

Modeling Interaction Join Point Adaptations Independent of Pointcut Models using UML Stereotypes

Mariam Nainan
Universiti Tunku Abdul Rahman
Petaling Jaya, Malaysia
603-79582628
mariam@utar.edu.my

ABSTRACT

Several approaches to aspect-oriented modeling of interactions are based on modeling pointcuts that select join points (pointcut models) and modeling adaptations needed at those join points (adaptation models). A common limitation of most of these approaches is that they couple the two models together because identifiers in adaptation models reference those in pointcut models or the same type of diagram is used to represent both models. If it is possible to use pointcut and adaptation models independently of each other, aspect-oriented modeling would be more flexible because, for example, a diagram type can be chosen based on its appropriateness in expressing the model instead of having to consistently use the same type of diagram. This paper proposes a technique for creating interaction adaptation models that are independent of the pointcut models used to capture join points. The technique uses Unified Modeling Language profiles and stereotypes. Examples are also given of how the profile is used to define a generic adaptation model and how the generic model is specialized.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques

General Terms

Design, Languages.

Keywords

Design model, aspect-oriented modeling, interaction models.

1. INTRODUCTION

Beginning with its conceptualization at the programming level, the applicability and benefits promised with the aspect-oriented approach are currently also investigated for the earlier phases and activities of software development such as design and modeling. Currently there are many approaches for aspect-oriented modeling [2] based on the Unified Modeling Language (UML). Taking common features among these approaches for modeling

interactions and their adaptations, they may be categorized into 3 groups: (1) using a parameterized adaptation model where join points and other application-specific elements are specified through parameter binding such as in [3, 12], (2) specifying adaptations and marking selected join points for adaptation in the base model with special indicators such as in [4, 11], and (3) using a pointcut model for selecting join points together with an adaptation model such as in [7, 8, 14].

A drawback of the parameterized model approach is that it does not scale well because of the parameter bindings required. A limitation of the approach based on marking the base model is that the base model is coupled to the aspect model. Among the approaches that use a pointcut and an adaptation model, the two models usually are tied together and cannot be used independently. The coupling is due to identifiers in pointcut models that are referred to in the corresponding adaptation models. Furthermore, generally the type of diagram used to represent the pointcut and adaptation models are the same. For example, both are sequence diagrams or both are state machine diagrams.

If it is possible to model join point capture without being concerned about the type of diagram used for join point adaptation and choosing the diagram type according to how adequately it expresses the selection criteria, there would be greater flexibility in pointcut modeling. The Join Point Designation Diagram (JPDDs) [13] is an example of a technique used specifically for join point selection. It is not coupled to any adaptation modeling technique. Using JPDDs, a join point may be selected using different types of diagrams or even a combination of them. Similarly, it is envisaged that more flexibility would be achieved if it was possible to model join point adaptations independently of the pointcut model.

This paper proposes a technique that makes the interaction adaptation model independent of the pointcut model in two ways. Instead of referencing the selected join points and their context through identifiers, they will be referenced using stereotypes defined in a UML profile [10] for that type of join point. Secondly, the diagram used to represent the adaptation model is not tied to the same type of diagram used for the selection model. The paper also presents examples of how the profile is used to define a generic adaptation model and how the generic model is specialized.

The paper is structured as follows. Section 2 presents the proposed technique for modeling interaction adaptations. Section 3 illustrates the technique with examples. Section 4 relates the proposed approach to existing work and section 5 concludes.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM '09, March 2, 2009, Charlottesville, VA, USA.

Copyright 2009 ACM 978-1-60558-451-5/09/03...\$5.00.

2. PROPOSED MODELING APPROACH

In general the proposed technique for adaptation modeling is to define UML profiles for different types of join points. These profiles will then be applied in the creation of adaptation models. The technique is part of an overall approach for aspect-oriented modeling. In this section, first an aspect modeling profile is described. Next interaction join points are examined and the proposed technique for interaction join point adaptation modeling is presented. Since the adaptation is to be modeled independently of the pointcut, the assumption made is that some modeling technique, for example JPPD, is available to capture selected join points (or their static shadows [9]) and related contextual information.

