基于分类树的随机测试用例生成

徐 伟 王林章 李宣东

(南京大学计算机科学与技术系 计算机软件新技术国家重点实验室 南京 210093)

摘 要 随机测试(RT)已被用于在基于模型的测试中自动生成满足一定模型覆盖准则的测试用例集合。然而,完全随机的测试用例产生方法可能会导致大量冗余的测试用例。因此,在研究基于UML(Unified Modeling Language)活动图产生测试用例的基础上,提出了一种基于分类树的随机测试用例产生方法(CT-RT),运用分类树分析已运行测试用例的输入域,从而指导产生新的测试用例,并通过实验案例说明了对于UML活动图中的简单路径覆盖。该方法较之完全随机的方法大大减少了冗余测试用例的数量。

关键词 随机测试,自动生成测试用例,分类树,UML活动图

Classification Tree-based Random Test Case Generation

XU Wei WANG Lin-zhang LI Xuan-dong

(State Key Lab for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)

Abstract Random testing (RT) has been employed to automatically generate test cases for model-based testing, in which model coverage criteria are key techniques to evaluate test adequacy so as to control the random testing process. However, independent and complete random test case generation may produce redundant tests with respect to the contribution to model coverage. Given a Java program, its corresponding UML(Unified Modeling Language) activity diagram and a path-oriented model coverage criterion, this paper proposed an improved random test generation approach (CT-RT) directed by a classification tree constructed with the executed test cases. To satisfy the same model coverage criterion, experimental results show that CT-RT generates much fewer test cases than RT.

Keywords Random testing, Automatic test case generation, Classification tree, UML activity diagram

1 引言

随机测试(RT)是一种基本的软件自动化测试技术[1],它通过在程序输入域上随机产生测试输入来测试程序。测试覆盖是衡量测试充分度的指标之一,但由于测试输入随机产生,随机测试难以保证对程序代码的高覆盖率,同时随机测试也缺少对测试结果进行预测和评估的有效措施。而在基于模型的测试中,可以利用模型来评估测试结果,并且可以依据模型覆盖来控制测试的过程,因此基于模型的随机测试是一种切实可行的自动化测试技术。

如今,在软件开发中,特别是面向对象的软件开发,UML是面向对象建模和设计的重要工具,提供了各种UML模型图来对软件的结构或行为进行建模。其中,UML活动图描述了活动的顺序或并发控制流,可用来对操作过程和工作流进行建模,是一种重要的对软件动态行为的建模工具[2],被用在软件运行时验证和软件测试工作中[3-5]。文献[5]中提出一种基于UML活动图自动产生测试用例的方法。在该方法中,测试用例完全随机产生,当达到UML活动图测试覆盖率预期的要求后,即停止工作并从已产生的测试用例中筛选出满足测试覆盖率要求的测试用例集。由于该方法中的测试用例完全随机产生,针对测试覆盖的效果,可能会产生大量冗余的测试用例且耗费大量的执行时间。

针对该问题,本文提出了一种基于分类树的随机测试用例生成方法(CT-RT)。该方法通过对已随机产生的测试用例集的学习自动构建分类树,利用分类树划分测试用例的输入域,从而指导产生新的测试用例集。本文中使用一个实例比较了CT-RT方法和RT方法,结果表明CT-RT具有更好的性能,能提高随机测试用例的产生效率,避免产生冗余的测试用例。

本文结构组织如下:第2节介绍相关研究工作;第3节是基本概念说明,是本文工作的基础;第4节详细描述了基于分类树的随机测试用例产生方法,是本文主要工作;第5节描述了实例研究。最后给出了本文工作的总结和未来展望。

2 相关工作

提高随机测试的效率是随机测试研究中一个重要的方向。自适应随机测试(ART)技术被用于提高随机测试效率[6,7]。ART技术的主要内容是:基于导致错误的输入在输入域上存在聚集现象,因此通过在输入域上随机产生分布均匀的测试用例,可尽快发现软件错误。本文目的是提高随机

到稿日期:2008-01-22 本文受国家自然科学基金(No. 60603036)和江苏省自然科学基金(BK2007139)资助。

徐 伟 硕士研究生,研究方向为软件测试,E-mail:xuwei@seg.nju.edu.cn;王林章 博士,副教授,CCF会员,研究方向为软件工程、软件测试;李宣东 博士,教授,CCF会员,博士生导师,研究方向为软件工程、形式化方法、模型检验、软件测试。

