基于EDPN的类测试框架及测试用例生成技术的研究*)

龚红仿[1] 李军义[1] 邹北骥[1] 孙家广[2]

(湖南大学计算机与通信学院 长沙410082)[1] (清华大学信息科学技术学院 北京100084)[2]

摘 要 针对基于状态的面向对象软件的类测试过程中存在的不可预测、不可达状态、状态组合"爆炸"和测试用例"爆炸"等问题,提出了基于EDPN模型的类测试、类的交互测试和类的层次测试框架,设计了相应的测试模型;提出了基于EDPN的有标记的唯一输入输出(UIO)测试用例的自动生成方法,生成状态转移路径序列,测试类的状态;提出了基于扩展的EDPN的状态组合的标记关联递推法,生成扩展的正交阵列表,测试类的交互;提出了基于扩展的EMDPN的协同路径(copaths)测试用例的生成方法,生成协同路径序列,测试类的层次。

关键词 类测试模型,状态测试,交互测试,层次测试,测试用例生成,事件驱动Petri网

Research on a Testing Framework and Technology of Class Test Cases Generation Based on EDPN

GONG Hong-Fang[1] LI Jun-Yi[1] ZOU Bei-Ji[1] SUN Jia-Guang[2]

(College of Computer & Comm., Hunan Univ., Changsha 410082)[1] (College of Info. Science & Tech., Tsinghua Univ., Beijing 100084)[2]

Abstract In the technology of object-oriented software classes testing based-on statement, there are some questions such as uncertain, unreachable statement, combinatorial state explosion and test cases explosion. Aimming at the questions, a classes testing framework is proposed based-on EDPN, and the framework includes classes test, classes interaction test and classes hiberarchy test. It proposed a classes testing cases generation method of UIO based-on EDPN for generating statement transfer sequence, and a classes interaction testing cases generation method named Marked Associate Recursive of extended EDPN for generating Extended Orthogonal Array Matrix, and a classes hiberarchy testing cases generation method with cooperated paths (copaths) based-on extended EMDPN for generating copaths sequence.

Keywords Class testing models, Statement testing, Interaction testing, Hiberarchy testing, Test cases technology, Event-driven petri network(EDPN)

1 引言

软件测试是保证软件质量,提高软件可靠性的关键,其目的是以最少的时间和人力找出软件中潜伏的各种错误和缺陷[1]。Myers在文[2]中指出软件测试所花费的代价大约占软件开发总代价的50%以上。面向对象的软件测试是面向对象软件开发的不可缺少的一环,各种软件测试技术也层出不穷。面向对象软件的封装性、继承性、多态性和动态绑定机制使得面向对象的软件测试更加复杂,传统的软件测试技术已不再完全适用[3~6]。目前,面向对象的软件测试技术中应用最为广泛之一的就是以扩展的有限状态机(EFSM)为基础的面向对象系统的基于状态的测试。Doong,Turner和Kung分别在文[7~9]中描述了基于状态的面向对象软件的类测试技术,提出面向对象软件测试的基本工作就是类的实例化以及类之间的消息交互,一个对象向另一对象发送一系列消息然后检查结果对象的状态,看其是否处于正确的状态。Bourhfir和TorX分别在文[10,11]中系统地论述了基于EFSM测试的理论基础和测试方法。但是利用EFSM测试面向对象的类时遇到了不可预测、不可达状态、状态组合"爆炸"和测试用例"爆炸"等问题。Jorgensen在文[12]中提出了事件驱动的Petri网(Event-Driven Petri Network,EDPN)模型,并将该模型运用于面向对象软件集成测试和系统测试中。我们发现将EDPN及其扩展的模型运用于类测试的过程,可以解决以上问题。

本文提出了基于EDPN模型的面向对象软件的类测试、类的交互测试和类的层次测试框架,设计了相应的测试模型;提出了基于EDPN的有标记的唯一输入输出(UIO)测试用例的自动生成方法,生成状态转移路径序列,测试类的状态;提出了基于扩展的EDPN(Extended EDPN,EEDPN)的状态组合的标记关联递推法(Marked Associate Recursive Method,MARM),生成扩展的正交阵列表,测试类的交互;提出了基于扩展的EMDPN的协同路径(copaths)测试用例的生成方法,生成协同路径序列,测试类的层次。这些方法能较好地解决EFSM在面向对象软件测试中存在的问题。

2 类的测试框架及测试模型

McGregor在文[13]中分别从类测试基础、类的交互测试和类的层次测试等方面讨论了类的测试问题,但没有提出类

*)本项研究得到国家863基础研究类项目(项目编号:2002AA411510)和国家自然科学基金(项目编号60474070)资助。龚红仿 硕士研究生,讲师,研究方向:软件测试技术、计算机应用技术。李军义 讲师,博士研究生,研究方向:软件工程、计算机网络。邹北骥 教授,主要研究领域:计算机图形学、图像处理与多媒体技术、软件工程。孙家广 中国工程院院士,清华大学教授,主要研究领域:计算机图形学,CAD技术,软件工程学。