2.1 Aspect-Oriented Modeling Profile

Aspects, pointcuts, and adaptations are defined as stereotyped packages as in [15]. The UML metamodel extensions for these stereotypes are depicted in Figure 1. An aspect contains a pointcut and an adaptation. An example of an application of these stereotypes for an authentication aspect is shown in Figure 2 with the details suppressed.

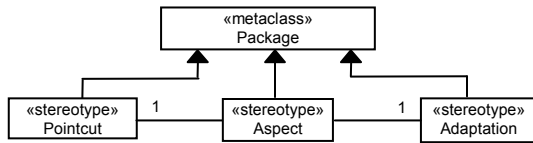


Figure 1. UML metamodel extensions for aspect, pointcut and adaptation

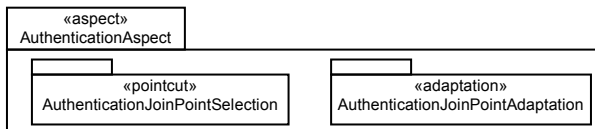


Figure 2. Authentication aspect model example

2.2 Interaction Adaptation Modeling

An interaction specifies how messages are passed between objects in order to accomplish a task. In the UML, a message defines a communication between a sender and a receiver object which are represented as lifelines. The communication may be in the form of a signal, a synchronous or asynchronous operation call, or a reply to an operation call. A message may also designate the creation or termination of an object.

A message is a join point that may be adapted by an aspect. With message join points, contextual information may be needed when modeling their adaptations. This may include the message signature and information about the sender and receiver objects.

2.2.1 Synchronous Operation Join Point Profile

In this paper, a UML profile for synchronous operation join point is defined. To represent the sender and receiver objects related to the captured join point, stereotypes are defined as shown in Figure 3. The stereotypes are extended from the metaclass *Lifeline*. The «sender» and «receiver» stereotypes are used to represent the lifeline objects that play those specific roles. Three stereotypes are defined as extensions of the metaclass *Message*. An

«operationCall» stereotype represents a synchronous operation call join point. An «operationReturn» stereotype represents a reply message after successful completion of an operation call. An «operationExecution» stereotype is used as a message but actually represents the sequence of messages that make up the execution of a method associated with an operation call. This execution is in the context of the receiver object. These stereotypes are required to describe the semantics of *before*, *after*, and *around* advice of AspectJ [1] as explained below. Further stereotypes may be added to the profile in the future, such as for events related to sending and receiving messages, for use in adaptation models represented in activity or state machine diagrams. In this paper, only adaptation modeling with sequence diagrams is considered.

In AspectJ, when a captured join point is adapted, the adaptation might make use of new structural elements introduced using intertype declarations. They may be new attributes, methods, or classes added in relation to the join point. Instead of introducing a pointcut and adaptation modeling technique for intertype declarations, the proposed approach is to include in the profile a stereotype to represent static structural introductions. The introductions will always be in the form of a new class even if only a new attribute or new method is to be introduced into an existing class. An extension of metaclass *Class* named *AspectualClass* is defined. It has a tag definition called *instance* of enumerated type *InstanceKind* with four values indicating the method of instantiation. An object of this type or class may be instantiated as a singleton. It may also be instantiated and associated with either a sender or receiver object. Once associated, the object exists as long as the corresponding sender or receiver object exists. This is similar to the *perthis* and *pertarget* association semantics of AspectJ. Alternatively it may be instantiated and associated with each join point captured.

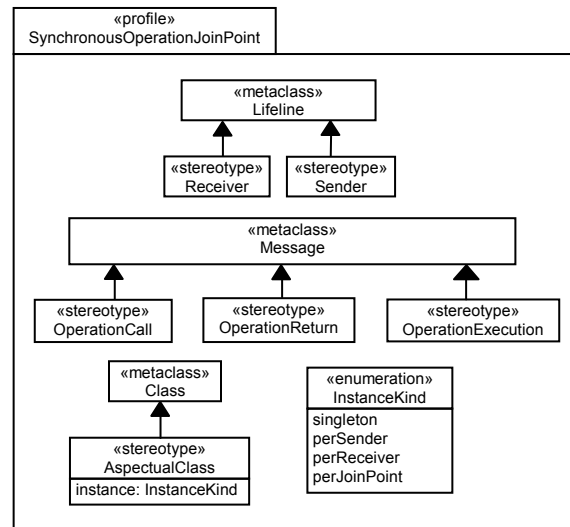


Figure 3. Stereotypes for synchronous operation join point and related context

2.2.2 Synchronous Operation Join Point Adaptation

Figure 5 shows a generic adaptation model for synchronous operation join points. This generic model will be specialized when creating adaptation models for specific aspects. This specialization is achieved through a UML package merge which

has a generalization/specialization relationship similar to class inheritance. Figure 4 shows an example of how an adaptation model for a logging aspect extends the generic synchronous operation join point adaptation model.

