Position Paper on Aspect-Oriented Modeling: Issues on Representing Crosscutting Features

Dominik Stein, Stefan Hanenberg, and Rainer Unland
Institute for Computer Science
University of Essen, Germany
{dstein | shanenbe | unlandR}@cs.uni-essen.de

ABSTRACT

This paper deals with the design of crosscutting features in general and in the UML in particular. We postulate a couple of issues that we think an aspect-oriented designer is faced with when designing crosscutting features. These issues include the independent specification of the crosscutting details and of where to crosscut, the designation of crosscut or referenced elements, the characterization of the composition strategy, and the abstract representation of crosscutting features in the overall design model. Then, we contemplate ways to obey these issues using the UML. In that contemplation, we concentrate on the design of the details of crosscutting features, i.e., the separate specification of the elements that crosscut a given decomposition and of the join points at which that crosscutting takes place, as well as the distinct designation of elements belonging to the crosscut decomposition that are used (e.g., referenced) by the crosscutting elements.

1. INTRODUCTION

A major issue in aspect-orientation in general and in aspect-oriented modeling with UML [17] in particular is the identification of "aspects" as a concept. Currently, an "aspect" is essentially perceived as a mechanism to diminish crosscutting. Different objectives pursued by different aspect-oriented techniques make it hard to refine a common "aspect concept" from the various "aspect mechanisms". AOP [14] with AspectJ [2] focus on the supplementation of given base code with particular (non-)functionality, for example; MDSOC [21] with Hyper/J [13] aim at the composition of multiple independently developed concern models; and Adaptive Programming [16] with DemeterJ or DJ [1] seeks to implement collaborative behavior among classes in an arbitrary class hierarchy.

Looking for appropriate UML representations, we decided in our AODM [18] [19] to represent AspectJ's aspect construct by stereotyped UML classes because we felt that AspectJ's aspects structurally resemble UML classes. Clarke chose in her SODM [5] to represent hyperslices in Hyper/J (or rather, subjects in SOP [11]) by means of UML packages because UML packages may hold entire concern models (i.e., class hierarchies). For Adaptive Programming, one might propose to use UML collaborations to represent crosscutting entities, which seems particularly befitting when it comes to represent adaptive methods.

We think that the reasons presented for one or the other proposal are quite convincing in one or the other case – i.e., in the context of a particular aspect-oriented technique. We feel, however, that no "right" solution may be found as long as the aspect-oriented community has not defined a common and precise notion of what an "aspect" (as a concept) constitutes.

We believe that the community of aspect-oriented modelers using UML can help to develop that commonly accepted "aspect" notion since we are working with widely accepted and commonly used concepts, which are well-defined by the UML specification. Comparing these to the characteristics of aspects appears to be a promising way to unveil the spirit of aspects.

To get there, we disregard any questions concerning the appropriate representation of aspects for a moment and take a closer look on the design of crosscutting features. That way, we are trying to obtain important findings on the constitutional properties of aspects. After all, an aspect is merely the encapsulation of crosscutting. The real work is done by its crosscutting features. In this paper, we are not looking for the most matching representation for crosscutting features on some abstract (meta-)level, though. Instead, we consider suitable ways to graphically represent the details of crosscutting features. Therefore, we postulate a couple of issues that we think an aspect-oriented designer is faced with when designing crosscutting features in section 2. We consider these design issues to be specific to aspect-orientation (abbr., "ao-specific" issues) and therefore, require aspect-oriented modeling notations to provide solutions for these issues. In section 3, we contemplate ways to achieve these issues using the UML. To do so, we consider the UML concepts we used in our AODM and explain why we deem these concepts appropriate – or not appropriate – to design crosscutting features in the UML. Section 4 concludes and presents a short discussion on remaining issues in the overall design of aspect-oriented software.