的测试框架的概念。面向对象软件从宏观上看是类与类之间的相互作用。在面向对象系统中，系统的基本构造模块是封装了的数据和方法的类和对象，而不再是一个个能完成特定功能的功能模块。每个对象有自己的生存周期，有自己的状态。消息是对象之间相互请求或协作的途径，是外界使用对象方法及获取对象状态的唯一方式。对象的功能是在消息的触发下，由对象所属类中定义的方法与相关对象的合作共同完成，且在不同状态下对消息的响应可能完全不同，对象的状态可能被改变，产生新的状态。对象中的数据和方法是一个有机的整体，测试过程中不能仅仅检查输入数据产生的输出结果是否与预期的吻合，还要考虑对象的状态。经验表明类测试是必需的，是发现错误的最重要手段。因此，我们提出类的测试框架与测试模型如图1所示。

(1)基于 EDPN 的类测试：通过唯一输入输出测试类的状态及状态转移；

(2)基于扩展的 EDPN 的类的交互测试：通过优化正交阵列测试系统测试类的交互；

(3)基于扩展的 EMDPN 的类的层次测试：通过生成协同路径方法测试类的层次。

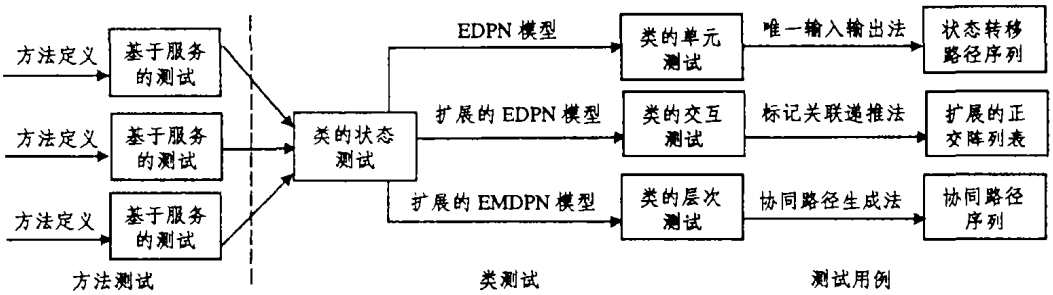

图1　类的测试框架及测试模型

3　基于 EDPN 的类测试

3.1　生成测试用例的 UIO 方法

面向对象软件测试技术中，EDPN 模型的应用还相对较为薄弱。EDPN 模型是对传统的 Petri 网的改进，修改了使 Petri 网能够更接近地表示事件驱动的系统，解决了表示事件静止的 Petri 网标记问题。

定义1(EDPN)　是一种多向图(P,D,S,In,Out)，包括三个节点集合 P、D 和 S，以及两个输入输出函数集合 In 和 Out。其中：P 是端口事件的集合，可以理解为发送的消息调用和行为的输入输出；D 是数据地点的集合，可以理解为 EFSM 模型中的状态；S 是转移的集合，可以解释为行为；In 是$(P\cup D)\times S$ 的有序对偶集合，Out 是 $S\times(P\cup D)$的有序对偶集合，In 和 Out 定义的 S 中的转移输入输出到端口事件地点 P 和数据地点 D。S 中的若干转移构成转移序列，在测试过程中，总能够通过转移的输入和输出，构造输入和输出测试序列。在运用 EDPN 模型生成测试用例的过程中，必须对 EDPN 做标记，以便能够有效地处理传统 Petri 网中的事件静止问题[12]。

在进行类测试时，需要考虑类测试系列的充分性。基于 EDPN 的状态覆盖率是类测试充分性的标准之一，能提高测试人员对测试质量的信任度。然而，穷举测试用例是不可能的，只能从无限的测试用例中选出有限的能达到覆盖要求的测试用例。基于 EDPN 的有标记的唯一输入输出方法就是这种方法，所产生的测试用例就是一系列的标记构成的标记序列，其步骤可表述为：(1)为每个数据地点设计一个输入元素 $d_i\in D, d_i<>d_0$(d_0为初始标记)，作为一个唯一输入输出 $MUIOS_i$ 的消息预测序列；(2)为每一个端口事件地点设计一组唯一的事件输入输出序列 $EUIOS_i$ 作为测试消息序列；(3)按标记的个体项连接状态序列 $MUIOS_i$ 和 $EUIOS_i$，即把 $MUIOS_i$ 序列置前，$EUIOS_i$ 序列置后，以它们中的元素为列构成列矩阵，该矩阵的每一行就是一个标记，行向量组就是 EDPN 的标记序列，即所生成的测试用例。用这种方法产生的测试用例同时包括预测，也就是运行消息预测序列、事件输入序列、事件输出序列以验证被测对象是否达到预期的效果。

