基于UML状态图测试的充分性准则*)

占学德　缪淮扣
(上海大学计算机学院　上海200072)

摘　要　本文描述了基于UML状态图生成测试用例的充分性准则。在测试时，循环被执行一次是很不充分的，本文提出了使循环分别执行0次、1次、2次，即ZOT循环覆盖准则，在此基础上本文还提出了全ZOT路径覆盖准则，并给出了这些覆盖准则的应用，讨论了这些覆盖准则之间的包含关系。

关键词　UML状态图，测试准则，基于规格说明的测试

Adequacy Criteria for Testing Based on UML State Diagram

ZHAN Xue-De　MIAO Huai-Kou
(School of Computer Engineering and Science, Shanghai University, Shanghai 200072)

Abstract　This paper presents these adequacy criteria that generate test cases based on UML diagram. It isn't very sufficiency that a loop is executed only one time while testing. This paper proposes ZOT loop coverage criteria, namely a loop is executed zero, one and two times, At the same time, full ZOT path coverage criteria is proposed. The applications of these coverage criteria are given and the subsumption relationships of coverage criteria are discussed.

Keywords　UML state diagram, Test criteria, Specification-based software testing

1　引言

在航天、金融、医疗等许多领域所使用的计算机软件对安全可靠性方面的要求是相当高的，软件测试是保证软件的安全可靠性的关键技术之一，而测试覆盖准则在软件测试中又起着非常重要的作用。

由于UML的易理解性和实用性，它被广泛地应用于面向对象的分析与设计，因而基于UML产生测试用例的研究就显得尤为重要。对于面向对象的类的测试：先要进行类内部的测试，然后再进行类之间的测试。而类内部的测试可分为两种：方法内部的测试(即传统的单元测试)和方法间的测试。基于UML状态图生成测试用例就是针对类的方法间的测试，我们所描述的测试准则适用于在这种场合。本文所涉及的UML状态图不含层次和并发结构。

在不同的软件测试层次和不同的软件测试方法中，已经有不少的软件测试充分性准则被提出[1~3]。Hong Zhu等人在文[1]中总结了软件单元测试的覆盖与充分性准则，B. Haworth等人在文[2,3]中提出了面向对象系统的充分性准则。在基于UML状态图生成测试时，目前还没有一个很充分的测试准则。基于规格说明的测试准则已有不少的研究，如文[4~7]，在文[4,5]中这些准则可以用于基于UML状态图生成测试。本文在这个准则的基础上主要提出了ZOT循环覆盖准则和全ZOT路径覆盖准则。这样使基于UML状态图生成测试的覆盖准则更加充分。

根据测试用例产生的依据，软件测试可分为基于程序代码的测试和基于规格说明的测试。在测试含有循环的问题时，不论是基于程序的测试还是基于规格说明的测试，基本上都是保证循环被执行一次。对于一些常见错误都完全有可能测试不出来。例如，编写程序完成以下功能：通过键盘输入整数，当输入零时结束输入，统计共输入了多少个非零整数。下面是用C++写的一个程序段：

```
int x=1,i=0;
while(x!=0)
  {  cout<<"请输入一个整数:";
     cin>>x;
     cout<<"输入的数是: "<<x<<"\n"<<"\n";
     i=1;};
cout<<"输入非零整数的个数是:"<<i<<"\n";
```

在这个程序中有一个很明显的错误，那就是语句i=i+1中掉了"i+"。在测试这个程序时，如果循环只执行一次，那么这个程序在测试时不会发现这个错误；但是如果循环执行两次，这个错误就会被发现。实际上，在基于规格说明的测试中，也存在同样的问题，下文会具体地讨论。对于含有循环的问题，测试一次很不充分，测试时执行循环的所有次数显然是不现实的，特别是无限次的循环将是不可能的。事实上，分析一个比较简单的循环问题时，把循环执行2至3次以后，这个循环的基本规律也就把握了。基于不完全归纳法的思想，对于循环的测试，本文提出使一个循环分别执行0次、1次、2次(Zero-One-Two)。

在文[4]中给出了一条覆盖准则，即"完全序列准则"，那就是：测试用例集应该包含遍历规格说明图中的迁移的"meaningful sequences"的测试，而测试工程师对这些序列的选择是基于经验和领域知识等。当有循环时，迁移序列的数量是非常大的，甚至于是无限的。这条准则要求工程师按照自己的经验和领域知识，从庞大的或者是无限的数量中去选择部分的迁移序列。这条准则的可操作性是比较弱的，不同工程师的经验和领域知识是不一样的，在进行软件自动化测试时更是难以实现。本文结合ZOT循环覆盖准则，提出了全ZOT路

*)本课题得到国家自然科学基金项目(60173030和60373072)资助。占学德　讲师、博士生，主要研究领域为软件形式化方法、软件测试。缪淮扣　教授，博导，主要研究领域为软件工程、软件形式化方法。

径覆盖准则。这个准则不仅容易操作，而且能较好地应用于软件自动化测试中。

本文通过 UML 状态图的一个例子，给出了本文所描述的6条测试准则的具体应用。最后还讨论了6条测试准则之间的包含关系。

2 UML 状态图

UML 用于面向对象系统的分析与设计时，主要通过各种图来描述系统的规格说明，其中的状态图是重要的部分。基于 UML 状态图生成测试用例的方法属于基于规格说明的测试。