2. AO-SPECIFIC DESIGN ISSUES

A major issue in designing and modeling is to concentrate at one problem at a time and abstract from others. A good design model only displays the information essential for a specific purpose and abstracts from others. To achieve this in aspect-oriented design we distinguish between the abstract declaration of crosscutting features and join point collections (Figure 1, B), which are part of the specification of aspect container elements (Figure 1, C), and the details of their specifications – that is, the specification of the exact elements that are to be injected into a given decomposition (Figure 1, A), and the specification of the set of locations (so-called "hooks", cf. [8]) where that injection has to take place (Figure 1, D). For a detailed differentiation of the kinds of crosscutting and hooks, see [10]. We believe that we can design the specifications of crosscutting features and join point collections independently

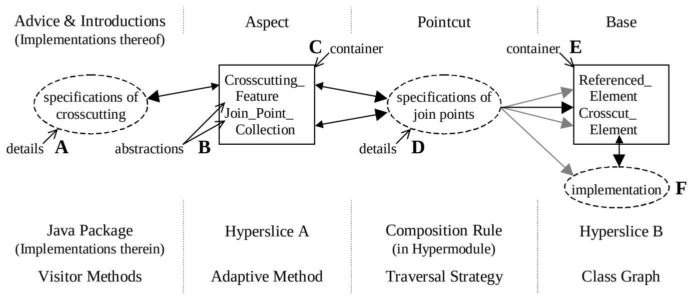

Figure 1. Aspect-Oriented Design Issues (non-UML diagram)

from each other as well as from the specification of the aspect element containing them. Furthermore, we deem their specification to be independent from the specification of the crosscut elements, their container elements, and their implementations (Figure 1, E and F) – contemplating that this separation of concern is one of the basic ideas of aspect-orientation [9]. In this section, we elaborate why we think that these issues can and should be designed separately in distinct design models, and where dependencies arise between these separate specifications. We are deriving a couple of postulates that we think a modeling notation must meet to provide for the design of crosscutting features.

2.1 Separation of Details and Hooks

When looking at the specification of crosscutting features in current aspect-oriented programming techniques, we identify the specification of the elements that crosscut a given decomposition (Figure 1, A) and the specification of the set of join points where that crosscutting takes place (Figure 1, D) to be two fairly independent issues that can be seen as distinct design problems. We do this because we contemplate that the reasoning on the crosscutting details can be accomplished not knowing where exactly that crosscutting is realized; and vice versa, we suppose we can reflect on the join points at which crosscutting should occur while neglecting what exactly is to be inserted at these points.

For example, when we implement a particular synchronization strategy we do not want to determine yet which objects it should be applied to. Oppositely, we could identify the actions that need to be synchronized first, and leave it to a third party to implement the synchronization strategy. In a third case, we have both a set of different synchronization strategies and a set of join point collections that designate actions to be synchronized. Now, we are able to combine crosscutting details and hooks of crosscutting in any manner, and thus we are set to realize any synchronization requirement by simple hooking the right implementation onto the right set of join points. In conclusion, the separate treatment of crosscutting details and points of crosscutting is a very important issue to achieve incremental programming (cf. [22]) – and after all that's what we're heading for in aspect-oriented programming. Reflecting on these observations, we demand:

A modeling language must allow

(1) the specification of the structural and behavioral elements that crosscut a given decomposition and

(2) the specification of the set of join points ("hooks") where that crosscutting takes place.

A modeling language must allow those specifications to

(3) be modeled separately by distinct design means so that they can

(4) be reused and combined in different contexts.

Even though we may reason on the specification of crosscutting details and the specifications of the hooks of crosscutting separately there certainly exist strong correlations between these two issues. In particular, dependencies arise from the charge of the latter to designate elements in the environment of the crosscut decomposition that are used by the former. In the following, we take a closer look on that designation issue.

2.2 Designation of Involved Elements

In general, crosscutting features do not interact with the crosscut decomposition solely by the elements they crosscut. Rather, they also interact with elements in the static or dynamic context of those crosscut elements. In a synchronization strategy, for example, we may want to coordinate a couple of actions depending on some attribute's value or some object's state (presupposing the attribute or object belongs to the crosscut decomposition). To do so, we need a reference to the attribute or object. Since all we have, though, is the reference to the join point, the reference to the referred element must be specified in relation to that join point.