3.2　UML 状态图与 EDPN

UML 状态图利用“超态”引进了文氏图描述层次结构的能力以及其中的并发状态图解决了并行程序的执行问题。所谓“超态”又称为组合状态，组合状态包含有其它状态或子状态。组合状态可划分为“与状态”(and-state)和“或状态”(or-state)。与状态表示其子状态可同时发生的组合状态，也称为正交状态或者并发状态；或状态表示其子状态相互排斥的状态[14~16](如图1)。

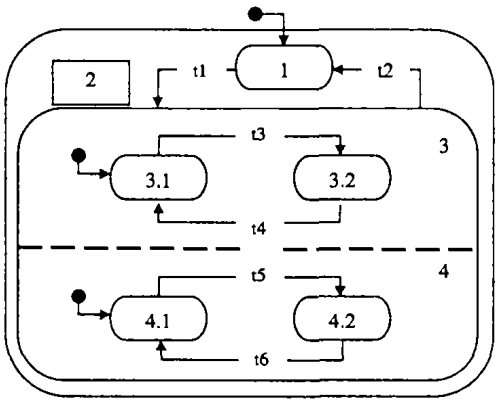

图2　状态图

EDPN 利用其较好的耦合性和基于标记的事件驱动执行方式，可以很方便地运用到类测试技术中。只需要将 UML 状态图转换成对应的 EDPN 图，就可以运用 UIO 方法生成测试用例。

3.3　用 EDPN 生成测试用例

(1)应用 EDPN 对类进行状态测试，就必须将 UML 状态图转换成 EDPN 图(图3)。转换时，将状态图的“团点”变成数据地点，用圆圈表示；状态转移变成 EDPN 转移，用小矩形表示；引起状态转移的事件成为端口输入事件地点，用三角形表示；与状态转移关联的输出变成端口输出事件地点，也用三角形表示。为了阐述问题的方便，图中没有给出端口输出事件。

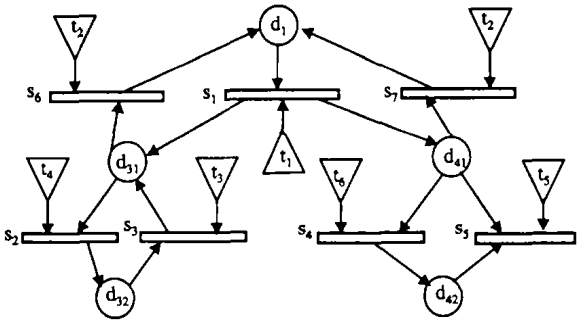

图3 图2的 EDPN 图

(2)设计样本场景标记。首先要对照测试的类，列表描述 EDPN 中的元素，如基于事件驱动的转移、事件静止状态、端口输入输出的事件等。然后描述样本场景标记，以生成测试用例。假设以上示例就是要作标记的样本，根据状态图，状态1是初始状态，所以先对 d_1 初始化，即作初始标记。表1中设计了样本场景的标记。

表1 样本场景的标记

步骤	d_1	d_{31}	d_{32}	d_{41}	d_{42}	t_1	t_2	t_3	t_4	t_5	t_6	描述
m0	1											没有允许发生的转移
m1	1					1						s_1 允许发生，s_1 被触发
m2		1		1								没有允许发生的转移
m3		1		1					1		1	s_2，s_4 允许发生，s_2，s_4 被触发
m4			1		1							没有允许发生的转移
m5			1		1			1		1		s_3，s_5 允许发生，s_3，s_5 被触发
m6		1		1								没有允许发生的转移
m7		1		1			1					s_6，s_7 允许发生，s_6，s_7 被触发
m8	1											回到初始状态

(3)运行 EDPN。运行 EDPN 必须遵循：只有当一个转移的所有输入地点中至少有一个记号(上表中用"1"表示)，该转移才允许发生；当触发一个转移时，要从该转移的每个输入地点分别删除一个记号，且在其每个输出地点增加一个记号。执行时，d_1 被初始化(步骤 m_0)，网络处于事件静止状态，如果事件 t_1 发生(步骤 m_1)，s_1 允许发生，s_1 被触发，状态被转移到地点 d_{31} 和 d_{41}，如果有输出端口事件地点，则产生相应的输出。此时，d_{31} 和 d_{41} 均增加一个记号，同时 d_1 和 t_1 中的记号被删除。由于在 UML 状态图中，2是两个并发状态3和4，因而图2中3.1和4.1同时被激活。这样继续执行，最后再回到初始状态 d_1。

(4)自动生成测试用例。按照上述过程，可以自动生成测试用例。用伪代码描述的算法如下：

```
Generation_Testing_Case (DataSite curDataSite){
  //S' 初始化为空集，存放被触发过的转移序列
  if S'=S then all s∈S are triggered and return;
  for each di∈D {
    if(di=curDataSite) then
      if(ti∈P is not marked)then //ti 与 di 相关的端口输入事件
        add a mark to ti;
      S'=S'∪{si};//与 di 相关的转移 si 被触发
      make di' as an output destination DataSite of si;
      delete the marks of di and ti;}
  for each destination DataSite di'
    Generation_Testing_case(di');
}
```

