

# Incorporating Aspects into the UML

Mark Basch  
University of North Florida  
Department of Computer and  
Information Sciences  
Jacksonville, FL 32224-2645  
(904) 620-2985  
basbm0001@unf.edu

Arturo Sanchez  
University of North Florida  
Department of Computer and  
Information Sciences  
Jacksonville, FL 32224-2645  
(904) 620-2985  
arturo@acm.org

## ABSTRACT

As aspect-oriented programming techniques move into mainstream use, it is likely that more software developers will be modeling systems with aspect-oriented features using the Unified Modeling Language. And with the Object Management Group considering revised standards for UML 2.0, it is an appropriate time to consider UML standards for aspect-oriented elements. This paper considers how aspect-oriented systems should be modeled in the UML, including a proposal for new UML elements that will be necessary to accommodate aspects.

## Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – *object-oriented design methods, computer-aided software engineering (CASE), modules and interfaces.*

## General Terms

Design. Documentation. Standardization. Languages.

## Keywords

Modeling Aspects. Aspect-Oriented Programming. UML. Software Process.

## 1. INTRODUCTION

Over the past decade, most of the research into aspect-oriented programming has been focused on developing programming techniques to incorporate aspect-oriented principles into the final realization of software systems. However, until very recently, little attention was given to how to model aspects in the software development process. After all, without programming languages that support aspects, there was no reason to consider aspect-oriented development.

However, with the advancement of AspectJ [12] and other aspect-oriented programming languages, we may be reaching the point where aspects will be incorporated into a wider family of large-scale software projects. The analysis and design of those projects will require modeling standards for aspects in the Unified Modeling Language. Since the Object Management Group is considering proposals for a new release of the UML standards from current version 1.4 to version 2.0 [14], it seems

appropriate to propose new modeling elements that accommodate aspects.

This proposal was submitted as part of a master's thesis [5] (submitted to and accepted by the University of North Florida's Department of Computer and Information Sciences in December 2002) that considered where aspects are unveiled during the software development process and how those aspects can be designed during the software development process using UML.

## 2. BACKGROUND

In the thesis, we considered the real estate project discussed by Kulak and Guiney in their book [13]. As we took this project through the development phases prescribed by the Unified Process and modeled features of the system as aspects, we determined that existing UML modeling elements were insufficient to design the system. As explained in Section 3, we propose the inclusion of new modeling elements in UML 2.0 to accommodate aspects.

Our proposal assumes the project is being developed using AspectJ, although the UML elements are easily adaptable to any other aspect-oriented development environment.

We use the terminology originally described by the developers of AspectJ in their 1997 paper on aspect-oriented programming [11], in which system modules are known as “components” and “aspects” are modules that cross-cut the components throughout the system.

In AspectJ, the term “join points” has a specific meaning but in previous research on aspect-oriented programming, join points have been used with various meanings. For our purposes, join points will refer to connection points where aspects cross-cut components.

The use of UML elements in this proposal is based on procedures outlined by the Object Management Group's specifications for the current version of UML [15], on previous UML guidelines described in various reference manuals, and on previous research submitted on aspects in the UML.

### 3. ASPECTS IN THE UML

In developing our proposal for modeling aspects in the UML, we established two principles suggested by previous research last fall. First, as suggested in [18], we established the principle that an aspect should be considered as its own encapsulated module separated from the system as a whole, and also separated from other aspects. Secondly, as suggested in [20], we determined that a key objective in modeling the system would be detailing the join points where aspects cross-cut components.

While some researchers have suggested diagramming methods that extend current UML elements, we found that the existing mechanisms do not adequately describe the unique nature of aspects. So we have proposed incorporating new elements to represent join points and to represent the aspects themselves (for the purpose of showing cross-cutting relationships with components).

#### 3.1 Aspect Packages

In order to accommodate the separation of aspects and components, we use UML packages. Each aspect is encapsulated within its own package, and all the functionality of the aspect can be modeled within the package. This should include class diagrams and interaction diagrams at a minimum, as well as anything else needed to specify the aspect's functionality.

According to UML specifications [15], the purpose of a package is to provide a grouping mechanism. It further states that "the package construct can be used for organizing elements for any purpose; the criteria to use for grouping elements together into one package are not defined within UML." In this case, the purpose of using packages is to separate and encapsulate the aspects.