In conclusion, aspect-oriented designers deal with three different kinds of elements: (1) Elements that are specific to the crosscutting concern and that are to be *added* to the crosscut decomposition; (2) elements that are specific to the crosscut concern and that are *referenced* by the crosscutting concern; and (3) elements that are "specific" to both concerns and that need to be *joined* in some way (these are the actual "crosscut elements"). Designers need to be able to designate all three kinds of elements in both the specification of the crosscutting details and in the specifications of the crosscutting hooks (new to aspect-orientated design is the designation of crosscut and referenced elements). This allows designers to reason on the their dependencies, e.g., if a particular join points specification designates all elements from a (eventually crosscut) decomposition that are needed by a particular crosscutting feature (i.e., its specification). Note that both the specification of the crosscutting details and the specifications of the crosscutting hooks need to specify how referenced and crosscut entities relate to each other – the former to fully describe the environment in which the crosscutting details are executed (recall that models are supposed to display all information for a specific purpose; note how this design principle compares to the "declarative completeness" requirement in Hyper/J), the latter to unambiguously identify them in the crosscut decomposition. Reflecting on these observations, we demand:

A modeling language must provide graphical means to

(5) designate elements – and their relationships to each other – belonging to the crosscut decomposition that are crosscut or referenced by the crosscutting concern

(6) in both the specification of the crosscutting details and in the specification of the crosscutting hooks.

Note that both crosscut and referenced elements must be present in the crosscut decomposition for the crosscutting details to execute. Since it is part of the join points specification to identify these elements in the crosscut decomposition, join points specifications in fact describe all characteristics a given decomposition must possess for a crosscutting feature to take effect. In doing so, join points specifications outline a particular perception of the crosscut decompositions (which possibly describes a dynamic snapshot of that decompositions). Note how that extra level of indirection in the design of crosscutting (resulting from making the perception of the crosscut entities explicit rather than relating crosscut entities and crosscutting features directly) helps designers to determine the exact deployment environment of their aspects and thus promotes aspect reuse.

Apart from the dependencies concerning the appropriate designation of crosscut and referenced elements, the specification of crosscutting details and the specification of crosscutting hooks further interdepend by the exact way the crosscutting details in the former are to be composed to the crosscut elements designated in the latter. In the next section, we consider what a modeling language must be capable of to delineate the way of composition.

2.3 Characterization of Composition

In aspect-orientation, "crosscutting" can mean addition, merging, or overriding of elements. Elements specified only by the crosscutting concern need to be *added* to the crosscut decomposition. Elements specified by both the crosscut and crosscutting concern need either to be *merged*, or the elements specified by the crosscutting concern *override* their counterparts specified by the crosscut concern. Hence, the type of crosscutting characterizes how the elements specified in the details (Figure 1, A) relate to the elements designated in the hooks (Figure 1, D). We believe that the type of crosscutting is confined by the semantic of the crosscutting details, and thus restrains the reading of the crosscutting details. Therefore, the type of crosscutting should be designed with the crosscutting details rather than with the join points specification.

For example, a synchronization strategy that simply defers the execution of actions is at best merged to (i.e., before the) crosscut actions. A synchronization strategy on the contrary, that possibly discards the execution of actions at all (depending on the behavior performed by other actions) needs to override the original actions. Hence, we need to determine in our design models describing the crosscutting details what type of crosscutting we are specifying. For the specification of join points (e.g., the actions to be synchronized), though, it should be of no importance what type of crosscutting is specified by the crosscutting details (e.g., if synchronization is accomplished by merging or overriding). Recall in this matter our explanations in section 2.1. So, reflecting on these observations we postulate:

A modeling language must provide means to

(7) attribute the specification of the crosscutting details with the type of crosscutting, which characterizes how the elements specified with the crosscutting details are to be composed to the crosscut elements designated by the join points specification.

Note that we abstract from reconciliation problems due to conflicting specifications of (crosscutting and crosscut) elements that are to be merged. We believe we can do that, since we regard the crosscutting of a particular crosscutting feature to a particular set of join points, which is rigidly defined by a join points specification. If the crosscutting details are to be assigned to a new (unknown) decomposition, designers are pointed right away to the elements that need to be reconciled (or adapted) by simply comparing the join points specification with the elements (meant to be crosscut and referenced) in the respective decomposition.

