

UML Class-Diagram-Based Adaptive Aspect

Abstract :

AOP has the ability to encapsulate crosscutting concerns in system and improves the abstract level of software, making it easier to implement and maintain. This paper makes an extension to the popular AOP language, AspectJ, and defines adaptive Aspect by making the encapsulation independent to the software structure. It also gives an algorithm that transforms the extensive syntax to standard AspectJ syntax based on UML class diagram.

Key words:

Adaptive , AOP , UML

1. Introduction

AOP (Aspect-Oriented Programming) as a new methodology of programming has interested both academic and industry community. It has the ability to encapsulate crosscutting concerns in system, and hence improves the abstract level of software, making it easier to implement and maintain. In this paper, we present extended syntax to the popular AOP language, AspectJ. The extended syntax allows the encapsulation of concerns independent to software structure and makes the encapsulated module adaptive, thus enhance the power of AOP in software construction and evolvement.

The rest of the paper is organized as follows. Section 2 briefly introduces the idea of Aspect-Oriented Programming and also some AspectJ syntax. Section 3 introduces adaptive programming then presents an UML class diagram-based adaptive Aspect. The transform of the extended syntax to standard AspectJ syntax is implemented in section 4, an algorithm is also given in this section. Section 5 gives a sample using adaptive Aspect. Then we describe future work and summarize the contributions of this paper in the final section.

2. AOP and AspectJ

AOP was a new methodology of programming presented by professor Gregor Kiczals in ECOOP97[1]. The main idea of AOP is to developing mechanisms that explicitly support modularizing crosscutting structure thus makes it possible to encapsulation tangled code in application.

AspectJ [2,3] is an aspect language built on top of Java developed by Xerox's Palo Alto Research Center (PARC). It partitions the construction of a software system into two parts: one is the system architecture that is implemented by a kind of regular OO language (in AspectJ, java language); the other is modularized crosscutting concerns called Aspect. The AspectJ compiler is in fact a compile-time weaver, which couples the java module and the Aspect module in compile-time.

An Aspect is a class-like module. It consists of three parts:

1. Introduction. Introduction is AspectJ's form for modifying classes and their hierarchy.
2. Pointcut. Pointcut identify certain join points in the program flow where join points are certain well-defined points in the execution of the program. For example, the Pointcut "call(void Point.setX(int))" identifies any call to the

method setX defined on Point objects. AspectJ defines 15 kinds of Pointcuts and supports their logical calculation. The table below gives some kinds of Pointcut defined in AspectJ.

Pointcut	Meanings
<code>call(Signature)</code>	Picks out a method call join point based on the static signature
<code>execution(Signature)</code>	Picks out a method execution join point based on the static signature
<code>get(Signature)</code>	Picks out a field get join point based on the static signature
<code>set(Signature)</code>	Picks out a field set join point based on the static signature
<code>target(TypePattern)</code>	Picks out all join points where the target object is an instance of a type of the type pattern, or of the type of the identifier
<code>This(TypePattern)</code>	Picks out all join points where the currently executing object is an instance of a type of the type pattern, or of the type of the identifier

Table1: Some Pointcut designators in AspectJ

3. Advice. Advice defines pieces of aspect implementation that execute at well-defined points in the execution of the program.

3. Adaptive Aspect

3.1 Adaptive Software and Demeter

We use professor Karl Lieberherr's definition to define adaptive software: "A program is called adaptive if it changes its behavior according to its context". Adaptive is achieved variously depends on the different views of context. Mohamed G.Gouda and Ted Herman view context as environment, or a set of "input" variables and presented some operators for combining adaptive programs [4]. The context could also be run-time environment, concurrency properties and so on.

Karl's Demeter project [5,6] did much about adaptive programming. The key idea of Demeter is to making the programs structure-shy by using partial information about the implementation-specific class structure when writing the behavior. Thanks to the high level of abstraction, behavior can be adaptive to the evolving software structure.

3.2 Adaptive Aspect

The idea of adaptive aspect mainly gets from Demeter project. Since Pointcuts are data-structure-dependent, we define adaptive Pointcut by using partial information about class structure. First we give some definitions as below:

Definition1. Class graph: is a well-defined UML class diagram.