测试用例的模型覆盖率，将模型的覆盖信息反馈到测试用例的输入域中，需要对输入域更深入的分析，而非简单地让测试用例分布均匀。

分类树方法作为一种范畴-划分的方法已经被用在基于规约构建完备的测试套件[8-10]。但这些方法需要测试人员识别规约中的重要方面作为分类的依据，从而构建分类树来对测试输入域进行划分。这里，分类树由基于规约的人工或者半自动化构建而成。而本文工作则是将活动图中的路径作为分类依据，先通过对已运行的测试用例样本集的学习自动构建分类树，再基于分类树自动产生新的测试用例。

基于搜索的测试数据产生技术作为软件测试领域一项很重要的研究工作，已经取得了许多研究成果[11]。已有的比较成熟的基于搜索的测试数据生成技术，如符号执行、遗传算法等，大多需要对程序结构进行分析，而本文工作是基于程序和模型的黑盒测试，故不适用。同时，本文的方法适用于黑盒测试，因此是基于模型产生黑盒测试用例技术的重要补充。

3 基本概念

本文的工作是基于 UML 活动图自动产生测试用例的后续工作[5]，所使用的分类树方法源自机器学习与数据挖掘中的决策树分类器[12]。下面分别介绍这两部分的内容。

3.1 UML 活动图

UML 活动图代表了活动发生的序列，它借鉴了流程图、状态转换图、工业工作流图和 Petri 网的模型，对工作流以及工作流与软件系统之间的交互建模。它通常关联一个用例或者一个类，可用于创建方法之间的控制流图。对于活动图 $\boldsymbol{D}$，其形式化定义参照文献[5]，我们使用活动和转换序列来表示它的行为(Behavior)。该序列称为运行段，其表示形式为 $\rho=\mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} \mu_n$，其中 μ_i 是 $\boldsymbol{D}$ 中的状态，是 $\boldsymbol{D}$ 中活动的子集；μ_0 仅包括初始活动；μ_n 仅包含终止活动；$\mu_i(1\leqslant i\leqslant n)$ 是 μ_{i-1} 状态经过转换 t_{i-1} 后产生的新状态。

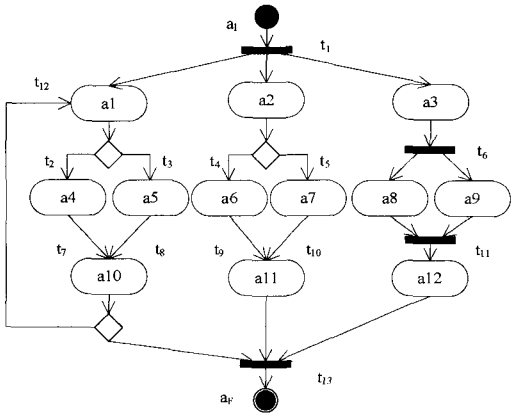

图 1 一个活动图例子

更进一步，为了定义活动图的测试充分性准则，我们定义了活动图中的路径。活动图中的一条路径是一个状态和转换集的序列，表示为 $\sigma=\mu_0 \xrightarrow{\tau_0} \mu_1 \xrightarrow{\tau_1} \cdots \xrightarrow{\tau_{n-1}} \mu_n$，其中 μ_i 是 $\boldsymbol{D}$ 中的状态，是 $\boldsymbol{D}$ 中活动的子集；μ_0 仅包括初始活动；μ_n 仅包含终止活动；考虑到并发，τ_i 是 $\boldsymbol{D}$ 中的转换子集，包含活动图中在当前状态 $\mu_i(0\leqslant i<n)$ 上同时可进行的所有转换；$\mu_i(1\leqslant i\leqslant n)$ 是 μ_{i-1} 状态经过 τ_{i-1} 中的所有转换后产生的新状态。如果路径 σ 中没有重复执行出现，则称之为简单路径。显然，对于活动图来说，一个运行段是一条路径的具体执行序列，一条路径可能对应于多个运行段。

例：在图 1 所示的活动图中，有 4 条简单路径，如图 2 所示。

$\sigma_1=\{a_I\} \xrightarrow{\{t_1\}} \{a_1,a_2,a_3\} \xrightarrow{\{t_2,t_4,t_6\}} \{a_4,a_6,a_8,a_9\} \xrightarrow{\{t_7,t_9,t_{11}\}} \{a_{10},a_{11},a_{12}\} \xrightarrow{\{t_{13}\}} \{a_F\}$

