基于元数据和反射的面向方面软件演化方法

何成万[1,2],张立军[1],张 慧[1]
(1.武汉工程大学计算机科学与工程学院,湖北武汉 430073;2.智能机器人湖北省重点实验室,湖北武汉 430073)

摘 要: 面向方面软件中的基础程序的结构信息发生变化后,会导致意外的连接点丢失问题.其原因在于连接点的定义紧紧地依赖于基础程序的结构,这种紧密的耦合严重阻碍了面向方面软件的演化.提出一种基于元数据和反射的面向方面软件演化方法.在连接点定义和基础程序之间加入概念层,以实现连接点定义和基础程序结构的解耦.概念层模型用于描述基础程序的逻辑信息、物理信息,以及这两类信息之间的映射关系.基于Java Annotation元数据机制在基础程序中使用逻辑信息对程序结构进行标注,同时,使用逻辑信息定义连接点.根据基础程序中定义的元数据对基础程序进行转换,自动生成元对象,基于反射机制实现基础程序演化后连接点的正确匹配.详细阐述了函数、构造函数的转换算法.该方法较好地解决了面向方面软件中由于基础程序结构信息的变化而导致的连接点丢失问题,有助于构建可适应的面向方面软件.

关键词: 面向方面软件; 意外的连接点丢失; 软件演化; 元数据; 反射
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2011) 08-1771-07

An Approach to Aspect-Oriented Software Evolution Based on Metadata and Reflection

HE Cheng-wan[1,2], ZHANG Li-jun[1], ZHANG Hui[1]
(1. *School of Computer Science and Technology, Wuhan Institute of Technology, Wuhan, Hubei* 430073, *China*;
2. *Hubei Provincial Key Laboratory of Intelligent Robot, Wuhan, Hubei* 430073, *China*)

Abstract: The change of base program structure in aspect-oriented software will lead to the accidental join point miss. The reason is that the definition of the join point is tightly dependent on the base program's structure. Such a close coupling hinders the evolution of the aspect-oriented software heavily. This paper proposes an approach to aspect-oriented software evolution based on metadata and reflection. A conceptual level is added between the join point definition and the base program's structure in order to achieve decoupling. The conceptual model is used to describe the logical information, physical information, as well as the relationship between the two kinds of information. The structure of base program is annotated by logical information based on Java annotation metadata. Simultaneously, join points are also defined by the logical information. The base program is transformed based on the defined metadata and meta objects are generated automatically. On the basis of reflection, the join points can be matched correctly after the evolution of base program. Transformation algorithms for method and constructor are described in detail. This approach solves the accidental join point miss problem effectively, and can be helpful to construct adaptable aspect-oriented software.

Key words: aspect-oriented software; accidental join point miss; software evolution; metadata; reflection

1 引言

随着用户需求的变化以及软件运行环境的变化,软件系统需要不断的演化(evolution),以适应这种新的需求和变化.如何实现软件演化是软件工程领域的一个研究热点[1,2].面向方面的程序设计(AOP, Aspect-Oriented Programming)[3]方法把软件系统的功能和非功能需求、平台特性等诸多不同的关注点相互独立,实现了更好的模块化.同时,它把横切关注点织入(weaving)到基础程序(base program,实现系统的功能、不包含aspect的程序)的实现机制也为软件演化提供了一种新的途径[4,5].

在AOP应用程序中,一个Aspect包括两部分:切入点(pointcut)和通知(advice).一个切入点是一组连接点

收稿日期:2010-08-09;修回日期:2011-03-22
基金项目:国家自然科学基金(No.60873024);湖北省自然科学基金(No.2009CDB293);武汉大学软件工程国家重点实验室开放基金(No.SKLSE2010-08-25)

(join point)的集合,而一个连接点是程序流中的一个特定的执行点;通知(advice)是在连接点之前(before)、之后(after)、或前后(around)被执行的代码.AspectJ[6]是目前被广泛使用的 AOP 语言.