So far, we are regarding the details of the specifications of crosscutting (i.e., the specification of the elements that crosscut, of the set of locations where they crosscut, and of how they crosscut). However, when it comes to reason about the dependencies between these specifications in the overall design model, we call for succinct design notations that abstract from the specification details so our perception is not blurred. In the following, we examine how such abstractions should look like.

2.4 Abstractions

Just as any programming language, aspect-oriented programming languages make use of identifiers like attribute names or operation signatures to reference a particular crosscutting property of some aspect entity. These identifiers generally give an overall idea of what the identified crosscutting property is for and what is does. At best, programmers can guess from these identifiers whether the particular identifier refers to a specification of crosscutting details (Figure 1, A) or a join points specification (Figure 1, D); they can observe which elements are used (i.e., crosscut or referenced) within that specification; further, they can recognize how the crosscutting details are to be composed to the crosscut decomposition. That way, programmers grasp at a glance all information that is needed to appropriately deploy the identified property.

A design notation should provide likewise concise notations to identify crosscutting features and join point collections. These notations ought to get as close as possible to their corresponding programming language constructs to ease their handling and improve their comprehensibility. At the same time, they are required to display all information (mentioned above) that is essential to contemplate on the role of the crosscutting feature or join point collection in the overall design model. That way, the design notation enables designers to assess the dependencies between crosscutting features, join point collections, and crosscut and referenced elements on an appropriate level of abstraction. Reflecting on these considerations, we demand:

A modeling language must provide appropriate abstractions

(8) for the specification of the structural and behavioral elements that crosscut a given decomposition and

(9) for the specification of the set of join points ("hooks") where that crosscutting takes place.

By these abstractions, a modeling language must enable designers to apprehend

(10) what the specification is for and what is does (or allocates)

(11) what elements are used or exposed (i.e., crosscut and referenced) by the specification details;

(12) how the crosscutting details are to be composed to the hooks.

After we have postulated these twelve issues regarding the aspect-oriented design of crosscutting features, we now have a look how we can achieve these postulates in the UML. Due to limitations of space we concentrate on the discussion of the most appropriate depiction of the details of a crosscutting feature rather than on the most befitting representation of their abstractions. Note, though, that we consider current aspect-oriented programming languages to provide sufficient ideas to develop suitable abstractions for crosscutting features and, thus, the task is to find matching meta-level representations rather than appropriate visualizations.

3. AO-DESIGN WITH THE UML

In this section, we contemplate on the possibilities provided by the UML to specify the details of crosscutting features. We neglect any considerations on how to represent these cross-cutting features on a more abstract level (e.g., the "aspect level", the "use case level", etc.) or at meta-level. Instead, we examine how to exploit existing UML concepts to graphically depict the specification of the elements that (eventually) crosscut a given decomposition and the specification of the join points at which that decomposition is crosscut by distinct design means. Further, we evaluate how to designate the crosscut and referenced elements the specifications refer to within those design means. Our examinations consider the concepts we are using in our AODM. We discuss why we think that the details of crosscutting features should be defined in that way.

3.1 Details of Crosscutting

Designers need to be provided with graphical means to design structural or behavioral details of crosscutting (Figure 1, A) independently from the decomposition being crosscut (Figure 1, E and F) as well as from the specification of the join points (Figure 1, D). We believe that for the design of elements involved in structural and behavioral crosscutting (Figure 1, A) current modeling techniques in the UML are ample enough to model a wide range of crosscutting concerns. We consider possible shortcomings (e.g., in behavioral modeling; cf. [15]) to be not specific to aspect-oriented modeling and consequently we consider them to be beyond the scope of this paper. The question to solve, though, is to find a suitable container element that may hold the crosscutting details and provides a namespace for its elements. Design in the UML should always take place within a distinguished namespace to avoid name-clashes (e.g., during automated design validation). UML classifiers, UML packages, and UML collaborations are examples for (i.e., subtypes of) UML namespaces.

