

A Metamodel for Aspect-Oriented Modeling

Christina Chavez Carlos Lucena

March 20, 2002

1 Introduction

Aspect-oriented programming (AOP) is reaching maturity after almost a decade of research [9, 1, 7, 8, 10, 2]. The term AOP has been used in [6] to denote “the space of programmatic mechanisms for expressing crosscutting concerns”, including Kiczales et al. work as well as related work (adaptive programming, composition filters, subject-oriented programming and multi-dimensional separation of concerns). We use the term *aspect-oriented modeling* to denote the space of modeling elements for specifying crosscutting concerns at a higher level of abstraction. Aspect-oriented modeling (AOM) should be built upon a conceptual framework that we refer to as the *aspect model* [4]. We adopt a terminology for AOP based on the core concepts presented in [8] as the main representatives for modeling elements: component (base element), aspect (crosscutting element), join point, crosscutting, and weaving.

It is quite natural to investigate UML’s suitability as a modeling language for supporting Aspect-Oriented Modeling. First, UML [3] is acquainted to be the *industry-standard modeling language* for specifying, visualizing, constructing, and documenting the artifacts of software systems for the software engineering community. Second, UML is a *general purpose modeling language* to be usable in a wide range of application domains. As such, it includes a rich set of modeling techniques for analysis and design as well as structural and behavioral modeling, organized into several complementary views, expressed as diagrams, to model a system. Third, UML is an *extensible modeling language* to facilitate domain-specific modeling. As such, there is a set of built-in extension mechanisms to extend or customize the UML for a specific domain or process.

AOP represents a paradigm shift in terms of programming rationale and, consequently, in terms of high-level modeling. This way, any extension required on a modeling language to support aspects and crosscutting corresponds to the definition of a new language, with additional syntax, semantics and pragmatics; therefore, we are talking about not subsets of UML-related families of languages, but rather, supersets. In this context, using built-in extension mechanisms provided by the UML (such as stereotypes, tag values and constraints) to support AOM is rather a compromise solution to support the transition from pure OO models to OO models combined with aspect models (the paradigm shift), than

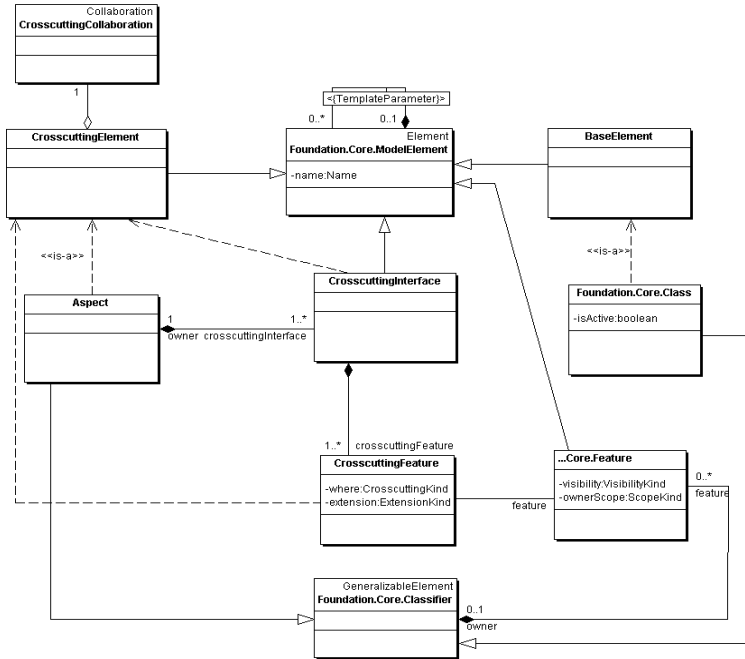


Figure 1: Backbone.

a conceptually sound solution. For us, the inviting challenge is the extension of the UML metamodel to cope with AOM.

In the next section, we present a flavor of the extensions to the UML metamodel that we have been working on.

2 Structural Modeling

Figure 1 shows the backbone we provide for structural modeling.

2.1 Base Elements

BaseElement describes modeling elements that may participate in a *Crosscutting* relationship with one or more elements of type *CrosscuttingElement*.