Definition2. Reachability (->): In a class graph, class A-> class B IFF. A->A1 & A->A2 & ... & A-> Ai & Ai->B | A is associated to B or super classes of B | A or its subclasses are associated to B or super class of B. (Note: Dependency relationship is quite subtle in UML and we ignore dependency when searching reachability information in class graph. Aggregation is a special kind of associate. See also [7])

Definition2. Path: we say there is a path between class A and class B if A->B

Definition3. Derived(TypePattern) = { C | C is a subclass of classes specified by TypePattern }

Definition4. Super(TypePattern) = { C | C is a super class of classes specified by TypePattern }

Definition5. From S To F [Bypass B] [Through T] = {C | C is a class that belongs to the paths which the start end is one element of S and the final end is one element of F. if “Bypass” keyword is appeared, eliminate the classes occur only in the path which include one or more element of B; if “Through” keyword is appeared, only classes those are in the path which include all elements in T are added }

We add definition 3, 4, and definition 5 as typepatterns to AspectJ, specially, to Pointcut designators that are relevant to objects: this(typepattern), target(typepattern) and args(typepettern, ...). For example:

Pointcut(): “this(From A to B)”

Picks out all join points where the currently executing object is an instance of class belonging to the path from class A to class B.

4. Implement of Adaptive Aspect in Rational Rose

Rational Rose is a prevailing modeling tool in Object-Oriented field. It supports UML and provides extensibility interface to manipulate UML model built in Rose. There are two reasons we use Rose to implement Adaptive Aspect, first, we can transform adaptive declaration to standard Aspect syntax by utilizing its’ extensibility interface to search the class diagram. Second, it’s also an attempt to modeling AOP in a Graphic-User Interface (GUI).

In AspectJ, Aspect is a class-like module consisted of introduction, Pointcut and advice. We added it to the UML metamodel as a subelement of Classifier element (Figure1).

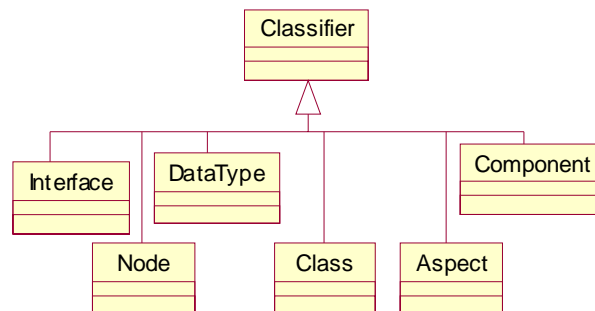


Fig.1: Aspect as a metamodel element

Aspect can have introductions, Pointcuts and advices. The relationship between Aspects can be generalization. The relationship between aspect and ordinary classes can be dependency. An aspect is shown in Rose as a class rectangle with stereotype <<aspect>>. A sample model using Aspect is given in Fig.2.

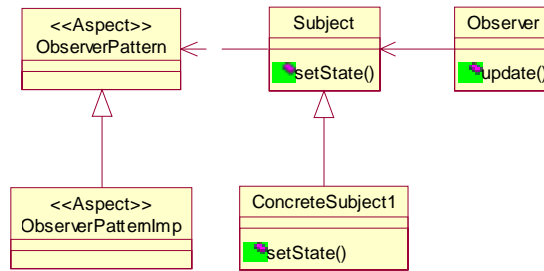


Fig.2: A sample model using Aspect

We extend Rose by adding two functions act on Aspect: specification and code generation for Aspect. In the phase of code generation, if extended syntax is scanned, we use the search algorithm act on class diagram to transform it to standard AspectJ syntax. We give the algorithm deals with clause “Form S To F [Bypass B] [Through T] below:

Algorithm: transform adaptive typepattern to standard typepattern.

SEARCH(S ,F ,B ,T): In which S is a set of start class, F is a set of final class, B is a set of bypass-class and T is a set of via-class. The algorithm returns a set that contains classes included in the satisfied path.

Step1. if B != NULL, SEARCH(S, F, B, T) = SEARCH (S , T , B) SEARCH (T , F , B) ;

Step2. SEARCH(S, F, B):

while: S contains class that has not been marked, for each unmarked class i in S:

 If i ∈ B , mark i = false in all sets constructed in the algorithm.

 If i ∈ F , mark i = true in all sets constructed in the algorithm, add i to the result set R.

 Else, construct a reachable set Mi for class i, Mi = {m | m is subclass of i or i associate to m} and mark i = indefinite;

 S = Mi, goto Step2;

JUDGE(S);

Step3: JUDGE(C):

For each class i in C:

 If i has a reachable set Mi, for each class m in Mi, JUDGE(m);

 If the reachable set Mi of i contains a class marked true, mark i = true and add c to the result set R;

 If the reachable set Mi of i contains no class marked true, mark i = false;

Step4: Classes in the set R are all classes meet the condition.

