一种基于UML扩展的AO建模方法*)

吴春明 郑志强 余建桥
(西南农业大学信息学院 重庆400716)

摘 要 随着AOP技术的日渐成熟,面向方面建模(AOM)已成为当前研究的热点。AOM利用建模语言为系统进行基于AO的分析和表示,允许开发人员在系统开发与设计的初始阶段就将问题分解为核心组件与横切组件,并将横切关注点模块化独立的aspect,这可使系统结构更加清晰,为下一个阶段的代码生成、系统维护带来便利和保障。本文利用UML的可扩展性,提出了一种基于AspectJ语法概念,通过扩展UML元模型元素来支持AO建模的方法。该方法利用UML自身的扩展机制(版类、标签值和约束),对class、association等元模型元素进行扩充,分别表达AspectJ系统模型中的aspect、pointcut等概念。最后利用Rational Rose CASE工具包对该方法进行了验证,并通过编制的脚本为模型中的aspect生成了AspectJ语法格式的代码框架。

关键词 AOM,UML扩展,元模型

An Approach to Aspect-Oriented Modeling Based on UML Extension

WU Chun-Ming ZHENG Zhi-Qiang YU Jian-Qiao
(College of Information,South-West Agricultural University,Chongqing 400716)

Abstract With the gradual perfection of Aspect Oriented Programming, recent researches have been focusing on Aspect Oriented Modeling (AOM). In this paper, based on extension mechanisms of the unified modeling language (UML), sush as stereotypes, tagged values and constraints, we propose a UML extension for modeling AO systems characterized by AspectJ constructs and supporting AspectJ system modeling by extending UML meta-model elements. To describe aspects and pointcuts and other notions in AspectJ system, this approach extends the "Class", "association" and other UML meta-model elements, and it is also tested here by using Rational Rose © CASE tool package. Using Rose Script we compiled to generate an AspectJ skeleton for the aspect-diagram for the model designed using the proposed extension.

Keywords Aspect-oriented modeling, UML extension, Meta-model

1 引言

AOP(Aspect-Oriented Programming)提供了精确捕获系统横切关注点的机制[1]。随着AOP技术的日渐成熟,AO的思想已不再局限于编程层次,而开始影响到软件开发的各个阶段。当前,AOM(Aspect-Oriented Modeling)成为有关AO的研究热点。在2002年荷兰举行的第一届国际面向方面软件开发大会上,还成立了专门进行AOM研究的工作组(workshop)[2]。

AOM利用建模语言为系统进行基于AO的分析和表示,允许开发人员在系统开发与设计的初始阶段就将问题分解为核心组件与横切组件,并将横切关注点模块化为独立的aspect,这可使系统结构更加清晰,为下一个阶段的代码生成、系统维护带来便利和保障。

目前,AO本身还没有正式的建模技术和建模工具,研究人员分别从建模语言、工具、方法以及思想等不同角度对AO系统的建模进行了研究。如文[3]开发了一种基于XML的aspect描述语言,用来在开发工具与aspect联结器间进行信息交流;文[4]提出了一种用于在设计阶段处理横切需求的组件模式,该模式利用了UML模板;文[5]利用UML状态图和类图来为代表横切关注点的不同子模块进行建模和内部联结,该方法不需对UML进行扩展;文[6]对支持AOM的CASE(Computer Aided software Engineer)工具进行研究,提出了一种基于元模型的OpenTool技术,利用OTScript元语来扩展UML的属性和结构,以支持AOP;文[7]提出了一种对UML版类包(profile)进行扩展以支持AO建模的方法。

本文在文[7]的基础上,对UML元模型元素进行了基于AspectJ语法概念的扩充,以支持AspectJ的系统建模。

2 UML的可扩展性

UML(Unified Modeling Language)是一种用来指定、可视化、构建和记录软件系统的建模语言,被OMG (Object Management Group)组织确定为面向对象的标准建模语言。UML一个重要特征是它提供了丰富的扩展机制,使其作用域不仅局限于面向对象的分析与设计,还能适应不同类型的系统、领域和方法。

