一种用于类测试的新方法

阳展飞　吴　桐　沈晨华　钱乐秋
(复旦大学计算机科学与工程系,上海 200437)

摘　要　面向对象程序的复杂性和类对象间广泛的相互依赖性使得对这样的程序的测试变得非常困难，而类又是面向对象测试中最基本的单元。对类的测试又可以分为3个级别(方法内测试,方法间测试,类内测试),难点主要集中在类内测试上。文章提出一种基于OSD(对象状态图)的新的算法用来产生进行类内测试的方法调用序列。这种算法是对传统的类测试的两种方法基于规格说明的测试和数据流测试的综合,可以使产生测试用例的复杂度大大的降低。

关键词　类的状态变量　原子类状态图　复合类状态图　方法内测试　方法间测试　类内测试　类内引用-调用对

文章编号　1002-8331-(2003)09-0091-04　文献标识码　A　中图分类号　TP31

A New Method Used in Class Testing

Yang Zhanfei　Wu Tong　Shen Chenhua　Qian Leqiu
(Dept. of Computer Science and Engineering, Fu Dan University, Shanghai 200437)

Abstract: The complexity and interdependencies of object-oriented programs makes testing such programs difficult.This paper analyzes the testing of classes of object-oriented programs which fall into three levels: intra-method testing, inter-method testing, intra-class testing.The paper then focus on intra-class testing by presenting a new method that is an integration of dataflow testing and state testing based on OSD to produce method invocation sequences for intra-class testing.

Keywords: state variable, AOSD, COSD, intra-method testing, inter-method testing, intra-class testing, intra-class def-use pair

1　引言

在软件界,对面向对象技术的使用越来越广泛。这种技术最主要的一个优点就是一般的面向对象语言都能够非常方便的重用类,一些可以实例化的、能够信息隐藏的模块。一个类就是一个用来定义一组对象所具有的属性和操作的模板。它包括一些属性,通常称为数据成员或称为实例变量,和一些定义在这些属性上的操作,成员函数或方法组成。文章称这些操作为方法。不可否认,面向对象技术对整个软件开发周期的分析、设计、实现这几个阶段都带来的巨大的好处,但是对于测试阶段而言却并非如此[4]。面向对象技术中的封装、继承、多态等机制给软件测试和软件维护提出了一系列的问题和挑战[6],[8]。有关面向对象测试的许多概念的详细介绍也可以在文献[6]和文献[8]中找到。

对于传统的过程语言,一般将测试分为单元测试、集成测试和系统测试三个级别,但是实践表明这种划分对面向对象语言来说不是非常严格。对于面向对象语言来说,类是一个最基本的单元。类之间通过继承形成层次化的类关系结构,子类可以对父类的属性进行继承,并对父类属性进行重载,覆盖。类把描述对象特征的属性(instance variable)和对这些属性的操作(member function)封装在一起,实现了信息隐藏。正因为类的举足轻重的作用,因此在面向对象的测试中任何一个有意义的测试单元都不能小于一个类的实例对象。但是类测试并不能和过程性语言中单元测试相提并论的,它里面又包括了与传统的单元测试和集成测试相对应的成份。将在第三节进行详细说明。

面向对象的类测试的另外的一个非常重要的特点就是在进行测试时必须考虑到类的对象的当前状态对类的成员方法的执行情况的影响。由于执行前对象的状态的变化,同样的一个成员方法可能执行完全不同的功能,在文献[5]中有对状态测试的详细介绍,并对它的优缺点进行的分析总结。

该文接下来的几节安排如下:第二节介绍类的状态测试的一些基本概念包括原子对象状态图,组合对象状态图;第三节比较简单地介绍类测试的测试级别和类内测试中考虑的def-use对;第四节提出一种新的用来产生方法调用序列的算法;最后是结论部分。

2　对象的状态

在文献[1]中,D.kung等通过一个很有启发意义的例子非正式的介绍了类的状态的概念。这里通过这个例子来说明对象状态的有关概念。把这个示例程序放到了附录中。该程序的成员方法returnQtrs()中有一个小的错误,正确的程序应该添加一条语句:allowVend=0。

