构造面向对象系统依赖图技术研究

Study on Constructing Object – Oriented Program System Dependency Graph

杜 林 江海燕 （山东教育学院 计算机科学与技术系 山东济南 250013）

摘 要： 本文分析了现有面向对象图形表示方案存在的问题，在此基础之上，提出了基于波动效应分析构造面向对象系统依赖图的方法，通过引入波动效应分析，来完善面向对象程序的语义和减小构图的复杂性，在波动效应分析的基础之上，构造类图、改造面向过程的系统依赖图，结合两个图来描述面向对象程序。通过类图描述不同类之间的关联关系和类的内部定义，在类图中表达过程依赖，改造面向过程的系统依赖图用于表达控制依赖和数据依赖。文中给出了计算波动效应和构造系统依赖图的算法描述。

关键词： 波动效应 类图 系统依赖图 过程依赖 数据依赖 控制依赖

1 现有面向对象图形表示方案分析

1.1 过程型程序表示方案

对于过程型程序，通过系统依赖图可以很好地表示它的控制依赖、数据依赖、参数传递等特性，并且可以进行过程间分析，系统依赖图是在控制流图、控制依赖子图、数据依赖子图、过程依赖子图和程序依赖子图的基础上建立起来的一种语法分析树表示。

1.2 面向对象程序表示方案

面向对象程序除了包含一些过程或方法外，还存在继承、多态以及动态联编等多种关系或者特性，所以适合过程

化程序的系统依赖图不能全面描述这些面向对象的概念。对此一般采用两种策略，下面对这两种策略进行分析，并指出它们存在的问题：

(1) 针对不同的目的，从不同角度出发，仅仅描述面向对象程序的部分特性。比如通过类层次图来表示类之间的继承关系；可见方法类层次子图用来静态反映 CHS 中各个节点的可见方法；虚函数调用图可用来解决 C + +/Java 语言中复杂的虚函数调用问题；动态面向对象程序依赖图只用来表示面向对象程序的动态行为；面向对象的程序依赖图可用来表示面向对象语言中的多态性和动态联编问题，但是不能表示面向对象的过程间分析；程序依赖网只表示并发面向对象程序。

(2) 综合考虑面向对象的全部特性，扩充基于过程化程序的系统依赖图的语法语义[1]，从而全面表示面向对象程序，虽然概念清晰，但是构图复杂。扩充后的系统依赖图是过程依赖子图、类依赖子图、类层次子图、控制依赖子图和数据依赖子图这几种表示方法的并集。利用该图，能够反映面向对象程序特性，可以处理过程间的数据流和控制流，能够表示参数传递，能够进行过程间分析。但是构造 OOSDG 非常复杂，而且容易出错。所以在实际构造中，针对研究的不同侧重点，忽略了面向对象程序的有些特性，这样就造成了不精确。举例如下：下面两种情况都造成了不精确。

其一，在 OOSDG 中，对应于类的每个成员变量只构造一个参数节点：因为类的成员变量在类的每个对象中都有一份独立拷贝，所以造成在一个类的不同对象之间产生了错误的数据依赖，也就是说，类的成员变量在一处定义，而在多处使用。

其二，为每一个消息处理方法构造一个调用点：当程序中存在多态时，这就相当于用一些类型不同但名字相同的对象来表示多态对象，但是构造 OOSDG 的算法不能区分这种类型不同但名字相同的情况，这就产生了错误。

2 本文构造面向对象系统依赖图的方案

在分析以上两种策略的基础之上，基于以下原则

构造适合表示面向对象程序的系统依赖图。

(1)完善面向对象程序的语义,更好地表达面向对象语言的特性:引入波动效应分析方法。

(2)在波动效应分析的基础之上构造面向对象系统依赖图,结合 C + +语言的特点,把系统依赖图的描述范围缩小到 main( )中的语句、谓词以及经过波动效应分析所得到的类、实例、成员方法和成员变量等单元。

