# Designing Aspect-Oriented Crosscutting in UML

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

## ABSTRACT
The Aspect-Oriented Design Model (AODM) is a new design model for the development of AspectJ programs with the UML. It extends existing UML concepts using standard UML extension mechanisms to provide aspect-oriented concepts as they are found in AspectJ. Further, the AODM specifies how an aspect-oriented design model may be transformed into an ordinary UML design model. It demonstrates how this weaving mechanism may be represented with help of existing UML concepts. The AODM facilitates the perception of aspect-oriented crosscutting in AspectJ programs and carries over the advantages of aspect-orientation to the design level. This paper explains the reasoning behind the AODM and demonstrates how it may be used to design aspect-oriented crosscutting in general.

## 1. INTRODUCTION
Aspect-oriented programming (AOP) [4] is a new software development paradigm that aims to increase comprehensibility, adaptability, and reusability by introducing a new modular unit, called "aspect", for the specification of crosscutting concerns. AspectJ [2] is a programming language that supports the aspect-oriented programming paradigm by providing new language constructs to implement crosscutting code. While with AspectJ a suitable aspect-oriented *programming* language is available, no feasible *modeling* language is presently at hand that supports the design of AspectJ programs. The Aspect-Oriented Design Model (AODM) [7] fills this gap. The AODM is a design model that extends the Unified Modeling Language (UML) [5] with aspect-oriented design concepts as they are specified in AspectJ, such as join points, pointcuts, pieces of advice, introductions, and aspects. The approach further reproduces AspectJ's weaving mechanism in the UML. Doing so, the AODM provides a complete set of means to help developers design and comprehend aspect-oriented crosscutting in AspectJ programs. Its application carries over the advantages of aspect-orientation to the design level and facilitates adaptation and reuse of existing design constructs. Its tight adherence to the UML assures an immediate understanding of aspect-oriented design models and enables rapid support by a wide variety of CASE tools. This paper explains the reasoning behind the AODM and demonstrates how it may be used to design aspect-oriented crosscutting in general.

The remainder of this work is structured as follows. Section 2 deals with the design of aspect-oriented crosscutting. It describes how crosscutting entities are designed with the AODM. Further, it explains how the entities being crosscut are designated. Section 3 presents a UML implementation of a weaving mechanism. That mechanism specifies how an aspect-oriented design model may be transformed into an ordinary UML design model. Section 4 gives a final review on the benefits and the problems of the AODM.

## 2. ASPECT-ORIENTED DESIGN
The design of aspect-oriented crosscutting includes the specification of crosscutting elements as well as the designation of elements being crosscut. Therefore, the AODM introduces new model elements with crosscutting effects on either type structure or behavior. The AODM provides these model elements with extra meta-properties to hold (so-called) weaving instructions. Weaving instructions specify which model elements in a base design model are to be crosscut. In the following, the specification of the crosscutting model elements is presented at first. Afterwards, the designation of the elements being crosscut (i.e., the specification of weaving instructions) is described.

### 2.1 Specification of Crosscutting Elements
AOP introduces a new modular unit, called "aspect", to serve as container for crosscutting elements. In the AODM, aspects are represented as UML classes of a special stereotype (called «aspect», in imitation to AspectJ's aspects). As such, they are instantiatable entities (analogous to aspects in AspectJ; note that aspects are instantiated "on need" rather than "on demand") which may contain crosscutting design constructs provided by the AODM (see sections 2.1.1 and 2.1.2 for details). Besides that, they may contain ordinary features such as attributes and operations. Further, classes of stereotype «aspects» may participate in association, generalization, and dependency relationships just like standard UML classes. Aspects are represented in the AODM as special stereotypes of standard UML classes, because other model elements which could serve as container for crosscutting elements (such as packages and collaborations) cannot be instantiated (and thus have no runtime semantic) and/or cannot own ordinary features (such as attributes and operations) or participate in relationships (such as associations, generalizations, and dependencies).