状态测试是面向对象软件测试的一个非常重要的方面。它与通常的控制流测试和数据流测试的不同在于它着重于对象的状态依赖行为而不是控制结构和个体数据。传统的功能测试和结构测试能够发现一个方法内部的错误,但是对于在面向对象中常见的由于对象之间或同一个对象内部成员方法的交互所引起的错误,它们往往是无能为力的。例如在附例的程序,要发现前面提到的错误必须测试各个成员方法的交互情况。

作者简介:阳展飞,硕士研究生,主要研究方向为软件测试。吴桐,硕士研究生,主要研究方向为软件测试。沈晨华,复旦大学计算机科学与工程系硕士研究生。钱乐秋,复旦大学计算机科学与工程系教授,博士生导师,主要研究方向为软件复用和软件测试。

大家都知道类的本质是属性,成员方法只是定义在这些属性上的操作。在一个类中,成员方法往往是依赖于某一个或某一些数据成员的,可能因为数据成员的取值不同,同一个方法可能执行完全不同的功能。像这样的能对某个或多个方法的功能造成影响的数据成员称为状态变量,如附例中的 allowVend,curQtrs。而像 totalQtrs 就不是一个状态变量,因为任何一个成员方法的具体执行情况都不依赖于它。

一个变量的所有状态的集合就是根据成员方法的行为而对变量定义域进行了一个分划(partition)。其正规定义如下:假设讨论的类为 C,v 是它的一个变量,v 的定义域为 D,S_1,S_2,...,S_k 为 D 的子集。那么称$\{S_1,S_2,...,S_k\}$为 v 的状态集当且仅当:

(1)$S_1,S_2,...,S_k$ 为 D 的一个分划。即对于任意的自然数 i,j ($\leq j<i\leq k$)有 $S_i\cap S_j=\phi$ 并且 $S_1\cup S_2\cup ...\cup S_k=D$。

(2)对于任意的自然数 i,j ($\leq j<i\leq k$),v 取 S_i 中的任意两个不同值时,所有的方法的功能都是一致的,而当 v 分别取 S_i、S_j 中的值时有某个或某些方法的功能不一致。

其中条件 1 就是数学中分划的概念,而条件 2 则是作这个分划的依据。有了这个依据,就能够保证在当前状态确定的情况下,任一个成员方法的执行效果都是唯一被确定的。

这是一个成员变量状态的情况,一个对象的当前状态指的就是各个状态变量的当前状态的组合。回到附例中,可以看到一个售货机有 4 种状态 ([0,0],[0,0]),([0,0],[1,M]),([1,M],[0,0]),([1,M],[1,M])。其中,变量的顺序为(allowVend,curQtrs),M 表示实现语言里面对非负整数的最大值。

一般而言,一个对象的状态的改变是由成员方法的调用所引起的。可以通过确定成员方法执行前的状态(pre-state)和执行后的状态(post-state)对它进行分析。从而可以得到类的状态和状态转化图,附录程序的状态和状态转化图如图 1。

注意到,如果一个类有 k 个状态变量,每个变量又分别有 $L_1,L_2,...,L_k$ 个状态,那么该类就有 $L_1\times L_2\times...\times L_k$ 个状态。随着变量数以及每个变量状态数的增加,类的状态的数目的增加是非常快的,实际经验表明在这时使用类状态和状态转换图基本上是不可行的。因此在文献[1]中,作者们提出了 AOSD(原子对象状态图),COSD(复合对象状态图)的概念。一个 AOSD 可以看成是一个具体的变量的状态和状态转换图,而 COSD 是单个 AOSD 的组合,详细介绍可以参照文献[1]。在文献[1]中还提出了一种利用符号执行从 C++源代码来获得各个状态变量的状态集并进而利用逆向工程方法提取状态依赖信息,生成 COSD 的算法。这里并不打算对此作过多的说明。只是将 OSD 作为算法的输入。

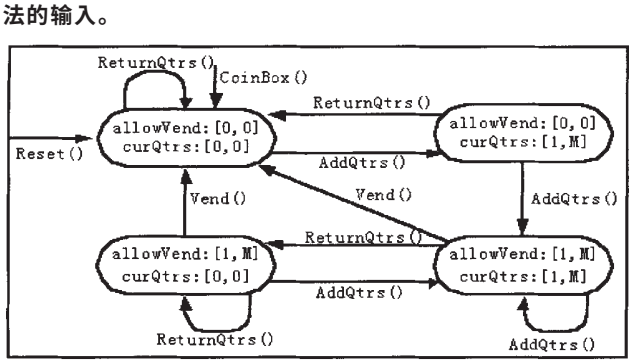

图 1

3 类测试

尽管类是面向对象测试中最基本的单元,但是对它的测试却比传统的单元测试包括更多的内容。在文献[2]中,Harrold 和 Rothermel 把对类的测试分成三个级别:

