Extending UML with Aspects: Aspect Support in the Design Phase

Junichi Suzuki
Department of Computer Science,
Graduate School of Science and Technology,
Keio University
Yokohama City, 223-8522, Japan
+81-45-563-3925
suzuki@yy.cs.keio.ac.jp

Yoshikazu Yamamoto
Department of Computer Science,
Graduate School of Science and Technology,
Keio University
Yokohama City, 223-8522, Japan
+81-45-563-3925
yama@cs.keio.ac.jp

Abstract
Aspect-Oriented Programming (AOP) has been considered a promising abstraction principle to reduce the problem of code tangling and make software structure clean and configurable. This paper addresses the aspect support in the design level while it has been focused mainly in the implementation/coding phase. We propose an extension to Unified Modeling Language (UML) to support aspects properly without breaking the existing UML specification. This allows developers to recognize and understand aspects in the design phase explicitly. Also, we propose a XML-based aspect description language, UXF/a. It provides the interchangeability of aspect model information between development tools such as CASE tools and aspect weavers.

1. Introduction
Aspect-Oriented Programming (AOP) [1, 2] is a promising abstraction principle to reduce the problem of code tangling and make software structure clean and configurable [1, 2]. It has been applied to various domains such as object interaction, memory management, persistence, distribution, fault tolerance, concurrency, etc. An aspect in the sense of AOP is a means to specify policy or strategy for software functional components, which is orthogonal to them. Separating aspects from functional components avoids that non-functional components cross-cut and therefore tangle groups of functional components. This explicit separation of concern allows to manage software complexity well and improve its quality by increasing modularity beyond an object in the sense of Object-Oriented Programming (OOP).

This paper addresses the aspect support in the design level, while it has been focused mainly in the implementation/coding level. We propose an extension to Unified Modeling Language (UML) to support aspects properly without breaking its existing metamodel specification. We also propose a XML-based aspect description language called UXF/a (UML eXchange Format, aspect extension). It provides the interchangeability of aspect model information between various development tools such as CASE tools and aspect weavers.

The remainder of this paper is organized as follows. Section 2 describes the benefits of capturing aspects in the design level, and expressing them in the XML format. Section 3 describes our extension to the UML metamodel. Section 4 presents our aspect description language based on XML. We conclude with a note on future work in Section 5 and 6.

2. Aspects in the Design Level
This section describes our motivation to support aspects in the design level and describe them in XML. Then, we outline the underlying technologies, UML, XML and UXF.

2.1 Benefits of Capturing Aspects in the Design Phase
Aspects can be identified at the design and implementation phases, though the inter-component tangling tends to occur at the implementation/coding phase [3]. When aspects are identified, or emergent, at the implementation phase, developers often add or change aspects manually (see Figure 1) and maintain them in the source code level. Few methods have been proposed for expressing aspects in the design level. Supporting aspects at the design phase streamlines the process of aspect-oriented development (Figure 1) by facilitating:

- Documentation and Learning

  Supporting an aspect as a design construct allows developers to recognize it in the upper level of

