文章编号:1001 - 9081(2009)03 - 0686 - 04

一种基于概念模型的切点定义方法

葛君伟,张 鹏,方义秋
(重庆邮电大学 计算机科学与技术学院,重庆 400065)
(xiaozhang206@yahoo.com.cn)

摘 要:针对面向方面编程中存在的切点软化问题,提出一种可视化的切点定义方法,切点定义依据一种概念模型而不再依赖于基础程序的具体结构,实现了切点定义与基础程序的解耦,提高了切点鲁棒性。概念模型是对 AspectJ 切点表达式的可视化的抽象概括,通过扩展 UML 元模型实现,文中称之为切点模型。同时,扩展了 Rational Rose,使其支持切点模型的建模,并实现了由切点模型自动生成方面框架代码。

关键词:面向方面编程;切点软化;AspectJ;UML 扩展
中图分类号: TP311.11 **文献标志码**:A

Approach to pointcut definition based on conceptual model

GE Jun-wei, ZHANG Peng, FANG Yi-qiu
(*College of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing* 400065, *China*)

Abstract: This paper proposed a visual approach to pointcut definition to tackle the fragile pointcut problem of aspect-oriented programming. The pointcut definition was based on a conceptual model not directly on how the base program was structured, which had effectively decoupled the pointcuts from the base program's structure. Therefore, it improved the robustness of the pointcut. The conceptual model was, as an abstraction of the AspectJ pointcut descriptor, achieved through extending the UML meta-model elements. Therefore, it was called pointcut model. The Rational Rose was also extended, so as to support the pointcut model and automatically generate the aspect skeleton code from the model.

Key words: aspect-oriented programming; fragile pointcut; AspectJ; UML extension

0 引言

面向方面编程[1](Aspect-Oriented Programming,AOP)技术是一种新兴的编程技术,它对传统编程技术进行了扩展,将软件系统中影响多个模块的横切关注点封装到一个可重用模块 aspect 中,并通过切点在编译或运行时将 aspect 自动织入到相应连接点处。但现有 AOP 技术的切点定义与基础程序结构耦合过紧,对基础程序看似安全的修改都可能导致切点意外匹配或丢失特定连接点,造成切点软化损坏[2],严重阻碍了面向方面软件的后期演化。因此,如何有效解决切点的软化损坏,创建鲁棒性更好的切点,已成为 AOP 领域的研究热点。

导致切点软化损坏的原因在于:1)切点定义所依赖的层次太低,与基础程序具体结构耦合太紧,开发人员不得不在代码层次处理切点问题;2)目前的切点语言表达力较弱,对连接点集合的处理粒度较差,基于枚举的切点定义粒度过细,而基于模式的切点定义粒度过粗;3)判断是否出现连接点匹配错误时,对开发人员的工具支持不够,开发人员只能在代码层次人工寻找匹配错误。

针对以上原因,本文提出一种可视化的切点定义方法,将切点软化问题提升到概念层次解决。首先,对 AspectJ 的切点表达式进行抽象概括,定义一种概念模型,本文称之为切点模型。切点定义依据此模型,不再依赖于具体的基础程序结构,实现了与基础程序的解耦;其次,模型中的元素依据 AspectJ 的切点语义对基础程序中的各种代码实体抽象分类并对这些代码实体施加一种概念约束,以此保证切点模型能正确表示 AspectJ 的切点语义。抽象分类屏蔽了连接点集合的粒度问题,概念约束限制了切点捕获的连接点集合。

这样,通过切点模型,我们就把切点软化产生的连接点匹配错误问题转化为切点模型与基础程序之间的匹配问题。基础程序演化后,只需通过检查切点模型中元素的抽象分类以及概念约束是否仍能与基础程序保持一致,即可判断是否出现了连接点匹配错误。若保持一致,则不需修改切点;否则修改模型与基础程序保持一致,生成新的切点。

1 基于 UML 切点模型的定义方法

本文所提切点模型是 AspectJ 切点表达式的抽象概括,切点表达式由切点指示符逻辑组合而成。目前,本模型目前仅支持 call,within,target,get 和 set 这几个常用的静态切点指示符组合的切点表达式。