方法内的测试 Intra-method testing 对单个的成员方法作测试。这个级别的测试相对于传统的对过程性测试中的单元测试。

方法间的测试 Inter-method testing 对公用方法(public method)和它直接或间接调用的方法的测试。它相对于传统的集成测试。

类内测试 intra-class testing 对公用方法之间的交互进行测试。由于用户对类的对象的调用的不确定性,这部分的测试是非常复杂的。这也超出了传统测试所覆盖的范围。这个级别的测试是为了确保方法调用序列能够恰当的交互。

由于对类的测试大部分的工作集中于基于规格说明的黑盒测试,而黑盒测试并不能保证充分的代码覆盖,因此需要用基于代码的方法对它们进行补充。数据流分析就是这样的一种方法,其核心是对定义-引用对的分析。相对于类的三个测试级别有三个级别的定义-引用对:方法内的定义-引用对,方法间的定义-引用对,类内的定义-引用对[2]。文章将对它们进行定义。

由于面向对象技术具有的信息隐藏和封装机制,一个类通常只有私有属性,只能通过对象的共有方法来访问这个类的对象。该文不考虑公有方法之间有相互调用的情况。同时,有些方法的唯一功能就是告诉用户"这个值是多少?"。像这样的方法也不在考虑范围内。接下来,给出一些定义:

定义-应用对:设 v 是程序中某一变量,d 是在程序中对变量 v 进行赋值(定义)的语句,u 是程序中引用变量 v 的语句,并且从定义点 d 存在一条执行干净路径可以到达引用点 u,则称有序对(d,u)是变量 v 的一个定义-引用对。所谓干净路径是指在这条路径上没有对 v 进行重新赋值。

方法内的定义-引用对:M 是类的一个公共方法,d,u 都在 M 中,存在程序 P 调用 M,并且满足在 P 对 M 的一次调用过程中,(d,u)是一个被执行的定义-引用对。

方法间的定义-引用对:M_0 是类的一个公共方法,设$\{M_0,M_1,M_2,...M_n\}$ 为 M_0 被调用是直接或间接调用的类中的私有方法集,$d\in M_i,u\in M_j$,其中 $M_i,M_j\in\{M_0,M_1,M_2,...M_n\}$。如果存在程序 P 调用 M_0,使得 P 对 M_0 的一次调用中(d,u)是一个被执行的定义-引用对并且或者 $M_i\neq M_j$ 或者 M_i、M_j 为对同一方法的不连续的两次调用。

类内的定义-引用对:M_0、N_0 是类中两个公共方法,设$\{M_1,M_2,...M_n\}$,$\{N_1,N_2,...N_m\}$分别为 M_0、N_0 被调用时直接或间接调用的私有方法集,d,u 分别属于 $\{M_0,M_1,M_2,...M_n\}$,$\{N_0,N_1,N_2,...N_m\}$中的某个方法。如果存在 P 调用 M_0、N_0,使得(d,u)为 P 中的定义-引用对,并且在 d 执行后在 u 执行前,对 M_0 的调用已经终止。

对于方法内的和方法间的定义-引用对可以使用 PLR[4]等算法来计算。而对于类内定义-引用对而言,不能直接应用这些算法,因为这需要完整的程序以用来构造过程间的控制流图。要计算类内定义-引用对,必须考虑公用方法调用序列中的交互情况。在文献[2]中,作者提出了一种算法用来产生依赖图,这种图能够是他们考虑所有的交互情况并使 PLR 等算法能够得到应用。但是并不打算详细讨论这些,文章的思想是利用 OSD(包括 AOSD,COSD)来产生能够应用类内数据流分析的公用方法调用序列。这就是下一节要说明的。