In the AODM, the details of structural and behavioral cross-cutting on a given decomposition are specified by means of UML collaborations. UML collaborations represent a mighty design concept because designers may specify both structural and behavioral aspects of a design model using the full range of UML's modeling facilities (cf. [3]). Therefore, no limits are set in collaborations for designing crosscutting elements. Besides these syntactical contributions, collaborations also contribute to the semantics of aspects. That is, because both collaborations and aspects describe a particular view on a system model with respect to a particular usage or concern. However, while aspects render (crosscutting) characteristics that are to be newly introduced into the system model, collaborations represent projections (or restrictions) on characteristics that already exist in a system model (cf. [17], p. 2-122). Due to this problem, we currently have to weave models (i.e., supplement the crosscut decomposition with the features specified in an aspect collaboration) before we may check the model for integrity, conformity, and well-formedness. However, we believe that weaving constitutes a core concept of aspect-orientation (like inheritance in object-orientation, for example). Therefore, a weaving relationship should be defined on the meta-level characterizing the weaving process and possible restrictions. We then advocate for a more tolerant interpretation of collaborations in the UML in the sense that collaborations may also specify new model elements, which are then implicitly added to designated base model elements by means of weaving relationships. Further, we propose to relinquish the conception that collaborations are generally supposed to describe some kind of usage or interaction. Instead, they should be seen as general descriptions of arbitrary aspects of some software system – may they be behavioral or purely structural.

We believe that UML collaborations are more appropriate to serve as namespace for crosscutting entities – compared to, for example, UML packages – because we feel that collaborations already have a somewhat crosscutting nature. That is, UML collaborations are intended to describe model elements in parts; they concede that model elements may have further properties defined somewhere else (a model element can participate in multiple collaborations at a time). That character particularly pertains to the design of structural properties. Packages, on the contrary, are meant as a general grouping mechanism where fully specified model elements are separated into disjunctive groups (a model element cannot belong to more than one package at a time). The model elements contained in a package can be imported, accessed or used "friendly" by elements from other packages, but they always stay distinct model elements – any joining or merging is not intended. On the other hand, we can borrow and reuse a lot from the existing UML specification on collaborations to develop means for aspect-oriented modeling. For instance, the UML specification already specifies an association between a classifier "role" in a collaboration and some "base" classifiers in the crosscut decomposition (cf. [17], p. 2-118). Furthermore, the specification provides us with collaboration runtime semantics (collaboration instance sets) and explicitly foresees collaborations templates – or parameterized collaborations (as we use them to designate crosscut elements; see next section).

3.2 Crosscut and Referenced Elements

A new ao-specific issue that needs to be solved by an aspect-oriented modeling language relates to the appropriate representation of elements belonging to the crosscut decomposition that are used (crosscut or referenced) within the crosscutting concern and, furthermore, how those elements are distinguished from the elements belonging to the crosscutting concern. Designers must be able to clearly distinguish these kinds of model elements in both the specification of the crosscutting details (Figure 1, A) and the specification of the crosscutting hooks (Figure 1, D). In the following, we derive suitable means to designate referenced and crosscut elements in the crosscutting details by investigating general UML practices. In the next section, we adopt the outcomes of this section for the designation of referenced and crosscut elements in join points specifications.

Looking for appropriated ways to distinguish crosscut and referenced elements as well as elements specific to the crosscutting concern (abbr., "cc-specific" elements) within the crosscutting details, we consider it useful in our AODM to differentiate between structural and behavioral crosscutting. In [20], we point out that in structural crosscutting the element being hooked onto and the element being affected coincide, while in behavioral crosscutting they do not. For example, structural crosscutting hooks onto a classifier and affects its signature by adding new attributes or operations. Behavioral crosscutting, however, hooks onto a message (e.g., an operation call) and affects the action sequence that is to be activated by that message. Our AODM reflects on this observation and hooks behavioral crosscutting onto UML links (being a structural element that is repeatedly passed during the execution of behavior), and defines the crosscutting details in standard UML collaborations (cf. [18] [19]). We believe that this disunion helps us to identify the most appropriate representation of referenced elements because (in case the behavior is merged to the original behavior, and does not override it) we can abstract from the crosscut element.