现有的 AOP 实现还不能很好地支持软件演化,基于 AOP 的软件演化理论与方法还存在众多的挑战性课题.其中的一个典型问题称为意外的连接点丢失(accidental join point miss)问题[7,8].主要表现为现在的联结点模型(JPM,Join Point Model)不能适应基础程序的演化.原因在于切入点(pointcut)的定义紧紧地依赖于基础程序的结构,这种紧密的耦合严重阻碍了软件的演化.如果基础程序的结构发生了变化,就有可能导致这个问题的发生.

重构[9]是指在不影响程序行为的前提下对软件结构进行的重组.它是提高软件质量、促成软件演化的重要方法之一.重构会导致基础程序结构的变化,比如类名的变更、函数名的变更、变量名的变更等.在 AOP 系统中,这些变化都有可能导致连接点的丢失.

为了解决上述的连接点丢失问题,虽然采用手工的方法修改切入点最为简单,但是很容易出现错误.因为基础程序结构的一个变化,有可能涉及到多个切入点的定义,如果不能把受影响的相关切入点定义都作修改,就会引起程序执行错误.支持 AOP 的可视化编程环境也不能检测出所有丢失的连结点,因为在连接点定义中可使用通配符,增加了自动检测的难度.本文的目的是在现有的 AOP 语言的基础上,提出并实现一种有效解决连接点丢失问题,支持面向方面软件演化的方法.当基础程序结构发生变化时,在不修改切入点定义的前提下,保证连接点不会产生丢失.

为解决连接点丢失问题,我们做了一些前期研究工作,相关成果发表在文献[10,11]中.这些研究成果在一定程度上能解决连接点丢失问题,但还有需要完善的地方,如:不支持 AOP 中的引入机制、没有实现变量的反射[12]、反射机制的实现比较复杂等.我们在国家自然科学基金"基于元数据和契约式设计的 Aspect 安全组合机制及其支撑工具"等项目的支持下,在前期研究的基础上,对面向方面软件中的连接点丢失及其相关问题进行了更深入的研究,进一步改进和完善了我们的方法.

本文首先通过一个例子介绍意外的连接点丢失问题和相关研究,然后阐述我们提出的基于元数据[13]和反射的面向方面软件演化方法、以及程序的转换算法,最后从通用性和有效性等方面对我们的方法进行讨论.

2 问题描述及相关研究

连接点丢失问题首先由 C.Koppen 等人在文献[7]中提出,A.Kellens 等人在文献[8]中对这一问题作了进一步的阐述.这个问题具体表现为面向方面软件中的基础程序的结构信息(如类名、函数名、变量名等)发生变化后,原来能够匹配的连接点变得不能匹配.

下面用一个简单的例子来说明这个问题.程序 1 是 HelloWorld.java(基础程序)和 MyAspect.java(Aspect)的定义,该程序表示在执行 HelloWorld 中的 greet()函数的前后分别输出字符串"before greeting..."和"after greeting..."

程序1 HelloWorld 类和 MyAspect 的定义

```
public class HelloWorld {
    public void greet(String str) {
        System.out.println(str);
    }
}


aspect MyAspect {
    pointcut atgreet():
        execution(void HelloWorld.greet(*));
    before():atgreet(){
        System.out.println("before greeting…");  }
    after():atgreet(){
        System.out.println("after greeting…");  }
}
```

以上程序开发完成后,假设需要对 HelloWorld 类作如下修改:原函数 greet()的函数名变更为 new_greet(),但其功能不变,即在其执行前后分别输出字符串"before greeting…"和"after greeting…".修改后的 HelloWorld 类如程序 2 所示.

程序2 修改后的 HelloWorld 类

```
public class HelloWorld {
    public void new_greet(String str) {
        System.out.println(str);}
    }
```

HelloWorld 类的结构信息修改后,如果不修改切入点的定义,则程序执行的结果和预期的结果会有差别:new_greet()函数被执行前后,并没有分别输出字符串"before greeting…"和"after greeting…".其原因是由于原来的 greet()函数更名为 new_greet()函数,导致和切入点中定义的连接点匹配不上了.这种现象称为意外的连接点丢失.