UML 状态图描述了一个特定对象的所有可能状态以及由于各种事件的发生而引起状态之间的迁移。大多数面向对象技术都使用状态图来描述一个对象在其生命周期中的行为。UML 状态图由状态和迁移构成，状态图中的每一个状态是可达的。初始状态是对象的初始状况，代表一个状态图的起始点，是一个伪状态，它是迁移的初始源，而不能是迁移的目标，用一个实心的圆表示，本文用 init 标识。终止状态是对象的最后的状态，代表一个状态图的终止点，它是迁移的最后目标，而不能是迁移的源，用一个圆中套一个小实心圆表示，本文用 fina 标识。迁移使系统从一个源状态到达一个目标状态，源状态和目标状态可以相同，一个迁移由三部分组成：事件、条件、动作，其格式如下：

事件[条件]/动作

其中每一部分都是可选的。当系统处在一个状态（当前状态）时，如果“事件”被接受，并且“条件”的值为真，则这个迁移被激活，就执行这个迁移的输出“动作”，并且指定的目标状态变成当前状态。

在基于状态测试的策略下，从状态图出发，根据不同的测试准则来产生测试用例[5,8,9]。图1是两个选手击球游戏的状态图。

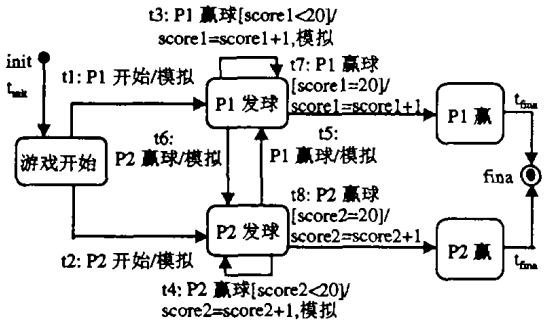

图1 两个选手游戏的状态图

3 基于 UML 状态图的测试准则

在软件测试中，只能说已经找到了多少错误，永远也不知道还有多少错误没有找到。在软件测试中还有一个非常重要的问题就是什么时候停止测试。测试者是根据具体的测试准则和测试策略来产生某个软件产品的测试用例集，也就是说这些测试用例满足预先给定的测试准则，当用这些测试用例去运行这个软件以后，就可以说这个软件产品在这个测试准则下被充分地测试了，此时测试就可以停止了。遗憾的是不能说这个软件产品被充分地测试了。因而测试准则是否充分在软件测试中起着至关重要的作用。

测试需求就是那些应该被满足或者被覆盖的软件产品的具体元素。例如，人们所熟悉的基于程序测试的语句覆盖，它的测试需求就是那些应该被执行的语句。测试准则就是使测试用例集满足测试需求的规则的集合。语句覆盖这条测试准则就是所有的可执行语句都至少被执行一次。

本文共讨论和分析了6条测试准则：状态覆盖准则、迁移覆盖准则、全谓词公式覆盖准则、迁移对覆盖准则、ZOT 循环覆盖准则、全 ZOT 路径覆盖准则。它们的定义将在下面的几个小节中详细地给出，这里的每条测试准则所需要的测试用例的个数通常是不相同的。这些准则的定义都是基于 UML 状态图的。

3.1 状态覆盖准则

状态覆盖准则：测试用例集 T 应该使 UML 状态图中的每一个状态至少被访问一次。

UML 状态图中的每一个状态是可达的，因而每一个状态被访问一次是很容易做到的。状态覆盖准则是这些测试准则中最简单、最容易被满足的测试准则，需要的测试用例也往往是最少的。

3.2 迁移覆盖准则

迁移覆盖准则：测试用例集 T 应该使 UML 状态图中的每一个迁移都至少被激活一次。

先使系统到达某一个状态（当前状态），如果一个迁移的“事件”被接受，并且这个迁移的“条件”的值为真，则这个迁移被激活。迁移覆盖准则是这些测试准则中比较简单、比较易被满足的测试准则，需要的测试用例也往往是比较少的。

3.3 全谓词公式覆盖准则

在定义全谓词公式覆盖准则之前，先引入谓词决定谓词公式（文[4,5]中的 predicate 和 clause 分别表示了通常的谓词逻辑中的谓词公式和谓词。本文中的谓词公式和谓词分别表示了文[4,5]中的 predicate 和 clause）的概念。在一个谓词公式 F 中有一个谓词 P，如果其余的谓词都有一个具体的值，则或者 P、F 有相同的真值（P 真 F 也真，P 假 F 也假），或者 P、F 有相反的真值（P 真 F 假，P 假 F 真），那么我们就说谓词 P 决定谓词公式 F 的真值，此时，我们把谓词 P 叫做谓词公式 F 的主要谓词，同时，还把其余的谓词叫做谓词公式 F 的次要谓词。如果谓词 P 决定谓词公式 F 的真值，则显然不允许 P 是真的时候 F 是真，同时 P 是假的时候 F 也是真。