4 基于扩展的 EDPN 的类的交互测试

4.1 扩展的 EDPN 模型(EEDPN)

正交阵列测试系统[13](Orthogonal Array Testing System，OATS)是一种特殊的抽样方法，该方法通过定义一组交互对象的配对方式的组合，以尽力限制测试配置的组合数目的激增。下面通过一个一般例子(图4)来扩展 OATS，优化 OATS 方法。该例子由类族 A 中的发送者、类族 C 中的接收者和类族 P 之间的交互所构成，而且每个类都有一个与它相关联的状态转换图，其中的实现细节并不重要。

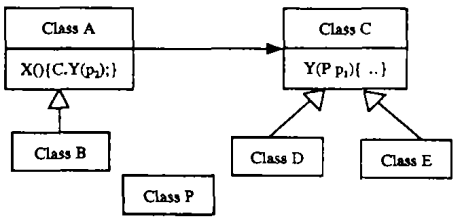

图4 应用了 OATS 的一个一般例子

我们对 EDPN 模型进行扩展，给出如下定义：

定义2(EEDPN) 是一种多向图($EDPN$、E)，其中 $EDPN$ 独立的描述每个类族，E 是连接若干个 $EDPN$ 的有向边的集合，且满足：如果有向边 $e=(u,v)\in E$，那么 $u,v\in P\cup D\cup S$。这里 P、D 和 S 是 $EDPN$ 中的节点集合。

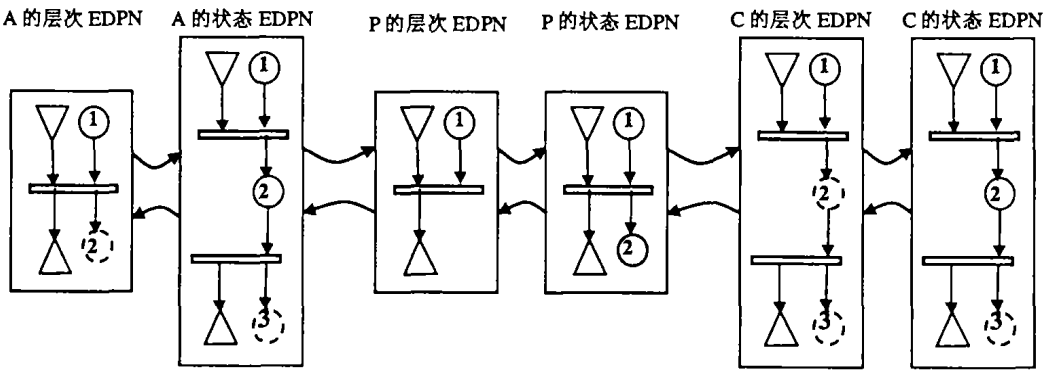

图5 图4的一般例子的 EEDPN 示意图

图中类族 A 两个层次，其中类 A 有2个状态，类 B 有3个状态；类族 P 一个层次，2个状态；类族 C 三个成员，其中类 C

有2个状态，类 D、类 E 都有3个状态。每个类族的层次和每个类的状态分别用相应的整数值表示阵列中的域值和状态值。根据定义2，作出图4对应的 EEDPN 图如图5，其中一个类族的层次和相应的状态数分别用不同的 $EDPN$ 表示，且所有的状态 $EDPN$ 中的虚线节点表示仅仅只能由相应的类族 $EDPN$ 中的虚线节点所达到的状态。为了讨论的方便，约定 $EEDPN$ 中的所有节点都称为"状态节点"，节点中的整数值称为"状态级别"或者"阵列值"。

4.2 生成交互路径的标记关联递推法(MARM)

在类的交互测试中，关键问题是寻找交互路径，以一条交互路径为正交阵列的一行，路径中的节点值为阵列值构成正交阵列。我们讨论的一条交互路径是指在两两交互的 n 个类族中依次取一个且仅取一个状态节点，构成一条有向路径。n 个类族至少需要 $TCNum=max\{m_i * m_{i+1}\}(1\leqslant i<n)$ 条有向路径才能保证测试的充分性，从而构成一个 $TCNum\times n$ 的矩阵。由于我们的矩阵是非标准的，因而称为扩展的正交阵列(Extend Orthogonal Array，EOA)，基于扩展的正交阵列生成测试用例的方法称为扩展的正交阵列测试系统(Extended OATS，EOATS)。

我们提出基于 EEDPN 的标记关联递推法生成交互路径。设 C_{i-1}，C_i，$C_{i+1}(1<i<n)$ 为任意相邻的3个类族，m_{i-1}，m_i，m_{i+1} 分别为对应的状态节点数。由于 EEDPN 中的每个状态节点都是可标记的，对于 C_i 中的某个状态节点，用正整数 k 标记该节点($1\leqslant k\leqslant m_i$)，且用与 k 交互的所有前驱节点和后继节点的状态级别来标记 k，分别构成 k 的含有 m_{i-1} 个元素的前驱节点集 $PrevSet$ 和含有 m_{i+1} 个元素的后继节点集 $NextSet$，k 的入度与出度也就相应地分别为 m_{i-1} 和 m_{i+1}。通过对 k 进行标记，就是为了保证与 k 关联的每条有向边至少覆盖一次。下面给出生成交互路径的标记关联递推算法：