2.2 Crosscutting Elements

CrosscuttingElement describes modeling elements that may participate in a *Crosscutting* relationship with one or more elements of type *BaseElement* (Figure 2).

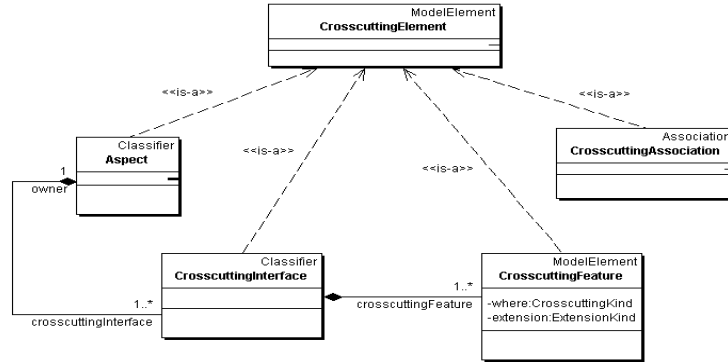


Figure 2: CrosscuttingElement.

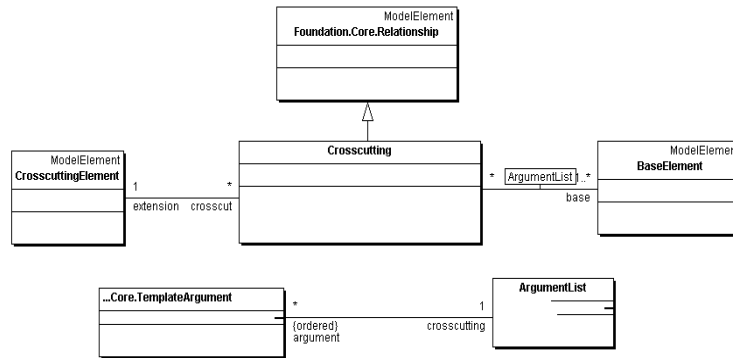


Figure 3: Crosscutting Relationship.

2.3 Crosscutting Relationship

Crosscutting is a relationship that denotes a new extension mechanism for combining incremental descriptions of structure and behavior localized in *CrosscuttingElement* into a *BaseElement*.

In the metamodel, the Crosscutting relationship is a directed relationship, from a *CrosscuttingElement* to one or more base elements, with a list of actual parameters that indicate points in the base elements where the additional structure and behavior will be combined (Figure 3).

Some rules of well-formedness defined are:

1. The number of base elements must be equal to the number of crosscutting interfaces of the aspect.
2. The number of arguments in each set of top-level arguments must be equal

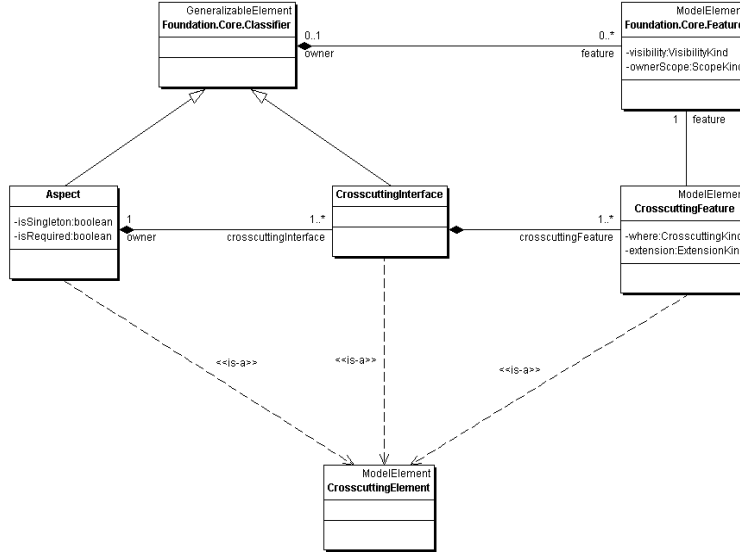


Figure 4: Aspect, CrosscuttingInterface, CrosscuttingFeature.

to the number of formal parameters of each crosscutting interfaces.