例如，对于谓词公式 $A \wedge B$，当谓词 B 取真时，谓词 A 就决定了谓词公式 $A \wedge B$ 的真值（显然当 A 取真时，谓词 B 也决定这个谓词公式的真值），即 A 取真时，$A \wedge B$ 也取真，A 取假时，$A \wedge B$ 也取假。也就是当谓词 B 取真时，谓词 A 就是谓词公式 F 的主要谓词。又对于谓词公式 $A \vee B$，当谓词 B 取假时，谓词 A 就决定了谓词公式 $A \vee B$ 的真值（显然当 A 取假时，谓词 B 也决定这个谓词公式的真值）。下面是全谓词公式覆盖准则的定义：

全谓词公式覆盖准则：测试用例集 T 应该包含 UML 状态图中的每一个迁移的谓词公式 F 的每一个主要谓词 P 取真和取假各一次（次要谓词都有一个具体的值）的测试。

假如一个迁移的谓词公式是 $A \wedge B$，当谓词 B 取真时，A 是主要谓词，B 是次要谓词，则在测试用例集中应包含 B 取真时 A 取真和取假各一次的测试。又当谓词 A 取真时，B 是主要谓词，A 是次要谓词，则在测试用例集中应包含 A 取真时 B 取真和取假各一次的测试。这样，当一个迁移的谓词公式是 $A \wedge B$ 时，满足全谓词公式覆盖准则的谓词 A、B 的真值组合有三种情况：A 取真 B 取真、A 取真 B 取假、A 取假 B 取

真，去掉了一组重复的真值。

当一个迁移的谓词公式只有一个谓词 P，则满足全谓词公式覆盖准则的谓词 P 的真值组合有两种情况：P 取真、P 取假。当一个迁移没有监视条件时，它的谓词公式就是一个永真式，那么满足全谓词公式覆盖准则的测试只需要包含这个谓词公式为真（即这个迁移被激活一次）的情形就可以了。

对于较复杂的谓词公式，当选定某一个主要谓词之后，次要谓词的真值如何确定呢？这个问题在文[4]中给出了具体的方法。

满足全谓词公式覆盖准则的这些测试用例是来自于合法的和非法的迁移，某一个时刻仅有一个迁移是合法的。非法迁移的测试就是强行地使之满足非法迁移的前置条件，这种测试能有效地测试潜行路径。被测实现（IUT）在某一个状态接受了一个没有被规定的事件而产生的路径，或者是被测实现接受了逃避监视条件的迁移而产生的路径，这样的非法路径就是潜行路径[8]。假如基于状态图1的被测实现在状态“P1发球”接受了非法迁移“P1赢球[score1≠20]/score1= score1＋1”而到达状态“P1赢”，这条路径是潜行路径。满足全谓词公式覆盖准则的测试用例集包含了这样的非法迁移的测试。

3.4 迁移对覆盖准则

在软件中的许多错误可能是因为软件工程师没有完全地理解规格说明中状态序列之间的复杂的相互作用关系，上面的准则孤立地测试了迁移，而没有测试迁移的序列，因此一些错误也许不能被充分地测试。可能因为一个非法的迁移序列被允许，合法的迁移序列不被允许，而导致典型错误的出现。为了检查这些类型的错误，引入迁移对覆盖准则。迁移对覆盖需要所有的迁移对都是使能的。S_i，S_j，S_k 表示三个状态，用 S_i：S_j 表示两个状态 S_i 和 S_j 之间的一个迁移。

迁移对覆盖准则：测试用例集 T 应该包含 UML 状态图中的每一对邻近迁移 S_i：S_j 和 S_j：S_k 的测试。

考虑下面的图2，在迁移对覆盖准则中测试状态 $S6$，共有6个测试需求：(1)从 $S1$经 $S6$到 $S3$；(2)从 $S1$经 $S6$到 $S4$；(3)从 $S1$经 $S6$到 $S5$；(4)从 $S2$经 $S6$到 $S3$；(5)从 $S2$经 $S6$到 $S4$；(6)从 $S2$经 $S6$到 $S5$。在测试用例集中应该包含下面的6组迁移对：(1)$t1$和 $t3$；(2)$t1$和 $t4$；(3)$t1$和 $t5$；(4)$t2$和 $t3$；(5)$t2$和 $t4$；(6)$t2$和 $t5$。

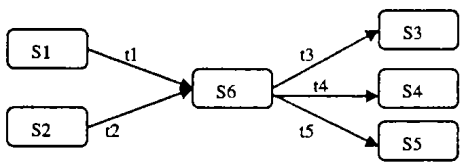

图2 迁移对覆盖

3.5 ZOT 循环覆盖准则

在测试含有循环的问题时，不论是基于程序的测试还是基于规格说明的测试，以往的测试准则基本上都是保证循环被执行一次。完全有可能对于一些常见错误无能为力，在引言中给出的程序就是一个例子。实际上，在基于规格说明的测试中，也存在同样的问题。在状态图1中从状态“P1发球”经迁移 $t3$：“P1赢球[score1＜20]/score1＝score1＋1，模拟”到达状态“P1发球”是一个循环，我们称为循环①。如果在状态图中迁移 $t3$误写为：“P1赢球[score1＜20]/score1＝1，模拟”，当循环只执行一次时，这种错误也不能被发现。