For example, Figure 1 shows two aspects which realize timing and billing features for a simple model of telephone connections (the telecom model is shown in Figure 2, right side; the example is take from [1]). The aspects contain crosscutting elements that alter the type structure and the behavior of the telecom model (by means of collaboration templates of stereotype «introduction» and operations of stereotype «advice», respectively; see sections 2.1.1 and 2.1.2 for further details). The **`Timing`** aspect further contains a "pointcut" element (which is a generalized "procedure" of pure weaving instructions; see section 2.2.1 for further explanations) that is used by the crosscutting operations **`advice_id04`** and **`advice_id06`** (these dependencies are explicitly indicated by means of «use» relationships). Apart from the crosscutting elements, the aspects contain ordinary attributes and operations. The aspects are supplied with tagged values that specify how the aspects are to be instantiated (in this example, both aspects are instantiated once per Java Virtual Machine). Finally, a «domi-

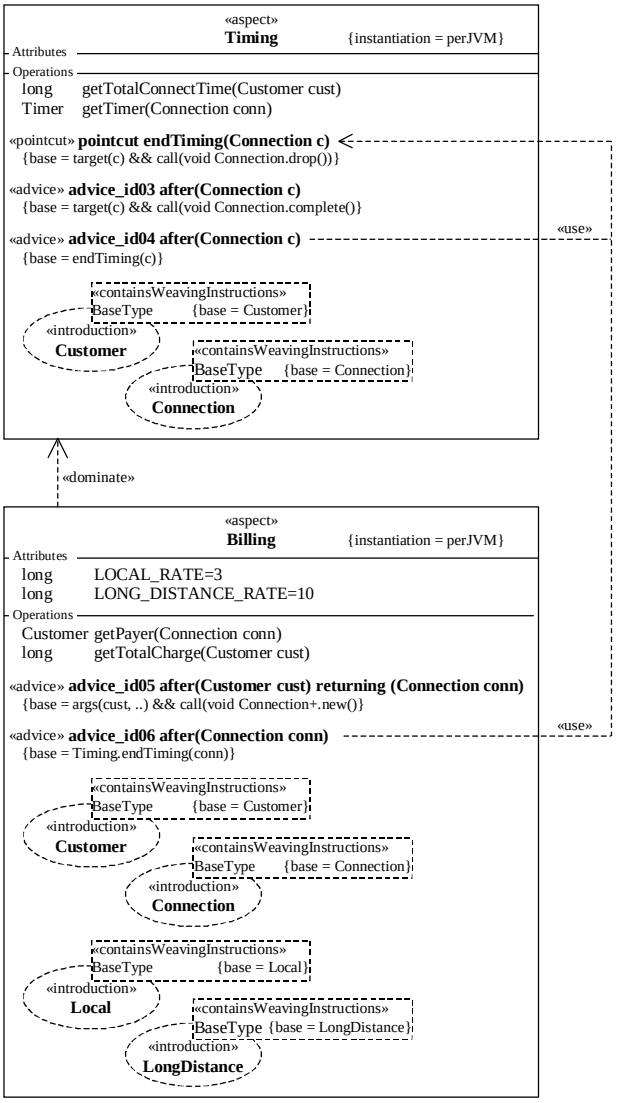


**Figure 1: An Aspect-Oriented Design Model**

nate» dependency indicates that the crosscutting effects on behavior of the **`Billing`** aspect precede the crosscutting effects on behavior of the **`Timing`** aspect.

### *2.1.1 Structural Crosscutting*

In the AODM, crosscutting of the type structure of a program (implemented, for example, by means of introductions in AspectJ) is specified by means of collaboration templates.

In standard UML, a collaboration describes a particular slice or a projection of a design model (cf. [5]). It specifies a set of instances together with their members and relationships (i.e., a structural context) and a set of interactions that describes some behavior performed by these instances. A template is a parameterized model element that is used to generate other model elements by binding its template parameters to actual arguments. The AODM adopts the collaboration concept to specify crosscutting features (such as attributes and operations) and relationships (such as association, generalization, or realization relationships) and the template concept to generate (i.e., insert) them into the existing base class structure.