(3)构造类图、改造面向过程的系统依赖图,结合两个图来描述面向对象程序。

下面首先计算面向对象程序波动效应,然后在此基础之上构造系统依赖图。

3 计算面向对象程序波动效应

程序中的语句之间是相互联系的,一条语句的执行会对其它语句造成直接或是间接的影响,这就是程序中的波动效应。在面向对象程序中,我们把类、实例、成员方法、成员变量等具有独立语义的单元作为研究对象,一个单元的变动会对其它单元造成影响,比如修改了一个类的定义,则这个类的实例和子类都会受到影响,这就是面向对象程序中具有独立语义的单元之间存在的波动效应。

完整波动图构图复杂,与传统系统依赖图的复杂度是同一个级别的,就样就与我们想通过波动效应分析缩小问题范围的初衷不符。

面向对象程序中各个单元之间的关系要么是直接的要么是间接的,对于间接的关系可以通过多个单元间的直接关系来表达,也就是说通过单元间的传递性来表达。在此借鉴聚类中处理传递性的有关方法,通过使用矩阵记录波动效应,通过矩阵操作反映传递性,从而计算完整波动效应。

算法描述:计算指定单元的波动效应所波及到的单元,即确定波动源头后,波动到的单元。

步骤一:确定参与波动效应分析的所有单元及波动效应的发起单元(波动源头)。

步骤二:定义两个矩阵 REA 和 REO:其中 REA 记录了所有单元的直接波动效应;REO 是在零矩阵的基础上,把波动效应的发起单元所对应的矩阵对角线元素置为 1。

步骤三:计算波动效应的传递性:REO = REO * REA,修改矩阵 REO 中的元素值,把非 0 元素值都置为 1。

步骤四:如果矩阵 REO 发生变化,则返回步骤三,否则进行步骤五。

步骤五:当矩阵 REO 不再发生变化时,得出波动效应分析结果,即波动效应的发起单元引起的波动效应所波动到的单元集合,此集合包含的单元为:矩阵 REO 中元素值为 1 所对应的单元。

4 构造基于波动效应分析的系统依赖图

在此对以上波动效应分析所得到的语义单元构造面向对象系统依赖图。

构图方案如下:构造类图、改造面向过程的系统依赖图,结合两个图来描述面向对象程序。通过类图描述不同类之间的关联关系和类的内部定义,因为在下一步的系统依赖图中描绘了过程依赖图,所以为了避免重复,在类图中简化了过程依赖图的处理,使得方法和过程具有相同的地位。改造面向过程的系统依赖图用于表达控制依赖和数据依赖。

4.1 构造类图

类图结合了传统方法中类依赖子图和类层次子图和过程依赖子图的功能[2],描述了不同类之间的继承交互和类的内部定义,以及多态和动态联编等,并简化了图形表示。

4.1.1 描述类、实例、成员方法、成员变量间的关系

类图包含类首部节点和类实例节点、类中的成员方法首部节点、成员变量结点,把实例结点、成员方法首部节点和成员变量节点连接到该方法所属类的首部节点表达成员关系。方法和过程具有相同的地位,对每一个方法和过程只提供入口节点,不进入内部。扩大方法入口节点含义,隐藏参数节点及参数之间数据传递,不同方法的参数节点之间的数据依赖通过方法之间的数据依赖表示。为该方法中所引用的成员变量附加成员变量节点;为该方法所引用的类实例附加实例节点;当类实例调用某个方法时,在每个方法节点附加表示消息接收对象的实例节点,表示该对象的状态变化。

4.1.2 描述类的继承交互

通过把类的首部节点和与其具有继承关系的类的

首部节点连接起来表达继承关系。当一个类和另一个类交互时，通过类首部节点和类成员边可以方便耦合。为了清晰地体现继承层次，减少回朔，如果子类中修改了继承自父类的虚方法或者使用了继承自父类中的方法，则该方法只在子类中描述，并在父类首部节点和子类中的方法首部节点增加关联边。这样做使得继承机制和虚方法的表达不需要在父类方法和子类方法间增加关联边，只需要添加类首部节点和方法首部节点的关联即可。