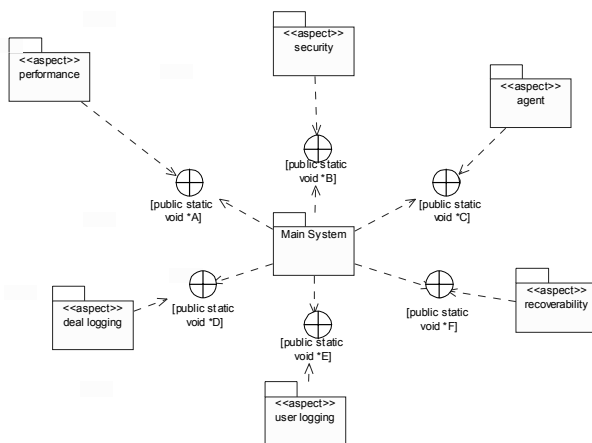


Figure 1: Aspect packages with join points.

Figure 1 illustrates a system in which all the major functionality is contained in one package called Main System, while each of six aspects is separated into its own package. Each aspect package is labeled with the stereotype <<aspect>> to distinguish it from a component package.

The package diagram also indicates that the aspects will cross-cut components in the main system at certain join points. Circles with a cross inside (to indicate their cross-cutting nature) indicate the join points. The definitions of the join points are contained in brackets below the circles. The definitions in Figure 1 correspond to AspectJ pointcut definitions, which are used to describe where the aspects will cross-cut system components. However, the syntax can be easily adapted for definitions required by other aspect-oriented languages.

An important modeling consideration here is that the join points are modeled outside both the aspect and component packages. That way, these points belong to neither concern individually but are used to indicate the cross-cutting relationship of an aspect and a component. The package diagram gives an overview of where to find the join points in the system.

The links from the aspects and components to the join points show a dependency relationship between the concerns and the join points. The cross-cutting features of the system are dependent upon both the implementation of the join points and the underlying aspect-oriented programming model.

#### 3.2 Class Diagrams for Aspects

Within the aspect package, we will use a class diagram to show which component classes are cross-cut by the aspect.

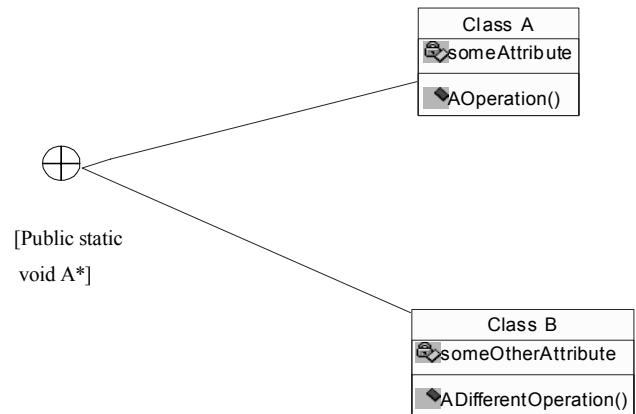


Figure 2: Aspect class diagram

Figure 2 models a class diagram for a generic aspect. The relationships are drawn between a join point and the component classes, since the aspect's relationship with the component occurs through the join point. The definition of the join point in Figure 2 shows that this aspect will cross-cut at methods beginning with the letter "A."

With this diagram, we can see at a glance which components are cross-cut by the aspect.

The class diagrams in the main system package would not model the relationship between aspects and components, since there is no aspect class that can be modeled and no additional insight into the relationships between aspects and classes could be drawn from a components class diagram. Interaction diagrams in the main system package will demonstrate those relationships

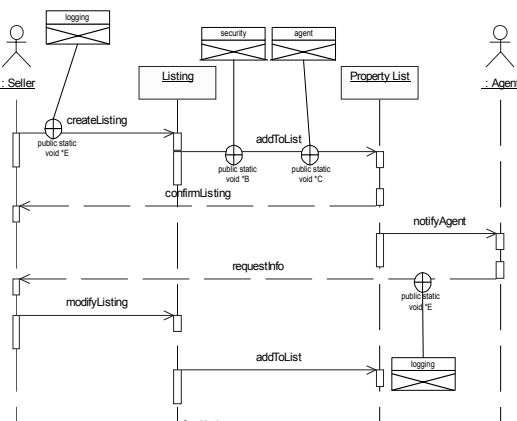
### 3.3 Interaction Diagrams for Aspects

Interaction diagrams – that is, sequence and collaboration diagrams – are a key design element when modeling aspects in the UML. The diagrams describe the specific relationships between aspects and components and the points at which they cross-cut. This would be modeled in interaction diagrams within the main system package.