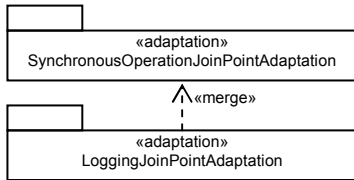


Figure 4. Generalization/specialization relationship between adaptation packages

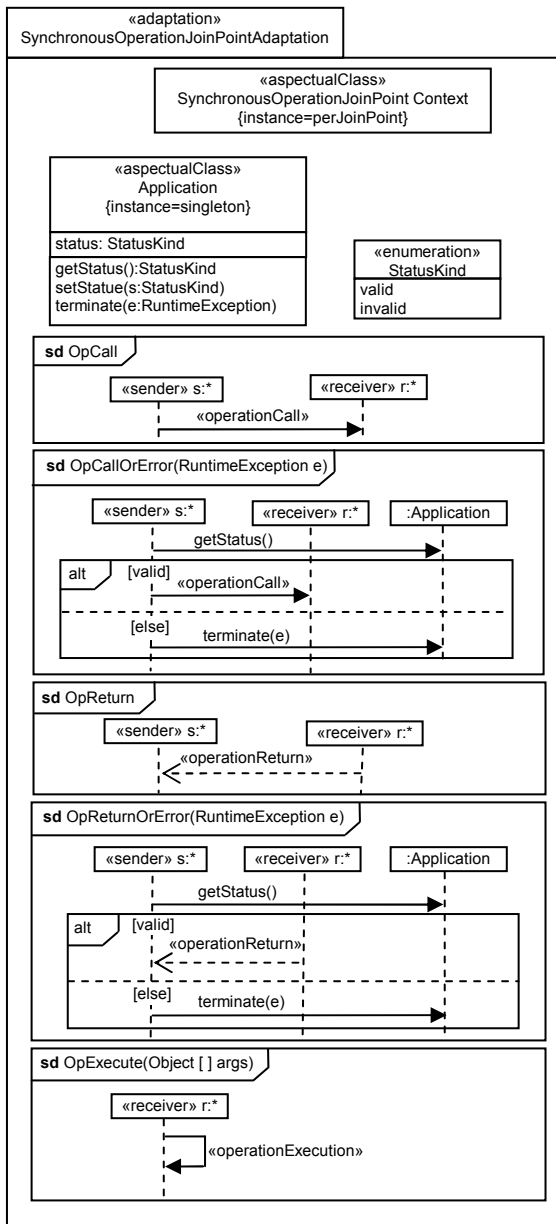


Figure 5. Generic synchronous operation join point adaptation

In the generic synchronous operation join point adaptation model in Figure 5, a class called *Application* is introduced. It is used to represent the overall application execution status and is instantiated as a singleton. The class contains a *status* attribute which is used to indicate whether the application status is valid or not. The status is set to invalid whenever an adaptation raises an exception. The *getStatus()* and *setStatus()* methods retrieve and set the *status* attribute respectively. A *terminate()* method is used to terminate the application by throwing a *RuntimeException* object passed as a parameter.

The stereotyped messages defined in the *Synchronous-OperationJoinPointAdaptation* profile are used in the sequence diagrams in Figure 5. The first sequence diagram contains only a message with stereotype «operationCall». It is used to indicate the point in the sequence of messages in the adaptation where the captured operation call will occur. In the second sequence diagram, first the status of the *Application* object is checked. Then the interaction either proceeds with the captured operation call or terminates the application by raising an exception passed as an argument to the sequence diagram. The third sequence diagram indicates the point in the adaptation after the return from a captured operation call. The fourth sequence diagram is similar to *OpCallOnError* but involves a reply message rather than the operation call. All of the above are applied in the context of the sender object. The last sequence diagram allows adaptation in the context of the receiver object, that is, during the execution of a call operation. It may receive arguments.