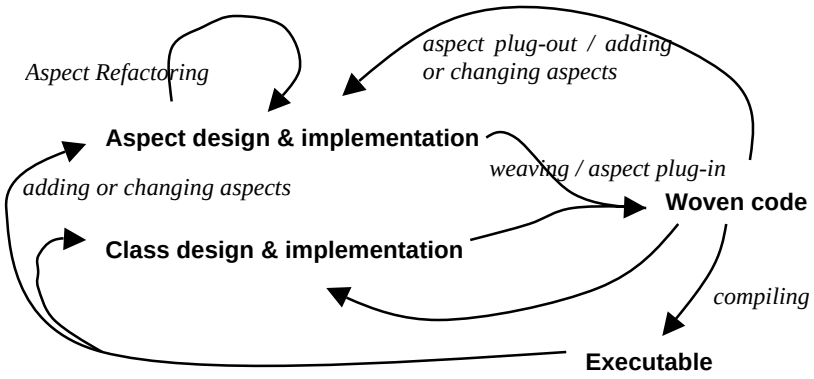

Figure 1: A typical process of aspect-oriented development

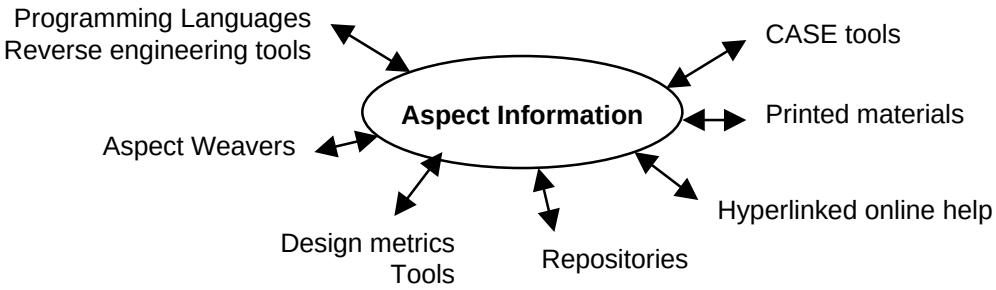

Figure 2: Interchangeable Aspect Information

abstraction at the earlier stage of development process. Aspect designers and people learning aspect-oriented software can learn and document aspect models in more intuitive way. For example, they can visualize aspect models using a CASE tool that supports visual modeling.

- Reuse of aspects

  The ease of documentation and learning leverages the reuse of aspect information, how an aspect is designed and how it is intended to affect to classes. It's easy to imagine more sophisticated ways of using aspects, such as aspect-aware CASE tools, hyperlinked documents and pattern catalogues that collects well-known and feasible aspects. These would increase the reusability of aspect-based design.

- Roundtrip development

  With the aspect support, the incremental and roundtrip development of aspect-oriented software is possible, e.g. aspect code-design model translation, model-aspect code, woven code-model translation, aspect refactoring (see also Figure 1).

2.2 Unified Modeling Language (UML)

UML [4] is the union of the previous leading object modeling methodologies; Booch, OMT and OOSE. It is the state of the art convergence of practices in the academic and industrial community, and has been a standard modeling language by the Object Management Group (OMG) [5]. UML defines 9 diagrams for modeling a given problem domain in terms of various perspectives:

- Structural diagrams:
  - Class diagram
  - Object diagram
- Behavioral diagrams:
  - Use Case diagram
  - Sequence diagram
  - Collaboration diagram
  - Statechart diagram
  - Activity diagram
- Implementation diagrams:
  - Component diagram
  - Deployment diagram

Using these diagrams with the fine level of abstraction, complex systems can be modeled through a small set of nearly independent diagrams. UML defines semantics and notation for every construct in the diagrams.

2.3 Benefits of Describing and Interchanging Aspect Information with XML

For software design phase, model interchange is a quite important capability, because there are few application-

neutral exchange format between development tools. Such an application-neutral format facilitates:

- Interchangeability and reuse of aspect description:

  Software models are dynamically changed in the analysis, design, implementation, revision and maintenance phases. Software tools used in each phase usually employ their own proprietary formats to describe model information. For example, current aspect weavers use their own language to describe aspects. An application-neutral format allows aspect information to be interchangeable between development tools throughout the lifecycle of software development (see Figure 1 and 2). Once encoded with the format, the aspect information can be reusable for a wide range of different development tools with different strengths. This seamless tool interoperability increases our productivity to design aspects.

- Intercommunications between aspect designers:

  An application-neutral aspect description format serves as a communication vehicle for aspect designers. They can communicate their modeling insights, understandings and intentions on an aspect design with each other. For example, we may write down an aspect model into an electronic mail to share it. This capability simplifies the circulation of aspect models between aspect designers.

2.4 UML eXchange Format (UXF)

The most important factor in interchanging an aspect design model is that the semantics within the model should be described explicitly and transferred precisely. To resolve this issue, we developed UXF/a (UXF, aspect extension), an interchange format for aspect design models, which is based on XML (eXtensible Markup Language). UXF/a is an extension to the UXF (UML eXchange Format), which is the format to describe UML models [6, 7, 8], to capture aspect information. UXF is carefully designed to be simple and well-structured enough to encode, publish, access and interchange UML models. XML is a sophisticated subset of SGML (Standard Generalized Markup Language) and provides the following advantages:

- Application neutrality (vender independence)
- User extensibility
- Ability to represent arbitrary and complex information
- Validation scheme of data structure
- Human readability

3. Extending the UML metamodel

This section describes our extension to the UML metamodel for supporting aspects. We added new elements for the aspect and woven class to the metamodel, and reused an existing element for the aspect-class relationship.