The aspect packages would also include interaction diagrams that display the self-contained sequence of events that take place when an aspect is invoked. Since the events of the aspect do not involve a call to a component, the interaction diagrams encapsulated within an aspect package do not need to show relationships with the components. However, a circumstance can arise in which an aspect cross-cuts another aspect. In that case, the aspect interaction diagrams would need to show a relationship with elements from another aspect package.

To demonstrate the interaction of aspects and components, we used an example from the real estate project, in which a use case described a sequence of events in which a seller of real estate added a property to the agency's list of properties for sale. We used a sequence diagram to model the flow of events, which included four aspects which cross-cut components during the sequence.

We attempted to create the sequence diagram using existing UML elements but found they were insufficient to accurately model the interaction. A sequence diagram consists of a series of objects with messages drawn to show relationships between the objects. When aspects are introduced into the system, it is most likely that the aspects would be invoked during the messages, which correspond to methods of an object-oriented implementation. However, the UML makes no provision for introducing an element that connects directly to a method. The only connections in a sequence diagram are the messages that connect two objects. So we introduce another new modeling element, representing an aspect, to solve the problem.

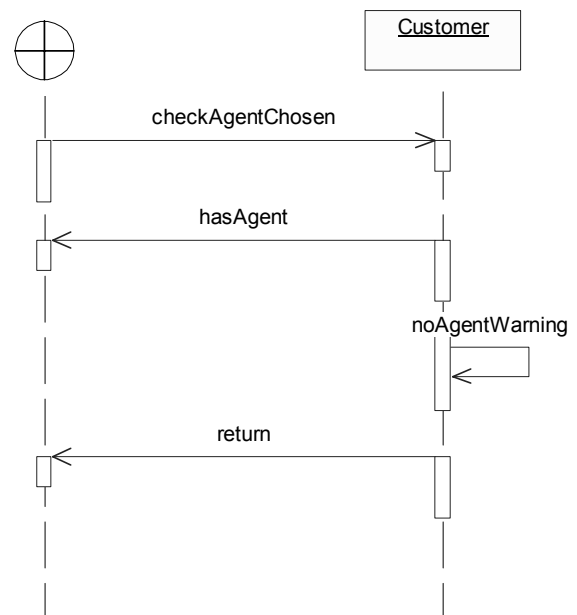


**Figure 3: Sequence diagram with aspects and join point elements**

The sequence diagram in Figure 3 represents a simple yet effective approach to the problem. The boxes with crossed lines on the bottom half (indicating cross-cutting) represent aspects, and the circles with the crosses inside, as indicated previously, are join points. The join points reside on the line of the message where they occur in components in the main system, and they are connected to the aspects which cross-cut the components by a simple association.

We have added new elements for aspects and join points because they are unique elements that cannot be modeled accurately with existing UML elements. In Figure 3, the aspects and join points are clearly represented by unique symbols that distinguish them from other modeling elements. The distinct elements would allow for implementation of the system using various aspect-oriented languages.

The interaction diagrams in the main system package, such as the sequence diagram in Figure 3, would be perhaps the most important artifact that models the cross-cutting relationship between aspects and components. But aspect-oriented programming also creates the need for a second distinct type of sequence diagram that would be included in the aspect packages, modeling the behavior of the aspect itself.



**Figure 4: Aspect sequence diagram**

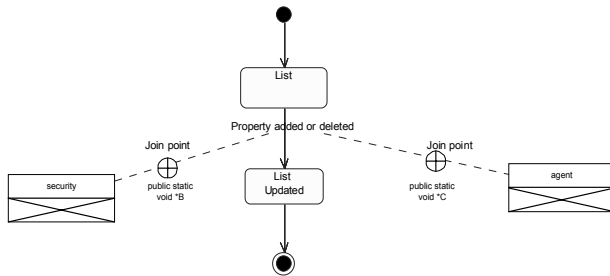
Figure 4 displays the sequence diagram for an aspect created in the real estate system. This aspect flashes a warning to users who attempt to make a transaction without using a real estate agent, warning them that they might need an agent to ensure proper execution of the transaction.