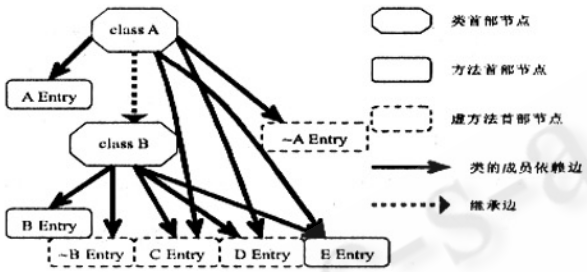

图1　类　图

```
class A{                    class B:public A{
public:                     public:
A();                        B();
virtual ~A();               ~B();
virtual void C();           void C();}
virtual void D();           void D();
void E();                   ......
}                           E();}
```

图1是对上面代码构造类图比如对下列代码构造类图。当一个类实例化另一个类时(例如 new)，存在对另一个类构造函数的潜在调用，为了表示这种潜在的调用关系，利用一条调用边以及参数边将该调用点和另一个中的构造方法节点连接起来。

4.1.3　描述多态、动态联编

通过虚方法首部节点和多态调用边来完整地表示多态性，利用多个调用边把调用节点连接到对每个可能目标调用的方法节点上，以多个具有相同规约的多态性节点表示可能目标中的动态选择，表示了所有可能性。

4.2　改造系统依赖图

通过简化的过程依赖图、控制依赖图和数据依赖图构造系统依赖图。扩大方法入口节点的含义，不同方法的参数节点之间的数据依赖通过方法之间的数据依赖表示，由直接指向调用点的数据依赖边完成，从而在简化构图的基础上解决了调用环境的问题。在过程依赖图中只提供过程入口节点，不进入过程内部[3]。

4.2.1　过程依赖图

对于每一个过程都对应一个过程依赖子图，图中的节点表示语句或者谓词表达式，数据依赖边表示语句以及谓词表达式之间的数据信息流，控制依赖边表示一个语句及谓词表达式执行时依赖的控制条件。每个过程依赖子图包含一个入口节点 entry 表示过程的入口。在类图中，还附加了表示消息接收者的实例节点以及调用边，以表示实例的状态变化。

4.2.2　控制依赖图

每个方法的控制依赖描述包含在这层表示中。因为这是一种静态表示方法，所以某些信息还不能够完全得到解决。例如，不能唯一确定哪些消息的接收者将是动态联编的。这种表示会识别具有多态性的类的集合，而不是形式上定义的接收者，是一种完全的静态表示。但是对于静态程序切片，切片中包含与调用点实际相关的所有虚方法的实现，因此不需解决动态联编的问题，只需考虑虚方法的实现。

4.2.3　数据依赖图

在数据依赖图上，对象被加到表示中。这使得把消息动态联编到对象中的特定方法得到完全解决。其它传统面向过程类型的数据依赖信息也得到表示。

4.3　构造表示面向对象程序的系统依赖图

通过以上构造类图、改造面向过程的系统依赖图，结合两个图来描述面向对象程序。通过类图描述不同类之间的关联关系和类的内部定义，在类图中简化了过程依赖图的构造，避免了和系统依赖图的重复构造，类图中的方法和过程具有相同的地位。改造面向过程的系统依赖图用于表达控制依赖和数据依赖。在此基础上，构造表示面向对象程序的系统依赖图。图中的节点和边的表示如图2所示：