That is, within a design model describing crosscutting behavior (to be merged), we only deal with model elements specific to the crosscutting concern and those referenced from the crosscut decomposition. Considering that in our AODM the details of behavioral crosscutting are specified using a standard UML collaboration, we can compare that specification to the way ordinary behavior (e.g., of standard UML operations) is specified. When specifying the behavior of a standard operation in the UML, though, we designate referenced elements by means of UML's «parameter», «local», «global», «association», or «self» stereotypes to indicate why a given entity is visible to the particular behavior. Adopting this UML practice, we advocate to use these stereotypes to designate referenced or cc-specific model elements in a crosscutting design model, too – possibly complemented by new stereotypes like «advice» or «indirectneighbor» to suit needs of particular aspect-oriented programming techniques. Having found feasible means to designate *referenced* model elements within the specifications of crosscutting details, we now investigate how we can designate *crosscut* elements regarding the specification of structural crosscutting.

Unlike to behavioral crosscutting, structural crosscutting hooks onto and affects the same model elements (i.e., classifiers in the AODM). Therefore, we have to deal with crosscut, referenced and cc-specific model elements in the same design model. To distinguish the former one from the latter two, we chose to use UML's template mechanism and designate the crosscut elements as template parameters – similar to Composition Patterns [7]. That way, we exploit the "instantiation semantics" of UML templates as we may define new properties (like features and relationships) that are to be appended to the crosscut element by binding the elements to be crosscut to the template parameters. Note that "binding" in the UML does not result in visual introduction of elements (cf. [17], p. 2-73). In this concern, binding in the UML conforms to our understanding of weaving, which should be specified syntactically and semantically on the meta-level in a like manner (cf. section 3.1). Nevertheless, we call for a more liberal usage of binding (or rather weaving) since currently an entity may be bound to (i.e., crosscut by) one template at most (cf. [17], p. 2-55). Furthermore, a bound (i.e., crosscut) entity is currently not allowed to add any information of its own (cf. [17], p. 2-26). These rules conflict with weaving semantic of most aspect-oriented programming techniques, though.

3.3 Hooks of Crosscutting

A new demand to aspect-oriented modeling is the precise specifications of the join points where crosscutting takes place in the crosscut decomposition (Figure 1, D), and principal effort should be taken to find an appropriate answer. Apart from specifying the set of locations in the given decomposition at which crosscutting takes place, designers should also be able to designate those elements of that given decomposition that are of relevance to crosscutting concerns and could be used (i.e., referenced) in their models (see section 3.2). These elements should be modeled in their relationship to the (set of) locations of crosscutting.

The notational means to represent sets of locations are strongly related to the concepts that have been chosen to serve as location (hook) for structural and behavioral crosscutting. In our AODM, we identify UML classifiers to be hook for structural crosscutting and UML links to be hook for behavioral crosscutting. As graphical means to designate hooks in base design models, we introduce a new "crosscut" relationship that connects the hooks (i.e., a classifier or link) with the crosscutting details. In [20], we argue though that using a relationship to designate hooks quickly blur design models as soon as all crosscut relationships need to be shown (in particular when using type patterns or wildcards in pointcuts or introductions in AspectJ or in match or bracket rules in Hyper/J). Considering the design issues postulated above (see section 1), relationships impose even more problems. For example, relationships directly relate elements in the crosscutting concern to elements in the crosscut concern. Therefore, they cannot be considered really independent from the crosscutting details, and they hardly can be reused in another crosscutting. Furthermore, relationships cannot be aggregated to new relationships (as pointcuts in AspectJ may be combined to aggregate new pointcuts). Relationships further fail when it comes to specify requirements to the environment of a crosscut entity (e.g., in the specification of a traversal strategy in Adaptive Programming, or in behavioral crosscutting in AspectJ when crosscutting to a message is suppose to take place only if the message is sent by a particular type of object.).

