基于领域特征的AOP编织实现方法

李 淼[1] 白 勇[2] 张 波[1]

(重庆工业职业技术学院 重庆 400050)[1] (重庆电力高等专科学院 重庆 400053)[2]

摘 要 面向方面编程(AOP)通过横切关注点和编织技术来实现软件。基于领域特征的领域分析、设计过程获得领域的特征和特征关系,并在领域实现的前期实现特征的代码编制。采用面向方面编程技术,对领域特征中任务(Role)的关系分析、归类后,按面向方面编程技术提供的关注点(Concerns)、通知(Advice)、横切(Crosscutting)、编织(Weaving)等方法,针对领域特征中的Role进行编织,组成完整的可变化特征,以适应不同软件产品的需要。采用面向方面编程技术不会对已封装的Role产生耦合,增强了代码的易用性和可维护性。

关键词 面向方面,AOP,领域特征

AOP Weaving Method Based on Domain Feature

LI Miao[1] BAI Yong[2] ZHANG Bo[1]

(Department of Computer Science, Chongqing Industry Polytechnic College, Chongqing 400050, China)[1]

(Chongqing Electric Power College, Chongqing 400053, China)[2]

Abstract Aspect-oriented Programming provides crosscutting concerns and weaving method to implementing software. Domain analysis and domain design process based on domain feature provide domain feature and feature relationship, And prophase of domain implementing completed coding of any feature. Using aspect-oriented programming technique, analyzing role and relationship of roles of domain feature, use method that concerns, advices, crosscutting and weaving, weaving role of domain feature, composing feature of variation. Adapt requirement of different software product. Using aspect-oriented programming strive for lowest possible coupling, and increasing usability and maintainability.

Keywords Aspect-oriented, AOP, Domain-feature

1 引言

软件复用的研究和实践表明领域工程是其中的关键,即可复用软件资产(包括体系结构和构件等)的生产阶段,主要包括领域分析、领域设计和领域实现这3个活动[1]。经过领域分析与设计,形成领域特征Feature[2,3];领域实现的软件制品是软件体系结构(Architecture)、组件(Component)和类(Class)[5],组件和类是实现级的较小粒度的可复用软件制品。

AOP(Aspect-Oriented Software Programming)技术,是由Xerox Palo Alto研究中心(Xerox PARC)的研究人员于20世纪90年代末提出的一种新的程序设计思想和模型[6],它使用关注点(Concerns)、通知(Advice)、横切(Crosscutting)、编织(Weaving)等技术[6]。这些新技术、新思想在软件编程中,对减少代码混乱、软件功能灵活组织等方面都显示出了突出的优越性[6]。

在领域工程的最后阶段,即领域工程实现阶段,需要对各种可复用软件制品进行组装和维护[2]。本文论述了采用AOP技术来实现这一目标的一种方法。本文第2部分论述了特征的相关内容,并归类整理。第3部分讨论AOP技术的关键部分。第4部分综合本文的第2,3部分,以AspectJ为载体,在Java平台下实现领域特征的编织实现方法。

2 领域特征

2.1 特征的内容

领域特征(Feature)是系统中用户最终可见的、显著或特色的行为、能力、特点等[1],描述的是用户对系统的理解。领域特征不是对传统需求规约模型的完全替代,而是形成了一种互补的关系。特征包含一系列的任务(Role)。例如,一个音像出租系统中的视频出租特征如表1所示[5],完成每一个任务又由若干个实现方法组成。

表1 领域特征构成

特征 Feature	任务 Role	方法 Method
视频出租 VideoRental	合同 Contract	constructor Contract(Customer, Video) setCustomer(Customer) Customer getCustomer() setVideo(Video) VideogetVideo()
	合同维护 Contract maintenance	addContract(Contract) removeContract(Contract) Contract[] getAllContracts
	租借状态 Rented item	boolean isRented()
	租借人 Renter	int rents(Video[])

