基于消息序列的形式化面向对象软件测试技术

赖祥伟　张为群

（西南师范大学计算机与信息科学学院　重庆400715）

摘　要　本文分析了UML交互图和状态图中的消息提取机制，构造了使用形式化描述语言RAISE对UML图中提取的消息进行描述、规约和精华，最终生成测试用例的测试流程，提出了基于消息序列的面向对象软件自动测试方法，并且在此基础上提出了检测OO系统容错能力的测试用例构造方法。

关键词　面向对象技术，软件测试，UML，RAISE

一、面向对象软件测试的问题及现状

进入90年代，面向对象技术规范因其相对于原有的过程化程序设计语言(procedural languages)的许多优点已为业界所广泛应用，这也给原有的软件测试技术带来了许多新的问题。

过程化程序设计是把程序看作是工作在一系列特定数据集上的过程和函数的集合(即程序＝数据结构＋算法)，而OOP是把程序作为彼此独立而又相互协作的对象的集合。类作为属性(attribute)、方法(operation)的集合本身就是一个传统意义上的程序，在OOP中程序被看作以类为最小单位元素的实体，类间通过消息机制相互作用(即OOP＝类＋对象＋继承机制＋消息传递)。大量研究表明传统的软件测试技术不完全适用于OO软件的测试，针对OO自身的特点必须对传统的软件测试技术作相应的调整，一般认为主要包括以下四个方面：

1、测试的定义必须扩大包括用于OOA和OOD模型的错误发现。注意测试OOA和OOD模型的正确性和一致性。

2、单元测试和集成测试的策略必须作很大的改变。

3、测试用例的设计必须考虑OO软件的独特特征。注意由于继承、封装、多态等OO方法带来的测试问题。

4、采用适合的软件度量标准。现在常用的度量标准包括：

二、统一建模语言UML与形式化方法RAISE

统一建模语言(UML)在1997年被国际标准化组织(OMG)接纳为正式官方标准后已成为新一代面向对象软件设计的事实标准。其与Rational统一过程(或其它软件工程开发过程)的配合使用更被业界许多企业所采用。

形式化方法指为保证复杂系统的可靠性而对其进行精确描述和验证的基于数学的语言、技术和工具，其关键在于形式化的规约语言。由于形式化方法对于解决保证软件开发过程中的一致性和正确性问题具有较好的效果，但是由于其要求开发者具有较深的数学和逻辑学基础使得其难以被广大的普通开发者所使用，这也阻碍了它的进一步发展和推广。

面向对象的软件开发方法不能够解决软件开发过程中的正确性问题，而形式化方法难以被广泛应用。将两者结合起来，共同为软件开发过程服务已成为当前软件开发的一大趋势。

三、基于消息序列的形式化面向对象软件测试技术

在面向对象的软件测试过程中，基于类的单元测试可以使得类本身的可靠性得到一定程度的验证。在后续的集成测试阶段，由于消息被广泛地用于类间调用和系统控制，故以消息机制为驱动的测试是实施集成测试的一个重要途径。

本文提出的测试流程主要采用的方法是从UML图中提取并形式化地描述消息结构和操作结果，通过RAISE形式化规约和精化处理，并在此基础上生成适当的测试用例用于测试。具体流程如图1所示。