他们的描述的内容如下：

4.3.1　节点的语义表示

节点包括单元节点和语句节点，单元节点包括首

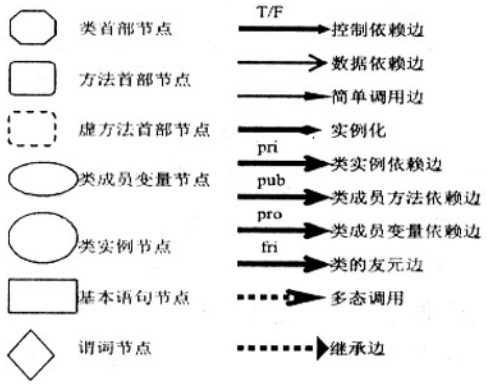

图2 图 例

部节点和非首部节点。首部节点包括:类首部节点、成员方法首部节点、虚方法首部节点,非首部节点包括实例节点和成员变量节点。在增加节点的基础上增加相应的连接边[4]。

4.3.1.1 首部节点:表示某种实体单元(类和方法/过程)的开始节点

(1) 类首部节点:一个类首部表示类中所有方法和数据属性的组合,参与表示类间的继承层次。类首部包含一系列定义在该类中的方法和指向继承自超类方法的指针,且和所有的超类或子类连接。类首部还包含类的数据成员的一些信息。

(2) 方法首部节点:一个方法(包括友元)首部是相应方法的入口节点,方法首部通过封装信息(例如,定义该方法的类、类的成员变量属性等)向表示中溶入另外的含义。方法首部也可构成某个类成员关系边的一个终点。

(3) 虚方法首部节点:有些面向对象语言允许类的所有方法都是动态联编的。C++是通过在方法声明的前面加上关键字 virtual 来规定该方法是动态联编的。可以用类似的方法来区分动态联编或者静态联编的方法。这使得理解多态性和动态联编的表示更加容易。一个虚方法是一个类的特定成员,如果对此方法的一次调用是多态的,它会得到动态的表示,在 OSDG 中描述了调用点可能调用到的所有虚方法实现。所以,用一种增强的表示方法来表示虚方法首部,虚方法首部和多态性选择边在清晰地表示动态联编时起到很重要的作用。

4.3.1.2 非首部节点:包括实例节点和成员变量节点

(1) 实例节点:具体表现类的行为,通过调用边调用类的成员方法和成员变量。

(2) 成员变量节点:为了完善面向对象程序的语义,为成员变量单独构造一个节点。

4.3.1.3 语句节点

(1) 基本语句节点:基本语句节点表示的语句,可以完成部分计算,也包括调用节点、过程返回或退出节点。一个调用节点表示在某个调用位置对方法或过程的一次调用。对依赖图中不同的调用边(也就是简单调用边、实例边和多态性调用边)来说,这是开始点。过程返回或退出节点表示在该条控制流路径中的最后一条被执行的语句。

(2) 谓词节点:谓词节点表示条件语句中的谓词部分,通过谓词节点和控制依赖边来表达控制关系。

4.3.2 边的语义表示

(1) 数据依赖边:数据依赖边用来描述程序中清晰的数据信息流,包括循环结构的信息流。

(2) 控制依赖边:这是一些在表示中显示节点之间控制依赖关系的单向边。这里要注意的是,这些控制依赖边的起点总是谓词或者方法节点。

(3) 调用边:有 3 类调用边:A. 简单调用边:表示一次对方法或对自由标准过程的调用。简单调用边表示能够静态联编的调用,表示调用位置和被调用的方法(过程)之间的控制流和数据流。B. 实例化调用边:在面向对象的程序中,实例化意味着创建了一个类的实例。这是靠调用一个类的构造函数来完成的,类的构造函数初始化对象状态并把它带进系统.。用一条特定的边来表示实例调用,实例边把实例语句类首部节点连接起来。C. 多态调用边:如果在编译时不能解决对一个特定方法的调用,则称这种调用是动态联编的,在表示多态性和动态联编时要使用多态调用边。多态调用边连接了调用位置和特定多个类的具有相同规约的虚方法首部节点,因此也具有多态选择的特性。