OMG组织将模型标准结构分为以下四个层次[8](见图1)。

M0:用户对象层(user object layer),该层是对实际运行中对象的抽象,由用户希望描述的信息组成,这些信息典型的

*)重庆市教委科学技术研究项目(项目号:030201)资助。吴春明 硕士生,主要研究方向为面向对象编程与网络技术;郑志强 硕士生,主要研究方向为软件工程与数据库;余建桥 硕士生导师,博士,教授,主要研究方向为数据库与人工智能。

指数据。

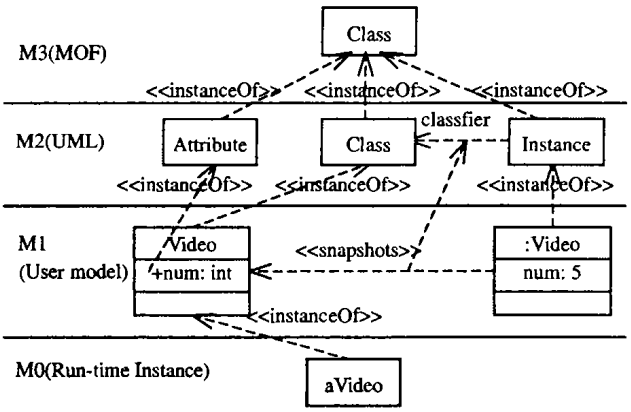

图1 UML 原模型层次结构

M1：用户模型层(user model layer)，该层是对建模者开发的 UML 模型的抽象，它描述了某一领域的相关信息，同时也是描述 M0层结构的 UML 模型。

M2：元模型层(meta model layer)，该层是 UML 元模型定义所在的层。这一层的概念被 UML 建模者所使用，如 Class、Attribute 等。同时，它也包括了从面向对象到面向组件等相关的概念，这也是构建 UML 的基础。

M3：元元模型层(meta-meta model layer)，该层包含了 UML 所依赖的大多数最基本元素，同时定义了用于指定元模型的语言。该层在 OMG 术语中被称为 MOF(meta object facility)。

以上结构中，某一层的实体总是其上一层中一个事物的实体。可见，对 M2层上的元模型进行改变或扩充将直接影响到用户模型。为能定义特定领域规范的标准元模型，UML 引入了版类包的概念。版类包包含了为特定领域或目的定制的模型基本元素，它利用版类、标签值和约束来扩展元模型。版类是对已存在的 UML 模型元素的分类，它用于引入模型元素的新类型，每个版类定义了一系列属性(通过“标签值”实现)和规则(通过“约束”实现)，这些属性和规则被由该版类所修饰的元素接受和使用。

UML 版类包中预定义了一系列的版类、标签值和约束。此外，它还允许用户根据实际需要进行特殊的定义和扩展。

3 基于 AspectJ 语法的 UML 扩展方法

AspectJ 是通用的标准 Java 语言扩展，由 Xerox 公司于2001年3月推出，它是第一种也是目前较为成熟和流行的 AOP 语言。AspectJ 中定义了多种结构和语法来支持 AO 的概念，如 aspect、pointcut、advice 等。

3.1 aspect

Aspect 是一种和 OOP 中的类相似的概念，它将联结点、切点、advice 等元素进行合并、封装，以构成单独的横切单元。自然地，它可通过为 UML 中的“Class”元模型元素添加名为〈〈aspect〉〉的版类来表示。为了与元模型元素 class 可视化地加以区别，本文创建了如图2MannersAspect 所示的图标。由于 aspect 是横切类的单元，本文为 class 图标加一斜线代表这种横切关系。

与 Java 中的类相似，在 AspectJ 中，一个 aspect 也能被抽象。一个抽象的 aspect 具有一个或多个抽象的切点，这样的切点仅有一个标记符号，其定义来源于引用它的一个实际的 aspect。这可用布尔标签值{abstract}来修饰。