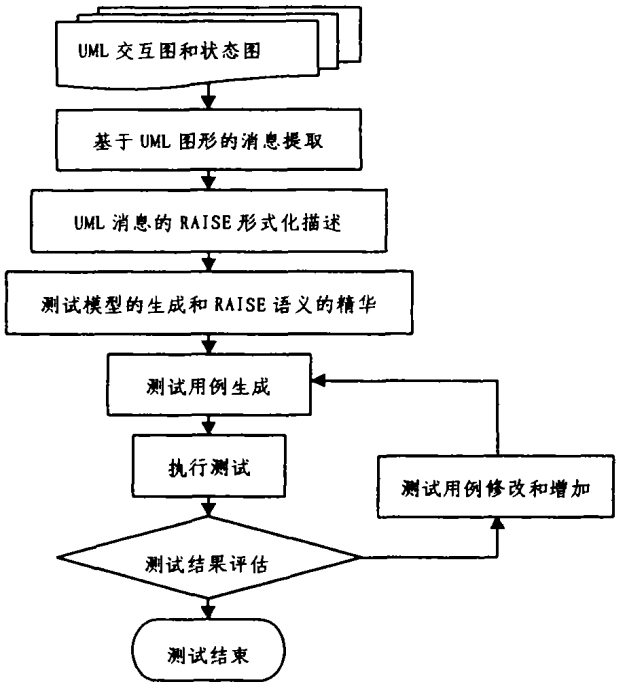

图1　基于UML图的形式化测试流程

需要解决的问题主要是如何提取并描述软件消息流程以及RAISE的规约、精华方法以及测试用例的生成技术。

3.1　基于UML图和形式化方法RAISE的消息提取和描述机制

在UML中，设计者采用多种图形描述系统的构造情况，其中与消息传递相关的图形主要是交互图(包括顺序图和协作图)和状态图。两者分别用于描述系统中不同抽象层次的关系。交互图的抽象较为宏观，更适用于描述类(对象)间的关系。状态图的抽象层次趋于微观，更适用于描述模块内部状态变化情况。当然这种划分只是相对而言，两种图同样可以描述其它层次的关系。

定义1　使用四元组M＝(S,R,O,P)表示一个OO系统的消息，其中S定义消息的发送者或产生者(dispatcher)，R定义被消息刺激的接受者对象(receiver object)，O指接收消息的方法(operation)，P提供操作成功所必需的信息(主要指

参数 parameters)。

使用 RAISE 建立的消息模型的表示为:

```
scheme MESSAGE =
 class
  type
    object,oper,para,
    message(S :object,R :object,O :oper,
            P :{|ps :para_set·(∀ p:para⇒p∈para_set)})
 end
```

(1)交互图的信息描述　交互图包括顺序图和协作图两类等价的描述图形,两者可以等价地互相转化。顺序图(sequence diagram)强调消息的时间顺序。协作图(collaboration diagram)强调发送和接收消息的对象之间的结构组织。交互图一般包括对象、链和消息三种元素。由于顺序图和协作图在逻辑上的等价,它们所产生的消息的形式化描述是相同的。

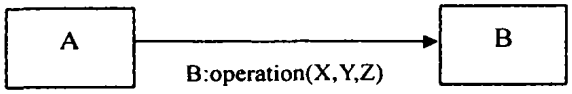

图2

在交互图中通常以链为消息提取的基本标志。图2是一个最简单的协作图,其与 RAISE 形式化描述的对应关系为:

S →A ;R→B; O→B:operation;P→{x,y,z}

通过继承 MESSAGE 的定义,图1中的消息可以描述为:

```
scheme MESSAGE1 =
Extend MESSAGE with
  class
    value
      message(A,B,B:operation,{x,y,z})
end
```

(2)状态图的信息描述　状态图(statechart diagram)显示了一个状态机。它强调从状态到状态的控制流。状态(state)是指在对象的生命周期中的一个条件或状况,在此期间对象将满足某些条件、执行某些活动或等待某些事件。事件是对一个在时间和空间上占有一定位置的有意义的事情的规格说明。在状态机的语境中,一个事件是一个激发的产生,激发能够触发一个状态转换。事件可能包括信号、调用、时间推移和状态的一次改变四种情况,其中信号和调用可以等价于消息;而时间推移和状态的一次改变并非完全等价于一个 OOP 中的消息,但是由于其固有的特点,绝大多数情况下均可以以消息的形式所描述。

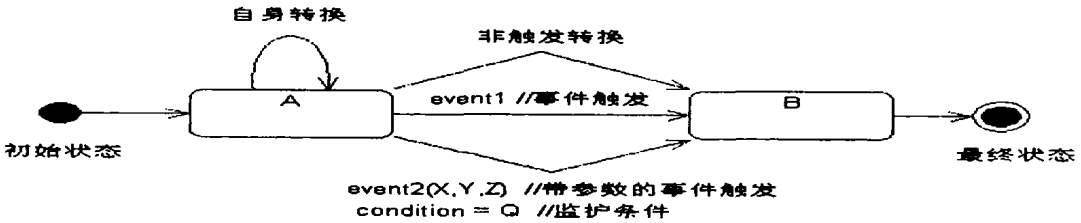

图3

如图3所示,在状态图中状态转换包括非触发转换、事件触发转换和带参数的事件触发转换三种,而每一种状态转换都可能带有相应的监护条件。

在状态图中使用箭头为消息提取的基本标志,其对应关系为:

S :A ;

R :B/A(自身转换);