到稿日期:2008-07-10

李 淼 副教授,主要研究方向为软件复用技术、面向方面编程等。

领域分析与设计完成后，就会得到某一领域的一系列特征(Feature)以及这些特征的任务(Role)。

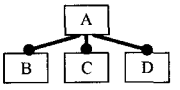
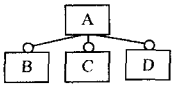
2.2 特征分类

不变特征：领域内，所有软件产品共有的特征。

可变特征：领域内，不同的软件产品具有不同的特征。

2.3 特征的关系[3]

表2 特征关系

关系名称	关系说明	图示
强制 Mandatory	如果选择了特征A，强制特征B必须被它的父特征A包含	
可选 Optional	如果选择了特征A，特征B可以被它的父特征A有选择的包含	
选一 Alternative	如果选择了特征A，必须从可选择的特征B、C、D中选择一个特征	
或 Or	如果选择了特征A，必须从可选择的特征B、C、D中选择一个或多个特征	

特征中的任务Role也会有强制、可选、选一、或这样的关系，这里不作详细的讨论。

3 AOP技术

面向方面程序设计(Aspect-Oriented Programming)和面向对象结合起来可以很好地完成对真实系统的横向和纵向的两个维度的建模，从而更加容易地构建稳定且易维护的软件系统。

3.1 AOP提供的技术支持[6,7,9]

切入点(Pointcuts)

切入点是指一个或多个连接点，可以理解成一个点的集合。切入点的描述比较具体，而且一般会跟连接点上下文环境结合。

通知(Advice)

指定到达特定切入点处应执行的代码。主要有3种基本类型。

前通知(before advice)是指在连接点之前，先执行通知中的代码。

后通知(after advice)是指在连接点执行后，再执行通知中的代码。后通知一般分为连接点正常返回通知和连接点异常返回通知等类型。

环绕通知(around advice)是一种功能强大的通知，可以自由改变程序的流程、连接点返回值等。

介绍(Introduction)

介绍是指给一个现有类添加方法或字段属性。介绍还可以在不改变现有类代码的情况下，让现有的类实现新的接口，或者为其指定一个父类，从而实现多重继承。

织入(weaving)

织入是把解决横切问题的切面模块，与系统中的其它核心模块通过一定策略或规则组合到一起的过程，并且不影响原系统的模块。

拦截器(interceptor)

拦截器是用来实现对连接点进行拦截，从而在连接点前或后加入自定义的切面模块功能。

目标对象(Target object)

指基于拦截器机制实现的AOP框架中，位于拦截器链上最末端的对象实例。

AOP代理(proxy)

是指在基于拦截器机制实现的AOP框架中，实际业务对象的代理对象。这个代理对象一般被切面模块引用，AOP的切面逻辑正是插入在代理对象中来执行的。

3.2 AOP的技术优势

AOP为开发者提供了一种描述横切关注点的方法，它允许定义程序代码执行顺序的关系，我们可以从几个方面来阐述它的优点。

(1)代码集中易于理解。使用AOP技术可以将需要出现多次的公用代码集中到一处实现，这样代码的修改范围就可以得到严格的控制，减少代码修改对系统稳定性的影响，减少代码的冗余度和耦合度，增强可读性，提高软件质量，解决了面向对象编程跨模块造成的代码混乱和代码分散问题。

(2)模块化横切关注点。每个关注点都使用最小耦合来进行处理，这样模块化处理的系统更容易理解和维护。

(3)系统容易扩展。Aspect模块根本不知道横切关注点，因此很容易通过建立新的方面加入新的功能。另外当向系统中加入新的模块时，已有的方面自动横切进来，使系统易于扩展。

(4)更好的代码重用性。AOP把每个方面实现为独立的模块，模块之间是松散耦合的。松散耦合的实现通常意味着更好的代码重用性，AOP在实现松散耦合这方面比OOP做得更好。