1.1 切点元模型的构造

从切点指示符自身的意义考虑,within 主要用于表示切点在哪个范围内挑选连接点,而这个范围一般多为一个或多个类的集合;call,get 和 set 则说明在 wihtin 集合内挑选什么类型的连接点或连接点集合,如函数的调用,字段的获取或修改等;target 则进一步说明在 within 集合中这些类型的连接点中切点具体需要匹配哪个或哪些连接点,实质上 target 中的类型即为连接点集合中函数或字段的实现者或拥有者。在实际开发中,经常把这几个指示符组合一起形成有效的切点表达式来选择需要的连接点集合,如:within(TypePattern) &&

收稿日期:2008 - 10 - 06;**修回日期**:2008 - 11 - 25。
基金项目:重庆市自然科学基金资助项目(2005BB2059);重庆市教委科技研究项目(KJ040511)。
作者简介:葛君伟(1961 -),男,重庆人,教授,主要研究方向:面向方面软件; 张鹏(1983 -),男,河南周口人,硕士研究生,主要研究方向:面向方面程序设计; 方义秋(1963 -),女,重庆忠县人,副教授,主要研究方向:软件工程。

call(mehtod signature && target (Type) , get (signature) && target(Type) , set(signature) && target(Type) 等。

在这里若我们把切点指示符 call,get 和 set 选择的连接点集合看成一种资源,则可以认为 within 所表示的类集合请求使用了这些资源,而 target 所代表的这些类则是这些资源的实现者。根据实际需要将这三个集合按照切点语义组合在一起,就构成了切点需要的连接点集合,基于以上思想,本文构建了如图 1 所示的切点元模型。

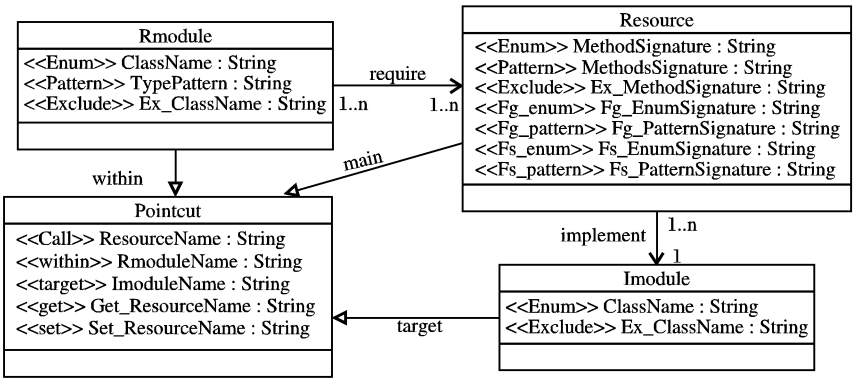

图 1　基于 UML 扩展的切点元模型

本文在元模型中加入 Pointcut 元素是基于如下考虑的,一是为了在模型中实现可视化切点定义;二是考虑要从切点模型中自动生成 AspectJ 切点代码,模型中必须有 Pointcut 元素存在;三是实现了切点模型与切点表达式的对应,使得切点模型简单易于理解,表达直观明了。

由元模型可以获知,在定义的切点模型中,应当显示 Rmodule, Imodule, Pointcut 和 Resource 这四种类型元素。Rmodule 是 Resource 的请求者,由于 Rmodule 可能包含多种类型的连接点,因此它可以对多个 Resource 发出请求。Imodule 是 Resource 的实现者,它可以实现多个 Resource。由于 Resource 中可以同时包含函数调用连接点以及字段调用连接点,因此,它可以接受多个 Rmodule 的请求,但是只能由一个 Imodule 实现。只有将以上三个元素有效组合才能形成切点定义的连接点集合,因此,本文将其他三个元素与 Pointcut 之间的关系设定为 UML 中的一般化关系。