第1步：定义一个 n 元数组 P 存放所有交互路径。在每一个类族 C_i 的 $EDPN$ 中，对每一个状态节点 k 用所有的前驱节点和后继节点标记。

第2步：对每一个前驱节点 j 或后继节点 l，如果 $j=k$，寻找一个不同于 k 的最小后继节点 l 构成三元组(j,k,l)，否则，寻找一个等于 j 或者 k 的最小后继节点 l 构成三元组(j,k,l)。

第3步：分别从前驱节点集和后继节点集合中删除节点标记 j 和 l。如果 k 的前驱节点集和后继节点集不空，则转到第2步。

第4步：如果 k 的前驱节点集或后继节点集任意一个不空，则为空集合重新标记最小或最大级别的节点，转到第2步。

第5步：将每一次两两交互的三元组按照前一三元组的后两个分量与后一三元组的前两个分量分别对应相等的原则，用笛卡儿乘积合并成四元组。如此反复，得到交互路径 n 元组集合 P。

第6步：以交互路径集合 P 中的每一个元素为行构成一矩阵，则得到类的交互测试用例——扩展的正交阵列表，算法结束。

该算法所生成的交互路径不受类族的个数和每个类族的状态节点数限制，可以方便灵活地扩充，还可以在很大程度上避免盲目的状态组合而产生的测试用例"爆炸"的问题。

4.3 扩展的 OATS 方法(EOATS)生成类的交互测试用例

结合 McGregor 提出的 OATS 方法和 MARM 方法，我们优化 OATS 方法，提出基于 EEDPN 的扩展的 OATS 方法生成类的交互测试用例。描述如下：

(1)确定所有的因素和每个因素的层次。根据 UML 的类图(或状态图)，确定所有的因素、每个因素相应的层次以及状态数，做出类的层次级别以及相应的状态数目表(如表2)。

表2 一般例子中类的层次级别以及相应的状态数目表

类族	类	层次级别	状态数
A	A	1	2
	B	2	3
P	P	1	2
C	C	1	2
	D	2	3
	E	3	3

(2)作出 EEDPN 图。按照第一步的分析作出相应的 EEDPN 图(如图5)。

(3)用 MARM 方法求出交互路径，并作出扩展的正交阵列表。求出图4所示一般例子的交互路径 $P_1\sim P_8$，作出正交阵列表如表3。

表3 用 MARM 方法生成的扩展的正交阵列表

测试用例	A	A 状态	P	P 状态	C	C 状态
	1	2	3	4	5	6
P_1	1	1	1	2	1	1
P_2	2	1	1	2	2	3
P_3	1	2	1	1	1	2
P_4	2	2	1	1	2	1
P_5	2	3	1	1	3	1
P_6	1	1	1	2	3	2
P_7	2	2	1	1	2	2
P_8	2	3	1	1	3	3

(4)基于映射和表中的行构建测试用例。通过阵列中每一行的级别编号解码并还原成每个因素的各个独立列表，从而将正交阵列解释还原成测试用例。例如：第8行解释为测试用例 P_8，通过将处于状态1的类 P 的一个实例传送给处于状态3的类 E 的一个实例，处于状态3的类 B 的一个实例将要发送消息。

5 基于扩展的 EMDPN 的类的层次测试

5.1 EMDPNG 模型

Katayama 在文[17～19]中讨论了并行程序的测试问题，提出了基于协同路径的并行程序测试用例生成方法。我们发现将 EMDPN 与测试并行程序的协同路径结合起来，能很好地解决类的继承测试问题。

事件消息驱动 Petri 网(Event and Message Driven Petri Network，EMDPN)模型是对传统的 Petri 网的扩展，对 EDPN 的改进，修改了使 Petri 网能够更接近地表示事件驱动的系统，增加了描述对象之间的消息通信的消息地点集合。

定义3(EMDPN) 是一种多向图$(EDPN,M)$，其中 $EDPN$ 独立地描述每个类族，M 是消息地点的集合。

由于 EMDPN 是一种有向图，因此其结构提供面向对象软件数据流分析框架。尽管有四类节点，但这里仅仅关注其连通性，不考虑节点类型，也就是将所有节点看成同一类。基于 EMDPN 的图(EMDPNG)中的节点指上述四类节点，边表示

消息和转移之间的连接。

定义4(EMDPNG) 是一种多向图(N,E,s,f,In,Out)，其中$N=D\cup S\cup M\cup P$是节点的集合，E是其边的集合。如果有向边$e=(u,v)\in E$，那么$u,v\in N$。s,f分别是满足下列条件的开始节点和终止节点：