$\sigma_2=\{a_I\} \xrightarrow{\{t_1\}} \{a_1,a_2,a_3\} \xrightarrow{\{t_2,t_5,t_6\}} \{a_4,a_7,a_8,a_9\} \xrightarrow{\{t_7,t_{10},t_{11}\}} \{a_{10},a_{11},a_{12}\} \xrightarrow{\{t_{13}\}} \{a_F\}$

$\sigma_3=\{a_I\} \xrightarrow{\{t_1\}} \{a_1,a_2,a_3\} \xrightarrow{\{t_3,t_4,t_6\}} \{a_5,a_6,a_8,a_9\} \xrightarrow{\{t_8,t_9,t_{11}\}} \{a_{10},a_{11},a_{12}\} \xrightarrow{\{t_{13}\}} \{a_F\}$

$\sigma_4=\{a_I\} \xrightarrow{\{t_1\}} \{a_1,a_2,a_3\} \xrightarrow{\{t_3,t_5,t_6\}} \{a_5,a_7,a_8,a_9\} \xrightarrow{\{t_8,t_{10},t_{11}\}} \{a_{10},a_{11},a_{12}\} \xrightarrow{\{t_{13}\}} \{a_F\}$

图 2 活动图(图 1)中简单路径

对于简单路径 σ_1，运行段 $\{a_I\} \xrightarrow{t_1} \{a_1,a_2,a_3\} \xrightarrow{t_2} \{a_2,a_3,a_4\} \xrightarrow{t_4} \{a_3,a_4,a_6\} \xrightarrow{t_6} \{a_4,a_6,a_8,a_9\} \xrightarrow{t_7} \{a_6,a_8,a_9,a_{10}\} \xrightarrow{t_9} \{a_8,a_9,a_{10},a_{11}\} \xrightarrow{t_{11}} \{a_{10},a_{11},a_{12}\} \xrightarrow{t_{13}} \{a_F\}$ 是该路径的一条具体的执行序列。

在对 UML 活动图的测试中，本文考虑的测试充分性准则是简单路径覆盖，即活动图中所有可行的简单路径必须被覆盖。当大量测试用例产生并执行后，可根据该覆盖准则选择一组测试用例集。

3.2 分类树

分类树(classification tree)，或曰决策树(decision tree)，是一个类似流程图的树结构，其中每个内部节点表示一个属性上的测试，每个分枝表示一个测试输出，每个叶节点代表类别。分类树是数据分类的一种模型。在机器学习和数据挖掘中，通过对训练样本的归纳学习而得到决策树，进而由决策树提取分类规则，从而可以对新的数据进行预测分类[12]。

本文将每条测试用例对应的活动图中的简单路径作为分类准则，利用分类树对测试用例的输入域进行分类。分类树中的每个属性代表测试用例的一个输入参数。这里，输入参数必须是简单数据类型，在 Java 语言中，有 boolean，char，byte，short，int，long，float，double 和 String。如果输入参数是复杂数据类型，我们可以分解得到其中的所有简单数据类型。例如，我们可以将一个类变量分解为成员变量的集合，通过对成员变量的赋值来完成对类变量的赋值。但是，这里我们暂不考虑一些高级数据结构，如容器、数组等，这类数据目前还不能作为我们方法的输入。为了处理方便，我们简单地将测试输入参数的数据类型分为两大类：一类是集合表示的数据类型(简称集合类型)，如 boolean 和 String；其余的都归为区间表示的数据类型(简称区间类型)，包括 byte，char，short，int，long，float 和 double。因此，本文中分类树的每个内部节点表示测试用例输入中的一个简单数据类型的参数，每个分枝代表对本节点属性值的一个划分(子集或子区间)，每个叶节点表示一个分类，标记为活动图中的一条简单路径。最顶部的节点是分类树的根节点。同时，为了后面分类树的更新，每个节点需要保存生成该节点的测试用例集合。下面，基于树的定义[13]来形式化定义一棵分类树：