In [20], we therefore suggest to use "join point designation diagrams", which represent collaboration templates that outline a particular perception of the crosscut decomposition (see Figure 2 for an example). In doing so, join point designation diagrams designate crosscut elements as template parameters (in the AODM that means either for classifiers or for message; in Figure 2, we designate the classifier `CrosscutType` as well as the message `CrosscutOp` to be crosscut elements). Furthermore, they nominate all elements from the crosscut element's environment that can be referenced within the crosscutting (cf. section 3.2). These elements are allocated in their relation to the crosscut element (for example, Figure 2 designates `SomeAssociatedType` whose operation `op1` may be invoked by the crosscutting details; further, it requires the `CrosscutType` to provide an attribute `att1` and two operations `op1` and `op2` so that they may be referenced

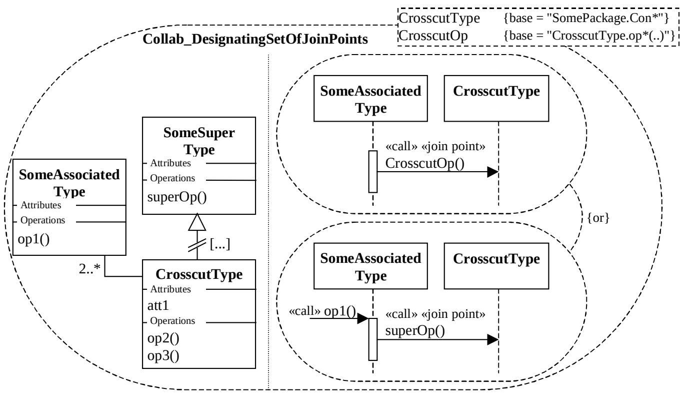

Figure 2. Join Point Designation Diagram (cf. [20])

within the crosscutting concern). To emphasize entities that are exposed to the crosscutting feature, we could deploy UML's special «parameter» stereotype for association ends. Hence, join point designation diagrams picture all model elements that a given decomposition must provide for a crosscutting to take place. This may even go beyond the designation of referenced and crosscut elements (in Figure 2, for example, we define the `CrosscutType` to be a (not necessarily direct) child of `SomeSuperType`; further, we require the crosscut message `CrosscutOp` to be sent from no other objects than objects of `SomeAssociatedType`, etc.). This helps designers to determine the exact deployment environment of their aspect and promotes aspect reuse.

4. SUMMARY & REMAINING ISSUES

In this paper, we postulated a couple of design issues specific to aspect-orientation and elaborated how we can use the UML to achieve these issues. Our explorations focused on the design of the details of crosscutting features, i.e., the specification of the elements that (eventually) crosscut a given decomposition, the specification of the join points at which that decomposition is crosscut, and the designation of elements from the crosscut decomposition that are used (i.e., crosscut or referenced) by the crosscutting feature. We explained what UML concepts we exploit in our AODM to achieve these issues. In conclusion, we think that the UML provides sufficient abstractions to specify crosscutting details (Figure 1, A) and to designate crosscut and referenced elements therein. These UML abstractions, however, may require some adjustment in their semantics and relaxation of their well-formedness rules to suit new necessities specific to aspect-oriented design. A whole new issue is the specification of the hooks of crosscutting (Figure 1, D) and it is a challenge to find new ways to appropriately represent that specification in a UML design model. Our future research is going to emphasize the development of thorough means for the detailed specification of sets of join points as well as their abstract representation on higher level of designs.

Beyond that, there remain a couple of further issues that we have not discussed in this paper. For example, we disregarded the appropriate representation of crosscutting features on any more abstract level (e.g., "aspect level" or "use case level") or on the meta-level in the UML. That is, we concentrated on the low-level design as the last step before implementation and skipped any design issues of higher levels (cf. [12]). Further, we neglected the appropriate representation of aspects as a whole in the UML. We did not deal with the indication of crosscut elements in the crosscut decomposition (see [20] for a proposition). These issues need to be resolved to obtain an overall aspect-oriented development process spanning from the problem analysis to the implementation of code (cf. [6]) that yields flexibility, comprehensibility, traceability, reusability, and so forth (cf. [4]).