5. A Sample

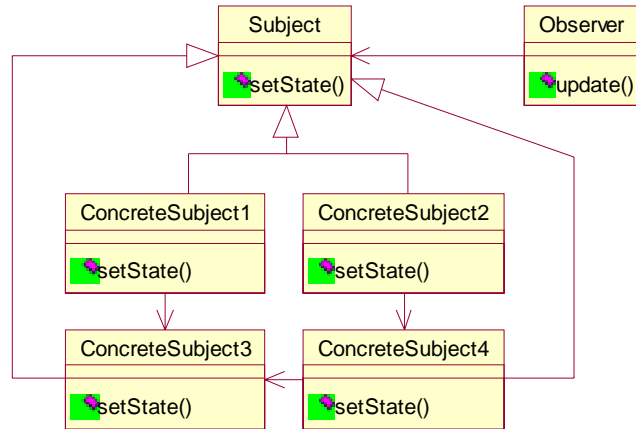


Fig3. Class diagram of an Observer pattern

Consider the observer pattern depicted in Fig3. An observer needs to observe various subjects. If subjects are selective, say, we need to observe ConcreteSubject2 and all the concreteSubjects it contains, using standard Aspect, we should define Pointcut as:

```
pointcut stateChanges():
    (target(ConcreteSubject2) || target(ConcreteSubject4) ||
    target(ConcreteSubject3)) && call ( void setState() )
```

This definition rigidly binds the join point to software structure, so the Pointcut needs to be changed when software structure is changed. Suppose the structure is evolved as shown in Fig.4 while the concern does not change, Pointcut needs to be redefined.

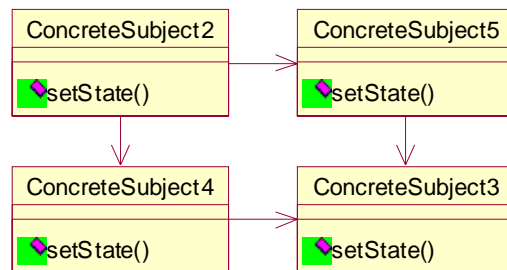


Fig4. The evolved Observer pattern

Using the UML class-diagram-based adaptive Aspect, we define the Pointcut as follows:

```
pointcut stateChanges():
    target(From ConcreteSubject2 to ConcreteSubject3) && call (void setState())
```

In the class graph corresponding to Fig3, the Pointcut is transformed to:

```
target(ConcreteSubject2) || target(ConcreteSubject4) ||
target(ConcreteSubject3)) && call ( void setState() )
```

In the class graph corresponding to Fig3, the Pointcut is transformed to:

```
target(ConcreteSubject2) || target(ConcreteSubject4) || target(ConcreteSubject3)
|| ConcreteSubject5) ) && call ( void setState())
```

It is clear that the extended syntax has high-level abstraction and hence improves the ability of AOP in software implementation and evolvement.

6. Future work

The adaptive aspect model we present in this paper is based on UML class diagram and we manipulate class diagram in Rational Rose. This is reasonable when we design a software system. In the implement phase, we can also get the class structure of the system, say, using reflection mechanism in Java. So the extended syntax can be transformed without UML model and Rational Rose, this is the future work the author plan to do.

7. Conclusion

This paper presents extended syntax to AspectJ and endues Aspect the ability of adaptive. We also give a search algorithm based on UML class diagram to transform the extended syntax. We hope this work will make AOP more useful and practical.

8. Reference

- [1] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin. Aspect-Oriented Programming. Proceedings of the European Conference on Object-Oriented Programming (ECOOP), June 1997.
- [2] Tutorial: Aspect-Oriented Programming with AspectJ, Bill Griswold, Erik Hilsdale, Jim hugunin, Wes Isberg, etc. Available at <http://Aspectj.org>
- [3] The AspectJ Programming Guide, the AspectJ team, available at <http://aspectj.org>
- [4] Adaptive Programming, M.G.Gouda and T.Herman, IEEE Transactions on Software Engineering, Vol.17, no.9, September 1991.
- [5] Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns, Karl Lieberherr, PWS Publishing Company, 1996
- [6] Efficient Implement of Adaptive Software, Jens Palsberg and, Cun Xiao and Karl Lieberherr, ACM Transactions on Programming Languages and Systems, 1995
- [7] OMG Unified Modeling Language Specification, OMG, available at <http://www.omg.org>