3.2 aspect 与 class 之间的关系

在 AspectJ 中，aspect 的行为由切点、advice 以及具体的服务代码所决定。在 UML 模型中，这种行为直接表现为一个 aspect 与一个核心类之间的关系，这种关系可用元模型元素“association”表示。由于 aspect 的行为必须依赖于某一具体的类，因此每一个 aspect 都必须至少与一个核心类相关联。此外，根据它们影响核心类的方式不同，又可将其分成不同的类型，这可用不同的标签值来加以区分。

以下是两者间几种可能的关系，随着 AOP 语言的不断发展，这种关系可能会更加复杂，这也有待于进一步研究。

a) Control：表示当 aspect 控制核心类的行为时。如一个负责同步的 aspect 可能会改变核心类的行为方式；

b) Track：表示当 aspect 负责跟踪核心类的功能或方法调用。如一个 aspect 的功能仅是简单地打印出用户操作的顺序，这样的关系可被定义成〈〈Track〉〉；

c) Customize：修饰当 aspect 负责定制一个核心类时的情况。如一个 aspect 可根据操作系统的不同，相应地定制一个核心类的特定行为；

d) Validate：修饰当一个 aspect 负责确认或验证某个方法的前提条件。如 aspect 负责检查客户端的 session 是否过期，它与核心类间关系即可用〈〈validate〉〉表示；

e) HandleError：修饰当一个 aspect 负责处理系统中可能发生的错误；

f) HandleException：修饰当一个 aspect 负责处理系统中可能发生的例外。

3.3 joinpoint(联结点)和 pointcut(切点)

联结点是程序执行过程中一些特殊的点，它指明对象在何处接受一个方法的调用。而切点是一个或多个联结点的组合，通过“&&”(and)、“‖”(or)和“!”(not)等操作符将联结点进行连接。切点指明了一个 aspect 的 advice 在何处与核心类的功能进行结合。根据定义，一个联结点实质上也是一个切点，因此，本文未对二者进行区分，而是将联结点作为参数的形式放在了切点方法中。

切点是 aspect 中一个重要元素，虽然本身并不执行具体功能，但它指定了 aspect 如何与核心类进行关联，这非常类似于展示给用户的接口(interface)，利用此接口来决定 aspect 中的哪一个 advice 将被执行。因此，本文将一个切点模型化为 UML 元素“Class”，用〈〈pointcut〉〉版类表示，它通过〈〈has pointcut〉〉关系与它所在的 aspect 进行关联。

同样，为了与元模型元素 class 进行区分，本文为切点设计了如图2callSayMessage 所示的图标，而将 pointcut 与 aspect 间的关系设计成通用化(Generalization)的表示方法，其中 aspect 作为子类的形式指向超类 pointcut。

此外，文[1]中指出：一个切点也能被抽象；如果一个 aspect 有至少一个抽象的切点，那么这个 aspect 也是抽象的；继承于抽象 aspect 的子类应提供切点的具体定义；抽象的切点可由标签值{abstract}来修饰。

3.4 advice

Advice 指定在切点上将要执行的具体功能，以代码体的形式被定义在 aspect 中，它和切点一起指定了联结的规则。在 AspectJ 中，advice 分为“before”，“after”，“around”，“after throwing”，以及“after returning”等几种类型。Advice 以类似于 Java 类中方法的形式存在，由于 aspect 是通过对元模型元素 class 扩展而得，advice 可通过为 UML 原模型元素“Opera-

tion"添加相应的版类来表示。本文完全采用 AspectJ 中已定义的一系列 advice 来作为版类名称，分别描述如下：

a)〈〈before〉〉：修饰该 advice 在联结点之前执行；

b)〈〈after〉〉：修饰该 advice 在联结点之后执行；

c)〈〈around〉〉：修饰该 advice 替代联结点处代码执行；

d)〈〈after returning〉〉：修饰当从一个联结点调用返回时，该 advice 才执行；

e)〈〈after throwing〉〉：修饰当从一个联结点调用返回且有例外发生时，该 advice 才执行。

3.5 Code Introduction

Code Introduction 为核心类添加新的变量、数据或方法，在语法格式上等同于类中的属性变量。因此，这里不为其进行特别的设置，同样将它看作 aspect 的属性。