O :NULL (在状态图中无法表示调用的方法和操作)

P :NULL(非事件触发)/event(事件触发)∪ Q(监护条件)∪{ NULL(非触发转换和事件触发)/{X,Y,Z}(带参数的事件触发)}

图3中的消息可以描述为:

```
scheme MESSAGE2 =
Extend MESSAGE with
  class
    value
      message1(A,A,NULL,{condition=Q}), //设自身转化为非触发转换
      message2(A,B,NULL,{condition=Q}), //非触发转换
      message3(A,B,NULL,{ event1∪(condition=Q)}), //事件触发转换
      message4(A,B,NULL,{ event2∪(condition=Q)∪x∪y∪z}) //带参数的事件触发转换
end
```

3.2 基于消息序列的测试用例生成技术

(1)基于 UML 交互图和状态图形式化描述的测试用例生成技术　对于一个被测软件而言,系统完成某一特定功能往往需要一系列的步骤和操作,其间当然也会需要和产生一系列的激励消息。

定义2　令 M*=(M,R)表示一个用于测试的完整消息,其中 M 表示一个 OO 系统中的消息(参见定义1),R 表示消息 M 激励系统进行操作所产生的运算结果或动作。

M* 可以被形式化的定义为:

```
scheme TESTMESSAGE =
Extend MESSAGE with
  class
    type
      result,
      testmessage(message,result)
end
```

定义3　对于有限的顺序测试消息集合 Σ={M*1,M*2…M*n},当软件系统处于初始状态 S,顺序的接受消息激励 Σ,软件系统将从初始态 S 经过状态 S1、S2…,最终转移到状态 Sn,简记为 $S \xrightarrow{\Sigma} Sn$,则称有序消息集合 Σ 为软件系统的一个测试用例。

定义4　对于一个软件系统的测试用例的集合称为该软件系统的测试用例集。

定义测试消息的合并规则为:

规则1　若存在测试消息 M*1=(S1,R1,O1,P1,RL1)、M*2=(S2,R2,O2,P2,RL2),其中 R1=S2,可以认为存在测试消息 M*3=(S3,R3,O3,P3,RL3),其中 S3=S1,R3=R2,O3==O1∪O2,P=P1∪P2,RL=RL1∪RL2。

形式化的定义规则1的消息合并规则:

```
scheme UNITE =
  Extend TESTMESSAGE with
  class
    value
    unit :testmessage×testmessage→testmessage
  axiom forall
    ∀ M1(S1,R1,O1,P1,RL1),∀ M2(S2,R2,O2,P2,RL2):unit
(M1,M2)≡M3(S1,R2,{O1∪O2},{P1∪P2},{RL1∪RL2})
end
```

规则2　对于交互图和状态图的某个测试消息 M*=(

S,R,O,P,RL)而言,P 为测试用数据集合,O∪RL 为软件对该测试用例的正确反映。

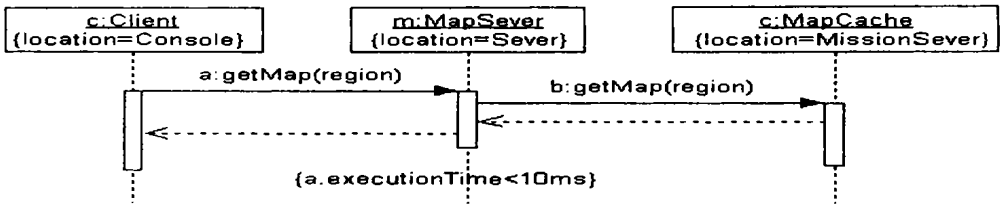

图4

图4为一个简单的 UML 顺序图,描述了一个网络地图查询服务系统的工作顺序流程。其中三个主体分别处于不同的物理位置,由客户端向服务器发送调用地图位置消息,服务器再从地图存储缓冲区中调用地图具体信息返回客户端。使用 RAISE 可以表示为:

```
scheme GETMAP =
  Extend TESTMESSAGE with
class
    value
     message1 (Client, MapSever, getMap, {(a.executionTime <
10ms)∪region},NULL),
     message2(MapSever,MapCache,getMap,{region},NULL),
     message3(MapCache,MapSever,NULL,NULL,NULL),
     message4(MapSever,Client,showMap,NULL,showmap),
    tm ≡unite(unite(unite(message1,message2),message3),mes-
       sage4)
       ≡ (Client, Clinet, {getMap, showMap}, { region ∪
       a.executionTime<10ms},showmap)
end
```