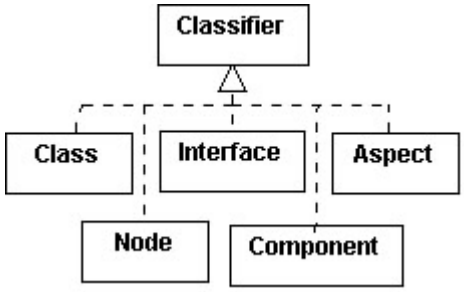

Figure 3: Aspect as a metamodel element derived from Classifier

3.1 Aspect

First of all, the aspect should be added to the UML metamodel. It is no doubt the aspect is a construct derived from the `Classifier` element, which describes behavioral and structural features [4] (Figure 3). All the `Class`, `Interface` and `Component` are kinds of `Classifier`.

Aspect can have attributes, operations and relationships. Attributes of an aspect is used by its set of weave definition. Operations of an aspect is considered as its weave declarations. Relationships of an aspect includes generalization, association and dependency. If the aspect weaver uses the kind of weaves, e.g. introduce and advice weaves in AspectJ [9], it is specified as an constraint for the corresponding weave (operation) declaration.

An aspect is shown as a class rectangle with stereotype <<aspect>> as depicted in Figure 4. The operation list compartment of the rectangle means the list of weave declarations. Each weave is displayed as an operation with the stereotype <<weave>>. A signature of weave declaration shows a designator; which elements (e.g. classes, methods and variables) are affected by the aspect. Figure 4 does not show the stereotype <<weave>> and the kind of weaves in the operation compartment, because the CASE tool we are using does not display the stereotypes and constraints for operations. Other CASE tool such as Rational Rose can display a weave declaration like:

```
<<weave>>{introduce}
    Subject.attach():void
```

The attribute list of a rectangle symbol maps into the list of attributes of the aspect.

3.2 Aspect-Class Relationship

The UML metamodel defines three primary relationships derived from the `Relationship` metamodel element: `Association`, `Generalization` and `Dependency` (Figure 3). The relationship between an aspect and classes that the aspect affects is a kind of dependency. The dependency relationship states that the implementation or functioning of one or more elements requires the presence of one or more other elements [4]. The derived metamodel