There is one feature of this diagram that distinguishes it from typical sequence diagrams. Normally in a sequence diagram, an actor representing a user of the system initiates the action. But when an aspect is invoked, there is no outside actor involved.

The join point, an internal feature of the system, initiates the action, and at the end of the sequence, control of the system is returned to the join point and the main system's program moves on. So we insert a join point into the sequence diagram to show how the sequence begins and ends.

### 3.4 Aspects and Statechart Diagrams

According to some UML manuals [3] [6], statechart diagrams are generally used to model the behavior of a single object and are not as useful to model interactions of components. So designers may not want to use the diagrams to model the interactions of aspects and components. However, according to UML specifications [15], state machines "can be used to model the behavior of individual entities (such as, class instances) or to define the interactions (such as, collaborations) between entities." This suggests statechart diagrams could be another tool used to model cross-cutting relationships between aspects and components. So it is worthwhile to consider a UML specification for such a diagram.



**Figure 5: Statechart diagram with links to aspect**

Figure 5 displays a statechart of a property list used by the hypothetical real estate agency, which transitions to an updated state every time a new property is added to the list or a property is deleted. The transition involves two cross-cutting aspects, a security aspect that determines if the user is authorized to modify the list and the previously-mentioned agent warning to check if the user is being assisted by a real estate agent.

The statechart diagram indicates the interaction with links to the two aspects at the transition points, with the join points indicated on the association lines.

### 3.5 Aspects and Other UML Diagrams

The previous sections cover the types of UML diagrams that would need to be used to adequately model a system developed with aspect-oriented techniques. The only other type of diagram that a developer might consider would be an activity diagram, which is used to analyze a sequence of events. But as noted in the UML specifications [15], the activity diagram's "primary focus is on the sequence and conditions for the actions that are taken, rather than on which classifiers perform those actions." Since in the consideration of aspects we are particularly interested in the elements (classifiers) that perform certain actions, this would indicate activity diagrams are not useful for modeling the interaction of an aspect cross-cutting a component.

## 4. RELATED WORK

In addition to the principles of encapsulating aspects separately [18] and focusing on join points [20], we looked at a number of proposals for modeling aspects with UML that have been put forth in the past year.

As in our proposal, it is argued in [7] that packages should be a major design element of aspects, as they allow for separation of concerns and re-usability of aspects. In [21], a proposal is made for simple icons to represent aspects and join points.

Several researchers have also made proposals that clearly distinguish elements of aspects in the UML, making them "first-class citizens"[10]. Proposals in [9] and [16] call for specific designations of AspectJ pointcuts.

Although we feel aspects are best modeled with a new set of UML elements, several researchers have proposed ways to use the UML extensibility mechanisms to model aspects [8][19].

Proposals have been made to model aspects using just about every available type of UML diagram, including statecharts [1], activity diagrams [4] and use case diagrams [2].

## 5. CONCLUSIONS

While incorporating aspects into the software development process can be handled seamlessly, we found in our research that modeling those aspects in the UML, using existing standards and tools, is more complex and needs more refinement. We have proposed the inclusion of two new modeling elements to incorporate aspects in the upcoming revision of the UML from version 1.4 to version 2.0. Those two elements represent the concepts of join points, which are the points at which aspects cross-cut components in the main program, and the aspects themselves, for the purpose of modeling the interaction between aspects and components.

We have also proposed that modeling the functionality of aspects should be encapsulated into distinct UML packages, for the purposes of maintaining separation of concerns. Within the packages, we can use class diagrams to model the relationship between aspects and components of the main system, and sequence or collaboration diagrams to model the functionality of an aspect. Interaction diagrams (sequence or collaboration diagrams) will also be the key diagrams used in the main system packages to model the relationship between components of the system and aspects.

At the present time, few major systems are under development using aspects. But as more software developers incorporate the principles of aspect-oriented programming into projects and aspect-oriented programming languages such as AspectJ are more fully refined, it will become more important to establish standards for developing aspects during the software development life cycle and for modeling them in the UML.