4 基于 OSD 的方法调用序列的产生算法

尽管算法主要考虑的是第三个级别的测试问题，还是首先说明一下类的状态图对第一，第二两个级别的测试的帮助。首先要说明的是许多传统的测试方法都可以在这两个级别的测试上进行引用[9]。但是由于对类，尤其是对类的成员方法的规格说明通常很不正规或者根本没有，因此即使是传统的黑盒测试方法有时也无法使用。利用类状态和状态转化图可以确定各个方法的前状态和后状态，从而可以为测试带来很多有意义的信息。例如在利用由源代码得到的图 1 中，可以发现在先进行两次投币动作，然后再取消，接着再在 S3 状态下调用 Vend 是有悖常理的。这说明了 AddQtr 存在着问题，实际上这个问题是由于这个方法的规约说明的不明确而引起的，而这个错误只有通过方法间的交互才能发现。进一步分析，实际上整个状态 3 就是一个不合理的状态，就不应该存在。这也为第三步的测试打下了基础，能够更好地产生测试序列。而对于这两个级别的基于数据流分析的白盒测试就可以采用传统的对过程性语言的方法，在此不予说明。

由于前面提到的复杂性原因，因此并不直接采用类的状态和状态转化图，而是以 AOSD 为基础来构造算法。首先给出几个相关定义：

Pre[M]：方法 M 的所有前状态（pre-state）的集合。

Post[M]：方法 M 的所有后状态（post-state）的集合。

OUT[M]：方法 M 中所有那些至少有一个定义可以到达方法外的变量的集合。

IN[M]：方法 M 中所有那些至少有一个引用可以从方法外到达的变量的集合。

Out[v,M,s]：在当前状态为 s 的情况下，方法 M 中的被执行到的关于变量 v 的 d 的集合。

IN[v,M,s]：在当前状态为 s 的情况下，方法 M 中的被执行到的关于变量 v 的 u 的集合。

假定每个状态变量的 AOSD 都是连通的（如果不连通可以分别考虑他们的各个连通子图），并且假定存在初始状态 S_0 和终结状态 S_x（就是变量的值可以被观察到的一个状态）。同时在算法中也并不对以上定义的几个集合进行具体计算，但是他们是可以从相应变量的 AOSD 中比较容易得到的。

下面给出该算法：

```
INPUT COSD
 For every two public method M,N (possibly the same as M)
  IF IN[N]∩OUT[M]=ϕ THEN
   For every variable v∈IN[N]∩OUT[M]
    For every state S_b∈Post[M]
     For every state S_e∈Pre[N]
     IF Out[v,M,s]≠ϕ 和 IN[v,M,s]≠ϕ THEN
     BEGIN
      Find a method sequence N_1,N_2,...,N_i,M that can be applied on the initial state and results S_b;
      Find a method sequence N_1,L_1,L_2,...,L_j that can be applied on S_e and results S_x;
      IF there exist a state path S_b,S_1,S_2,...,S_k,S_b and a method sequence M_1,M_2,...,M_{k+1} such that :OUT [v,M_1,S_b]=OUT[v,M_2,S_b]=...OUT[v,M_k,S_{k-1}]=OUT[v,M_{k+1},S_k]=ϕ THEN
      BEGIN
       OUTPUT the sequence N_1,N_2,...N_i,M,M_1,M_2,...M_{k+1},N,L_1,L_2,...L_j;
       COUNTINUE;
      END
     END
```

算法的输入为被测试类的 COSD，但是对某一个具体的变量而言实际上用到的是它的 AOSD。对于任意的两个方法（可能相同），如果 IN[N]∩OUT[M]=ϕ 那么他们之间不存在类内定义-引用对。否则，对其中的每一个变量 v，考虑所有可能在 M 中定义，在 N 中引用的定义-引用对。根据对状态的定义，对于 M 每一个前状态 S_b，Out[v,M,s]或者为空或者只有一条语句。如果 Out[v,M,s]为空，则表示在当前状态为 s 的情况下 M 中不存在对 v 的定义。而对于 N 的每一个后状态 S_e，IN[v,M,s]或者为空或者是一些能一次被执行到的语句。如果 IN[v,N,s]为空，则表示在当前状态为 s 的情况下 N 中不存在对 v 的引用。因此只有当 Out[v,M,s] 和 IN[v,N,s]都不为空的时候，M 和 N 之间才可能有定义-引用对。这个如果存在一条对于变量 v 来说是干净的路径，那么就存在一个关于 v 的类内定义-引用对，因此产生一个测试序列来对它进行测试。

