

A UML Profile for Aspect Oriented Modeling

Omar Aldawud
Lucent Technologies
Naperville, IL
oadawud@lucent.com

Tzilla Elrad
Illinois Institute of Technology
Chicago, IL
elrad@iit.edu

Atef Bader
Lucent Technologies
Naperville, IL
abader@lucent.com

Abstract

AOP has matured to become Aspect Oriented Software Development (AOSD) that means the community recognizes the importance of applying aspect orientation to all phases of software development life cycle. Once an initial decomposition of the problem domain identifies software components and the corresponding aspectual properties that cut through these components we would like to be able to express this initial decomposition and carry it to the next life cycle phase. For this refinement process to be effective it must preserve the initial semantics.

In this Position paper we propose an initial discussion on UML profile for AO modeling, which we believe will set the stage for AOSD modeling and hence move us towards a truly aspect oriented software systems. We also initiate the discussions regard modeling the orthogonal concerns in the aspect-oriented software system, where loosely coupled concerns in different dimensions can be easily modeled and reasoned about in isolation of the core components and aspectual components.

1 Introduction

AOP [15],[16] is a paradigm that complements the object-oriented technology. The premise of aspect-oriented technology is separation of concerns; where certain design requirements tend to cut across group of core functional components. Concurrency, security, logging, debugging and fault tolerance are examples of such concerns. AO technology attempt to modularize these scattered implementations in order to avoid the code-tangling phenomena associated with it

AOP offers very little to support the developers in expressing their AO systems with a formal modeling technique. What we are after here is a formal modeling methodology that all AOSD stakeholders can use to communicate, and design their software systems. As modeling and UML profile are essential to OO applications it seems natural to extend it to AO. What is needed is a modeling language that fully supports AOSD, an essential requirement on modeling languages for AOSD, is that it shall be capable of expressing both “Core” components and “Aspectual” components as well as the associations between them and among themselves. This should be generalized to any number of dimensions to support multi dimensional separation of concerns.

The UML [3][4] is a graphical language for specifying, visualizing, constructing, and documenting the artifacts of software systems. UML is a standard modeling language endorsed by Object Management Group (OMG) in 1997 as UML 1.1. UML is the graphical notation that defines the semantics of the object meta-model (model of UML itself), and defines a notation for capturing and communicating object structure and behavior.

UML supports extension mechanisms [4][19],[20] (stereotypes, tagged values, and constrains) to allow tailoring UML to fit the needs of a specific domain, UML profile allow the stakeholders of a certain domain to express the semantics of their systems using a well-defined set of extensions.

2 Modeling AOSD

Benefits of modeling have been outlined in the literature [3][4]. Why do we model? UML summary [3] outlines the following benefits: provide structure for problem solving, experiment to explore multiple solutions, furnish abstractions to manage complexity, reduce time-to-market for business problem solutions, decrease development costs, and manage the risk of mistakes.

Benefits of modeling AOSD systems also been examined by the researchers. In [1] we demonstrated that when aspects are identified at an early stage of the development life cycle it makes *their* design components more reusable, and it makes automatic code generation possible for AOP systems with higher levels of separation of concerns at the generated code. [8] Stresses that capturing aspects at the design phase streamlines the process of AO development, it helps learning and documenting aspects. [8],[9] Also outlines that capturing aspects at the design phase makes round trip development possible of AO systems, and helps to maintain consistency of requirements, design, and implementation.

As a starting point we propose a simple extension to UML packaged in a UML profile for AO, this profile introduces basic AOSD terminology to UML to enable the visual representations of AOSD systems. We started examining AspectJ® constructs [15] as a starting point for constructing this profile, but by no means should the profile be restricted to AspectJ® constructs.

3 UML Profiles

The Object Management Group (OMG) defines UML modeling language in a four-layer architecture [3][4]. The meta-meta-model level defined by the MOF (meta object facility), the UML meta-model is defined and standardized on top of the MOF, the meta-model layer specifies the modeling language, the model layer defines the model, and the user object layer defines instances of the model. The standard metamodels represent areas under which specific domains exist that require a specialization of their domain. In order to provide mechanisms for defining domain standards specializing the standard metamodels, UML introduces the notion of Profile, which is specified in [white paper].