1.2　代码实体的抽象分类与概念约束

切点模型中 Rmodule,Imodule 及 Resource 对代码实体的抽象分类通过将枚举以及模式匹配设定为模型元素的属性实现,同时为模型元素定义了 Exclude 属性用于排除因模式匹配而造成的意外匹配的类或函数。模型元素中的属性,根据实际需要都是可选的,这样切点模型对连接点集合的控制粒度就更为灵活。

注意,由于 target 指示符只能与类型匹配[3],因此,Imodule 中只设定了 Enum 属性而没有定义 Pattern 属性。与类和函数的意外匹配相比,字段的意外匹配的几率相对较小,因此,本文并没在 Resource 元素里设定字段的 Exclude,而仅针对 get 和 set 设定了相应的枚举和模式匹配,如 Fg_enum, Fg_pattern, Fs_enum 和 Fs_pattern。

为了实现对切点所捕获连接点集合的限制,更好地改善面向方面程序的模块性,本文将 Pointcut 与 Rmodule, Imodule 以及 Resource 之间关系看成一种概念约束,并与 Pointcut 中属性的定义相对应,即只有 Pointcut 中定义了相应模型元素的名字,才能将该模型元素与 Pointcut 设定为一般化关系。注意,由于 Resource 元素可能既包含函数又包含字段,而对函数与字段的拦截一般认为是两种横切关系,需要分别定义两个切点来完成。因此,Pointcut 与 Resource 之间关系的设定不会存在重复,将在下文中举例说明。

1.3　元模型的实现

鉴于 UML 在建模领域的成熟度、普遍适用性以及丰富的可扩展机制,本文所提切点模型主要通过为 UML 相关元模型添加新的版类来实现。版类是在已定义的模型元素基础上构造的一种新的模型元素,与现有的模型元素有相似的结构,但是附加了一些限制,具有新的解释和图标。

对于切点模型中的元素 Rmodule, Imodule, Resource 以及 Pointcut 本文统一为 Class 元模型添加与模型元素名字相同的版类来实现,如: << Rmodule >> , << Resource >> 。同时,为了与 Class 图标区别,本文依次为以上版类创建了新的图标,如图 2 右边所示。而切点模型中元素的属性,通过为元模型 Attribute 添加与属性名字相对应的版类来实现。

关于模型中 Rmodule, Resource 及 Imodule 之间的关系,本文通过为元模型元素 Association 添加新的版类实现。根据切点语义,本文将 Rmodule 与 Resource 之间的关系 require 进一步细化为 << call >> , << get >> 与 << set >> 三种新的版类,而 Resource 与 Imodule 之间的关系则用 << impl >> 表示,意为 implement。以上三种元素与 Pointcut 之间的关系为一般化关系,为了切点模型更易理解,本文为元模型 Generalization 添加 << within >> , << main >> 和 << target >> 三种新的版类。

为了支持切点模型的建模,本文利用 Rational Rose 的扩展接口[4],编写了一个. ini 配置文件来定制本文上面扩展 UML 的版类和图标。同时,通过编制一个 Rose 脚本文件 Pointcut. ebs 实现了切点模型的代码自动生成功能。

1.4　举例说明

图 2 为本文在 Rose 中利用上面定义的版类和图标构建的一个切点模型。由于在一个面向方面软件系统中,可能存在多个横切关注点(如:日志、权限等),并且一个关注点会被多个横切关注点横切,因此,一个切点模型可由多个切点元模型组合而成,包含多个切点的定义。为了简单说明问题,除 Log 与 Security 外,本文没有给图 2 切点模型中的元素添加任何属性。

依据本文所构造的元模型,可以看出此切点模型定义了三个切点分别为 Log,Security 和 Notice。同时,根据切点模型中 Pointcut 的属性,我们可对以上 3 个切点做如下解读:

对于 Log 切点,它表示 UI 模块里所有与 BussinessAction 相关的函数调用连接点都会被捕获,以记录日志,而不考虑这些函数是何种类型的。Notice 切点则表示所有对 DataOperation 里字段的赋值都会被捕获,以提示可能存在的风险,这里不考虑是在哪个类里进行的赋值。而 Security 切点则表示 UI 模块对 DataOperation 里某些类型函数的调用会