3.6 aspect 与 aspect 之间的关系

AspectJ 是 Java 语言的扩展，同时它也保留了 Java 中类的继承机制，即一个 aspect 可以继承另一个 aspect。这可由 UML 中已有的通用化关系来表示。

此外，当两个 aspect 同时作用于一个核心类时，还需辨别彼此的优先关系。为了表现这种关系，文[1]提出用连接两个 aspect 的<<dominates>>版类来扩展 UML 的"association" 元模型元素。Association 箭头从高优先权的 aspect 指向低优先权的 aspect。如果设计时未指明这种关系，则意味着两者顺序无关紧要。

有关 aspect 彼此间是否还存在其它关联关系，还需在建模时进行更进一步的研究。

4 实验与结果

Rational Rose 是 IBM 公司的面向对象的可视化 CASE 工具，包括了对 UML、OOSE 及 OMT 的支持。同时，它也是一个软件开发集成环境，内置了对 Java，C++，VB 等软件代码双向生成功能。此外，Rational Rose 还提供了多种方法来扩展和定制它的能力以适应特定的软件开发需要[9]。

本文中，我们利用 Rational Rose 提供的扩展接口(REI)，编制了一个.ini 配置文件来定制前面的扩展版类，最后，又编写了.ebs 的 Rose 脚本程序来产生 AspectJ 的代码框架。

以下通过一个简单实例来验证前面所提出的扩展方法。该例取自文[10]。

4.1 问题提出

核心类为一个名为 HelloWorld 的 java 类，有两个名为 say()和 sayToPerson()的方法，其功能只是简单地将各自的实参输出显示。现要为其添加一些基本功能，在每个输出前显示"Good day"，每个输出后显示"Thank you"。现采用面向对象的方法，引入 mannerClass 类，其中定义了 beforesay()和 aftersay()方法，则最后 HelloWorld 类的代码将如下所示：

```
public class HelloWorld {
public static void say(String message) {
        mannerClass.beforesay();
        System.out.println(message);
        mannerClass.aftersay();
    }
    public static void sayToPerson(String message, String name)
        {
        mannerClass.beforesay();
        System.out.println(name + ", " + message);
        mannerClass.aftersay();
    }
}
```

为了将 mannerClass 类的功能作用于 HelloWorld 类，mannerClass 的代码将不得不横切 HelloWorld，如以上代码中的粗体部分。严重情况下，当 HelloWorld 类中有许多方法，或该系统有若干个类，而每个类都要用到 mannerClass 中的两个方法时，程序开发者将不得不在每个类中加入这些代码，这给系统调试、维护、修改和升级都带来了困难。

4.2 基于 AO 的解决方案

图2显示了在 Rational Rose 中针对该问题所建的逻辑视图设计，其中 MannersAspect 为设计的 aspect 名字，callSayMessage 为 MannersAspect 的切点。该方案可达到同样的效果，但无需对 HelloWorld 代码进行任何改变。