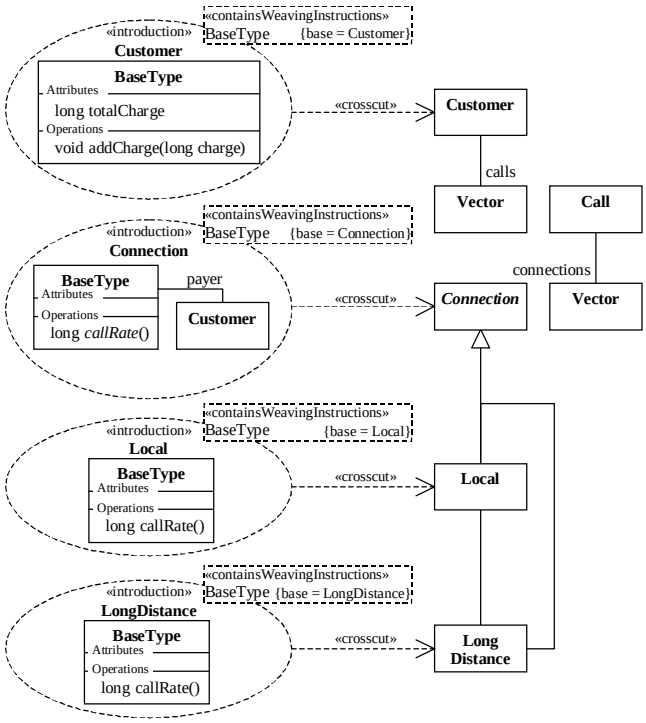


**Figure 2: Designing Structural Crosscutting**

The templates concept was chosen because with its help both features *and* relationships can be inserted into an existing base class structure. Other approaches (using stereotyped relationships, such as stereotyped generalization relationships between classifiers or stereotyped permission dependencies between packages) soon proved to be incapable to do so and thus were discarded.

Collaborations were chosen because they describe, rather then declare, features and relationships. That is, collaborations specify non-instantiatable *roles* of classifiers, together with the relationships (i.e., constraints) that must exist between them. So, during the instantiation (or binding) of a collaboration template, no instantiatable entity is generated in the base class structure. In contrast to this, other model elements that could be used to specify crosscutting features and relationships (such as packages and classifiers) define instantiatable entities. This means that during the instantiation of a template of these model elements, the base class structure would be supplemented with new instantiatable entities. This, however, does not go with the idea of crosscutting.

Within collaboration templates, designers may use the full range of UML's modeling facilities (cf. [3]) to specify crosscutting features (i.e., attributes and operations) and relationships (such as associations, generalizations, or constraining dependencies) that are to be introduced to the base class structure. The ADOM provides a special instantiation mechanism that ensures that the model elements generated from the collaboration template will not be ill-formed. In particular, this instantiation mechanism complements the existing base class structure according to the role specifications made in the collaboration template. Further, it extracts model elements that may not reside in actual collaborations (such as state machines and activity graphs, for example).

In the AODM, collaboration templates that specify crosscutting of the type structure of a design model are stereotyped with «introduction» (in imitation to AspectJ's introductions). In Figure 2 a

couple of such «introduction» templates are shown describing various features and relationships that are to be introduced to the base class structure. The crosscutting attributes, operations, and associations are specified using UML collaboration diagrams (in the UML, collaboration diagrams are used to describe the structural characteristics of a collaboration; cf. [3]). Each collaboration template has one template parameter (named "BaseType") which is bound to the base class being crosscut (in the AODM, binding is specified by means of special tags (named "base") that are attached to the template parameters (which are labeled to «containWeavingInstructions»); see section 2.2.1 for further explanations). The instantiation of collaboration templates of stereotype «introduction» is represented by means of crosscut relationships (see section 2.2.2 for further details on that relationship).

### *2.1.2 Behavioral Crosscutting*

In the AODM, crosscutting of the behavior of a program (implemented, for example, by means of advice in AspectJ) is visualized by highlighting messages in UML interaction diagrams. Behavioral crosscutting takes place at the *link* that is used to communicate the message (i.e., to communicate the stimulus that dispatches the action associated with the message). The crosscutting behavior itself is specified by standard UML collaborations.

Links were chosen because they represent principal points in the dynamic execution of a program (therefore, links correspond to join points in AspectJ). In the UML, links are instances of association relationships that connect two (or more) classifiers. No interaction can take place between two objects, if they are not connected by a link (i.e., if the sender object has no reference to the receiver object). Links can be interpreted as vertices in a simple object call graph. In that graph, control passes each link twice, once the control is passed "down" to the called instance, and once control flows back "up" again to the calling instance. In the AODM (in analogy to AspectJ), behavioral crosscutting may take place in either direction.