## 6. REFERENCES

- [1] Aldawud, O., A. Bader and T. Ellrad, Weaving with Statecharts, Aspect-Oriented Modeling with UML workshop at the 1<sup>st</sup> International Conference on Aspect-Oriented Software Development, April 2002.

- [2] Araujo, J., et al, Aspect-Oriented Requirements with UML, Aspect Modeling with UML workshop at the Fifth International Conference on the Unified Modeling Language and its Applications, September 2002.
- [3] Arlow, J., and I. Neustadt, UML and the Unified Process, Pearson Education Ltd., London, 2002.
- [4] Barros, J.P. and L. Gomez, Activities as Behaviour Aspects, Aspect Modeling with UML workshop at the Fifth International Conference on the Unified Modeling Language and its Applications, September 2002.
- [5] Basch, M.A., Incorporating Aspects into the Software Development Process in the Context of Aspect-Oriented Programming, master's thesis submitted to the University of North Florida, December 2002.
- [6] Fowler, M., UML Distilled, Second Edition, Addison Wesley Longman Inc., Reading, Mass., 2000.
- [7] Herrmann, S., Composable Designs with UFA, Aspect-Oriented Modeling with UML workshop at the 1<sup>st</sup> International Conference on Aspect-Oriented Software Development, April 2002.
- [8] Ho, W., et al, A Toolkit for Weaving Aspect-Oriented UML Designs, Proceedings of the 1<sup>st</sup> International Conference on Aspect-Oriented Software Development, 1, 1, (April 2002), pp. 99-105.
- [9] Jezequel, J., et al, From Contracts to Aspects in UML Designs, Aspect-Oriented Modeling with UML workshop at the 1<sup>st</sup> International Conference on Aspect-Oriented Software Development, April 2002.
- [10] Kande, M.M., J. Kienzle and A. Strohmeier, From AOP to UML – A Bottom-Up Approach, Aspect-Oriented Modeling with UML workshop at the 1<sup>st</sup> International Conference on Aspect-Oriented Software Development, April 2002.
- [11] Kiczales, G., et al, Aspect-Oriented Programming, Proceedings of the 11<sup>th</sup> European Conference on Object-Oriented Programming, 1, 1, (June 1997).
- [12] Kiczales, G., et al, Getting Started with AspectJ, Communications of the ACM, 44, 10 (October 2001), pp. 59-65.
- [13] Kulak, D., and E. Guiney, Use Cases, Requirements in Context, ACM Press, New York, 2000.
- [14] Object Management Group, Final Report of the UML 1.4.1 RTF, June 26, 2002.
- [15] Object Management Group, OMG Unified Modeling Language Specification, Version 1.4, September 2001.
- [16] Pawlak, R., et al, A UML Notation for Aspect-Oriented Software Design, Aspect-Oriented Modeling with UML workshop at the 1<sup>st</sup> International Conference on Aspect-Oriented Software Development, April 2002.
- [17] Rumbaugh, J., I. Jacobson and G. Booch, The Unified Modeling Reference Manual, Addison Wesley Longman Inc., Reading, MA, 1999.
- [18] Sapir, N., S. Tyszberowicz and A. Yehudai, Extending UML with Aspect Usage Constraints in the Analysis and Design Phases, Aspect Modeling with UML workshop at the Fifth International Conference on the Unified Modeling Language – the Language and its Applications, September 2002.
- [19] Stein, D., S. Hanenberg and R. Unland, A UML-based Aspect-Oriented Design Notation for AspectJ, Proceedings of the 1<sup>st</sup> International Conference on Aspect-Oriented Software Development, 1, 1, (April 2002), pp. 106-112.
- [20] Stein, D., S. Hanenberg and R. Unland, On Representing Join Points in the UML, Aspect Modeling with UML workshop at the Fifth International Conference on the Unified Modeling Language and its Applications, September 2002.
- [21] Zakaria, A.A., H. Hosny and A. Zeid, A UML Extension for Modeling Aspect-Oriented Systems, Aspect Modeling with UML workshop at the Fifth International Conference on the Unified Modeling Language – the Language and its Applications, September 2002.