由此看来对于含有循环的问题，测试一次是很不充分的，测试时执行所有次数的循环显然又是不现实的，特别是无限次的循环将是不可能的。事实上，分析一个比较简单的循环问题时，把循环执行2至3次以后，这个循环的基本规律也就把握了。根据不完全归纳法的思想，对于含有循环的测试，我们提出使循环分别执行0次、1次、2次。若通过了这样的测试，则循环的可靠性会明显增强。当一个循环被执行0次时，就相当于这个循环的初始状态，当然在有的软件规格说明中一个循环至少要执行1次，那么这个循环执行0次就是一个非法的路径，在测试时仍然应该测试。

在状态图1中，假如有一个非法迁移的条件是：“[score1＝1]”，无事件、无动作，源状态是“P2发球”，目标状态是“P1赢”。循环①执行1次到达状态“P2发球”，此时就有 score1＝1，就可以测试这个非法迁移。循环①执行2次到达状态“P2发球”，此时就有 score1＝2，就不能测试上面这个非法迁移。如果不允许循环①仅执行1次的话就不会有这种测试机会了。当一个循环分别执行1次和2次时，这个循环在被测实现中的作用通常是不同的，所以一个循环执行2次还是有必要让它执行1次。

ZOT 循环覆盖准则：测试用例集 T 应该包含 UML 状态图中的每一个循环分别执行0次、1次、2次的测试。

独立路径是指包括一组以前没有访问的状态和迁移的一条路径[10]。在执行一个循环时，循环体内的各条独立路径都应该执行。当一个循环体中有 n 条独立路径时，则针对于这个循环的 ZOT 循环覆盖准则所应包含的路径（我们称为 ZOT 路径）共有多少条呢？看图3。

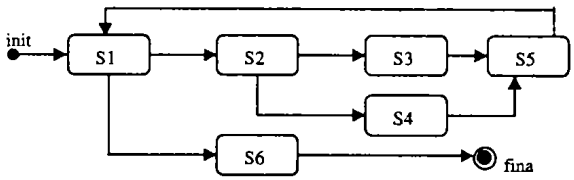

图3 ZOT 循环覆盖

图中的路径：$S1$→$S2$→$S3$（或者 $S4$）→$S5$→$S1$是一个循环，这个循环体中有2条独立路径。

当循环执行0次也就是这个循环体不执行时，有一条路径：(1)$init$→$S1$→$S6$。

当循环执行1次时，有两条路径：(2)$init$→$S1$→$S2$→$S3$→$S5$→$S1$；(3)$init$→$S1$→$S2$→$S4$→$S5$→$S1$。

当循环执行2次时，因为循环体内有2条路径，所以共有四条路径：(4)$init$→$S1$→$S2$→$S3$→$S5$→$S1$→$S2$→$S3$→$S5$→$S1$；(5)$init$→$S1$→$S2$→$S3$→$S5$→$S1$→$S2$→$S4$→$S5$→$S1$；(6)$init$→$S1$→$S2$→$S4$→$S5$→$S1$→$S2$→$S4$→$S5$→$S1$；(7)$init$→$S1$→$S2$→$S4$→$S5$→$S1$→$S2$→$S3$→$S5$→$S1$。

由上面的分析可以看出，当循环体中有2条独立路径时，针对于这个循环的 ZOT 路径共有：$1+2+2^2$条。当循环体中有 n 条独立路径时，针对于这个循环的 ZOT 路径共有：$1+n+n^2$条。

假如把上面的一个循环换成一个双层的循环，如果内层循环中的独立路径数是 n，那么针对于这个双层循环的 ZOT 路径共有：$1+(1+n+n^2)+(1+n+n^2)^2$条。也就是它的 ZOT 路径复杂度是 $O(n^4)$。

3.6 全 ZOT 路径覆盖准则

全路径覆盖就是要有足够的测试用例，覆盖状态图中所有可能的路径。在实际问题中，一个不太复杂的状态图，而全

部的路径可能是庞大的或者是无限的。要在测试中覆盖这样多的路径是不现实的。为解决这样的难题，只得把覆盖的路径数压缩到一定限度内，例如，状态图中的循环分别只执行0次、1次、2次。本节所介绍的覆盖准则就是包含 ZOT 循环覆盖准则的全路径覆盖准则。

全 ZOT 路径覆盖准则：当状态图中的每一个循环分别执行0次、1次、2次，而循环外的独立路径要求至少被访问一次时，测试用例集 T 应该包含 UML 状态图中从初始状态开始到终止状态结束的全部路径的测试。

对于一个状态图的全 ZOT 路径，可以把它简单地分为两部分：一部分是针对循环的 ZOT 路径，另一部分是针对循环外的独立路径。在执行一个循环时，循环体内的各条独立路径都应该执行；而循环外的独立路径要求至少被访问一次。如果在一个状态图中，针对循环的 ZOT 路径有 m 条，而针对循环外的独立路径有 k 条，则这个状态图的全 ZOT 路径有 MAX[m，k]条，即 m、k 的最大数。

下面以图3为例具体说明全 ZOT 路径：在3.5节中已经给出了针对循环的 ZOT 路径共有7条，从图3中可以看出，针对循环外的独立路径有2条，因而图3的全 ZOT 路径条数是 MAX[7，2]=7。这7条全 ZOT 路径可以是：