By utilizing links as fix points for behavioral crosscutting the AODM provides a sound instrumentality to crosscut any (inter)action that is communicated over a link, such as method and constructor invocations and object creations. In addition, the AODM specifies a couple of stereotypes to allow crosscutting of (intra)actions as well, such as method, constructor, and exception handler executions, class and object initializations, and field accesses. Usually, these (intra)actions are not communicated over links. However, they always come to pass in close conjunction with an (inter)action that is communicated over a link, or they may be designed as a "pseudo" (inter)actions that is communicated over a link. Crosscutting of (intra)actions then takes place at the link of the affiliated (inter)action.

Figure 3 demonstrates how the various crosscuttable actions are visualized in an (imaginary) UML interaction diagram. Note how (intra)actions that usually do not result in communications (such as method and constructor executions, object initializations, or field accesses) are indicated by special stereotypes (i.e., «execute», «initialize», «set», and «get»). Note further that the stereotypes «execute» and «initialize» are placed beneath the (inter)action (e.g., a «call» or «create» action) with which they are affiliated; the action stereotypes are arranged in a manner that signifies their order of processing.

In the AODM, the crosscutting behavior itself is specified in terms of standard UML interactions. The interactions are contained in standard UML collaborations that realize operations of special stereotype (called «advice», in imitation to AspectJ's advice). These operations set up a (formal and) structural context in which the crosscutting behavior is performed. This context may include instances that are part of the context of the behavior being crosscut. Such instances are specified as parameters. During the weaving process, invocation calls to these operations are inserted at every link at which the crosscutting behavior is to be executed.