被捕获,以检查是否有权限。

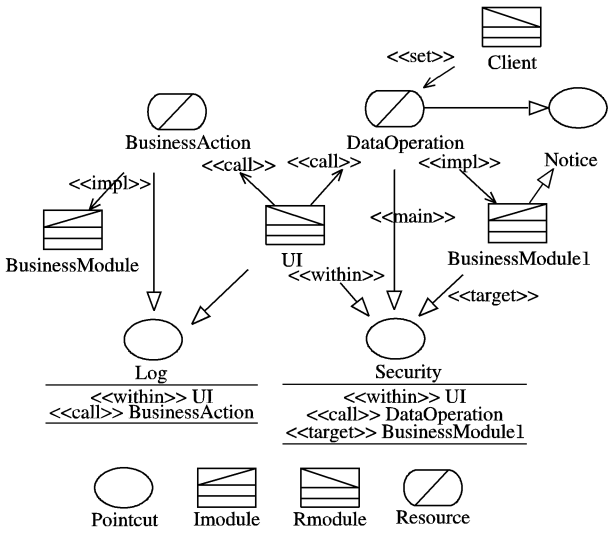

图 2 基于 UML 扩展的切点模型

从图 2 的切点模型可以看出,Pointcut 中属性的定义必须与模型中其他三个元素的关联相对应,以保证切点模型符合概念约束,捕获正确的连接点集合。如:切点 Log( )没有定义属性 << target >> ,因此,它与 BussinessModule 不存在任何关联。同时,图 2 的切点模型清晰地展现了基础程序中各种横切关系的分布,有助于开发人员之间的交流,准确掌握多个方面之间的交互。

在图 2 中,除了 Security,本文并没有标出 Log 和 Notice 与其他模型元素之间的关系。事实上,在对 AspectJ 的切点语义非常了解的情况下,可以省去 Pointcut 与其他模型元素之间关系的标注,因为它完全可从 Rmodule 与 Resource 之间的关系推理出来。

2 实验分析

我们通过文献[5]里的一个例子验证本方法的效用,同时进一步说明本方法如何应用。在下面的例子中,通过在 Client 类里调用 Line 和 Point 对象的 moveBy( ),setX( )或 setY( )函数来移动这些对象。同时,定义切点 change( )捕获对 moveBy( ),setX( )以及 setY( )的调用,以实现方面 UpdateSignaling 在对象每次移动后插入 update( )函数自动更新屏幕。

```
class Client{
  // calls methods of Point and Line class
}
interface Shape{
  public moveBy(int dx, int dy)
}
class Point implements Shape{
  int x, y;
  public int getX(){ return x; }
  public int getY(){ return y; }
  public void setX(int x){this.x = x;}
  public void setY(int y){this.y = y;}
  public void moveBy(int dx, int dy){
    x += dx; y += dy;
  }
}
class Line implements Shape{
  private Point p1, p2;
  public Point getP1( ) {return p1};
  public Point getP2( ) {return p2};
  public void moveBy(int dx, int dy){
    p1.x += dx; p1.y += dy;
    p2.x += dx; p2.y += dy;
  }
}
aspect UpdateSignaling{
  Pointcut change( ):
  call(void Point.setX(int))
  || call(void Point.setY(int))
  || call(void Shape.moveBy(int, int));
  after( ) returning: change( ){
    Display.update( );
  }
}
```

由本文第 1 章的分析可知,现有切点 change 鲁棒性较差,在基础程序演化时极易软化损坏。比如,若把 Point 类的域 x 和 y 修改为私有,则 Line 类必须通过调用 Point 的setX( )和 setY( )来更新自身的 Point 对象,因此,moveBy( )函数必须修改如下:

```
public void moveBy(int dx, int dy){
  p1.setX(p1.getX( ) + dx);
  p1.setY(p1.getY( ) + dy);
  p2.setX(p2.getX( ) + dx);
  p2.setY(p2.getY( ) + dy);}
```