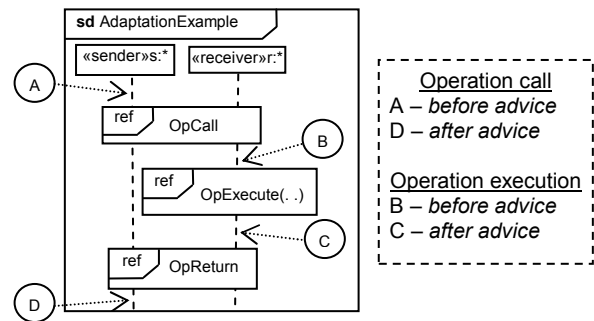


Figure 6. Positions for before and after advices

Figure 6 illustrates how the sequence diagrams are used to specify before and after advices of AspectJ. Any messages placed before *opCall* (position A) corresponds to before advice for the call. Any messages in the context of the receiver object that appears before (B) and after (C) the *OpExecute* represents before and after advice for operation execution respectively. Any messages placed after *OpReturn* (D) corresponds to after advice for the call.

If a *OpCall* is used followed by other messages in the context of the receiver object but no *OpExecute* is included, this represents the *around* advice that bypasses the operation execution. *Around* advice for a call operation is represented by excluding the *opCall* and all other interaction uses. The application of these interaction uses is illustrated in examples in the next section.

Adaptations may require information related to the context of a join point. For a synchronous operation join point, this includes the operation call signature, the names and types of the receiver and sender objects, and so on. To make this information available for adaptation modeling, a class called *SynchronousOperation-*

JoinPointContext is defined (Figure 5), inspired by the technique used in [5]. An instance of this class is created for each captured join point.

3. ILLUSTRATIVE EXAMPLES

3.1 Logging Aspect

The generic *SynchronousOperationJoinPointAdaptation* is used for modeling specific adaptation models as shown in Figure 7 for a Logging aspect. The *LoggingJoinPointAdaptation* merges with the *SynchronousOperationJoinPointAdaptation*. The merging follows the semantics of UML package merge where elements that do not match are incorporated into the merged package and the properties of those that match are merged. The logging adaptation is detailed in Figure 8. The adaptation makes use of a *Logger* class. After the merging, the effect is that the *LoggingJoinPointAdaptation* package will contain the *Logger* and *Application* classes and the combined set of sequence diagrams.

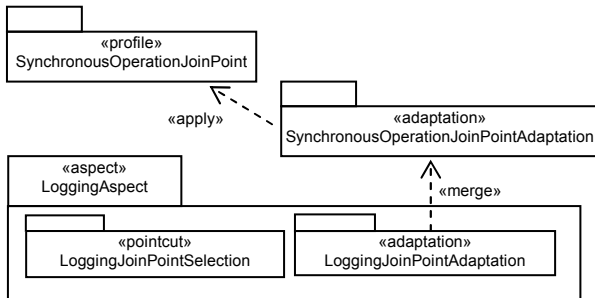


Figure 7. Logging aspect

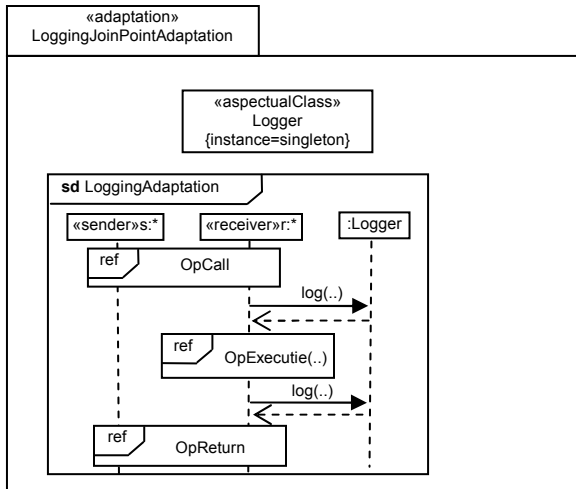


Figure 8. Logging join point adaptation model

The sequence diagram called *LoggingAdaptation* specifies that before and after the captured join point operation is executed, the *log()* operation of the *Logger* object must be called. The operation is called in the context of the receiver object. The log operation argument details (not shown) are obtained using the *SynchronousOperationJoinPointContext* class methods.

