support (T.M)	Weaving Support (T.W)	Code Generation (T.C)
Ortiz et al.	✓		
Conejero et al.	✓		
Groher et al.	✓		✓
Pawlak et al.	✓		✓
Stein et al.	✓		
Aldawud et al.	✓		✓
Clarke et al.			
France et al.	✓	✓	

Legend:

✓	supported
	not supported

Table 6. Tool Support