与切入点定义有关的研究大多集中在如何提高切入点定义的表现力上.其目的是提供一种基于高层次信息的切入点定义方法,使程序员能够更直接地定义切入点,这样的切入点描述称为语义切入点(semantic pointcuts).现在有很多这样的提案,如 AspectJ 中的 if 和 cflow[6]、描述数据依赖关系的 dflow[14]等.但是这些方法

的侧重点在于提高切入点的描述能力,对于基础程序的演化而导致的对切入点定义的影响考虑得较少.

文献[7]介绍了一种处理连接点丢失问题的方法:使用可视化的工具,自动检测并显示 AOP 应用程序演化前后的两个版本之间的切入点语义的区别.该方法使程序员能够快速准确地发现程序的错误,进而对程序进行修改,但它只适用于切入点语义变化的检测,没有从实质上解决意外的连接点丢失问题.

文献[15]指出,基础程序和 Aspects 之间应该是一种松散的耦合.它提出一种基于 UML(Unified Modeling Language)描述的设计信息的联结点模型,提高了 Aspect 的可重用性.虽然该方法对程序员而言显得更直观和易于理解,但存在的问题是如何把高层次的设计信息转换成低层次的代码,以及如何在这两者之间保持同步.

文献[8]提出了一种基于模型的切入点定义方法.它把脆弱的切入点问题转换到更容易解决的概念层,采用的方法是在切入点和基础程序之间加入一层概念层,切入点的定义不是使用基础程序的构造元素,而是使用概念层的模型元素,从而实现切入点和基础程序的解耦.

文献[16]介绍了一种解决连接点匹配问题的自动化方法.它通过分析对应于一个切入点的基础程序元素间的共同特征,抽取出相应的表达模式.然后把这些模式应用于下一个版本的程序,自动地建议程序员对切入点定义进行维护,增加需要包含的新连接点.

3 基于元数据和反射的面向方面软件演化方法

3.1 方法概述

面向方面软件演化的类型之一是基础程序结构信息的变化,如类名、函数名和变量名的变更,本文关注的是这种类型的软件演化,因为导致连接点丢失问题的是这种结构信息的变化.另外,面向方面软件的切入点定义包含了需要匹配的连接点,以 AspectJ 为例,切入点的定义包括函数的调用、构造函数的调用、变量的引用和赋值、异常处理、初始化等连接点,以及这些连接点的组合.基础程序结构信息的变化只涉及到构造函数名、函数名、变量名的变化,对异常处理、初始化等的切入点定义没有影响.而类名的变更可以归结为构造函数名的变更.另外,对于变量名的变更,我们采取和方法名变更一样的策略,通过调用该变量的 set()函数和 get()函数,实现变量访问时对通知(advice)的调用.

基于以上分析,为了突出重点,我们选取了 AspectJ 切入点定义的一个子集.虽然本文以 AspectJ 为例来说明我们的方法,但我们的方法同样也可应用到其它的基于 Java 的 AOP 语言中.

切入点::= call(函数名)|execution(函数名)
|call(构造函数名)|execution(构造函数名)
|handler(异常类名称)
|within(类型名)
|(切点 && 切点)|(切点||切点)

在分析第 2 节阐述的问题和相关研究成果的基础上,我们提出一种基于元数据和反射的面向方面软件演化方法(如图 1 所示).其基本思想是:在切入点和基础程序之间加入一层概念层,以实现切入点定义和基础程序结构的解耦.概念模型用于描述逻辑信息(概念、逻辑功能等)、物理信息(实现某各功能的类、函数等)、以及这两类信息之间的映射关系等.我们采用 Java Annotation 元数据标注机制,在基础程序中使用逻辑信息对函数、变量进行标注.同时,连接点的定义也使用逻辑信息描述.使用逻辑信息标注后的函数和变量分别称为反射函数和反射变量.