(1)*init*→*S*1→*S*6→*fina*；(2)*init*→*S*1→*S*2→*S*3→*S*5→*S*1→*S*7→*fina*；(3)*init*→*S*1→*S*2→*S*4→*S*5→*S*1→*S*6→*fina*；(4)*init*→*S*1→*S*2→*S*3→*S*5→*S*1→*S*2→*S*3→*S*5→*S*1→*S*7→*fina*；(5)*init*→*S*1→*S*2→*S*3→*S*5→*S*1→*S*2→*S*4→*S*5→*S*1→*S*6→*fina*；(6)*init*→*S*1→*S*2→*S*4→*S*5→*S*1→*S*2→*S*4→*S*5→*S*1→*S*7→*fina*；(7)*init*→*S*1→*S*2→*S*4→*S*5→*S*1→*S*2→*S*3→*S*5→*S*1→*S*6→*fina*。

对于上面的图3，全 ZOT 路径是7条，而如果循环次数是50次的话，那么它的全路径有 n^{50} 条，所以要全部测试它几乎是不可能的，而要测试所有的全 ZOT 路径是完全可以做到的。

4 测试准则的应用

基于 UML 状态图的测试中，上面共描述了6条测试准则，那么在什么情况下使用那些准则呢？在基于状态的测试方法中，状态覆盖准则和迁移覆盖准则都必须满足，否则对状态图的测试将极不完善。在含有循环的状态图中，ZOT 循环覆盖准则是必须满足的。对于另外的三个测试准则（全谓词公式覆盖准则、迁移对覆盖准则、全 ZOT 路径覆盖准则）来说，是全部的选择或者是部分的选择，可根据被测软件可靠性、安全性等软件利益要求的高低和软件测试的成本来权衡。

怎样具体地使用这些测试准则来产生测试用例呢？下面以状态图1为例，分别应用每条测试准则，给出测试路径，使它们所产生的测试用例集能满足相应的测试准则。

4.1 状态覆盖准则

对于上面的图1，在应用状态覆盖准则时，使每一个状态至少被访问一次，有下面的两条测试路径就能满足：(1)*init*→游戏开始→P1发球…→P1发球→P1赢→*fina*；(2) *init*→游戏开始→P2发球…→P2发球→P2赢→*fina*。在第(1)条测试路径中的"…"意味着从状态"P1发球"到状态"P1发球"经过迁移 $t3$ 通常需要循环20次（在另一测试路径中类同）。若这两条测试路径所产生的测试用例分别是 $tc1$ 和 $tc2$，则测试用例集 $T1=\{tc1,tc2\}$ 就能确保状态图1满足状态覆盖准则。

4.2 迁移覆盖准则

测试用例集 $T1$ 就不能使状态图1满足迁移覆盖准则，因为在这个测试用例集中不可能激活迁移 $t5$"P1赢球/模拟"和迁移 $t6$"P2赢球/模拟"。对于上面的图1，在应用迁移覆盖准则时，有下面的两条测试路径就能满足：(3) *init*→游戏开始→P1发球→P2发球…→P2发球→P2赢→*fina*；(4) *init*→游戏开始→P2发球→P1发球…→P1发球→P1赢→*fina*。在第(3)条测试路径中的"…"意味着从状态"P2发球"到状态"P2发球"经过迁移 $t4$ 通常需要循环20次（在另一测试路径中类同）。若这两条测试路径所产生的测试用例分别是 $tc3$ 和 $tc4$，则测试用例集 $T2=\{tc3,tc4\}$ 就能确保状态图1满足迁移覆盖准则。显然，测试用例集 $T2$ 也能确保状态图1满足状态覆盖准则。

4.3 全谓词公式覆盖准则

测试用例集 $T2$ 不能使状态图1满足全谓词公式覆盖准则，因为当对象处于状态"P1发球"时，测试用例集 $T2$ 中不可能激活非法迁移"P1赢球[score1＞20]"。在测试用例集 $T2$ 的基础上再增加两个测试用例，使之能满足全谓词公式覆盖准则。对于上面的图1，设计两条特殊的测试路径：(5) init→游戏开始→P1发球→"P1赢球[score1＞20]"→；(6) init→游戏开始→P2发球→"P2赢球[score2＞20]"→。在上面的两个测试路径中，各有一个非法迁移，这两个非法迁移的期望结果是产生动作"非法事件异常"。两个非法迁移的实现还需强制满足非法迁移的前置条件。若这两条测试路径所产生的测试用例分别是 $tc5$ 和 $tc6$，则测试用例集 $T3=\{tc3,tc4,tc5,tc6\}$ 就能确保状态图1满足全谓词公式覆盖准则。显然，测试用例集 $T3$ 也能确保状态图1满足迁移覆盖准则，更能确保状态图1满足状态覆盖准则。

4.4 迁移对覆盖准则

下面考虑图1中的状态"P1发球"，运用迁移对覆盖准则时，可以参看图4。在测试用例集中应该包含下面的9组迁移对：(1)$t1$和$t3$；(2)$t1$和$t6$；(3)$t1$和$t7$；(4)$t3$和$t3$；(5)$t3$和$t6$；(6)$t3$和$t7$；(7)$t5$和$t3$；(8)$t5$和$t6$；(9)$t5$和$t7$。因为迁移 $t3$ 的源状态和目标状态都是"$P1$发球"，所以图4中状态"$P1$发球"不仅到达的迁移中有 $t3$ 而且出发的迁移中也有 $t3$。根据规格说明可知：在上述的9组迁移对中，第(3)组迁移对"$t1$和$t7$"是非法迁移对。对于这样的迁移对也要进行测试，要确保被测实现中不含这样的非法迁移对。