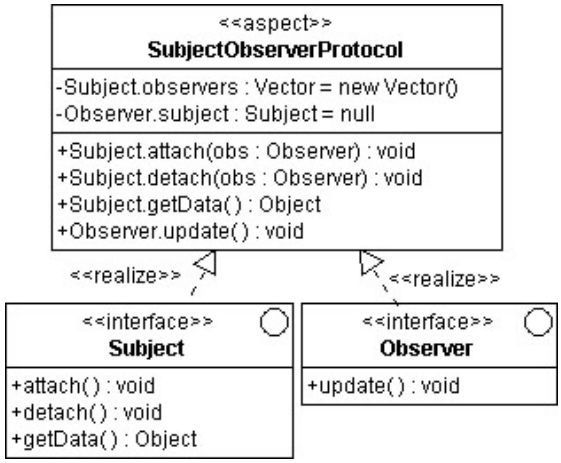

Figure 4: Notation of the aspect and aspect-class relationship

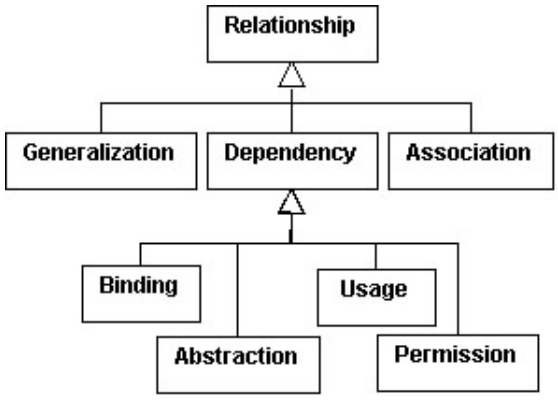

Figure 5: Some kinds of Relationship element in the UML metamodel

elements of `Dependency` are `Abstraction`, `Binding`, `Permission` and `Usage` (Figure 3). The aspect-class relationship is classified as a kind of the abstraction dependency. An abstraction dependency relates two elements that is the same concept at different levels of abstraction or from different viewpoints. The UML metamodel defines three stereotypes for the abstraction dependency: derivation, realization, refinement and trace. The aspect-class relationship is best-suited to the abstraction dependency with stereotype realization, <<realize>>. A realization is a relationship between a specification model element and a model element that implement it. The implementation model element is required to support the declaration of specification model element.

UML defines the notation for an abstraction dependency with the <<realize>> stereotype as a dashed generalization arrow. Therefore, the aspect-class relationship is shown as in Figure 4.

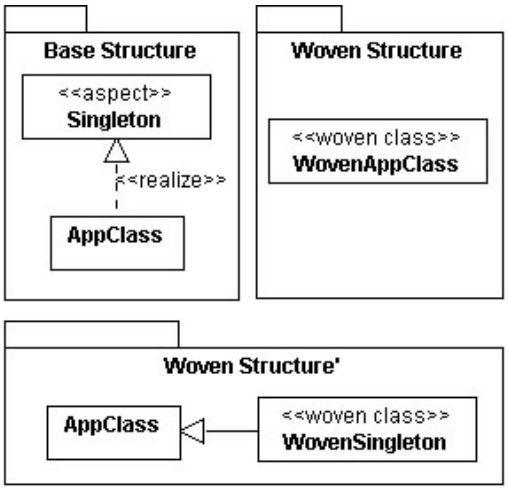

Figure 6: Woven class structures

3.3 Woven Class

Using an aspect weaver, aspect and class code are merged and then a woven class is generated (Figure 1). The woven class structure depends on the aspect weaver and programming language used (Figure 6). For example, AspectJ replaces an original class with the generated woven class. AOP/ST [10] generates a woven class derived from the original class [11]. There would be other alternatives such as using Mediator and Decorator design patterns [12].

We introduced the stereotype <<woven class>> into the `Class` element in order to represent a woven class (Figure 6). It is recommended that the woven class specifies the source class and aspect that are used to generate it, using a tagged value.

4. Describing and Interchanging Aspect Information with UXF/a

This section describes our XML-based format to express aspect information, UXF/a, and then presents some examples of aspect model interchange.

As described in Section 2.3, we use UXF/a to describe the aspect model information. UXF/a is developed based on our metamodel extension described in Section 3. We created a DTD to represent aspect structural information, and then merged it with existing UXF DTDs. All the DTDs are available at [13]. Figure 7 shows a simple UXF/a description. This description represents an aspect named Singleton and a class named AppClass, shown in Figure 6. It is generated from an aspect code of AspectJ using our aspect-UXF/a converter. The reverse translation, from a UXF/a description to AspectJ aspect code, has been also developed. These translators are built by extending the Doclet toolkit included in JDK (Java Development Kit).

```
<UXF Version="2.0"
     xmlns:UXF="http://www.yy.cs.keio.ac.jp/
                ~suzuki/project/uxf/">
  <UXF:Aspect>
    <UXF:Name>Singleton</UXF:Name>
    <UXF:Operation>
      <UXF:Name>AppClass.AppClass</UXF:Name>
      <UXF:Stereotype>
        <UXF:Name>weave</UXF:Name>
      </UXF:Stereotype>
      <UXF:Constraint>
        <UXF:Body>advice, before</UXF:Body>
        <UXF:ConstrainedElement>
          AppClass.AppClass
        </UXF:ConstrainedElement>
      </UXF:Constraint>
    </UXF:Operation>
  </UXF:Aspect>
  <UXF:Abstraction>
    <UXF:Stereotype>
      <UXF:Name>realize</UXF:Name>
    </UXF:Stereotype>
    <UXF:Supplier>Singleton</UXF:Supplier>
    <UXF:Client>AppClass</UXF:Client>
  </UXF:Abstraction>
  <UXF:Class>
    <UXF:Name>AppClass</UXF:Name>
  </UXF:Class>
</UXF>
```

Figure 7: Sample UXF/a description

The conversion between UXF/a and Java code can be available at aspect design/coding, class design/coding and woven class verification phases (see also Figure 1). This conversion capability helps both forward/reverse engineering within the roundtrip aspect development.