5. REFERENCES

[1] Adaptive Programming, http://www.ccs.neu.edu/research/demeter/

[2] AspectJ, http://www.aspectj.org

[3] Booch, G., Jacobson, I., Rumbaugh, J., *The Unified Modeling Language User Guide*, Addison Wesley, Reading, MA, 1999

[4] Chitchyan, R., Sommerville, I., Rashid, A., *An Analysis of Design Approaches for Crosscutting Concerns*, AOD Workshop at AOSD'02 (Enschede, The Netherlands, Apr. 2002)

[5] Clarke, S., *Composition of Object-Oriented Software Design Models*, PhD Thesis, Dublin City University, Dublin, Ireland, Jan. 2001

[6] Clarke, S., Harrison, W., Ossher, H., Tarr, P. *Subject-Oriented Design: Towards Improved Alignment of Requirements, Design, and Code*. in Proc. of OOPSLA '99 (Denver, CO, Nov. 1999), SIGPLAN Notices 34(10), 325-339

[7] Clarke, S., Walker, R.J. *Composition Patterns: An Approach to Designing Reusable Aspects*. in Proc. of ICSE '01 (Toronto, Canada, May 2001), ACM, 5-14

[8] Elrad, T., Aksit, M., Kiczales, G., Lieberherr, K., Ossher, H., *Discussing Aspects of Aspect-Oriented Programming*, in: ACM Communications, Vol. 44(10), Oct. 2001, pp. 33-38

[9] Elrad, T., Filman, R.E., Bader, A., *Aspect-Oriented Programming*, ACM Communications, Vol. 44(10), Oct. 2001, pp. 29-32

[10] Hanenberg, St., Unland, R., *A Proposal for Classifying Tangled Code*, 2nd AOSD-GI Workshop (Bonn, Germany, Feb. 2002)

[11] Harrison, W., Ossher, H., *Subject-Oriented Programming (A Critique of Pure Objects)*, in Proc. of OOPSLA '93 (Washington DC, Oct. 1993), SIGPLAN Notices 28(10), pp. 411-428

[12] Harrison, W., Tarr, P., Ossher, H., *A Position on Considerations in UML Design of Aspects*, AOSD-UML Workshop at AOSD'02 (Enschede, The Netherlands, Apr. 2002)

[13] Hyper/J, http://www.alphaworks.ibm.com/tech/hyperj

[14] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, Ch., Lopes, Ch., Loingtier, J.-M., Irwin, J., *Aspect-Oriented Programming*, in: Proc. of ECOOP '97 (Jyväskylä, Finland, Jun. 1997), LNCS 1241, pp. 220-242

[15] Kleppe, A., Warmer, J., *Unification of static and dynamic semantics of UML*, Technical Report, Klasse Objecten, July 2001, http://www.klasse.nl/english/uml/ unification-report.pdf

[16] Lieberherr, K., *Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns*, PWS Publishing Company, Boston, 1996

[17] Object Management Group (OMG), *Unified Modeling Language Specification*, Version 1.4, Sep. 2001

[18] Stein, D., Hanenberg, St., Unland, R., *A UML-based Aspect-Oriented Design Notation For AspectJ,* in: Proc. of AOSD '02 (Enschede, The Netherlands, Apr. 2002), ACM, pp. 106-112

[19] Stein, D., Hanenberg, St., Unland, R., *Designing Aspect-Oriented Crosscutting in UML*, AOSD-UML Workshop at AOSD'02 (Enschede, The Netherlands, Apr. 2002)

[20] Stein, D., Hanenberg, St., Unland, R., *On Representing Join Points in the UML*, 2nd AOSD-UML Workshop at UML'02 (Dresden, Germany, Sept./Okt. 2002)

[21] Tarr, P., Ossher, H., Harrison, W., Sutton, S.: *N Degrees of Separation: Multi-Dimensional Separation of Concerns*. In: Proc. of ICSE'99 (Los Angeles, CA, USA, May 1999), ACM, 107-119

[22] Wegner, P., Zdonik, S., *Inheritance as Incremental Modification Mechanism or What Like is and Isn't Like*, in: Proc. of ECOOP'88 (Oslo, Norway, Aug. 1988), pp. 55-77