此时切点 change 已软化损坏,出现连接点匹配错误,由 Line.moveBy( )引发的对 Point.setX( )和 Poin.setY( )的调用也被切点 change 捕获,同时触发 Display.update( )更新屏幕,但在 Client 类里并未移动 Point 对象。这个问题也可以通过 AspectJ 的切点指示符 cflowbelow( )来解决,把 advice 修改如下即可:

```
after( ) returning :
change( ) &&! cflowbelow(change( )){
  Display.update( );
}
```

这样 advice 只会匹配在类这一层次对 Point.setX( )和 Point.setY( )的调用。但由于切点 change( )本身易于软化损坏,因此,这种切点不能解决实质问题,关键是在出现连接点匹配错误时更难于发现。

本文利用基于切点模型的定义方法解决此问题。按照本文的思想,Client 类是一个资源的请求者,而这些资源拥有者为 Point 和 Line 类。这里我们仅关心 Client 类对 Point,Line 函数的调用,而忽略图形对象内部间的函数调用。基于以上分析,本文构建如图 3 所示的切点模型。

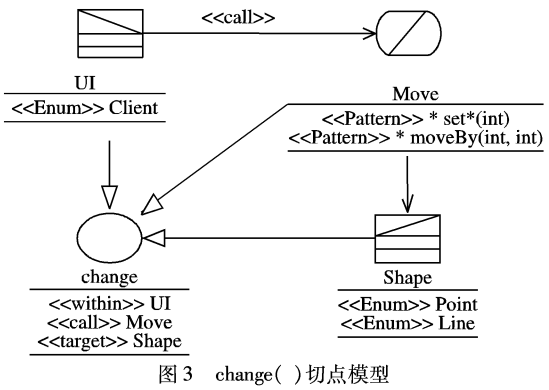

图 3 change( )切点模型

在构建好如图 3 所示的切点模型后,选定 change 图标,

运行 Pointcut. ebs 脚本,得到如图 4 所示的 Aspect 框架代码。

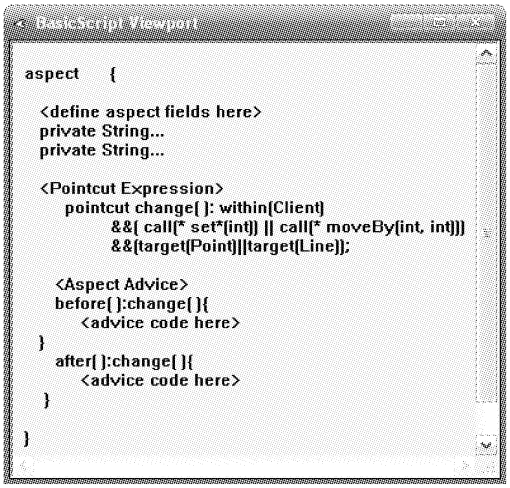

图 4　生成的 Aspect 代码框架

由图 4 可知,自动生成的切点 change( )准确表达了本文所要捕获的连接点集合。由于切点模型是切点表达式的抽象概括,因此本文仅能生成包含完整切点的方面代码框架,而方面的可见性、名字、字段以及通知的具体内容,仍需开发人员手工添加。

此时,考虑把 Point 类的域 x 和 y 修改为私有,由于模型元素的抽象分类及概念约束依然能与基础程序保持一致,因此不需修改切点。

若在基础程序中添加一个 Circle 类,同样实现了 Shape 接口。此时,切点模型与基础程序不再一致,模型元素 Shape 并未包含 Circle。此时,需修改模型,通过枚举的方式将 Circle 添加到 Shape 中。

当程序演化后期图形对象过多时,若不再关注图形的具体类型,此时须重新定义 change,删除 target 类型属性及 change 与 Shape 之间的关系,以符合概念约束的限制,表示所有图形移动的函数调用连接点都会被捕获。