UML provides the means by which one can extend it's meta-model (model of UML itself) to fit the needs of a modeling concern. It provides extension mechanisms that allow customizing and extending UML model elements (i.e. <<class>>). The UML extension mechanisms are stereotype, tagged values, and constrains. Stereotypes are classification of an existing UML model element, which has its own properties that are expressed as tagged values, constrains are restrictions placed on the stereotypes. UML profiles are predefined set of stereotype, tagged values, constrains, and graphical icons to allow modeling a specific domain.

There have been many UML profiles proposed to OMG, UML profile for real time systems UML-RT [UML-RT] is a UML extension to support Real time intensive application, the profile itself is based on concepts from [17] the profile introduces the notions Capsules, ports, connectors, protocols, and protocol role to UML. CORBA Profile [6] CORBA allows applications to communicate with each other regardless of the location and design, the UML Profile for CORBA provides a standard means for expressing the semantics of CORBA IDL using UML artifacts, which enable expressing these artifacts with UML tools. UML Profile for CORBA has been adopted by OMG in October of 2000. [18] Proposed a UML profile that extends the existing class diagram definition to support persistence modeling. A profile that uses UML for Relational DB, key, secondary key, table, file are examples of stereotypes proposed in this profile.

UML profile has 15 requirements documented in [5] one key requirement is that the resulting notation shall adhere to the syntax and semantics of UML, the profile shall not break existing UML modeling principles, We intend on addressing all of UML profile requirements in the near future.

4 UML Profile for AOSD

UML profile for AOSD will provide conventions and guidelines for applying and specializing standard UML to the graphical notation for AOSD (Specialize UML meta-model for AOSD, and CASE tool support). It would help in capturing every desired terminology of AOSD and provide “native” support for AOSD in UML. The profile will set a common means of expressing AOSD artifacts visually for all AOSD stakeholders.

Table 1 starts the list of stereotypes that we suggest for AOSD, the 1st row defines <<aspect>> as a stereotype for the base class <<class>, which is part of the <<classifier>>(Figure 1), the <<aspect>> stereotype will be used to model units of crosscutting concerns.

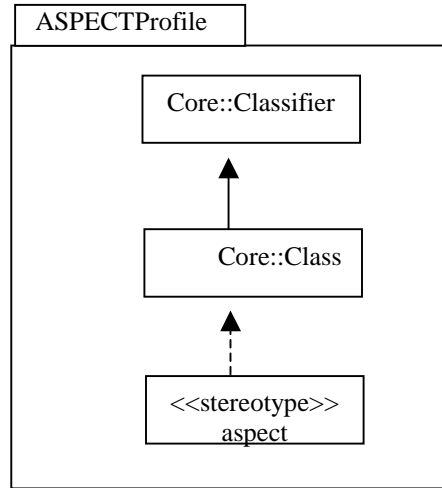


Figure 1 Virtual Metamodel for AOP Profile <<aspect>> stereotype

Extension mechanisms are based mainly on the concept of stereotype, which provides a way of classifying elements so that they behave as if they were instances of the new meta-model. For example when defining <<aspect>> as a stereotype of the classifier <<class>> it is guaranteed that <<aspect>> behaves the same way if it was created as an instance of <<class>>. The newly constructed stereotype becomes a model element and can participate in all relationships a model element can (ie. dependency relationship, inheritance, etc.).

Table 1 Candidates stereotypes for AOP modeling

Base Class/UML meta-model element	Stereotype	Usage
Class	<<aspect>>	Used to model “units of crosscutting implementation “(aspects)
	<<joinpoints>>	Requires further research
	<<advice>>	Requires further research
	<<pointcut>>	Requires further research
Association	<<Control>>	Used to model control relationships where the aspect code controls the functional components, i.e. synchronization aspect (Figure2).