$s\in N\wedge\forall\ u[u\in N\rightarrow(u,s)\notin E]$，$f\in N\wedge\forall\ u[u\in N\rightarrow(f,u)\notin E]$

面向对象的软件系统由多个类族(类与其派生类)交互而成，因而有多个EMDPN图。一个面向对象软件(OOP)的EMDPN图的集合表述如下：

$EMDPNGs(OOP)=\{EMDPNG_i=(N_i,E_i,s_i,f_i,In_i,Out_i]\mid 1\leqslant i\leqslant numClass(OOP)\}$

其中$numClass(OOP)$表示程序OOP的类族数。

5.2 EMDPNG中的类交互

当两个类族C_A和C_B同步传递消息时，分别用$EMDPNG_A$和$EMDPNG_B$表示C_A和C_B，用N_A和N_B表示相应的节点结合。我们定义类族间的同步消息集合。

定义5(同步消息集、消息传递集与消息等待集) 类族C_A和C_B间的同步消息集合由三元组$\langle a,b,X\rangle$集表示：

$CSync(EMDPNG_A,EMDPNG_B)=\{\langle a,b,X\rangle\mid a\in N_A,b\in N_B\}$

这里$\langle a,b,X\rangle$表示基于标记X的同步消息传递。其中"标记"是抽象的，可以解释为建模环境。例如，标记可以是属性被改变，或对象间的消息传递，或语句序列的执行等等。

同样，我们可以定义对象间的消息传递和消息等待集合：

$CMsg(EMDPNG_A,EMDPNG_B)=\{\langle a,b,Y\rangle\mid a\in N_A,b\in N_B\}$

$CWait(EMDPNG_A,EMDPNG_B)=\{\langle a,b,Z\rangle\mid a\in N_A,b\in N_B\}$

这里$\langle a,b,Y\rangle$表示在标记Y下节点a向节点b传递消息，$\langle a,b,Z\rangle$表示在标记Z下节点a等待节点b传递消息的可能性，这种可能性有"等待"或者"不等待"两种情况。

在较复杂的面向对象软件系统中，可能同时存在较多的同步消息传递，因而，我们将这些同步消息传递定义为同步消息序列集合。

定义6(同步消息序列集、消息传递序列集与消息等待序列集) 面向对象软件系统中的同步消息序列集为：

$CSyncs(EMDPNGs)=\{\langle a,b,X\rangle\mid\exists\ A,\exists\ B[\langle a,b,X\rangle\in CSync(A,B)\wedge A,B\in EMDPNGs]\}$

同样，定义消息传递和消息等待序列集为：

$CMsgs(EMDPNGs)=\{\langle a,b,Y\rangle\mid\exists\ A,\exists\ B[\langle a,b,Y\rangle\in CMsg(A,B)\wedge A,B\in EMDPNGs]\}$

$CWaits(EMDPNGs)=\{\langle a,b,Z\rangle\mid\exists\ A,\exists\ B[\langle a,b,Z\rangle\in CWait(A,B)\wedge A,B\in EMDPNGs]\}$

定义7(EMDPNIG) 基于EMDPN的交互图(EMDPN Interactions Graphs，EMDPNIG)由EMDPNG和交互组成，表示类族间的行为特征。

$Interactios(OOP)=\{CSyncs(EMDPNGs),CMsgs(EMDPNGs),CWaits(EMDPNGs)\}$

$EMDPNIG(OOP)=\langle EMDPNGs(OOP),Interactions(OOP)\rangle$

5.3 生成协同路径的算法

根据上面的讨论，我们设计出生成协同路径的算法如下：

```
Generation_Paths(&PA){
//生成基于EMDPNG的路径，PA为生成的路径集
Find a path pa=(s,a1,a2,…,am,f) of which s,f are the start and
final node by DFS;
PA={pa};
while(TRUE){
    if can't find a subpath then return;
    Find a subpath pa'=(ak,a'k+1,…,a'l-1,al)
        //ak,al∈pa∧(a'j∉pa,k+1≤j≤l-1)∧1≤k,l≤m;
    Replace the subpath of pa from ak to al with pa',and get anoth-
    er path pa=(s,…,pa',…,f);
    PA=PA∪{pa};}
}
Generation_Copaths(){
  //生成A和B两个基于EMDPNG的协同路径集TC
  Generation_Paths(Path(A));//生成类族A的路径集Path(A)
  Generation_Paths(Path(B));//生成类族B的路径集Path(B)
  TC=φ;
  for each α∈Path(A) {
      Numα=Σα(α,a). a∈α,a∈⟨a,b,X⟩,⟨a,b,X⟩∈CSyncs(A,
           B)∨CMsgs(A,B);
      for each β∈Path(B){
        Numβ=Σβ(β,b). b∈β,b∈⟨a,b,X⟩,⟨a,b,X⟩∈CSyncs
             (A,B)∨CMsgs(A,B);
        if Numα=Numb then ⟨α,β⟩ is a copath and TC=TC∪
        {⟨α,β⟩};
      }
  }
  if TC=φthen the copath is not existence and return;
  return the copath set TC;
}
```