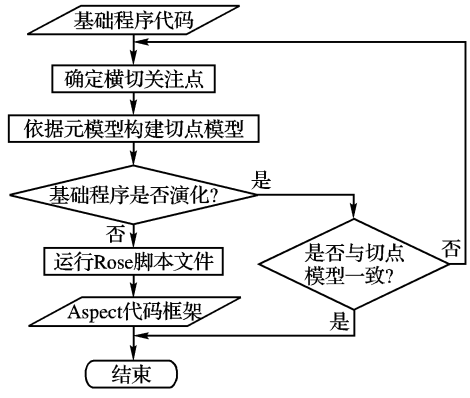

图 5　基于概念模型的切点定义流程

3　相关工作

文献[2]提出一种基于视图的切点定义方式,视图是代码实体的抽象概括,并通过两种约束与基础程序保持一致。基础程序演化后,通过视图工具 IntensiVE 检查视图是否与基础程序一致,判断是否切点是否软化损坏。本文所提切点模型与视图的作用相似,但切点模型是 AspectJ 切点表达式可视化的抽象概括,本身包含了切点语义,更易于理解面向方面程序中方面的交互。同时,切点模型的概念约束可限制切点捕获的连接点集合,有利于改善面向方面软件的模块性。与视图不同,本文的切点模型通过扩展 UML 和 Rational Rose 实现,具有更好的通用性和实用性。

文献[6-7]开发了 PCDiff 工具,用于比较不同版本的基础程序中与切点匹配连接点集合的变化,以此发现基础程序演化后是否出现了连接点匹配错误。但 PCDiff 有时不能准确发现基础程序连接点集合的变化,特别是基础程序中添加新的代码实体时。

文献[8-9]提出可在切点定义中引入 Java 标记,本文认为标记只能实现切点定义与基础程序的部分解耦,且在程序中出现连接点匹配错误时更难于发现。与之相反,切点模型在实现切点与基础程序解耦的同时,将连接点匹配错误可视化地展现出来。

文献[10-11]提出表达力更强的切点语言,提高了切点的鲁棒性。文献[12-13]利用推理逻辑编程来自动生成鲁棒性更好的切点。但以上几种方法,都存在切点软化损坏更加难于发现、调试的问题,并且需考虑编译器匹配问题。

文献[14]提出横切编程接口的概念,并通过一些设计规则[15]实现了基础程序代码与方面代码演化的相互独立。

4　结语

本文针对当前面向方面语言中存在的切点软化损坏问题,提出一种基于切点模型的定义方法。该方法一方面实现了切点定义与基础程序解耦,获得了鲁棒性更好的切点;另一方面通过模型对代码实体的抽象封装,屏蔽了连接点集合的粒度问题,并将连接点匹配错误可视化地展现出来,有效避免了切点软化损坏。最后,本文通过编写 Rational Rose 的脚本文件,实现了切点模型的代码自动生成。

目前切点模型仅支持几个常用的切点指示符,仍不能彻底解决切点软化损坏问题,本文下一步考虑将其余的切点指示符引入模型,如 execution,cflow,withincode 等。

参考文献:

[1] KICZALES G, LAMPING J, MENDHEKAR A, *et al*. Aspect-oriented programming[C]// Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97), LNCS 1241. Berlin: Springer-Verlag, 1997: 220-242.

[2] KELLENS A, MENS K, BRICHAU J, *et al*. Managing the evolution of aspect-oriented software with model-based pointcuts[C]// Proceedings of the 20th European Conference on Object-Oriented Programming (ECOOP'06), LNCS 4067. Berlin: Springer-Verlag, 2006: 501-525.

[3] COLYER A, CLEMENT A. Eclipes AspectJ 中文版[M]. 钱竹青, 邹正武, 译. 北京:清华大学出版社,2006.

[4] Rational Software Corporation. Rational Rose 2000e, Rose Extensibility User's Guide[Z]. Rational Software Corporation, 2000.

[5] KICZALES G, MEZINI M. Aspect-oriented programming and modular reasoning[C]// Proceedings of the 27th International Conference on Software Engineering (ICSE'05). New York: ACM, 2005: 49-58.