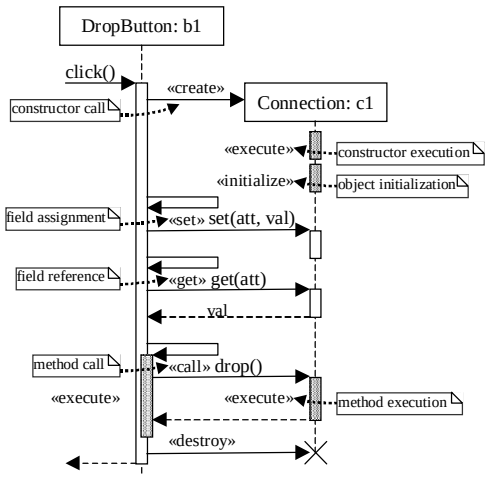


**Figure 3: Indicating Join Points in Interaction Diagrams**

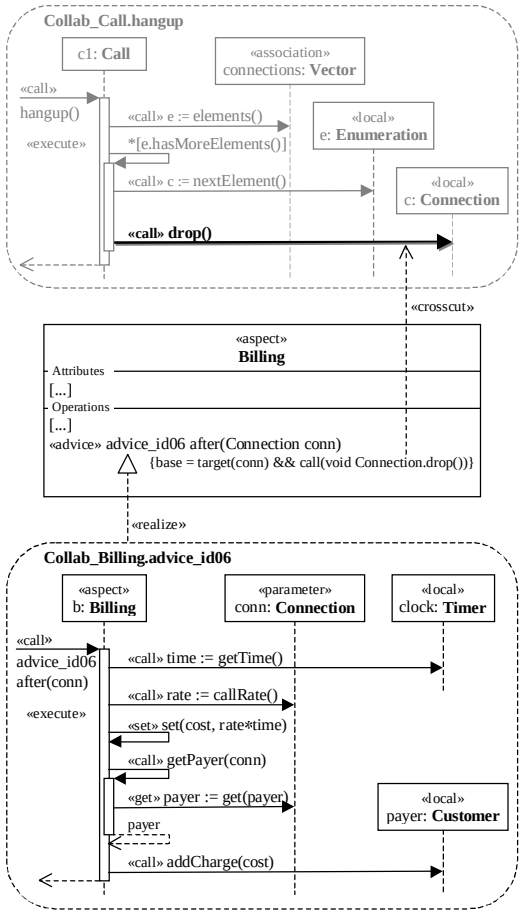


**Figure 4: Designing Behavioral Crosscutting**

Figure 4 demonstrates how crosscutting behavior is specified in the AODM using UML sequence diagrams (in the UML, sequence diagrams are used to determine the time ordering of interactions (i.e., messages) in a collaboration; cf. [3]). The collaboration realizes an operation of stereotype «advice», named **`advice_id06 after`**. In the AODM, advice are named with an unique "pseudo" identifier of style **`advice_id#`**, followed by a keyword indicating at which point relative to the execution of the action being crosscut the crosscutting behavior shall be performed (in imitation to AspectJ, these keywords are "before", "after", and "around"). The operation is adorned with a special "base" tag that contains a (so-called) link set expression. Link set expressions define the set of links at which the operation is to be executed and specify which instances contained in the dynamic execution context of the links are passed to the operation as parameters (for further explanations, refer to section 2.2.1). In Figure 4, the link set expression specifies that the operation crosscuts all method calls to the **`drop`** operation of the **`Connection`** class. This crosscutting effect is visualized by means of a crosscut relationship from the operation to the crosscut link (see section 2.2.2 for further details on that relationship). The link set expression specifies further that the receiver (or **`target`**) instance of the method call (i.e., the **`Connection`** instance in the upper collaboration in Figure 4) is passed to the operation as parameter (**`conn`**).

## 2.2 Designation of Crosscut Elements

In the AODM, the model elements being crosscut are specified by means of weaving instructions, which are stored in special meta-attributes in the aspect (e.g., with the collaboration templates of stereotype «introduction» or the operations of stereotype «advice»). The crosscutting effects of these weaving instructions are illustrated by means of a special crosscut relationship. This subsection explains how weaving instructions for structural and behavioral crosscutting may be specified in the AODM. Further, it introduces the new crosscut relationship.

### *2.2.1 Weaving Instructions*

In the case of *structural crosscutting* (rendered by collaboration templates of stereotype «introduction»; see section 2.1.1), weaving instructions define which classes in a base class hierarchy are to be crosscut. In other words, the weaving instructions specify the actual arguments (i.e., the base classes) to which the collaboration templates' parameters are bound. Note that this binding of the template parameters to actual arguments cannot be accomplished using the standard UML binding relationship, because then a base class could be crosscut at most once (the UML well-formedness rules state that "a model element may participate in at most one binding as a client" [5], i.e., as an actual argument). That is why the weaving (or binding) instructions are specified in special meta-attributes (named "base") which are attached to the collaboration templates' parameters. This meta-attribute contains no single type name but a general type pattern to allow binding to multiple base classes. Table 1 presents wildcards and operators that may be used to define those type patterns.

In the case of *behavioral crosscutting* (rendered by operations of stereotype «advice»; see section 2.1.2), weaving instructions specify which links in a base class collaboration are to be crosscut. In other words, the weaving instructions define the points in the dynamic execution of a program at which the operation of stereotype «advice» is to be run. Further, the weaving instructions specify which instances in the dynamic execution context of the links may be passed to the operation (i.e., its collaboration) realizing the crosscutting behavior as parameters. This time, the weaving instructions are stored in a special meta-attribute (named "base") with the method implementing the crosscutting behavior (i.e., implementing an operation of stereotype «advice»). The set of links being crosscut is specified by means of a link set expression. Table 2 contains examples of designators that may be used to select links and parameters in link set expressions (link set expressions in the AODM correspond to pointcut declarations in AspectJ; further designators may be found in [1]). In the AODM, weaving instructions are included into the implementation of the crosscutting operation (of stereotype «advice») because this way the weaving instructions become subject of inheritance. They may be declared abstract in super-aspects, and they may be overridden in subaspects (to gain from the latter possibilities, designers must use pointcuts; see next paragraph for further explanations).

| **Wildcards** | **Meaning** |
|---|---|
| **`*`** alone | all types |
| **`*`** in an identifier | any sequence of characters, not including "**`.`**" |
| **`..`** in an identifier | any sequence of characters starting and ending with "**`.`**" |
| **`+`** appended to an identifier | indicate all subtypes |
| **`[]`** appended to an identifier | indicate array types |
| **Operators** | **Meaning** |
| *TypePattern1* **`&&`** *TypePattern0* | all types in both type patterns |
| *TypePattern1* **`\|\|`** *TypePattern0* | all types in either type pattern |
| **`!`** *TypePattern* | all types not in *TypePattern* |