When modeling with AOSD in mind a model should consist of 2 views, the core view and the aspectual view, in order to link these views together we need to construct relationship stereotypes

(dependency, association, etc.). The problem is what are these relationships? The way we look at it, they depend on the underlying aspect itself, for example for synchronization aspect a control stereotype is required since the synchronization aspect controls access to the functional components. However a different aspect might require new association to be defined, we are currently researching the possibility of defining a generic association and then the user's model itself might tailor this association for its specific needs.

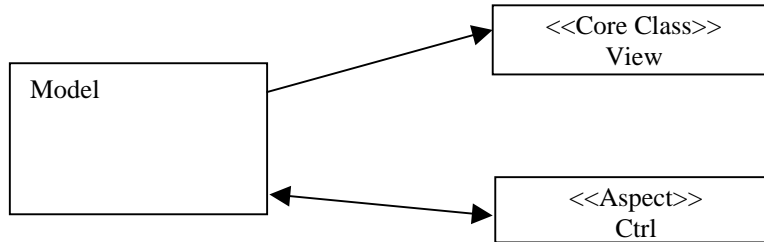


Figure 2-example modeling synchronization aspect

Since AOSD profile is a mean for all AOSD steak holders to express the semantics of their system, it's essential that this profile address AOSD from all angles, and try as much as possible to satisfy all AOSD methods (composition filters, AO application framework , Aspect J, etc.). We need to create a set of stereotypes to support AOSD and not a particular implementation of AOSD. For this profile to be complete it is necessary that all AOSD researchers and AOSD user community participate in defining it.

4.1 Aspects behavior

In [1][2] we proposed expressing the behavior of aspects by means of complex statecharts artifacts, for the sake of completeness we started looking at other UML behavioral elements. The UML behavioral Package (shown in Figure 3) is composed of 4 major packages: Collaboration, Use Case, State Machine, and Activity Graphs Package. All packages inherited from the Common Behavior Package, which specifies the core concepts required to describe a behavior, and it provides the infrastructure to support all other behavioral packages.

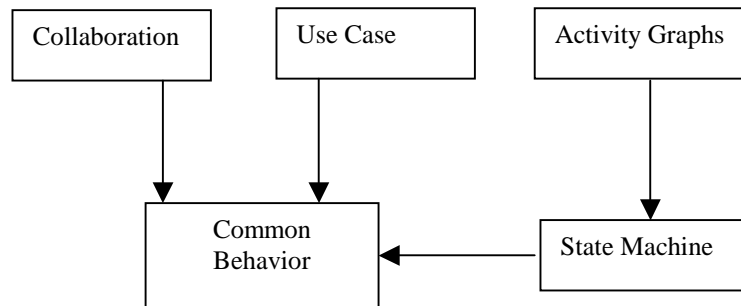


Figure 3 UML Behavioral Package

A Collaboration describes how classes (classifiers) and their association are to be used in a accomplishing a particulr task, in other words it defines a set of roles to be played by instances and links and communications between instances when they play the role. If we can determine what roles an ASPECT plays in acollaboration then we can easily describe its behavior using clooaboration diagrams. Lets go back to the synchronization aspect, the role of it is to control access to the funciotnal components, so it plays the <<control >> role. We have already introduced <<control>> as a stereotype of association model element,

we also introduced <<aspect>> as a stereotype of the <<class>> classifier, a collaboration diagram for the synchronization aspect would look like this:

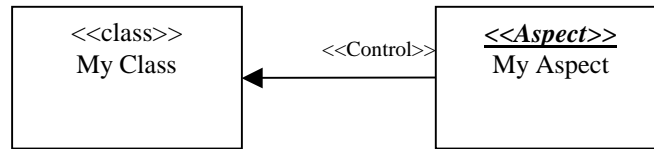


Figure 4. Collaboration Diagram

The behavior of an aspect in aspectJ is determined by pointcuts, advice [15] and the code that does the service, we are currently studying how to model this behavior and incorporating these constructs into the model, for example we know that multiple functional components have common code (cross cut) that is defined by an aspect, in other words these components <<cut>> into the aspect. It's too early for us to complete the description of the aspect behavior, but this is an area that we are actively researching.