结束语 本文提出了基于EDPN模型的面向对象软件的类测试、类的交互测试和类的层次测试框架，设计了相应的测试模型；提出了基于EDPN的有标记的唯一输入输出测试用例的自动生成方法测试类的状态；基于扩展的EDPN的状态组合的标记关联递推法生成扩展的正交阵列表，测试类的交互；基于扩展的EMDPN的协同路径测试用例的生成方法生成协同路径序列，测试类的层次。这些方法能较好地解决EFSM在面向对象软件测试中存在的问题。由于类的测试是一项复杂的系统工程，因而我们提出的测试框架和相应的测试模型还存在问题，如何简化模型使其更加直观，从而简化设计的算法?如何开发生成测试用例的工具，使得生成的测试用例更加自动化、规范化?在测试过程中，需要尽可能多的生成测试用例才能覆盖所有状态，解决不可达状态的问题，导致生成的测试用例过多而产生测试用例"爆炸"的问题，那么如何进一步优化测试用例，提高生成的测试用例的可行性?这些问题是今后进一步研究的课题。

参考文献

1 郑人杰，殷人昆，陶永雷. 实用软件工程(第2版). 北京：清华大学出版社，1997

2 Myers G. The art of software testing. John Wiley & Sons，1979

3 Binder R V. Testing Object-oriented Systems：A Status Report. American Programmer，1994

4 Barlay S，Strohmeire A. The Problematic of Testing Object-oriented Software. Building Quality into Software[C]，1995. 410

5 金凌紫. 面向对象软件测试技术进展. 计算机研究与发展，1998，35(1)：6～13

6 姬莹，等. 面向对象软件测试主要问题的探讨. 西安工业学院学报，2001，21(1)：31～37

7 Doong R，Frankl P G. The ASTOOT approach to testing object-oriented programs. ACM Transaction on Software Engineering and Methodology，1994，3(2)：101～130

8 Turner C D，Robson D J. The state-based testing of object-oriented programs. In：Proc. of Conf. on Software Maintenance，Montreal，1993. 302～311

9 Detail K. On object state testing. In：Proc COMPSAC'94，Tai-

wan,1994. 222～227
10 Bourhfir C, Dssouli R, Aboulhamid E M. Automatic Test Generation for EFSM-based System. http://citeseer.nj.nec.com
11 Coga N. Comparing TorX, Autolink. TVG and UIO Test Algorithms. Springer-verlag Berlin Heikelberg,2001
12 [美]Jorgensen P C 著. 韩 柯,杜旭涛 译. 软件测试. 北京:机械工业出版社,2003
13 [美]McGregor J D,Sykes D A 著. 杨文宏,李新辉,杨洁,等译. 面向对象的软件测试. 北京:机械工业出版社,2002
14 Kim Y G, Hong H S, Bae D H. Test Cases Generation from UML State Diagrams[J]. IEEE Proceeding on Software,1999,46(4):187～192
15 张毅坤,等. 基于 UML 状态图的类测试用例自动生成方法. 计算机工程,2003,29(21):91～93
16 林宏,曾一. 基于 UML 的面向对象软件测试框架. 重庆大学学报,2003,26(8):43～46
17 Katayama T,Furukawa Z,Ushijima K. Event Interactions Graph for Test-Case Generation of Concurrent Programs. In:Proc. 1995 Asia-Pacific Softw. Eng. Conf. (APSEC'95),1995. 29～37
18 Katayama T,Furukawa Z,Ushijima K. Design and Implementation of Test-Case Generation of Concurrent Programs. In:Proc. 1998 Asia-Pacific Softw. Eng. Conf. (APSEC'98),1998. 262～269
19 Katayama T,Itoh E,Furukawa Z,Ushijima K. Test-Case Generation of Concurrent Programs with the Testing Criteria Using Interactions Sequences. In:Proc. 1999 Asia-Pacific Softw. Eng. Conf. (APSEC'99),1999. 590～597

(上接第26页)