**Table 1: Defining Type Patterns (cf. [1])**

| **Designators (Examples)** | **Meaning** |
|---|---|
| **`call(`***Signature***`)`** | all links that are used to communicate a call action to an operation with signature pattern *Signature* (note: signature patterns may contain the same wildcards and operators as type patterns; see Table 1) |
| **`this(`***TypePattern***`)`**<br>**`target(`***TypePattern***`)`** | all links that are used for communications sent from/to instances matching *TypePattern*; that instance may be passed to the operation as parameter |
| **`args(`***TypePattern***`, ..)`** | all links that are used for communications that pass arguments matching *TypePattern*; the argument may be passed to the operation as parameter |
| **`within(`***TypePattern***`)`** | all links that are used for communications defined in classifiers matching *TypePattern* |
| **`cflow(`***LinkSetExpr***`)`** | all links that are located in the subtrees of the links designated by *LinkSetExpr* in a simple object call graph |
| **Operators** | **Meaning** |
| *LinkSetExpr1* **`&&`** *LinkSetExpr0* | all links in both link set expression |
| *LinkSetExpr1* **`\|\|`** *LinkSetExpr0* | all links in either link set expression |
| **`!`** *LinkSetExpr* | all links not in *LinkSetExpr* |

**Table 2: Defining Link Set Expressions (cf. [1])**

The AODM provides a special "*pointcut*" concept (in analogy to AspectJ's pointcut concept) to define generalized "procedures" of weaving instructions for behavioral crosscutting. These "procedures" of weaving instructions may be referred to in (other) weaving instructions – similar to the way ordinary operations are referred to in ordinary program code. The parameters of such a "procedure" refer to instances that are contained in the dynamic execution context of the links appointed by the weaving instructions. These instances may be passed to the crosscutting operations (of stereotype «advice») to which the weaving instructions are applied. Pointcuts may be overridden and may be declared abstract. In that case, their implementation (i.e., the weaving instructions) is (re)defined in subaspects. In regard to these characteristics, pointcuts are represented in the AODM as operations of special stereotype (called «pointcut»). In contrast to standard UML operations (and to crosscutting operations of stereotype «advice»), operations of stereotype «pointcut» specify purely weaving instructions. Pointcuts are visualized in the AODM by highlighting all links that are appointed by the pointcut's weaving instructions (i.e., by the link set expression in the pointcut's "base" meta-attribute). The AODM (analogous to AspectJ) does not provide means to define generalized "procedures" of weaving instructions for structural crosscutting.

### *2.2.2 Crosscut Relationship*

In the AODM, the crosscutting effects of weaving instructions are visualized by a special crosscut relationship. The AODM introduces that relationship in imitation of the extend relationship that is already provided by the UML [5]. It is no special stereotype of the extend relationship, though, because extend relationships may only exist between two use cases. Crosscut relationships, however, generally connect aspects and (base) classes.

Similar to extend relationships, the crosscut relationship is a directed relationship from an aspect to a (base) class indicating that the aspect *depends* on the presence of the (base) class in a way that the aspect's implementation or functioning requires the presence of the (base) class. At the same time, the relationship denotes that the (base) class is *affected* by the aspect in a way that the aspect's code is woven into the (base) class according to the weaving mechanism described in section 3. This characteristic makes (the extend relationship as well as) the crosscut relationship distinct from other relationships in the UML, such as the various kinds of dependency relationships.

Crosscut relationships and weaving instructions correlate to each other by a one-to-one mapping. So, the crosscutting effects of aspects may be determined either by drawing crosscut relationships or by specifying weaving instructions.

## 3. ASPECT-ORIENTED COMPOSITION

After crosscutting elements and weaving instructions are specified, an aspect-oriented design model may be transformed to an ordinary UML design model. For that purpose, the AODM implements a weaving mechanism, describing how collaborations realizing crosscutting and crosscut behavior may be merged (e.g., by tools) using standard UML means. Further, it specifies how weaving may be represented on a more abstract level.

### *3.1.1 Structural Crosscutting*

In the case of *structural crosscutting* (rendered by collaboration templates of stereotype «introduction»; see section 2.1.1), weaving is realized by instantiating collaboration templates. During the instantiation process, the base classes (specified in the template parameter's "base" tag) are supplemented with the features and relationships specified in the collaboration template so that the design model will not be ill-formed. This instantiation mechanism is defined on an informal (meta-meta-)level and cannot be rendered with help of standard UML concepts (just like the standard UML instantiation mechanism). However, it can be represented on a more abstract level using UML use cases. Please refer to [7] to see how this is accomplished.