4 实现方法

4.1 AOP与OOP结合的领域特征实现方法

面向对象编程(OOP，Object-Oriented Programming)能很好地将问题简化，并封装成类，能很好地实现抽象。但是在类封装后，实现商务规则的复杂关系，组合成软件产品的过程中，类之间会形成耦合，使得原有的类形成复杂的关系，更严重的情况将会形成代码混乱。

如本文第3部分描述，面向方面编程(AOP，Aspect-Oriented Programming)提供了横切、通知技术，能将一段程序切入到其它软件程序中而不会影响原模块，使得附加的程序代码可以独立于原系统模块。从原系统模块的角度看，没有任何变化。

将AOP技术与OOP技术结合，应用到领域实现阶段。从代码层入手，将两者的优势结合到一起。并找到了领域特征关系与AOP技术实现的方法。本文认为，实现领域特征关系的关注点是如何在不影响原模块特征代码的前提下，将新的实现特征关系的代码编织到软件产品中。从特征关系的分析发现，当Feature A执行后，会以一种方式执行Feature B。传统的实现方式是在原系统内建立一个链接关系类，当关系发生变化时，例如，需要执行Feature C，必然会修改原系统的链接关系类。类之间产生了新的耦合关系，不利于软件的维护。采用AOP技术，只关注横切位置，特征关系的实现代码

独立于原系统代码之外。任何关系的增、减、变化，都不会影响原系统代码。

4.2 特征(Feature)和任务(Role)的实现

通过本文第2,3部分的论述，领域工程中分析设计完成后，就得到了某个领域的特征集 Feature1，Feature2，……。组成每个特征的是一些任务 Role，Feature1 的任务集用 role11，role12，role13，……表示，Feature2 的任务集用 role21，role22，role23，……表示。

这些任务(Role)均独立设计完成，在 Java 平台下，仍然以传统的面向对象编程(Object-Oriented Programming)的类形式实现。如此代码实现的原因有3个：

(1)领域工程还没有完善、成熟的分析、开发工具。

(2)目前的软件开发工具提供的是面向对象的开发方法。

(3)软件的代码级重用仍然以封装类和组件为主。

用于本文的测试代码片段如下：

```
package Feature;
public class Role {
    public static void main(String[] args) {
        new Role(). role11();
    }
    public static void role11(){
        //Do any business rule
        outMessage("role11! \n");
    }
    public static void role12(){
//      Do any business rule
        outMessage("role12\n");
    }
    public static void role13(){
//      Do any business rule
        outMessage("role13\n");
    }
    public static void outMessage(String outputMessage){
        System. out. println(outputMessage);
    }
}
```

以上代码片段省略了实际应用系统的商务处理规则代码，仅以显示语句代码作为实际代码的仿真输出。

4.3 横切特征关系实现

本文以 Mandatory 关系为例，其它关系的实现只需要增加判断代码，这里就不作更详细的描述。

Feature A 的任务 role12 依赖 role11 而需要强制执行，在 AOP 中增加横切点(pointcut Mandatory)，在 role11 执行之后增加通知(after advice)并执行 role12。在 Java 平台下，采用 AspectJ 实现。本文的 Aspect 测试代码如下：

```
package CrossCutting;
public aspect cutrole {
    public pointcut Mandatory():
     call(void Feature. Role. role11());
    after() returning:Mandatory() {
     Feature. Role. outMessage("After Role11,Mandatory:");
     Feature. Role. role12();
    }
}
```