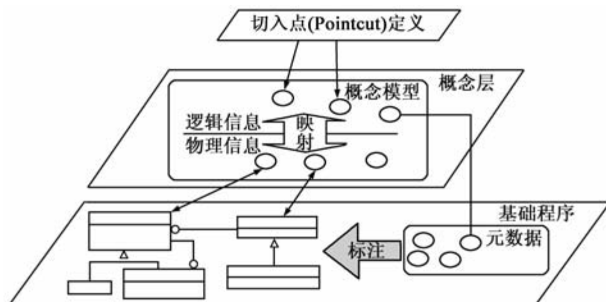

图1 基于元数据和反射的面向方面软件演化方法

为了实现基础程序演化后连接点的正确匹配,我们对标注后的源程序进行转换之后,再使用 AOP 语言提供的工具进行编译和执行.图 2 是采用我们的方法开发面向方面软件时的流程.

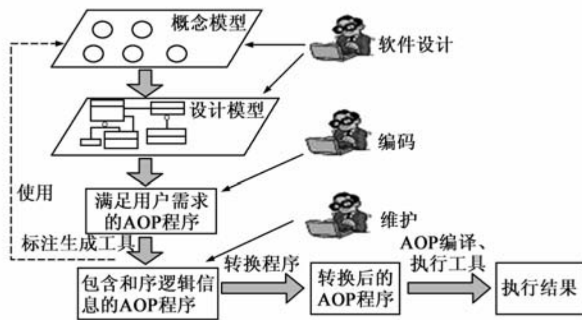

图2 开发流程

3.2 概念模型的描述及基础程序的标注

概念模型描述了基础程序的物理信息与逻辑信息的对应关系.由于 XML(Extensible Markup Language)很适合描述结构化信息,我们采用 XML 描述概念模型.XML 的 DTD(Document Type Definition)定义如下所示:

```
<! ELEMENT class ( method ) >
<! ATTLIST class name NMTOKEN JHJREQUIRED >
<! ELEMENT classes ( class ) >
<! ELEMENT method ( physicalInfo, logicalInfo ) >
<! ELEMENT physicalInfo ( modifier, returntype, name, paras ) >
<! ELEMENT logicalInfo ( modifier, returntype, name, paras ) >
<! ELEMENT modifier ( # PCDATA ) >
<! ELEMENT name ( # PCDATA ) >
<! ELEMENT para EMPTY >
<! ATTLIST para paratype NMTOKEN # REQUIRED >
<! ELEMENT paras ( para + ) >
<! ELEMENT returntype ( # PCDATA ) >
<! ELEMENT variable ( modifier, type, vname ) >
<! ELEMENT type ( # PCDATA ) >
<! ELEMENT vname ( # PCDATA ) >
```

程序 3 是一个概念模型的例子.它描述了程序 2 所示的类 HelloWorld 中的函数 new_greet()的物理信息,以及对应的逻辑信息.

程序3 一个使用 XML 描述的概念模型

```
<classes>
  <class name="HelloWorld">
    <method>
      <physicalInfo>
          <modifier>public</modifier>
          <returntype>void</returntype>
          <name>new_greet</name>
          <paras> <para paratype="String"/> </paras>
      </physicalInfo>
      <logicalInfo>
          <modifier>public</modifier>
          <returntype>void</returntype>
          <name>greet</name>
          <paras> <para paratype="String"/> </paras>
      </logicalInfo>
  </method> </class>
</classes>
```

为了实现基础程序的转换,以及基础程序的物理结构信息与概念模型所表示的信息之间的自动映射,我们使用 Java Annotation 元数据标注方法对基础程序进行了标注.表 1 是我们定义的用于描述逻辑信息的标注.

基于程序 3 的概念模型和表 1 的标注定义,我们可以对程序 2 的类 HelloWorld 中的函数 new_greet()添加标注.如程序 4 所示.