To illustrate the eventual aspect model composition, Figure 9 shows an example of a base interaction model to be composed with the logging aspect. Suppose *m1()* is the captured message

join point. The call to operation *m2()* is part of the operation execution of the captured message join point. Figure 10 shows what the resulting composed model would look like after applying the logging adaptation.

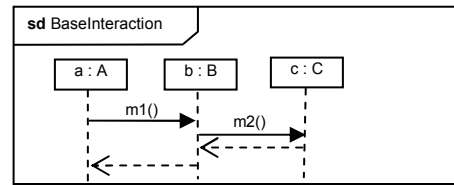


Figure 9. Base interaction model

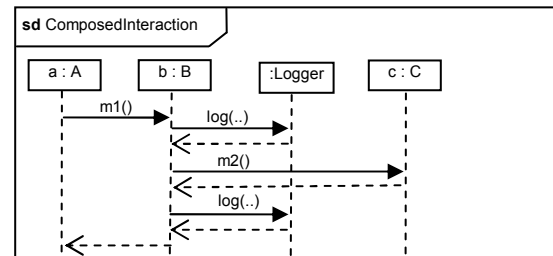


Figure 10. Composed interaction model

3.2 Authentication Aspect

The *AuthenticationJoinPointAdaptation* (Figure 11) also merges with the *SynchronousOperationJoinPointAdaptation*. The classes introduced are *User* and *Login*. Each time a *User* object is created, it is associated with the sender object connected to the join point. A *Login* object is created and exists only during the advice execution. The *IllegalAccessException* class is extended from *RuntimeException*. Before the call to the captured join point operation, the sender object checks if a *User* object already exists. If not, it calls the *authenticate()* method to perform a login and validation and set the *User* object created, if valid. The tagged value *withSender* for the *User* class will be interpreted as follows: an attribute *user* of type *User* and methods *getUser()* and *setUser()* will be introduced into the class of the sender object. This is to implement the intertype declaration semantics of AspectJ.

4. RELATED WORK

This section reviews closely related work. As mentioned earlier, the approach used in this paper to refer to context information in the adaptation model is to use an object of a special join point context class introduced in the generic adaptation model. The idea is taken from Cazzola [5] where behavioral adaptation is modeled using activity diagrams and actions. In their approach, stereotypes are defined and used in pointcut models but none were mentioned in the paper for use in the adaptation model. In [6], Cottenier *et al.* use action join points such as operation call action and model their adaptation in transition-centric state machines with concepts of state, transition, and decision actions. They use a reflective API to access join point context information. In this way, common identifiers need not be used in the pointcut and adaptation models. Fuentes and Sanchez [7] use an action language to define join point adaptation. The information related to the join point and its context is obtained through reflective actions. However, the

pointcut model must include specifications of the type of advice (*before*, *after*, or *around*) and whether it is at the point of sending or receiving a message.