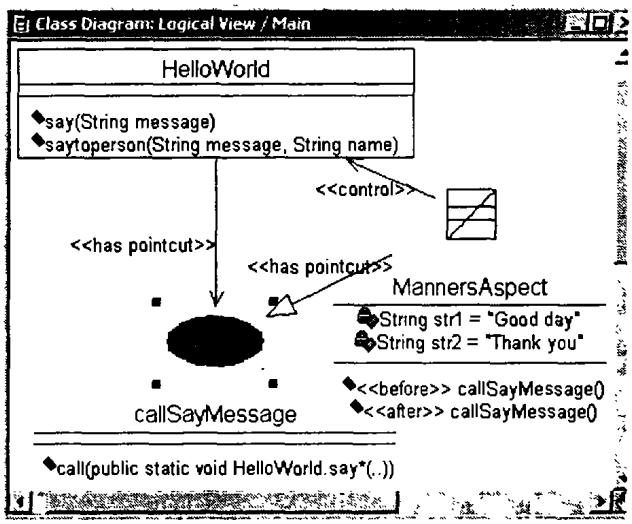

图2 MannersAspect 的逻辑设计视图

选择 callsayMessage 图标，运行.ebs 脚本程序，此时生成的代码框架如图3所示。

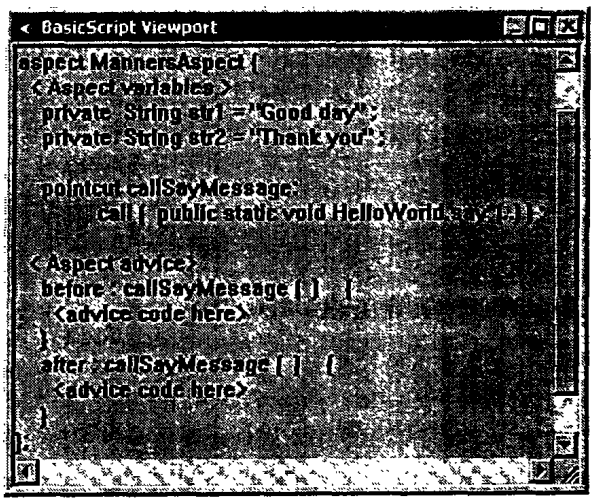

图3 由脚本自动生成的 aspect 代码框架

结语 本文通过扩展 UML 元模型元素来支持 AO 建模，对于减小 OO 和 AO 设计工具间的鸿沟进行了有益的尝试。

AOM 是当前有关 AO 的研究热点，并取得了许多可喜的成就。但目前仍有许多问题有待解决，例如，如何确定一个好的 AO 设计，或者说好的 AO 设计有什么特征，以及如何对这些特征进行量化分析等方面，都还有必要进行进一步的研究。此外，建立一种通用的 AO 建模标准也是目前急需解决的问题。

(下转第201页)

的代码就与其它代码分离开来。

2.2 记录和审计

记录和审计服务可以提供收集和分析信息系统的活动情况。使用它们的主要目的就是找到系统潜在的软件漏洞并发现原因。虽然记录和审计是安全中很重要的一环，但目前为止J2EE却没有提供标准的解决方案。目前通常的做法是，把程序的某些操作记录在日志文件中而一般的系统都不提供审计功能。虽然在J2EE中可以用编程性的安全来解决这个问题，但这引出了如下两个问题：1)要修改组件的源代码，这将极大地破坏组件的结构。记录和审计属于这种关注点的范畴，即可以很好地通过面向方面来实现。下面的程序清单演示了记录是怎样通过面向方面技术实现的。如果某个方法抛出给定的任何异常，就将这种情况记入日志。

```
aspect BankAspect{
    pointcut bankMethods();
    execution(public * bank.*(..)) && this(SessionBean);
    after() throwing (BankSecurityException e);
bankMethods(){
        Log log1 = Log.getInstance();
        log1.write(e);
    }
}
```

图5 使用AspectJ实现记录功能

审计功能也可以通过类似的方式实现，只要把对程序执行操作进行记录改为进行审计即可。下面就对容器和AOP实现安全特性的方法进行一个简要的对比：

表1 容器管理与AOP在实现安全方面的对比

	容器管理的安全		面向方面的安全
安全机制	说明性的安全性	可编程的安全性	
使用方式	被动式	宣告式	被动式
身份认证	局限于容器提供的功能	局限于容器提供的功能，如果使用第三方提供的服务则降低了通用性。	不用修改组件代码
记录和审计	不支持	需要修改组件代码	
结论	只能实现固定的几种安全特性。	宣告式并不能完全的解决模块分离问题。	因为不使用第三方服务，所以不存在兼容性，但各种特性的实现机制没有行业规范，所以通用性不好