参 考 文 献

1 Adya A, Bolosky W, Castro M, et al. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In OSDI '02, Boston, MA, 2002
2 AVAKI CORPORATION. Avaki 2.0 Concepts and Architecture. White paper,2001
3 Ball T,Rajamani S K. Automatically validating temporal safety properties of interfaces. In: the Proc. of the Intl. SPIN Workshop on Model Checking of Software,May 2001. 333～344
4 Beiriger J, Johnson W, Bivens H, Humphreys S,Rhea R. Constructing the ASCI Grid. In:Proc. 9th IEEE Symposium on High Performance Distributed Computing, 2000
5 Bolcer G. Magi: An Architecture for Mobile and Disconnected Workflow. White paper, 2001
6 Buyya R, Stockinger H, Giddy J,Abramson D. Economic models for management of resources in peer-to-peer and grid computing. In ITCom 2001, Aug. 2001
7 Castro M, Liskov B. Practical Byzantine Fault Tolerance. In: Proc. Usenix Symp. Operating Systems Design and Implementation (OSDI 99) Usenix Assoc., Berkeley, California. 1999
8 Castro M, Drushel P, Ganesh A, Rowstron A, Wallach D. Secure routing for structured peer-to-peer overlay networks. In OSDI '02, Boston, MA, 2002
9 Clarke D, Elien J, Ellison C, Fredette M, Morcos A, Rivest R. Certificate chain discovery in SPKI/SDSI. Journal of Computer Security, Jan. 2001
10 Clarke I, Sandberg O, Wiley B, Hong T. Freenet: A distributed anonymous information storage and retrieval system
11 European union data grid project. http://eu-datagrid.web.cern.ch/eu-datagrid/, 2001
12 Foster I, Iamnitchi A. On death, taxes, and the convergence of peer-to-peer and grid computing. In:2nd Intl. Workshop on Peer-to-Peer Systems (IPTPS'03), Feb. 2003
13 Foster I, Kesselman C, Nick J, Tuecke S. The physiology of the grid: An open grid services architecture for distributed systems integration,2002
14 Foster I, Kesselman C. The Grid: A New Infrastructure for 21st Century Science,1998
15 Foster I, Kesselman C, Tuecke S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications,2001, 15 (3):200～222
16 Foster I, Tuecke S. What is the Grid? A Three Point Checklist. July 2002
17 Foster I, Kesselman C, Tsudik G, Tuecke S. A security architecture for computational grids. In: ACM Conf. on Computers and Security, Nov. 1998
18 Chen Guihai, Shen Haiying, Xu Cheng-Zhong. Cycloid: A Constant-Degree and Lookup Efficient P2P Overlay Network
19 Humphrey M, Thompson M. Security implications of typical grid computing usage scenarios. In HPDC 10, Aug. 2001
20 Houston B. The "GnutellaMandragore" virus. www.exocortex.org/gnutella,2001
21 Iamnitchi A, Foster I. On fully decentralized resource discovery in grid environments. In: Intl. Workshop on Grid Computing. IEEE, Nov. 2001
22 Johnston W E, Gannon D, Nitzberg B. Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid. In:Proc. 8th IEEE Symposium on High Performance Distributed Computing, IEEE Press,1999
23 Karger D, Ruhl M. Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems. In:3rd Intl. Workshop on Peer-to-Peer Systems (IPTPS'04), Feb. 2004
24 Katzenbeisser S. Information hiding techniques for steganography and digital watermarking. Artech House Books, 1999
25 Kubiatowicz J, Bindel D, Chen Y, et al. OceanStore: An Architecture for Global-Scale Persistent Storage. In:Proc. of the Ninth intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), Nov. 2000
26 Ledlie J, Taylor J, Serban L, Seltzer M. Selforganization in peer-to-peer systems. In 10th EW SIGOPS, Sep. 2002
27 Litzkow M, Livny M, Mutka M. Condor - A Hunter of IdleWorkstations. In:Proc. 8th Intl Conf. on Distributed Computing Systems, 1988
28 Malkhi D, Naor M, Ratajczak D. Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In:Proc. of Principles of Distributed Computing (PODC 2002), 2002
29 Milojicic D S, Kalogeraki V, Lukose R, et al. Peer-to-Peer Computing, HP Technical Report, HPL-2002-57
30 Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A Scalable Content-Addressable Network. In: Proc. of the ACM SIGCOMM 2001 Technical Conference, San Diego, CA, USA, Aug. 2001
31 Rowstron A, Druschel P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In Middleware, Nov. 2001
32 Stevens R, Woodward P, DeFanti T, Catlett C. From the I-WAY to the National Technology Grid. Communications of the ACM, 1997,40(11):50～61
33 Stoica I, Morris R, Liben-Nowell D, et al. Chord: A scalable peer-to-peer lookup service for internet applications. Research report, MIT, Jan. 2002
34 Padmanabhan V N, Wang H J, Chou P A. Supporting Heterogeneity and Congestion Control in Peer-to-Peer Multicast Streaming. In: 3rd Intl. Workshop on Peer-to-Peer Systems (IPTPS'04), Feb. 2004
35 Welch V, Foster I, Kesselman C., et al. X.509 Proxy Certificates for Dynamic Delegation. 3rd Annual PKI R&D Workshop, 2004
36 Bell W H, Cameron D G, Capozza L, et al. Simulation of dynamic grid replication strategies in optorsim. In Grid 2002, Nov. 2002
37 Xu Z, Reps T, Miller B. Typestate Checking of Machine Code. In: Proc. of the 10th European Symposium on Programming, April 2001
38 Zhao B Y, Kubiatowicz K D, Joseph A D. Tapestry: An Infrastructure for Fault-Resilient Wide-Area Location and Routing. Technical Report UCB//CSD-01-1141, University of California at Berkeley Technical Report, April 2001