表 1 描述逻辑信息的 Java Annotation

标注的定义	描述	适用
@interface LogicalConsInfo { String modifier(); String name(); String[] paramtype();}	表示构造函数所对应的逻辑信息.标注的属性分别表示反射构造函数的逻辑修饰符、逻辑函数名、参数类型.	构造函数
@interface LogicalMethodInfo { String modifier(); String returntype(); String name(); String[] paramtype();}	表示函数所对应的逻辑信息.标注的属性分别表示反射函数的逻辑修饰符、逻辑返回值类型、逻辑函数名、参数类型.	普通函数
@interface GetMethodInfo{ String modifier(); String returntype(); String name();}	表示变量的 get 函数信息.标注的属性分别表示 get 函数的修饰符、返回值类型、函数名.	变量
@interface SetMethodInfo{ String modifier(); String name(); String[] paramtype();}	表示变量的 set 函数信息.标注的属性分别表示 set 函数的修饰符、函数名、参数类型.	变量

程序4 一个函数标注的例子

```
public class HelloWorld {
    @LogicalMethodInfo(
        modifier = "public",
        returntype = "void",
        name = "greet",
        paramtype = {"String"})
    public void new_greet(String str) {
        System.out.println(str);}
}
```

3.3 程序转换方法

由于函数、构造函数和变量的调用方式各不相同.下面分别对它们的转换方法进行描述.

为了更精确地描述程序的转换算法.我们给出如下定义:

定义 1 设有类 C,定义 sub(C)为其子类.

定义 2 设有类 X 或者函数 X,定义 name(X)为 X 的名称.

定义 3 定义类 C 的描述为 C is_c K FST KST MST.其中 K 表示类 C 的父类型.FST 表示类 C 的属性集.KST 表示类 C 的构造函数集.MST 表示类 C 的函数集.

定义 4 定义 annotation(X)为类 X 或函数 X 的标注集.

定义 5 定义 $f(X)$为类 X 或函数 X 的转换函数.

3.3.1 函数的转换

反射函数使用@LogicalMethodInfo 标注.根据程序中标注的逻辑信息,通过程序转换,在程序中自动添加

和逻辑信息一致的函数(称为逻辑函数),以保证连接点的正确匹配.另外,重新定义原有的反射函数,使用反射机制,实现对逻辑函数的自动调用.由于增加了逻辑函数的定义,原有的切入点定义能够被正确匹配.如图 3 所示.

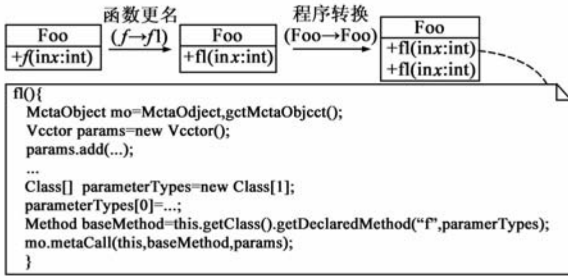

图3　函数的转换方法

元对象 MetaObject 包含以下形式的函数 metaCall(),完成对基础程序中逻辑函数的调用.元对象和基本对象的对应关系为 1∶n 的关系,即一个元对象处理所有反射函数的调用.

```
public Vector metaCall ( Object baseObject, Method baseMethod, Vector params ) {
    …
    baseMethod.invoke(baseObject, oparams); }
```

我们采用编译时反射(compile-time reflection)[12]完成反射机制中截取的功能.重新定义的反射函数包含以下处理:(1)封装执行逻辑函数调用时的实际参数和参数类型,实现反射机制中的具体化(reification)功能;(2)取得元对象 MetaObject;(3)调用元对象中的函数 metaCall(),完成对逻辑函数的调用;(4)从调用 metaCall()函数的返回值中取出返回值,实现反射机制中的反射功能.

函数转换的算法如算法 1 所示.

算法1　函数的转换算法

1. 从类 $C(C\ is_c\ K$ FST KST MST)中取得所有满足条件的函数 M = $\{M \mid M \in$ MST 且 annotation(M) = @LogicalMethodInfo$\}$.
2. For each m in M

　2.1. 把反射函数名 name(M)更名为逻辑函数名 @LogicalMethodInfo.name().

　2.2. 在原有类中重新定义反射函数.

　$f(C) = C'$

　$C'\ is_{c'}\ K$ FST KST MST′

　MST′ = $\{M \mid M \in$ MST$\}$ ∪

　$\{m \mid$ name(m) = @LogicalMethodInfo.name()$\}$

　2.3. 在重新定义的反射函数中,生成一下处理.