Also, we have developed additional two translators from a UXF/a description into popular CASE tools, Rational Rose [14] and MagicDraw [15]. Figure 7 is a screenshot of MagicDraw that displays a UML model generated from the UXF/a description in Figure 7. This translation shows an example of the tool interoperability and aspect model interchangeability as described in Section 2.3.

5. Future Work

UXF/a has been tested and improved with AspectJ and Java programming language. We have not evaluated the interchangeability of aspect model information between weavers. Further tests are planned to improve the interchangeability using other weavers such as AOP/ST.

6. Conclusion

This paper addresses the aspect support in the design phase, and proposes an extension to Unified Modeling Language (UML). Then, we proposes an aspect description language, UXF/a, based on XML. Our work facilitates the aspect

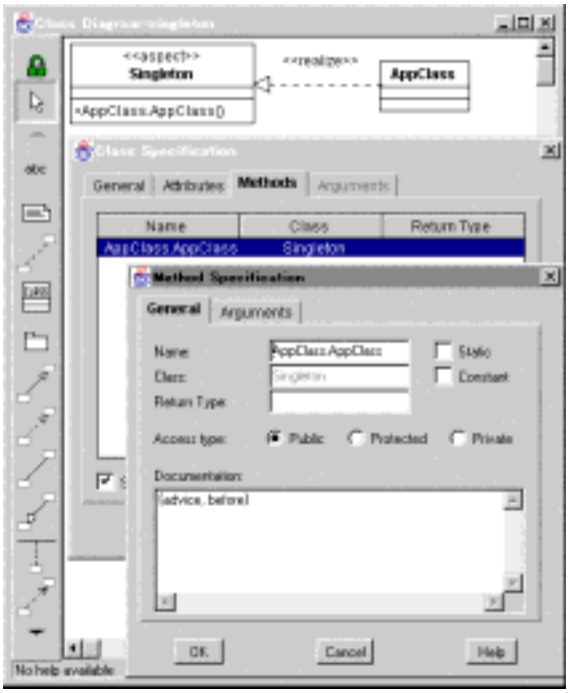

Figure 8: Sample screenshot that display an aspect model information generated from the UXF/a description in Figure 7 (the graphical positions of icons are changed manually)

documentation and learning, and increases the aspect reusability and interchangeability. We believe it shows a next logical step in the evolution of AOP.

7. References

[1] G. Kiczales et.at. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, LNCS 1241, Springer-Verlag, 1997.

[2] C. V. Lopes. D: A Language Framework for Distributed Programming. Ph.D. Dissertation, College of Computer Science, Northeastern University, November 1997.

[3] K. Czarnecki. Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models. Ph.D. Thesis, Technische Universität Ilmenau, Germany, 1998.

[4] Object Management Group, Unified Modeling Language Specification version 1.3 bata 1, 1999. available at uml.shl.com.

[5] www.omg.org.

[6] J. Suzuki and Y. Yamamoto. Making UML Models Interoperable with UXF. In *Proceedings of UML'98*, LNCS 1618, Mulhouse, France, June 1999.

[7] J. Suzuki and Y. Yamamoto. Managing the Software Design Documents with XML. In *Proceedings of ACM SIGDOC'98*, Quebec City, Canada, September 1998.

[8] J. Suzuki and Y. Yamamoto. Toward the Interoperable Software Models: Quartet of UML, XML, DOM and CORBA. In *Proceedings of IEEE ISESS'99*, to appear.

[9] C. V. Lopes and Gregor Kiczales. Recent Developments in AspectJ. In *ECOOP'98 Workshop Reader*, Springer-Verlag LNCS 1543, 1998.

[10] AOP/ST. available at www.germany.net/teilnehmer/101,199268/

[11] K. Böllert. Implementing an Aspect Weaver in Smalltalk. In *Proceedings of STJA '98*, 1998.

[12] E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[13] www.yy.cs.keio.ac.jp/~suzuki/project/uxf/

[14] www.rational.com/rose/

[15] http://www.nomagic.com/