运行结果如下：

```
role11!
After Role11,Mandatory:
role12
```

测试的结果可以看出

(1)本文叙述的方法达到了关系代码独立于原系统代码的目的，如图1所示。

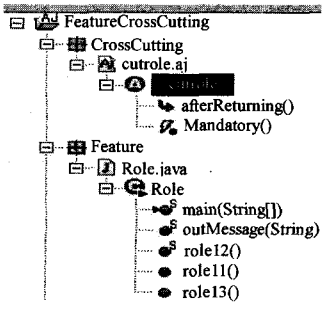


图1 代码独立

从图1中可以看出，特征包 Feature 和横切包 CrossCutting 相互独立，在没有 CrossCutting 包时，Feature 包程序仍然能独立运行，但是有依赖关系的程序没有运行；采用 Aspect 加上横切包 CrossCutting 后，具有依赖关系的程序得以运行。

(2)横切点和横切关系清晰，如图2所示。

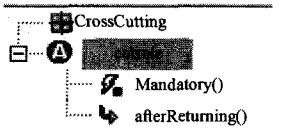


图2 横切点及横切关系

从图2可以看出，横切点名称 Mandatory，执行通知 After，清晰地展现了横切点和横切关系。

结束语 本文提出的基于领域特征的 AOP 编织实现方法能保持原系统的代码独立性，实现领域特征关系的编码。同时可应用到 Aspect C#，AspectC++。不足之处在于对 Aspect 的支持工具还不完善，而且与程序实现的语言相关；后续研究包括对复杂的交织关系进行深入研究；根据领域工程的研究进展，发现新的领域特征关系，研究新的实现方法。研究特征关系到代码实现的映射。

参 考 文 献

[1] 张伟，梅宏. 一种面向特征的领域模型及其建模过程. 软件学报，2003,14(8):1345-1356. http://www. jos. org. cn/1000-9825/14/1345. htm

[2] Kang K C, Lee J. Patrick Donohoe. Feature-Oriented Product Line Engineering. IEEE SOFTWARE. July/August 2002

[3] Beuche D, Papajewski H. Variability management with feature models. Science of Computer Programming archive, 2004, 53(3)

[4] Sochos P, Philippow I, Riebisch M. Feature-Oriented Development of Software Product Lines: Mapping Feature Models to the

Architecture // NODe 2004. LNCS 3263. Berlin Heidelberg: Springer-Verlag, 2004: 138-152

[5] Jansen A G J, Smedinga R, van Gurp J, et al. First class feature abstractions for product derivation// IEE Proc. -Softw. 2004, 151(4)

[6] Filman R E, Elrad T, Clarke S, et al. Aspect-Oriented Software Development. ISBN: 0-321-21976-7. October 2004

[7] AspectJ. in action: PRACTICAL ASPECT-ORIENTED PROGRAMMING. RAMNIVAS LADDAD. ISBN 1-930110-93-6. Printed in the United States of America

[8] Lee K, Kang K C. Feature Dependency Analysis for Product Line Component Design // ICSR 2004. LNCS 3107. Berlin Heidelberg: Springer-Verlag, 2004: 69-85

[9] 刘小龙，王忠. 面向方面编程研究. 开发研究与设计技术. 武汉工程大学计算机科学与工程学院

（上接第 290 页）

与 Pajek, JUNG 等大型网络分析工具相比，JPAC 采用 XML 文件保存网络的结构信息，具有简单易用的特点，通用性也更好。JPAC 功能强大，可以分析大型 Java 系统的结构，并生成与复杂网络相对应的随机网络，比较二者的异同。JPAC 可以计算基于复杂网络的各度量值，并用模拟退火算法分离出社区结构，亦即软件的拓扑意义上的模块结构。通过对生成的模块结构与原结构的比较，可以分析各种模块划分方法的优劣，从而对软件系统更好地理解、评估、预测、控制和改进。

结束语 如何提高软件的质量，始终是软件工程领域研究的重要方向。基于度量的量化管理是目前最有效的质量保证手段之一。随着面向对象方法的日益流行，软件系统规模的不断增大，如何度量面向对象大规模软件系统，成为一个亟待解决的问题。