　　取得元对象 MetaObject;

　　封装执行参数;

　　调用元对象中的函数 metaCall();

　　返回元对象调用的返回值;

　EndFor

3.3.2　构造函数的转换

当变更基础程序中的某个类名时,构造函数名会发生改变.构造函数使用 @LogicalConsInfo 标注,其转换方法如图 4 所示.新建一个以逻辑函数名命名的子类,在子类的构造函数中调用父类的构造函数.同时,在基础程序中使用类 Foo 生成类 NewFoo 的实例.程序转换后,原有切入点定义不需修改,就能正确匹配构造函数调用的连接点.

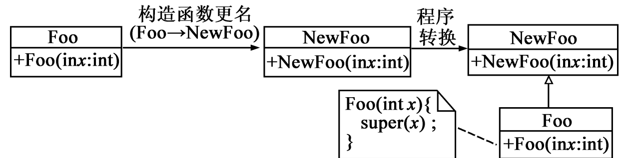

图4　构造函数的转换方法

构造函数转换的算法如算法 2 所示.

算法2　构造函数的转换算法

1. 类 $C(C\ is_c\ K$ FST KST MST)中检查是否含有被 @LogicalConsInfo 标注的构造函数.
2. 如有,则从类 C 中取得所有满足条件的构造函数 M = $\{M \mid M \in$ KST 且 annotation(M) = @LogicalConsInfo$\}$.
3. 定义类 C 的子类 sub(C),且 name(sub(C)) = @LogicalConsInfo.name().
4. For each m in M

　根据构造函数 M 所对应的逻辑信息,在 sub(C)中定义构造函数,并调用父类的构造函数.

EndFor

3.3.3　变量名变更时的程序转换

反射变量使用 @GetMethodInfo 和 @GetMethodInfo 标注.变量名变更时,切入点中定义的 set 和 get 连接点会产生丢失.我们采用和函数调用类似的策略,通过变量的 get 和 set 函数访问变量,实现对通知(advice)的调用.反射变量使用 Java 标注指定了 get 和 set 函数的逻辑信息,在进行程序转换时,如果类中没有对应的 set 和 get 函数,则自动生成相关 set 和 get 函数.转换方法如图 5 所示.

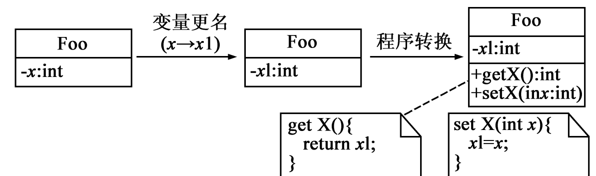

图5　变量名变更时的转换方法

变量名变更时的程序转换算法如算法 3 所示.

算法3　变量名变更时的转换算法

1. 类 $C(C\ is_c\ K\ FST\ KST\ MST)$中检查是否含有被 @GetMethodInfo 和 @SetMethodInfo 标注的变量.
2. 如有,则从类 C 中取得所有满足条件的变量 V = $\{V \mid V \in FST$ 且 annotation(V) = @GetMethodInfo 和 @SetMethodInfo$\}$.
3. for each v in V

　3.1. 检查类 C 中是否有和@ GetMethodInfo 的属性值一致的函数,如没有则追加.

　3.2. 检查类 C 中是否有和@ SetMethodInfo 的属性值一致的函数,如没有则追加.

EndFor

3.4 一个函数转换的例子

以程序 4 所示的添加标注后的 Java 程序为例.经转换程序转换后的程序如程序 5 所示.在类中增加了一个逻辑函数,同时重新定义了 new_greet()函数.new_greet()函数完成元对象的获取、执行参数的封装、元对象调用等处理.