There is a need to address the orthogonal modeling of Core components and aspectual components. Modeling orthogonal dimensions and specify how an event in one of these dimensions triggers events in others. For instance a call to a method in a core component may trigger a sequence of events – more likely in aspect components. This can be naturally extended to any number of dimensions.

5 Conclusion

We argue that the AOSD community is ready to think about moving to formal modeling of AO system development. An AOP UML profile will be one more step in closing the gap between OO and AO design tools. There are still basic issues to be resolved. A collective effort for the requirements and constraints for such an undertaking would help to achieve this goal.

Reference

- [1] Aldawud, Elrad, and Bader "Automatic code generation using an Aspect Oriented Framework", 15th European Conference on Object-Oriented Programming, <http://trese.cs.utwente.nl/ecoop01-aoom/>
- [2] Aldawud, Elrad, Bader, and Const, "Modeling Intra Object Aspectual Behavior", ICSE 2001, <http://www.rational.com/events/ICSE2001/ICSEwkshp/>
- [3] UML Summary V1.1, OMG, ad/97-08-03, www.rational.com/UML.
- [4] UML Syntax and Semantics Guide V1.1, OMG, ad/97-08-03, www.rational.com/UML.
- [5] OMG White Paper on the Profile mechanism V1.0. OMG Document ad/99-04-07
- [6] OMG, "UML Profile for CORBA Specification V1.0", OMG document ptc/00-1001, October 2000
- [7] Junichi Suzuki, Yoshikazu Yamamoto, "Extending UML with Aspects: Aspect support in the design phase" 3rd AOP Workshop at ECOOP 1999.
- [8] Siobhan Clarket, William Harison, Osshier, Tarr, "Separation Concerns throughout the Development lifecycle" 3rd AOP Workshop at ECOOP 1999
- [9] Siobhan Clarket "Extending UML Metamodel for Design Composition", 3rd AOP Workshop at ECOOP 1999
- [10] David Harel "From Play-In Scenarios To Code: An Achievable Dream. IEEE Computer, to appear. Preliminary version in Proc. Fundamental Approaches to Software Engineering (FASE), Lecture Notes in Computer Science, Vol. 1783, Springer-Verlag, March 2000, pp. 22{34.
- [11] W. Ho, F. Pennaneac'h, J. Jezequel, and N. Plouzeau, "Aspect-Oriented Design with the UML*,"
- [12] D Harel and M. Politi, Modeling Reactive systems with StateCharts. McGraw-Hill, New York, 1994.
- [13] UML Statecharts by Bruce Powell Dpouglass, ESP Jan-1999. "I-Logix [11] Harel, Daveid "Statecharts": a visual formalism for complex systems, "Science of Computer Programming. Vol. 8 (1987),P. 231
- [14] A.Bader, C. A. Constantinides, T. Elrad, T. Fuller, P. Netinant. Building Reus-able Concurrent Software Systems. International Conference on Parallel and Distributed Techniques and Applications (PDPTA'2000) special session on Distributed Objects in Computational Science. June 26 - 29, 2000. Las Vegas, Nevada, USA.
- [15] Gregor Kiczales, et. All, "An Overview of AspectJ", in proceedings of 15th ECOOP, 2001.
- [16] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In Proceedings of ECOOP '97. LNCS 1241. Springer-Verlag, pp. 220-242. [19] Kim Mens, Cristina Lopes, Badir Tekinerdogan, and Gregor Kiczales. Aspect-Oriented Programming. Report of the ECOOP '97 Workshop for Aspect-Oriented Programming.

- [17] B. Slic, G. Gullekson, and P. War. "Real-Time OO Modeling". John Wiley &sons, 1995.
- [18] Scott Ambler, "Persistence Modeling in the UML", Published in Aug, 1999 issue of Software Development <http://www.sdmagazine.com/>
- [19] Sinan Si Alhir "Extending UML" , Distributed Computing, 1998. PP. 28-32
- [20] Sinan Si Alhir "What is UML, <http://home.earthlink.net/~salhir/whatistheuml.html>