[6] KOPPEN C, STOERZER M. PCDiff: Attacking the fragile pointcut problem[C]// First European Interactive Workshop on Aspects in Software (EIWAS). Berlin, Germany: [s.n.], 2004.

[7] STOERZER M, GRAF J. Using pointcut delta analysis to support evolution of aspect-oriented software[C]// ICSM'05: Proceedings of the 21st IEEE International Conference on Software Maintenance. Washington, DC: IEEE Computer Society, 2005: 653-656.

(下转第 715 页)

条 PCIRCFG 测试序列。

遍历 PCIRCFG 生成测试序列的步骤如下:

1)首先访问根 RCFG 节点,以根节点为待处理节点 r。

2)基于测试覆盖准则,获得 r 的 RCFG 测试序列集合 P。

3)①若 r 为非根节点的叶子节点,将 r 的父节点作为待处理节点,重复此步骤 3)。②若 r 中的子节点都已处理完毕,将 r 的 RCFG 测试序列集合 P 与 r 所有的儿子节点的 PCIRCFG 测试序列集合进行路径合并,获得以 r 为根节点的子 RCFG 树的 PCIRCFG 测试序列集合。若 r 为根节点,终止;若 r 不是根节点,将 r 的父节点作为待处理节点,重复步骤 3)。③若 r 的子节点还未处理完,选择一个未处理的子节点作为待处理节点,返回 2)。

4.4　实例验证

选用完全 PCIRCFG 路径覆盖准则,图 2 将得到 24 条测试序列,采用本文改进的覆盖准则,则只需要 6 条测试序列就能满足所有消息节点至少覆盖一次的要求。6 条测试序列中,既经过 m1 又经过 m5 的序列有 5 条:P1 = (start(m0,A),m1(m0,A),start(m1,B),m3(m1,B),start(m3,D),end(m3,D),end(m1,B),m5(m0,A),start(m5,C),m1(m5,C),start(m1,B),m3(m1,B),start(m3,D),end(m3,D),end(m1,B),end(m5,C),end(m0,A)),P2 = (start(m0,A),m1(m0,A),start(m1,B),m3(m1,B),start(m3,E),end(m3,E),end(m1,B),m5(m0,A),start(m5,C),m1(m5,C),start(m1,B),m3(m1,B),start(m3,E),end(m3,E),end(m1,B),end(m5,C),end(m0,A)),P3 = (start(m0,A),m1(m0,A),start(m1,B),m3(m1,B),start(m3,F),end(m3,F),end(m1,B),m5(m0,A),start(m5,C),m1(m5,C),start(m1,B),m3(m1,B),start(m3,F),end(m3,F),end(m1,B),end(m5,C),end(m0,A)),P4 = (start(m0,A),m1(m0,A),start(m1,B),m2(m1,B),start(m2,C),end(m2,C),m3(m1,B),start(m3,F),end(m3,F),end(m1,B),m5(m0,A),start(m5,C),m1(m5,C),start(m1,B),m2(m1,B),start(m2,C),end(m2,C)m3(m1,B),start(m3,F),end(m3,F),end(m1,B),end(m5,C),end(m0,A)),P5 = (start(m0,A),m1(m0,A),start(m1,B),m2(m1,B),start(m2,C),end(m2,C),m3(m1,B),start(m3,F),end(m3,F),m4(m1,B),start(m4,D),end(m4,D),end(m1,B),m5(m0,A),start(m5,C),m1(m5,C),start(m1,B),m2(m1,B),start(m2,C),end(m2,C)m3(m1,B),start(m3,F),end(m3,F),m4(m1,B),start(m4,D),end(m4,D),end(m1,B),end(m5,C),end(m0,A))。由于已经对 m1 扩展过了,所以只经过 m1 的序列只要选一条:P6 = (start(m0,A),m1(m0,A),start(m1,B),m2(m1,B),start(m2,C),end(m2,C),m3(m1,B),start(m3,D),end(m3,D),m4(m1,B),start(m4,D),end(m4,D),end(m1,B),end(m0,A))。

5　结语

多态性是面向对象技术中的重要特征,对多态性的测试也对传统软件测试方法提出新的挑战。本文通过分析多态性在类图和协作图中的表示及其对测试的影响,提出了多态性扩展的结合类信息的函数间约束控制流图 PCIRCFG,并根据改进的覆盖准则给出了 PCIRCFG 线索的提取算法和实例描述。

参考文献:

[1] ALEXANDER R T. Testing the polymorphic relationships of object-oriented programs [D]. Fairfax, VA, USA: George Mason University, 2001.