程序5 转换后的 HelloWorld 类

```
public class HelloWorld {
    public void new_greet (String s){
    try{
        MetaObject mo = MetaObject.getMetaObject();
        Vector params = new Vector();
        params.add(s);
        Class[] parameterTypes = new Class[1];
        parameterTypes[0] = String.class;
        Method baseMethod = this.getClass().
          getDeclaredMethod("greet",parameterTypes);
        mo.metaCall(this,baseMethod,params);
        }catch(Exception e){e.printStackTrace();}   }
    public void greet(String str) {System.out.println(str);}
}
```

(1) 在同一个类中,反射函数名、逻辑函数名、以及变量的 set 和 get 函数名都必须具有唯一性,以保证程序转换的正确性;

(2) 基础程序中反射变量的访问需要通过变量的 set 和 get 函数完成,以保证变量名发生改变后对通知(advice)的正确调用.

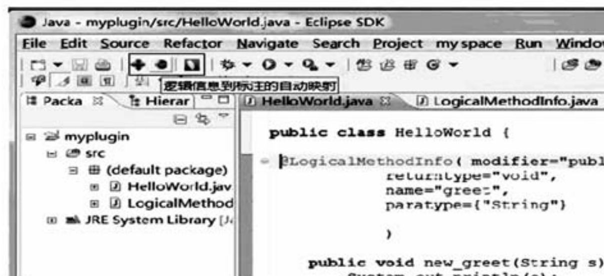

图6 支撑工具界面

4.4 和其它方法的比较

和文献[8]的方法类似,我们的方法也是在基础程序和连接点定义之间增加概念层,以实现两者的解耦.但是两种方法存在一些本质的区别:

(1) 我们的方法基于概念模型中定义的逻辑信息、物理信息及其两者之间的映射关系对基础程序进行转换,在不修改切入点定义的前提下,实现连接点的自动匹配;而文献[8]的方法没有对程序进行转换,概念层仅仅用于确认连接点不匹配的时机和原因,需要手动修改程序;

(2) 在实现方法上,我们的方法基于反射体系结构实现了元对象对基本对象的控制,程序具有较好的适应性和扩展性;文献[8]的方法是使用逻辑语言描述连接点定义,推理和发现不一致的连接点定义.

(3) 我们的方法采用的是通用的语言和技术,保证了方法的通用性;文献[8]的方法对于概念层的描述语言和 AOP 语言均有严格要求,有一定的局限性.

文献[16]介绍了一种在 AOP 程序演化后自动提示程序员对切入点定义进行维护的方法,程序员根据提示,对有可能需要维护的切入点定义进行修改.我们的方法不需要修改切入点定义,而且基础程序的转换可以使用工具自动完成.

5 结论

本文介绍了一种基于元数据和反射的面向方面软件演化方法,通过在切入点和基础程序之间加入概念层,实现连接点定义和基础程序结构的解耦.根据基础程序中标注的元数据,利用反射机制,实现了连接点的自动匹配.该方法有效地解决了面向方面软件中由于基础程序的演化而导致的意外的连接点丢失问题,有助于面向方面软件的演化.

4 讨论

本节从方法的通用性、有效性、使用的限制条件,以及和其它方法的比较等方面进行讨论.

4.1 方法的通用性

本文提出的基于元数据和反射的面向方面软件演化方法在实现时采用的是标准的平台和语言,如概念模型的描述、基础程序中结构信息的标注等都是采用被广泛使用的语言.另一方面,反射的实现采用的是编译时反射,即通过源程序的转换实现反射功能,对编程语言没有特殊要求.以上这些策略保证了方法的通用性.

4.2 方法的有效性

本文给出的切入点定义的集合涵盖了现有 AOP 语言所提供的切入点定义的核心部分,能够满足开发过程的需要.同时,由于追加的逻辑函数和重新定义的反射函数都是对原来的类或者子类进行修改,保证了对 AOP 语言中引入(introducetion)机制的支持.

为了验证方法的有效性,我们在 Eclipse 平台上开发了相关功能的插件,在工具栏中增加了三个插件按钮,分别完成程序的转换、概念模型到基础程序的逻辑信息映射、基础程序到概念模型的物理结构信息映射等功能,如图 6 所示.结果表明我们的方法是有效可行的.