根据规则2,通过 RAISE 精华后图4中所产生的测试用例的测试数据集为{(a.executionTime<10ms)∪region},测试的预期结果为 showmap。

在形式化地定义了 UML 交互图和状态图的形式化语义的基础上可以通过对 UML 图的形式化语义进行形式化的规约和进化生成相应的测试用例。这样不仅可以保证测试用例的正确推导,提供精确而完整的测试用例集,而且有利于测试用例的自动化生成。可以说形式化方法是该测试用于实际测试工作所不可缺少的重要手段。

(2)检验系统容错能力的测试用例生成 在 OO 软件设计过程中要求软件具有一定的适应性和容错能力,以适应环境和用户使用可能带来的危害。在过程化程序的测试中一般采用边界值检查等方法进行测试。对于 OO 软件而言,同样在用户输入界面测试的某些地方采用边界值等传统方法,但是这些测试相对于越发庞大的软件而言是远远不够的。

在前面所生成的测试用例只是针对正常的软件功能进行测试使用的用例,并没有包括对于错误情况的测试。通过以下的策略可以非常简单地将上文中生成的用例自动扩展生成针对软件容错能力的测试用例。

A. 遗漏法。在执行某一测试用例时故意遗漏一个或者多个参数(甚至是消息)以检查系统的反应。

B. 替换法。使用错误的参数或者消息替代测试用例序列中的正确参数或消息。

C. 冗余法。重复执行测试用例序列中的一个或者多个消息。

D. 交错运行法。同时执行软件的两个或多个测试用例,以检测软件各个功能或者同一功能的各个模块之间是否存在并发问题。

当然在测试过程中,测试并不要求软件系统能够对所有的测试用例作出与原有结果相同的反应。但是要求软件能够作出如报错、禁止操作等错误处理,把错误带来的危害和影响尽量降低到最小的程度,最基本的是不能导致系统的崩溃或者数据的丢失。

结论 本文主要讨论了使用消息驱动机制构造 OO 系统测试用例的方法和实现技术。选择采用流行的 OO 建模方法 UML 的交互图和状态图作为 OO 系统消息提取的基础,同时使用形式化方法 RAISE 对 UML 的交互图和状态图所提取的消息作形式化描述,通过对于形式化描述的规约和精华保证测试用例的正确生成。最后提出了检验系统容错能力的测试用例生成方法以保证测试的完整性。

基于规格描述的形式化软件测试方法是当前测试研究的热点之一。本文所提出的方法能够有效地协助解决测试用例的自动生成技术,但是并不能完全解决测试的全覆盖问题,因为在实际工程设计实现中出于软件开发成本的考虑,设计人员使用 UML 图并不能完整地包括软件系统中所有的消息。(实际上是用任何方法都无法达到软件系统的全覆盖测试)在以后的工作中需要进一步加强的研究方向包括:

(1)进一步扩展消息提取的途径和方法。

(2)讨论基于消息序列的软件测试的覆盖率问题,尽量达到较高的测试覆盖率。

(3)构造原型系统,实现形式化语言的自动规约和精华以及测试用例的自动生成。

参 考 文 献

1 The RAISE Method Group,George C,et al. The RAISE Development Method. TERMA Elecktronik AS,Denmark,1999

2 The RAISE Language Group. The RAISE Specification Language. Prentice Hall Intermational(UK)Ltd. 1995

3 Booch G, Rumbaugh J, Jacobson I. The Unified Modeling Language User Guide, Addisom-Wesley, 1999

4 Roger S. Pressman Software Engineering A Prectitioner's Approach Fourth Edition McGraw-Hill, 1997

5 Lano K, Haughton H. Objiect-Oriented Specification Case Studies. Prentice Hall, 1994

6 Binder R. Testing Object-Oriented Systems: A Status Report American Programmer, 1994, 7(4): 476～493

7 Wilde N I, Huitt R. Maintaining Object-Oriented Software Metrics, Pretice-Hall, 1994

8 面向对象软件测试理论与技术的研究:[博士论文]. 西安电子科技大学

9 李留英,王戟,齐治昌. UML statecharts 的测试用例生成方法. 计算机研究与发展,38(6)

10 兰毓华,毛法尧,曹化工. 基于 Z 规格说明的软件测试用例自动生成. 计算机学报,22(9)