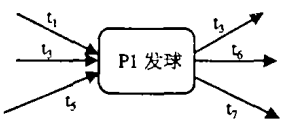

图4 图1中状态"P1发球"的迁移对覆盖

分析状态图1中的状态"$P2$发球"，也有9组迁移对：(10)$t2$和$t4$；(11)$t2$和$t5$；(12)$t2$和$t8$；(13)$t4$和$t4$；(14)$t4$和$t5$；(15)$t4$和$t8$；(16)$t6$和$t4$；(17)$t6$和$t5$；(18)$t6$和$t8$。状态图1中除了上面8组迁移对外，还有下面4组迁移对：(19)t_{init}和$t1$；(20)t_{init}和$t2$；(21)$t7$和t_{fina}；(22)$t8$和t_{fina}。

对于上面的图1，在应用迁移对覆盖准则时，有下面的6条测试路径就能覆盖上面的22组迁移对：(7) *init*→游戏开始→P1发球→P1赢；(8) *init*→游戏开始→P2发球→P2赢；(9) *init*→游戏开始→P1发球→P2发球→P1发球→P1发球→P2发球…→P2发球→P1发球→P2发球→P2赢→*fina*；(10) *init*→游

戏开始→P2发球→P1发球→P2发球→P2发球→P1发球…→P1发球→P2发球→P1发球→P1赢→*fina*;(11) *init*→游戏开始→P1发球…→P1发球→P1赢→*fina*;(12) *init*→游戏开始→P2发球…→P2发球→P2赢→*fina*。在这几条测试路径中的"…"都意味着要循环20次。若这6条测试路径所产生的测试用例分别是*tc*7、*tc*8、*tc*9、*tc*10、*tc*11和*tc*12,则测试用例集$T4=\{tc7, tc8, tc9, tc10, tc11, tc12\}$就能确保状态图1满足迁移对覆盖准则。显然,测试用例集$T4$也能确保状态图1满足状态覆盖准则和迁移覆盖准则。

4.5 ZOT循环覆盖准则

对于状态图1中的循环①:从状态"P1发球"经迁移t3到状态"P1发球",这里有三条测试路径:(13) *init*→游戏开始→P1发球→P2发球;(14) *init*→游戏开始→P1发球→P1发球→P2发球;(15) *init*→游戏开始→P1发球→P1发球→P1发球→P2发球。这三条路径分别使这个循环执行了0次、1次、2次。

对于状态图1中的循环②:从状态"P2发球"经迁移t4到状态"P2发球",这里有三条测试路径:(16) *init*→游戏开始→P2发球→P1发球;(17) *init*→游戏开始→P2发球→P2发球→P1发球;(18) *init*→游戏开始→P2发球→P2发球→P2发球→P1发球。这三条路径分别使这个循环执行了0次、1次、2次。

对于状态图1中的循环③:从状态"P1发球"经迁移t6到状态"P2发球",再经迁移t5到状态"P1发球",这里有三条测试路径:(19) *init*→游戏开始→P1发球→P1发球;(20) *init*→游戏开始→P1发球→P2发球→P1发球;(21) *init*→游戏开始→P1发球→P2发球→P1发球→P2发球→P1发球。这三条路径分别使这个循环执行了0次、1次、2次。

在上面的9条测试路径中,第19条路径完全包含在第14条路径中,这样三个循环的ZOT路径共有8条。若这8条测试路径所产生的测试用例分别是*tc*13、*tc*14、*tc*15、*tc*16、*tc*17、*tc*18、*tc*20和*tc*21,则测试用例集$T5=\{tc13, tc14, tc15, tc16, tc17, tc18, tc20, tc21\}$就能确保状态图1满足ZOT循环覆盖准则。

4.6 全ZOT路径覆盖准则

对于上面的状态图1,三个循环的ZOT路径共有8条,而循环外的独立路径有2条,则全ZOT路径覆盖共有8条,分别是:(22) *init*→游戏开始→P1发球→P2发球…→P2发球→P2赢→*fina*;(23) *init*→游戏开始→P1发球→P1发球→P2发球…→P2发球→P2赢→*fina*;(24) *init*→游戏开始→P1发球→P1发球→P1发球→P2发球…→P2发球→P2赢→*fina*;(25) *init*→游戏开始→P2发球→P1发球…→P1发球→P1赢→*fina*;(26) *init*→游戏开始→P2发球→P2发球→P1发球…→P1发球→P1赢→*fina*;(27) *init*→游戏开始→P2发球→P2发球→P2发球→P1发球…→P1发球→P1赢→*fina*;(28) *init*→游戏开始→P1发球→P2发球→P1发球…→P1发球→P1赢→*fina*;(29) *init*→游戏开始→P1发球→P2发球→P1发球→P2发球→P1发球…→P1发球→P1赢→*fina*。在这8条测试路径中的"…"意味着从状态"P1发球"到状态"P1发球"经过迁移*t*3通常需要循环20次,或者是从状态"P2发球"到状态"P2发球"经过迁移*t*4通常需要循环20次。