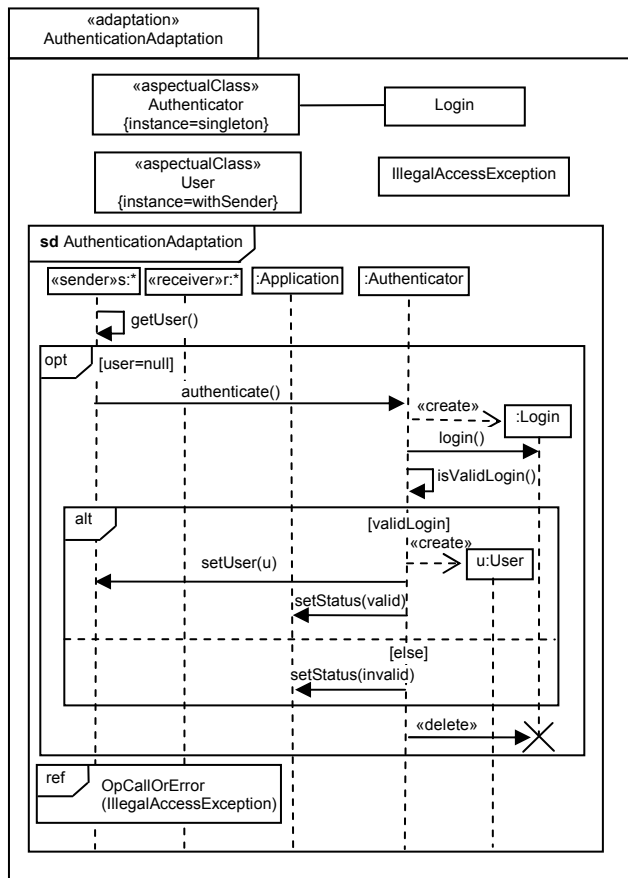


Figure 11. Authentication join point adaptation

5. CONCLUSION

This paper presented an approach to modeling interaction adaptation models that are specified independently of pointcut models. The technique used is to define a UML profile with stereotypes to represent a join point, its context, and structural introduction needed for adaptation. The aim is to allow modeling of the adaptation of join points and their context in a manner that is independent of how the capture of join point and context was modeled. The paper presented a profile defined for synchronous operation join point adaptation. Another objective of the approach is to be able to decide on the type of diagram used based on how well it expresses the model rather than being limited to a particular type of diagram. Future work includes defining UML profiles for other types of join points and different types of diagrams.

6. REFERENCES

[1] AspectJ Programming Language homepage. <http://www.eclipse.org/aspectj>.

[2] Aspect-Oriented Modeling homepage. <http://www.aspect-modeling.org>.

[3] Baniassad, E. and Clarke, S. 2004. Theme: An approach for aspect-oriented analysis and design. In Proceedings of the 26th International Conference on Software Engineering, IEEE Computer Society, 158-167.

[4] Basch, M. and Sanchez, A. 2003. Incorporating aspects into the UML. In 3rd Int'l Workshop on Aspect-Oriented Modeling at International Conference on Aspect-Oriented Software Development.

[5] Cazzola, W., Cicchetti, A., and Pierantonio, A. 2006. Towards a model-driven join point model. In Proceedings of the 2006 ACM Symposium on Applied Computing, ACM Press, 1306-1307.

[6] Cottenier, T., Van Den Berg, A., and Elrad, T. 2007. Stateful aspects: The case for aspect-oriented modeling. In 10th Int'l Workshop on Aspect-Oriented Modeling at Int'l Conference on Aspect-Oriented Software Development. ACM Press.

[7] Fuentes, L. and Sanchez, P. 2007. Towards executable aspect-oriented UML models. In 10th Int'l Workshop on Aspect-Oriented Modeling at Int'l Conference on Aspect-Oriented Software Development. ACM Press, 28-34.

[8] Grassi, V. and Sindico, A. 2006. UML modeling of static and dynamic aspects. In 9th Int'l Workshop on Aspect-Oriented Modeling at Int'l Conference on Model Driven Engineering Languages and Systems.

[9] Hilsdale, E. and Hugunin, J. 2004. Advice weaving in AspectJ. In Proceedings of the 3rd Int'l Conference on Aspect-Oriented Software Development. ACM Press, 26-35.

[10] Object Management Group. UML Superstructure Specification version 2.1.1, 2007. OMG Document formal/2007-02-05.

[11] Reddy, R. Solberg, A., France, R., and Ghosh, S. 2006. Composing sequence models using tags. In 9th Int'l Workshop on Aspect-Oriented Modeling at Int'l Conference on Model Driven Engineering Languages and Systems.

[12] Simmonds, D., Solberg, A., Reddy, R., France, R. and Ghosh, S. 2005. An aspect oriented model driven framework. In Proceedings of the 9th Int'l Enterprise Computing Conference, IEEE Computer Society, 119-130.

[13] Stein, D., Hanenberg, S., and Unland, R. 2006. Expressing different conceptual models of join point selections in aspect-oriented design. In Proceedings of the 5th Int'l Conference on Aspect Oriented Software Development, ACM Press, 15-26.

[14] Whittle, J., Araujo, J. and Moreira, A. 2006. Composing aspect models with graph transformations. In Proceedings of the 2006 Int'l Workshop on Early Aspects at International Conference on Software Engineering, ACM Press, 59-65.

[15] Zhang, G. 2005. Towards aspect-oriented class diagrams. In Proceedings of 12th Asia-Pacific Software Engineering Conference, IEEE Computer Society, 763-768.