### *3.1.2 Behavioral Crosscutting*

In contrast to weaving of structural crosscutting, weaving of *behavioral crosscutting* (specified by operations of stereotype «advice»; see section 2.1.2) can be rendered using standard UML concepts. Weaving of behavioral crosscutting is a matter of splitting and composing collaborations. Splitting always takes place in the collaboration describing the crosscut behavior, at the link at which the crosscutting behavior is to be executed. Depending on the direction in which the control flow shall is to be crosscut (see

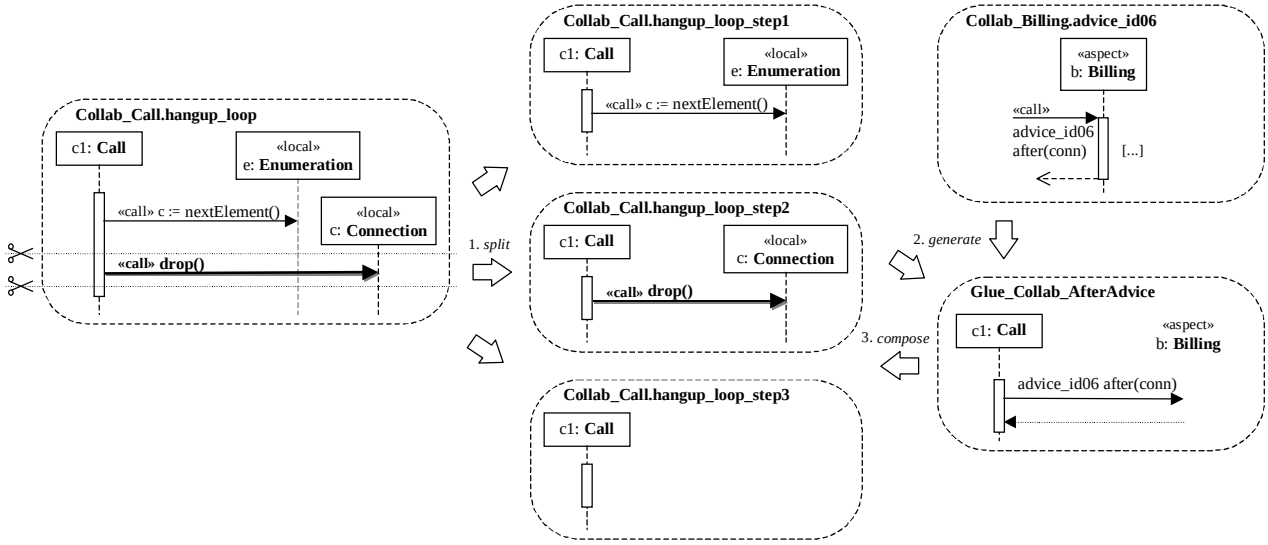


**Figure 5: Weaving Collaborations**

section 2.1.2), a new collaboration is composed to the split collaboration before or after (or "around") the link at which the collaboration was split. That collaboration invokes the operation (of stereotype «advice») that implements the crosscutting behavior. Composition of collaborations is accomplished according to standard UML rules (cf. [5]) by identifying and matching instances that participate in each collaboration being composed.

For example, Figure 5 demonstrates how the collaboration realizing the operation (of stereotype «advice») **`advice_id06 after`** from Figure 4 is woven into the collaboration that it crosscuts (for the sake of simplicity, the collaboration in Figure 5 describes only the (**`for`**-)loop of the collaboration specified in Figure 4). At first, the crosscut collaboration **`Collab_Call.hangup_loop`** is split into three collaboration fragments. One contains the behavior being crosscut, i.e., the link at which crosscutting has to take place (**`Collab_Call.hangup_loop_step2`**). The others describe the behavior before (...**`_step1`**) and after (...**`_step3`**) that link. Then, a glue collaboration (**`Glue_Collab_AfterAdvice`**) is generated that invokes the operation (of stereotype «advice») implementing the crosscutting behavior. This glue collaboration may be generated automatically from the instance sending the action being crosscut (i.e., **`c1: Class`**), from the operation (of stereotype «advice») being invoked (i.e., **`advice_id06 after`**), and from the aspect containing the invoked operation (i.e. **`b: Billing`**). The generated glue collaboration is composed to the collaboration fragments. Since the operation's signature contains the keyword "after", the glue collaboration is inserted behind the collaboration containing the behavior (i.e., the link) being crosscut (**`Collab_Call.hangup_loop_step2`**).