注：使用方式说明和对比

使用方式	释义	优缺点
被动式	服务提供者在服务使用者不知情的情况下把服务强加给服务使用者	1. 可以让使用者完全不必关注与己无关的事情。 2. 有些服务与使用者关系密切，不能使用这种方式。
宣告式	服务使用者必须显示的调用服务提供者提供的服务	2. 使用者对服务如果不甚了解，可能会对同类的多种服务无从选择。 3. 如果要使用的服务有很多种，会造成使用者内部代码混乱，并可能忘记调用某些服务。

结论 在安全体系结构的建立方面，容器和AOP各有利弊。但是在实现第二类需求时，容器却无能为力了，而AOP可以较好地完成任务。容器所提供安全实现机制可以让开发人员不必介入安全方面的事务，而把它留给安全专家来进行处理，应用程序安全属性是在应用程序的描述文件中进行定义的。另一方面我们只能利用容器提供的有限的安全特性，它不具备足够的灵活性来实现更复杂和更具动态性的安全策略。我们也可以使用基于程序的安全实现方法，但是这却导致了安全逻辑和功能逻辑互相交叉不能很好分离。最后形成的代码就极度混乱并且难以维护。而使用AspectJ解决安全问题却具有更加灵活和可扩展的特性，这就可以产生独立的并具有灵活性的安全模块并且可以把它很方便地置入未考虑安全的应用程序中。这种方法就让应用程序不必为了引入安全特性而修改源代码。综上所述，目前为止最为完美的解决方案就是把二者进行有机的结合。

参考文献

1 Security Functionality Requirements. National Institute of Standards and Technology, 1992 97. dustry, tark. Integrate security infrastructures with JbossSX, JavaWorld, April 2001
2 Information Technology Security Evaluation Criteria (ITSEC). Department of Trade and InLondon 1991
3 http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/aspectj-home
4 王妍. J2EE中的安全 [EB/OL]. http://www-900.ibm.com/developerWorks/cn/java/l-j2eeSecurity/index.shtml
5 Stark S. Integrate security infrastructures with JbossSX, JavaWorld, April 2001
6 JAAS page at Sun web site. http://java.sun.com/products/jaas
7 Resource Access Decision Facility Specification, OMG, 2001. http:omg.org/docs/ formal/01-04-01.pdf
8 徐迎晓. Java安全性编程实例[M]北京. 清华大学出版社

(上接第198页)

参考文献

1 The emerging Technologies That Will Change the World. MIT Technology Review, Jan./Feb. 2001 issue
2 Workshop on "Aspect-oriented Modeling with UML". http://lglwww.epfl.ch/workshops/aosd-uml/
3 Suzuki J, Yamamoto Y. Extending UML with Aspects: Aspect Support in the Design Phase. In: AOP Workshop at ECOOP'99, Portugal, 1999
4 Clarke S, Walker RJ. Composition Patterns: An Approach to Designing Reusable Aspects. In: Proc. of ICSE, 2001
5 Aldawud O, Bader A, Elrad T. Weaving with statecharts. http://www2.umassd.edu/swsoc/workshops/
6 Lions J M, Simoneau D, Pitette G. Extending OpenTool/ UML Using Metamodeling: An Aspect Oriented Programming Case Study. In: Second Intl. Workshop on AOM, 2002
7 Aldawud O, Elrad T, Bader A. A UML Profile for Aspect Oriented Modeling. In: OOPSLA 2001 workshop on AOP
8 OMG. Unified Modeling Language Specification. Version 1.3, 1999
9 Rational Rose 2000e, Rose Extensibility User's Guide. Copyright (c) 1998-2000 Rational Software Corporation
10 Laddad R. I want my AOP. http://www.javaworld.com/ javaworld/jw-01-2002/. 2002