若以上8条测试路径所产生的测试用例分别是*tc*22、*tc*23、*tc*24、*tc*25、*tc*26、*tc*27、*tc*28和*tc*29,则测试用例集$T6=\{tc22, tc23, tc24, tc25, tc26, tc27, tc28, tc29\}$就能确保状态图1满足全ZOT路径覆盖准则。显然,测试用例集$T6$也能确保状态图1满足状态覆盖准则、迁移覆盖准则和循环覆盖准则。

5 测试准则之间的包含关系

当满足第一个测试准则的任何测试用例集都满足第二个测试准则时,我们称第一个测试准则包含第二个测试准则。很显然测试准则之间的包含关系具有传递性,即如果第一个测试准则包含第二个测试准则,第二个测试准则又包含第三个测试准则,那么第一个测试准则就包含第三个测试准则。

满足迁移覆盖准则的测试用例集$T2$也能确保状态图1满足状态覆盖准则。实际上满足迁移覆盖准则的任何测试用例集都会满足状态覆盖准则。满足状态覆盖准则的测试用例集$T1$不能满足其它5条覆盖准则,所以状态覆盖准则不包含任何其它的测试覆盖准则。测试用例集$T2$除了能满足状态覆盖准则以外,不能满足其它的4条覆盖准则,也就是说迁移覆盖准则除了包含状态覆盖准则以外,不包含其它的4条覆盖准则。

满足全谓词公式覆盖准则的测试用例集$T3$也能确保状态图1满足迁移覆盖准则。实际上满足全谓词公式覆盖准则的任何测试用例集都会满足迁移覆盖准则。所以全谓词公式覆盖准则包含迁移覆盖准则。由测试准则包含关系的传递性,全谓词公式覆盖准则包含状态覆盖准则。测试用例集$T3$不能满足迁移对覆盖准则、ZOT循环覆盖准则和全ZOT路径覆盖准则,因而全谓词公式覆盖准则只包含状态覆盖准则和迁移覆盖准则,不包含其它的3条覆盖准则。

同样的道理,迁移对覆盖准则只包含状态覆盖准则和迁移覆盖准则,不包含其它的3条覆盖准则。

满足ZOT循环覆盖准则的测试用例集$T5$不能满足其它的5条覆盖准则,因而ZOT循环覆盖准则不包含任何其它的覆盖准则。

很明显,全ZOT路径覆盖准则包含状态覆盖准则、迁移覆盖准则和ZOT循环覆盖准则,它不包含全谓词公式覆盖准则和迁移对覆盖准则。

图5画出了这些准则之间的包含关系。

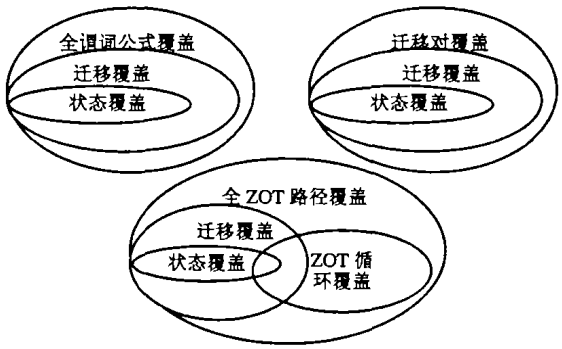

图5 覆盖准则之间的包含关系

结论与未来的工作 在测试时,循环只要求执行一次,显然是很不充分的,而循环的所有次数都要执行往往很难或者不可能实现。为了解决这个问题,本文提出了测试时使循环分别执行0次、1次、2次的覆盖准则,即ZOT循环覆盖准则。ZOT循环覆盖准则要求循环的执行不仅可以实现,而且又比原来循环只执行一次的测试充分得多。在此基础上本文还提出了全ZOT路径覆盖准则。全ZOT路径覆盖准则是具有可操作性的,而全路径覆盖准则是没有可操作性的。本文系统地描述了基于UML状态图生成测试用例的6条充分性准则。本文通

过一个 UML 状态图的实例，给出了本文所描述的6条测试准则的具体应用。最后还讨论了这6条测试准则之间的包含关系。

众所周知，数据流覆盖主要是用在基于程序的测试中。在未来，希望能把数据流覆盖也应用到基于状态图的测试中。这样就能使基于状态图的测试准则更加充分。

参 考 文 献

1 Zhu H, Hall P AV, May J HR. Software Unit Test Coverage and Adequacy. ACM Computing Surveys, 1997, 29(4): 366～427

2 Haworth B, Kirsopp C, Roper M, et al. Towards the development of adequacy criteria for object-oriented systems. In: Proc. of the 5th European Conf. on Software Testing Analysis and Review, Edinburgh, Scotland, Nov. 1997. 417～427

3 Haworth B. Adequacy criteria for object testing. In: Proc. of the 2nd Intl. Software Quality Week Europe 1998, Brussels, Belgium, Nov. 1998

4 Offutt A J, Xiong Y, Liu S. Criteria for Generating Specification-Based Tests. In: Proc. of 5th IEEE Intl. Conf. on Engineering of Complex Computer Systems (ICECCS'99), Las Vegas, Nevada, USA, Oct. 1999. 119～129

5 Offutt A J, Abdurazik A. Generating tests from UML specifications. In: Proc. of the Second IEEE Intl. Conf. on the Unified Modeling Language (UML99), Fort Collins, CO, IEEE Computer Society Press, 1999. 416～429