在该算法中，由于利用了 AOSD，每次只考虑一个状态变量来产生调用序列，因此对整个类的调用序列的产生的复杂度是各个变量复杂度的和，而不再是它们的乘积了，从而大大降低了复杂度。

5 结论

由于状态测试在面向对象测试中的重要地位，以及常用的对类测试方法存在的对代码覆盖不足的问题，将这两种方法结合起来提出了一种以类的状态图为基础的用来产生方法调用序列的算法。这种算法是针对类测试的类内测试的，它具有很好的代码覆盖率。而在对类的方法内和方法间测试方面，还可以利用类状态图来帮助进行基于规格说明的黑盒测试。实际上，文章提出了一种将白盒测试和黑盒测试，状态测试和数据流分析相结合的思想。（收稿日期：2002 年 3 月）

附录：

这是一个用 C++实现的关于售货机的例程。为了简便，假定它只具有非常简单的功能。它只收硬币，当前的硬币数为两个时可以进行售货。

```
Class CoinBox{
 private:
  unsigned int totalQtrs;//total quarters collected
  unsigned int curQtrs;//current quarters collected
  unsigned int allowVend;//1=vending is allowed
  unsigned int IsAllowVend(){return allowVend;}
  void Reset(){totalQtrs=0;curQtrs=0;allowVend=0;}
 public:
  CoinBox(){Reset();}
  void returnQtrs(){curQtrs=0;}//return current quarters
  void addQtrs(){
   curQtrs=curQtrs+1;//add a quater
   if(curQtrs>1)//if more then one quarter is
    allowVend=1;//collected,then set allowVend
  }
  void vend(){
   if(IsAllowVend()){//if allowVend
    totalQtrs=totalQtrs+curQtrs;//update totalQtrs
    curQtrs=0;                  //update curQtrs
```

```
        allowVend=0;              //update allowVend
      }
    }
  }
```

参考文献

1.D Kung,N Suchak J Gao et al.On Object State Testing[C].In :Proceedings,The Eighteenth Annual International Computer Software & Applications Conference,IEEE Computer Society Press,Los Alamitos,Calif,1993 :222~227

2.M J Harrold,G Rothermel.Performing data flow testing on classes[C].In 2nd ACM-SIGSOFT Symposium on the foundations of software engineering,New Orleans,LA (USA),1994 :154~163

3.M J Harrold,M L Soffa.Interprocedural data flow testing[C].In :Proceedings of the Third Testing,Analysis and Verification Symposium,1989 :158~167

4.Berard.Issues in the Testing of Object-oriented Software[C].In : Berard ed.Electro/94 International,Conference Proceedings Combined Volumes.IEEE Computer Society Press,Los Alamitos,Calif,1994 : 211~ 219

5.Robert V Binder.State-based Testing[J].Object Magazine,1995 ;5 (4) : 75~78

6.Robert V Binder.Testing Object-Oriented Software :A Survey[J].Journal of Software Testing,Verification and Reliability,1996 :125~252

7.Kuang-Nan Chang,D Kung,P Hsia et al.Object-Oriented Data Flow Testing[C].In :Proceedings 13th International Conference and Exposition on Testing Computer Software,USPDI,Silver Spring,Maryland,1996 : 97~100

8.David C Kung,Pei Hsia,Gerry Gao.Testing Object-Oriented Software[M].IEEE Computer Society Press,1998

9.Sang Chung,M Munro,W K Lee Y R Kwon.Applying Conventional Testing Techniques for Class Testing[C].In :Proceedings of 20th International Computer Software and Applications Conference :COMPSAC '96,IEEE Computer Society Press,Los Alamitos,Calif,1996 : 447~454

(上接 34 页)

产调度方法中。Agent 是一种主动对象。它与面向对象方法有所区别 :①Agent 可主动运行,有自己的目标和行为,可由外部激励或内部状态而启动,而 object 是纯被动的,只能由外部的消息控制。②Agent 是一个自治的实体,具有自己的知识和分析问题方法,能理解信息并控制自己的行为,而 object 只能机械地执行所规定的动作。③Agent 能根据推理规则进行信息的抽象,而 object 却不具备推理功能。