[2] PILSKALNS O, WILLIAMS D, ANDREWS A. Defining maintainable components in the design phase[C]// Proceedings of the 21st IEEE International Conference on Software Maintenance. Washington, DC: IEEE Computer Society, 2005: 49 - 58.

[3] ROUNTEV A, KAGAN S, SAWIN J. Coverage criteria for testing of object interactions in sequence diagrams[C]// Fundamental Approaches to Software Engineering, LNCS 3442. Berlin: Springer, 2005: 2 - 10.

[4] BINDER R V. 面向对象系统的测试[M]. 华庆一, 王斌君, 陈莉, 译. 北京: 人民邮电出版社, 2001.

[5] 赵炜, 曾一, 张利武. 基于 UML 活动图生成功能测试线索[J]. 计算机工程与设计, 2006, 27(22): 4328 - 4330.

[6] 颜炯, 王戟, 陈火旺. 基于模型的软件测试综述[J]. 计算机科学, 2004, 31 (2) : 184 - 187.

[7] 黄陇, 于洪敏, 陈致明, 等. 基于UML的软件测试自动化研究[J]. 计算机应用, 2004, 24(7): 135 - 137.

(上接第 689 页)

[8] KICZALES G, MEZINI M. Separation of concerns with procedures, annotations, advice and pointcuts [C]// ECOOP' 05: Proceedings of the 19th European Conference on Object-Oriented Programming, LNCS 3586. Berlin: Springer, 2005: 195 - 213.

[9] HAVINGA W , NAGY I , BERGMANS L . Introduction and derivation of annotations in AOP: applying expressive pointcut languages to introductions [C/OL]// European Interactive Workshop on Aspects in Software (EIWAS). New York: ACM, 2005 [2008 - 09 - 01]. http://prog. vub. ac. be. events/eiwas2005/Papers/EINAS2005-Wilke%20Havinga. pdf.

[10] GYBELS K , BRICHAU J . Arranging language features for more robust pattern-based crosscuts[C]// AOSD' 03: Proceedings of the 2nd International Conference of Aspect-oriented Software Development. New York: ACM, 2003: 60 - 69.

[11] OSTERMANN K, MEZINI C, BOCKISCH M. Expressive pointcuts for increased modularity [C]// ECOOP' 05: Proceedings of the 19th European Conference on Object-Oriented Programming, LNCS 3586. Berlin: Springer, 2005: 214 - 240.

[12] BRAEM M, CYBELS K, KELLENS A, *et al.* Inducing evolution-robust pointcuts [C/OL]// ERCIM Evolution Workshop, Lille, France, 2006 [2008 - 09 - 01]. http://ssel. vub. ac. be/members/mbream/publications/ericm06. pdf.

[13] TOURWE T , KELLENS A , VANDERPERREN W , *et al* . Inductively generated poincuts to support refactoring to aspects [DB/OL]. [2008 - 09 - 01]. http://prog. vub. ac. be/Publications/2004/vub-prog-tr-04-09. pdf.

[14] GRISWOLD W , SHONLE M , SULLIVAN K , *et al* . Modular software design with crosscutting interfaces [J]. IEEE Software, 2006, 23(1): 51 - 60.

[15] SULLIVAN K , GRISWOLD W , SONG Y , *et al* . Information hiding interfaces for aspect-oriented design[C]// Proceedings of the 5th Symposium on the Foundations of Software Engineering Joint with the European Software Engineering Conference (ESEC/FSE). New York: ACM, 2005: 166 - 175.