4.3 使用的限制条件

使用的限制条件包括以下两个方面:

参考文献

[1] Tom Mens,Jeff Magee,Bernhard Rumpe. Evolving software architecture descriptions of critical systems[J]. IEEE Computer,2010,43(5):42 – 48.

[2] 王映辉,王立福. 软件体系结构演化模型[J]. 电子学报,2005,33(8):1381 – 1386.
Wang Ying,Wang Lifu. Research about model and ripple effect analysis of software architecture evolution[J]. Acta Electronica Sinica,2005,33(8):1381 – 1386.(in Chinese)

[3] Gregor Kiczales,John Lamping,Anurag Mendhekar,et al. Aspect oriented programming[A]. Proceedings of ECOOP'97[C]. UK:Springer Verlag,1997.220 – 242.

[4] Walter Cazzola,Sonia Pini,Massimo Ancona. AOP for software evolution:A design oriented approach[A]. Proceedings of the 2005 ACM symposium on Applied computing[C]. New York:ACM Press,2005.1346 – 1350.

[5] Cazzola W,Chiba S,Saake G. Software evolution:A trip through reflective,aspect,meta-data oriented techniques[A]. Proceedings of the ECOOP 2004 Workshop[C]. UK:Springer-Verlag,2005.118 – 132.

[6] AspectJ team. The AspectJ programming guide [EB/OL]. http://eclipse.org/aspectj/,2010-08-09.

[7] C. Koppen,M. Stoerzer. Pcdiff:Attacking the fragile pointcut problem[A]. Proceedings of EIWAS 2004[C]. UK:Springer-Verlag,2004.1 – 8.

[8] Andy Kellens,Kim Mens,Johan Brichau,et al. Managing the evolution of aspect-oriented software with model-based pointcuts[A]. Proceedings of the 20th European Conference on Object-Oriented Programming (ECOOP'06)[C]. UK:Springer Verlag,2006.501 – 525.

[9] Tom Mens,Tom Tourwe. A survey of software refactoring[J]. IEEE Transactions on Software Engineering,2004,30(2):126 – 139.

[10] Chengwan He,Zheng Li,Keqing He. Towards trusted aspect composition[A]. IEEE 8th International Conference on Computer and Information Technology Workshops[C]. USA:IEEE Computer Society,2008.643 – 648.

[11] Chengwan He,Zheng Li,Keqing He. Using conceptual model and reflection mechanism to resolve the structural conflict in AOP application[A]. 2008 International Conference on Computer Science and Software Engineering[C]. USA:IEEE Computer Society,2008.77 – 80.

[12] Chiba S. A metaobject protocol for C++[A]. ACM Conference on Object-Oriented Programming,Systems,Languages and Applications[C]. New York:ACM,1995.285 – 299.

[13] National Information Standards Organization. Understanding Metadata[EB/OL]. http://www.niso.org/publications/press/UnderstandingMetadata.pdf.

[14] H Masuhara,K Kawauchi. Data flow pointcut in aspect-oriented programming[A]. Asian Symposium on Programming Languages and Systems[C]. UK:Springer-Verlag,2003.105 – 121.

[15] Walter Cazzola,JeanMarc J'ez'equel,Awais Rashid. Semantic join point models:Motivations,notions and requirements[A]. Proceedings of the Software Engineering Properties of Languages and Aspect Technologies Workshop (SPLAT'06)[C]. New York:ACM,2006.

[16] Raffi Khatchadourian,Phil Greenwood,Awais Rashid,Guoqing Xu. Pointcut rejuvenation:Recovering pointcut expressions in evolving aspect-oriented software[A]. IEEE/ACM International Conference on Automated Software Engineering (ASE 09)[C]. USA:IEEE Computer Society,2009.575 – 579.

作者简介

何成万 男,1967 年生于湖北荆门. 博士,教授. 研究方向为软件工程及软件开发环境、软件复用及软件构件技术.
E-mail:hechengwan@hotmail.com

张立军 男,1987 年生于湖北天门. 硕士研究生,研究方向为软件工程及软件开发环境.

张 慧 女,1986 年生于湖北天门. 硕士研究生,研究方向为软件工程及软件开发环境.