3. A Crosscutting has one extension and at least one base.

2.4 Aspects

Aspects are defined as parameterized model elements. Aspects comprise a set of local features and a set of crosscutting interfaces. A *crosscutting interface* models *join points* and crosscutting features defined over these join points. *Aspect* is a kind of *Classifier* that describes attributes and operations (*CrosscuttingFeature*) to be combined to other modeling elements (*BaseElement*), through some weaving mechanism, denoted by the *Crosscutting* relationship. An aspect has a set of interactions (*Interaction*), to describe interactions among instances and aspect instances. Figure 4 shows the metaclasses that describe aspects.

2.5 Crosscutting Interfaces

A crosscutting interface models an archetypical representation of a set of crosscut objects under the aspect's perspective (Figure 4). A crosscutting interface models interface-level, static *join points* and the crosscutting interface operations model crosscutting operations over these join points.

Each crosscutting interface describes the minimal set of properties required to the classes whose state and behavior are supposed to be extended by aspect's structure and behavior. A crosscutting interface specifies:

- a set of attributes and operations that the aspect introduces in the base classes.
- a set of signatures of class methods expected to be used by the aspect
- a set of aspect operations expected to refine class methods. The pronoun *base* denotes the crosscut base object.

A crosscutting interface isolates the part of an object that is relevant to an aspect from the rest of the object. Instances from different classes can be extended by the same aspect, as long as they provide operations expected by one of the aspect's crosscutting interfaces.

2.6 Crosscutting Feature

A crosscutting feature is a model element that describes a feature to be combined to one or more base elements. Each crosscutting feature has attributes that define an `ExtensionKind` (add, refine, override) and a `CrosscuttingKind` (before, after, around). Features that refine or override base operations also have interaction patterns associated (refine-before, refine-after, refine-around, override). Interaction patterns comprise behavioral modeling and are not presented here.

Aspect operations that refine or override are specified as parameterized elements, i.e., they contain one or more unbound parameters that represent operation name and signature. A base operation may be explicitly invoked inside the aspect operation's body using the pronoun *base*.

3 Final Remarks

There are similarities between our work and the Composition Patterns approach [5]. The composition patterns model evolved from the concepts found in subject-oriented programming [7] and composition is based on *merge semantics*; our aspect model has evolved from the core concepts of AOP [8] (we are very concerned about characterizing aspects as first-class entities and expressing the relationships among aspects and components, and also interacting aspects, under the base-aspect dichotomy perspective).

Ongoing work includes extensions to the metamodel to express structural relationships among aspects (precedence, aspect interaction, etc.) and behavioral modeling (aspect instances, join points, interactions).

References

- [1] M. Aksit, K. Wakita, J. Bosch, L. Bergmans, and A. Yonezawa. Abstracting Object Interactions Using Composition Filters. In R. Guerraoui, O. Nierstrasz, and M. Riveill, editors, *ECOOP'93 Workshop on Object-Based Distributed Programming*, pages 152–184. Springer-Verlag, 1994.

- [2] Aspect-Oriented Software Development Web site. <http://aosd.net/>.
- [3] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.
- [4] C. Chavez and C. Lucena. Design-level Support for Aspect-oriented Software Development. In *International Workshop on Advanced Separation of Concerns at OOPSLA*, 2001.
- [5] S. Clarke. Designing Reusable Patterns of Cross-cutting Behavior with Composition Patterns. In *OOPSLA'00, Workshop on Advanced Separation of Concerns*, 2000.
- [6] T. Elrad, R. E. Filman, and A. Bader. Aspect-Oriented Programming. *Commun. ACM*, 44(10):29–32, 2001.
- [7] W. Harrison and H. Ossher. Subject-Oriented Programming (A Critique of Pure Objects). In *OOPSLA93*, pages 411–428, 1993.
- [8] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin. Aspect-Oriented Programming. In M. Aksit and S. Matsuoka, editors, *European Conference on Object-Oriented Programming*, volume 1241 of *LNCS*, pages 220–242. Springer Verlag, 1997.
- [9] K. J. Lieberherr. *Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns*. PWS Publishing Company, Boston, 1996. ISBN 0-534-94602-X.
- [10] P. L. Tarr, H. Ossher, W. H. Harrison, and S. M. S. Jr. N Degrees of Separation: Multi-Dimensional Separation of Concerns. In *21st International Conference on Software Engineering*, pages 107–119, May 1999.