Just like the weaving mechanism for structural crosscutting, the weaving mechanism of behavioral crosscutting may be represented on a more abstract level using UML use cases. Please refer to [7] to see how this is accomplished.

# 4. SUMMARY AND REVIEW

The AODM demonstrates how the UML may be extended to allow the design of aspect-oriented crosscutting. The AODM uses standard UML extension mechanisms to derive aspect-oriented design concepts like they are defined in AspectJ (such as join points, pointcuts, pieces of advice, introductions, and aspects) from standard UML model elements. This way, the AODM provides means to specify crosscutting on type structure and on behavior of existing design models. Further, the AODM implements a weaving mechanism in the UML that generates standard UML design models from aspect-oriented design models according to the crosscutting specifications. Doing so, the AODM facilitates the perception of aspect-orientation in general and of AspectJ programs in particular. It carries over the advantages of aspect-oriented modularity (such as higher comprehensibility, adaptability, and reusability) to the design level. Its tight adherence to the semantics of AspectJ and to the specification of the UML allows an immediate and effective deployment by aspect-oriented software developers.

The tight adherence to AspectJ and UML, however, sometimes imposes some problems on the AODM specifications. One complication, for example, originates from the fact that collaboration templates of stereotype «introduction» (see section 2.1.1) are contained in the model element representing the aspect (i.e., in a class of stereotype «aspect»; see section 2.1). Doing so conflicts with the UML specification because templates may not be used directly in design models (instead, the model elements generated from the template can be used in models; cf. [5]). Therefore, the AODM provides a secondary instantiation mechanism for collaboration templates of stereotype «introduction» (beside the one that generates (i.e., introduces) the template's contents into the base class structure). That mechanism generates a collaboration inside the class containing the template and supplements the class with all elements needed so that the design model will not be ill-formed. This secondary instantiation mechanism occurs implicitly and is not explicitly visualized in aspect-oriented diagrams. Its only purpose is to avoid clashes with the UML specification. The complication could be resolved by excluding the design of structural crosscutting from the model element representing the aspect. This, however, would conflict again with standard AOP semantic.

Another quandary arises from the fact that pointcuts (see section 2.2.1) are represented as special stereotypes of standard UML operations (called «pointcut»), although they do not "specify a service that can be requested from an object to effect behavior" [5]. While this is perfectly all right with the UML specification version 1.3 (which states that "a stereotype shares the attributes, associations, and operations of its base class but it may have additional well-formedness constraints as well as a different meaning and attached values" [5]), it may become a problem in UML version 1.4 (which states that stereotypes define "virtual subclasses of UML meta-classes with new meta-attributes and additional semantics" [6]). It remains to be investigated to what extend pointcuts constitute "virtual" subclasses of operations (note that apart from this obstacle, the UML version 1.4 specifies a much better framework to implement the AODM extensions).

At last, the utility of the AODM needs to be verified in practical deployment. Based on the practical experience, revisions may become necessary. In that case, it needs to be examined to what extent the AODM specification can be amended to facilitate the design of aspect-oriented programs without clashing with the semantic of AOP and AspectJ and the specification of the UML. A possible amendment would be, for example, to summarize all crosscutting effects of one aspect on structure in a single collaboration template (having multiple parameters, each one specifying a distinct type pattern) rather than specifying one collaboration template for each type pattern (see Figure 2 for an example).

# 5. REFERENCES


[1] AspectJ Team. *The AspectJ Programming Guide.* http://aspectj.org/doc/dist/progguide/index.html, Sep. 2001

[2] AspectJ, http://www.aspectj.org, Ver. 1.0b, Sep. 2001

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

[4] 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, 220-242

[5] Object Management Group (OMG). *Unified Modeling Language Specification.* Version 1.3, Mar. 2000

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

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