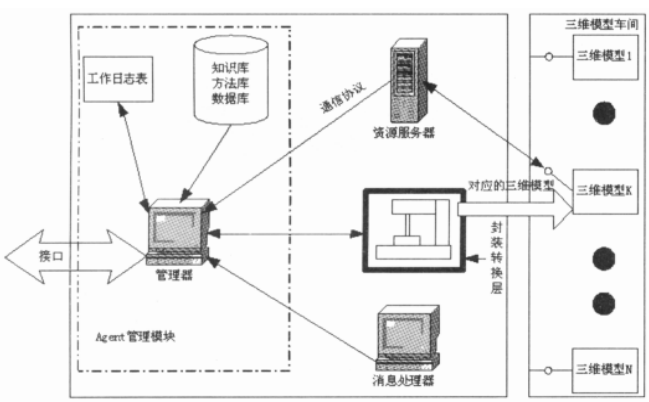

图 6 虚拟设备资源 Agent 结构

车间生产中提高效率、柔性、可靠性的关键技术就是作业调度和控制技术。为了更好的支持敏捷制造,其中的一种思想就是将虚拟车间的制造资源 Agent 化,对建立的 Agent 要求能完成两个任务 :首先,它能够代表客户工作 ;其次,它可以把设备资源包装起来,引导并代替用户对这些资源进行访问,成为便于通达这些资源的枢纽和中介,而且它还负责对制造资源的自治管理。Agent 化分两个步骤完成 :第一步,为设备资源原有控制器建造一个转换层,目的是把设备资源转变成标准的网络服务器节点 ;第二步,建造一个 Agent 管理器,它包括对制造资源的功能性的高层次的 '镜像 '[7]。如图 6 所示,为设备 Agent 结构,其中服务器负责本设备的加工任务的执行,而管理器则负责与外部的信息交流和服务器的工作日程管理。在三维模型车间中,通过对设备建立对应的三维模型,由资源服务器提供加工生产指令和管理信息,最终完成三维仿真环境下虚拟的加工过程。另一方面,在这个 Multi-Agent 单元中,人占有主导地位,而 Agent 管理器主要用来辅助管理节点上的各种资源和调度工具。Agent 管理器的辅助管理使得设计者在虚拟三维环境下把更多的注意力用在调度任务上,从而提高设计效率。

3 结论

基于多 Agent 的虚拟车间具有许多传统企业所不具备的优势。它的适应性强可以组织各种规模的生产 ;重构性能好,可以适应不断变化的市场要求 ;通过合作,可以集中全球范围内的资源和技术要求,增强企业的竞争力 ;可以降低风险,降低费用 ;合作也便于开发进入新市场,避开市场堡垒 ;在运行过程中采用双赢策略,使参与各方均受益。由于虚拟车间的伙伴分散性大,不利于资源全面优化和项目的监督控制 ;所以对车间的三维建模可以将一个逻辑上的车间变成一个视觉上形象的车间,通过网络,可以使这个车间在各个合作伙伴的监督控制之下。对虚拟车间进行三维建模和使用 Agent 技术可以解决在不同地域条件下,生产调度的可行性,能够将实际生产过程中存在的问题显示出来,供生产组织者进行正确的决策。避免或减少生产中的各种资源的浪费[6]。 (收稿日期 :2002 年 11 月)

参考文献

1.王洪海,陈幼平.虚拟制造环境下基于多智能体的敏捷调度策略的研究[J].计算机工程与应用,2002 ;38 (8) :74~76

2.易先平,陈凌.信息化的产物——虚拟企业的运营模式分析[J].商业研究,2001 ;(2) :143~144

3.张生芳,沙智华等.基于 VRML 数控车削加工全景仿真技术研究[J].大连铁道学院学报,2001 ;22 (4) :17~20

4.王新龙,肖田元等.虚拟制造及其应用[J].中国图像图形学报,1999 ;4 (1) :80~87

5.刘进江,周旭等.动态联盟公司异地生产计划调度与控制系统的研究[J].机械工业自动化,1998 ;20 (6) :1~5

6.G Chryssolouris,D Mavrikios,D Fragos et al.A novel virtual experimentation approach to planning and training for manufacturing processes-the virtual machine shop[J].Journal of Computer Integrated Manufacturing,2002 ;15 (8) :214~221