(4) 继承边:继承层次是面向对象设计方法的框架,继承边表示一个继承层次中类之间的关系,并按依赖的方向连接子类和它的超类。

(5) 类成员边:每个定义的类中都有固定数目的方法,这样,每个方法(包括该类的构造函数和析构函数)都属于该类,且只能通过该类的实例才可以访问。一条类成员边连接方法首部节点和定义该方法的类首部节点。友元作为类的特殊方法或者属性,在 OSDG 中以特殊的类成员边表示,但是其在切片计算过程中

具有与普通类成员边相同的处理方式。

4.4 算法描述：

把面向对象程序表示为一个二元组(M,C),M 表示主程序 main( ),C 为类的集合。首先根据源文件提取出类层次结构,然后根据类层次从底向上为每个类的方法构造过程依赖图 PrDG。计算某个方法过程依赖图时,首先判断该方法是否需要新的过程依赖图;然后构造类中声明的每个方法;如果一个方法直接或者间接调用了虚方法,则也把该方法当作虚方法对待。在为每个方法构造了过程依赖图后,算法调用 connect( )函数用连接边连接这些过程依赖图的调用点与每个被调用的方法入口节点,同时连接类图和主程序。

判断一个方法是否需要新的过程依赖图是在类图中声明一个新类时,首先识别出该类从所继承基类中继承的可见方法以及未重构的虚方法,然后对这些方法分别做标记。所有未标记过的方法都需要构造新的过程依赖图。算法描述如下:

```
input:    the abstract syntax tree of P = ( M, C)
output:   the OSDG of P
void ConstructOSDG( ) {
   for ( class Ci ( C) {
      for ( method m defined in Ci) {
         if ( m is " marked" )
            make Ci and the " marked" method of
            base class connected as membership;
         else{
            calculate the PrDG of m;
            make m as" marked"
         }
      }//end for1
   }//end for2
   connect( );
}//end ConstructOSDG
```

5 结束语

对于现有面向对象图形表示方案,有的是针对不同的目的,从不同角度出发,仅仅描述面向对象程序的部分特性,比如面向对象程序依赖图可用来表示面向对象语言中的多态性和动态联编问题,但是不能表示面向对象的过程间分析;有的是扩充基于过程化程序的系统依赖图的语法语义,从而全面表示面向对象程序,虽然概念清晰,但是构图复杂。通过引入波动效应分析方法,可以分析面向对象程序中类、实例、成员方法、成员变量间的波动关系,但是完整波动图构图复杂,与传统系统依赖图的复杂度是同一个级别的,就样就与我们想通过波动效应分析缩小问题范围的初衷不符。所以本文通过使用矩阵记录波动效应,通过矩阵操作反映波动的传递性,从而计算完整波动效应。在波动效应分析的基础之上构造面向对象系统依赖图,结合 C++语言的特点,把系统依赖图的描述范围缩小到 main( )中的语句、谓词以及经过波动效应分析所得到的类、实例、成员方法和成员变量等单元,这样既完善了面向对象程序语义,又减少了构图的复杂性。构造类图、改造面向过程的系统依赖图,结合两个图来描述面向对象程序。通过类图描述不同类之间的关联关系和类的内部定义,在类图中表现过程依赖图,方法和过程具有相同的地位。改造面向过程的系统依赖图用于表达控制依赖和数据依赖。下一步工作是基于面向对象系统依赖图计算面向对象程序切片,并研究相关算法和进行复杂度的分析。

参考文献

1 A. van Deursen, B. Elsinga, P. Klint, and R. Tolido. From Legacy to Component: Software Renovation in Three Steps. CAP Gemini White Paper, 2000.

2 Anand Krishnaswamy. Program Slicing: An Application of Object - oriented Program Dependency Graphs. Technical report, Department of Computer Science, Clemson University, 1994.

3 Rupesh Nasre. Slicing of Object Oriented Programs. Seminar Report, Department of Computer Science and Engineering Indian Institute of Technology Numbai, 2001.

4 李必信,王云峰. 基于简化系统依赖图的静态粗粒度切片方法. 软件学报,2002,12(2):204 - 211.