本文运用复杂网络理论解决这个问题。研究发现，软件系统中存在大量复杂网络现象。通过研究复杂网络所共有的一些特征，从元素级、模块级、网络级 3 个不同的粒度提出基于复杂网络的软件系统的各种度量。介绍了大型 Java 程序复杂网络描述和度量工具 JPAC。JPAC 可以分析大型 Java 系统的结构，并计算基于复杂网络的各度量值。通过计算这些度量值，可以定量地刻画和分析软件的结构和行为，以便更好地理解、评估、预测、控制和改进。

用 JPAC 分析了 JDK 各版本，我们发现其 JSCG 图具有明显的复杂网络的特征。我们计算了基于复杂网络中度量，并分析了 JDK 的演化。

下一步的工作是研究尽可能多的软件系统，提出度量的定量标准，并指导于实际的软件开发中。我们可以计算出各结点类的 z-score 的值和参与系数的值，并按分散系数的值将各结点分类，从而与面向构件的软件开发有效地结合起来。我们的工作也可应用于遗产系统构件的识别与提取、构件的度量等问题。

参 考 文 献

[1] Fenton N E, Pfleeger S L. Software Metrics A Rigorous and Practical Approach. Second Edition. Boston: PWS Pub., 1997

[2] Chidamber S R, Kemerer C F. A Metrics Suite for Object-oriented Design. IEEE Trans. on Soft. Eng., 1994, 20(6): 476-493

[3] Brito e Abreu F. The MOOD Metrics Set // Proc. of ECOOP'95 Workshop on Metrics. Aarhus, Denmark, 1995

[4] Strogatz S H. Exploring complex networks. Nature, 2001, 410: 268-276

[5] Albert R, Barabasi A-L. Statistical mechanics of complex networks. Rev. Mod. Phys., 2002, 74: 47-97

[6] Dorogovtsev S N, Mendes J F F. Evolution of Networks: From Biological Nets to the Internet and WWW. Oxford: Oxford University Press, 2003

[7] Newman M E J. The structure and function of complex networks. SIAM Review, 2003, 45: 167-256

[8] Potanin A, Noble J, Frean M, et al. Scale-free Geometry in Object-oriented Programs. Comm. ACM, 2005, 48: 99-103

[9] Valverde S, Ferrer-Cancho R, Sole′ R. Scale-free Networks from Optimal Design. Europhysics Letters, 2002, 60: 512-517

[10] Myers C. Software Systems as Complex Networks: Structure, Function, and Evolvability of Software Collaboration Graphs. Physical Rev. E, 2003, 68

[11] Wheeldon R, Counsell S. Power Law Distributions in Class Relationships // Proc. Third IEEE Int'l Workshop Source Code Analysis and Manipulation. 2003

[12] Tamai T, Nakatani T. Analysis of Software Evolution Processes Using Statistical Distribution Models // Proc. Int'l Workshop Principles of Software Evolution (IWPSE). 2002: 120-123

[13] Concas G M. Power-Laws in a Large Object-oriented Software System. IEEE Trans. on Soft. Eng., 2007, 33(10): 687-708

[14] 李冰，王浩，等，基于复杂网络的软件复杂性度量研究. 电子学报，2006, 34(B12): 2371-2375

[15] McCabe T A. Complexity measure. IEEE Trans. on Soft. Eng., 1976, 2(4): 308-320

[16] Guimera R, Amaral L. Cartography of complex networks: modules and universal roles. Nature, 2005, 433(7028): 895-900

[17] Newman M E J, Girvan M. Finding and evaluating community structure in networks. Phys. Rev. E, 2004, 69: 026113

[18] Newman M E J. Modularity and community structure in networks // Proc. Natl. Acad. Sci. USA, 2006, 103: 8577-8582

[19] 陈焘，王树森. 基于复杂网络的 JDK 代码结构演化研究. NASAC, 2006

[20] 汪小帆，李翔，陈关荣. 复杂网络理论及其应用. 北京：清华大学出版社，2006