6 Abdurazik A, et al. Evaluation of Three Specification-based Testing Criteria. In: Sixth IEEE Intl. Conf. on Engineering of Complex Computer Systems (ICECCS '00), Tokyo, Japan, Sep. 2000

7 Ammann P E, Black P E. A Specification-Based Coverage Metric to Evaluate Test Sets. International Journal of Reliability, Quality and Safety Engineering, World Scientific Publishing, Singapore, 2001, 8(4): 275～300

8 Binder R V. Testing Object-Oriented System: Models, Patterns, and Tools. Boston: Addison Wesley Longman, Inc. 2000

9 Kim Y G, Hong H S, Bae D H, Cha S D. Test Cases Generation from UML State Diagram, IEEE Proceedings - Software, 1999, 146(4): 187～192

10 郑人杰，殷人昆，陶永雷. 实用软件工程. 北京：清华大学出版社(第二版)，1997

(上接第208页)

方法将由无效状态变为关闭状态，释放资源。此时，该连接句柄既不能被再次使用也不与任何实际的物理连接相关。

4.2 连接事件通知机制及其扩展

在 JCA 规范中，应用服务器通过连接事件通知机制进行连接及事务管理，其中定义了五种类型的连接事件，包括：CONNECTION_CLOSED、CONNECTION_ERROR_OCCURRED、LOCAL_TRANSACTION_STARTED、LOCAL_TRANSACTION_COMMITED 和 LOCAL_TRANSACTION_ROLLEDBACK。

应用服务器为每个物理连接对象注册一个连接事件监听器(ConnectionEventListener)，每当监听到连接事件发生时就根据所发生的事件进行相应处理。例如，如果应用组件关闭了某一连接句柄，这一事件以资源适配器所定义的方式传递给它所关联的物理连接对象，该对象使用监听器的 connectionClosed 方法通知所有注册的应用组件，并传递一个 CONNECTION_CLOSED 类型的连接事件给应用服务器。应用服务器收到这一事件后判断是否需要将该连接返回连接池并调用事务管理器将代表该物理连接对象的资源从事务管理器中注销。同样，本地事务也是通过向应用服务器发送事件通知来告知该事务已经开始、提交或是回滚。

在商业应用中，经常有一些包含在全局事务中的应用组件调用了无需在这个特定的事务作用域中使用的连接，或者应用组件并不会事务性地使用到这些连接。在这些情况下，根据 JCA 规范的要求在进入事务作用域时主动获得这些连接的资源可能是没有必要的，而且如果一个事务中只用到了一个单独的连接却占用了多个连接，则可能阻止该事务的单阶段提交优化。因此，希望这些连接只有在需要它们的时候才被征用。我们利用连接事件通知机制的思想对实际应用中的事务管理进行了优化。通过增加一个连接延期(CONNECTION_PENDING)事件通知可以做到这一点，资源适配器可以用 CONNECTION_PENDING 事件来通知连接管理器何时要使用 ManagedConnection 以及何时需要在一个全局事务中使用它。在 OnceAS 中我们对 JCA 规范的标准接口 ConnectionEventListener 进行了扩展，如图5所示。

```
public interface javax.resource.spi.ConnectionEventListener {
  public void connectionClosed(ConnectionEvent event);
  public void connectionErrorOccurred(ConnectionEvent event);

  // Local Transaction Management related events
  public void localTransactionStarted(ConnectionEvent event);
  public void localTransactionCommitted(ConnectionEvent event);
  public void localTransactionRolledback(ConnectionEvent event);
}
public interface ConnectionListener extends javax.resource.spi.ConnectionEventListener{
  public static final int CONNECTION_PENDING;
  public abstract void connectionPending(ConnectionEvent) throws ResourceException;
}
```

图5 使用连接延期事件扩展接口

同时，通过在资源适配器中将事务资源设置为动态注册，使得应用服务器可以监视交互挂起事件，这样就可以使得物理连接对象只有在必要的时候才在全局事务作用域中被使用，从而提高了系统性能。

结论 本文研究了 J2EE 应用服务器使用 JCA 进行资源集成的性能问题及其优化。我们对连接池提出了应用程序连接请求队列，并使用 Reactor 模式对连接请求进行了优化，分析了基于 Reactor 模式的连接池对系统性能的提高。我们对事务管理的规范进行了扩展，提高了系统的灵活性及资源利用效率。本文提到的方法已应用到应用服务器 OnceAS 中，取得了很好的效果。

参 考 文 献

1 Sun Microsystems Inc. Java 2 Platform, Enterprise Edition Connector Architecture Specification. http://java.sun.com.

2 Sun Microsystems Inc. Java 2 Platform Enterprise Edition Specification. Version 1.3.2001.7

3 中科院软件所软件工程技术中心. OnceAS 1.0应用服务器技术白皮书. 2004.5

4 Bernstein P, Newcomer E. Principles of Transaction Processing. Morgan Kaufmann Publishers, 1997

5 Schmidt D C. Experience Using Design Patterns to Develop Reuseable Object-Oriented Communication Software. Communications of the ACM, 1995, 38(10): 65～74

6 Sun Microsystems Inc. Java Naming and Directory Interface Application Programming Interface. 1999