定义(分类树) 一棵分类树是一个三元组 $\boldsymbol{T}=(A,E,a_R)$，其中：

$A=\{a_1,a_2,\cdots,a_m\}$ 是一个有穷节点集合；

$E=\{e_1,e_2,\cdots,e_n\}$ 是一个有穷边集合；

$a_R\in A$ 是根节点；

A 中的每个节点是分类树 $\boldsymbol{T}$ 的内部节点(包括根节点)或者叶节点。每个内部节点被标上一个属性,叶节点则被标记分类。每条边 e 可以表示为一个三元组(a_I, a_O, v),其中 a_I 是 e 的入节点,a_O 是 e 的出节点,v 是标记在 a_I 上的属性的值域的子集或子区间。我们用节点和边的序列来表示 $\boldsymbol{T}$ 中的路径,如 $\pi=a_0 \xrightarrow{e_0} a_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{k-1}} a_k$,其中,(1)对于 $i=0,\cdots,k-1$,a_i 是边 e_i 的入节点,a_{i+1} 是边 e_i 的出节点;(2) $\boldsymbol{T}$ 中任意一个顶点在该路径序列中至多出现一次。在分类树 $\boldsymbol{T}$ 中,从 a_R 到树中任意一个其他的节点,有且只有一条路径;不存在从 a_R 到 a_R 的路径。我们定义 T 中以根节点为起点的路径为根路径,表示为 $\delta=a_0 \xrightarrow[\langle A_0, v_0\rangle]{e_0} a_1 \xrightarrow[\langle A_1, v_1\rangle]{e_1} \cdots \xrightarrow[\langle A_{k-1}, v_{k-1}\rangle]{e_{k-1}} a_k$,其中,$a_0=a_R$,$a_k$ 是根路径 δ 的终点。$\langle A_i, v_i\rangle$ 是属性-值对,其中,A_i 是节点 a_i 中标记的属性,v_i 是边 e_i 上标记的属性 A_i 的值域的子集或子区间。

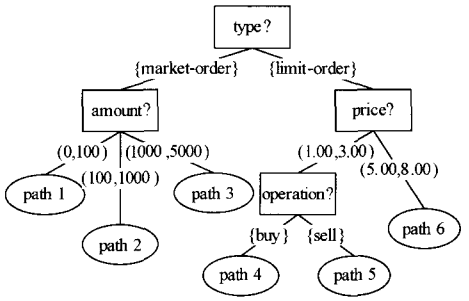

图 3　以股票订单作测试输入而构建的分类树例子

例:对于一个以股票订单作为输入对象的测试用例(该测试用例的详细说明见本文实例研究部分),股票订单包括客户编号、股票号、交易类型、订单类型、交易数额、交易价格等成员变量。以该订单作为测试输入而构建的一棵简单的分类树如图 3 所示,其中标记属性"*type*"的节点是树的根节点,"*type*?"→"*amount*?"→"*path* 1"是树中的一条根路径,⟨*type*,{*market-order*}⟩和⟨*amount*,(0,100)⟩是该根路径上的两个属性值对。

4　基于分类树的随机测试用例生成方法

基于分类树的随机测试用例生成方法步骤为:随机产生一些测试用例运行并获得这些测试用例对活动图中简单路径的覆盖信息。将简单路径信息作为分类的依据,将已运行的测试用例作为训练样本来学习构建分类树。然后基于分类树产生一批新的测试用例集运行程序,并且根据运行结果更新路径覆盖信息和分类树。如果覆盖率未达到要求,则继续基于更新后的分类树产生新的测试用例集。该过程重复进行,直到满足测试覆盖率或者测试资源被耗尽。

4.1　分类树的生成

本文中,首先随机产生一组测试用例,用作构建分类树的学习样本。这需要选出测试用例的所有输入参数,并且为每个输入参数设定相应的输入域(如果是区间类型,则设定该输入域的全区间,反之则设定输入域的全集),以便于随机测试用例的生成。同时,考虑到测试用例中的不同输入参数之间可能存在的相关性,例如在前文提到的股票订单中,如订单类型是市场订单,则股票交易价格必须为空,因此需要开发一个过滤器,帮助去掉不合理的、随机产生的测试用例。

如文献[5]所示,为了获取测试用例对模型的覆盖信息,需要根据活动图对程序进行插桩,将每个活动映射到程序中的一个方法。当测试用例驱动插桩后的程序运行后,通过匹配程序运行轨迹(方法序列)与活动图的行为可以检验程序运行是否满足活动图行为规约,同时也得到了测试用例对应的活动图中的简单路径。现在,测试用例作为学习样本,测试用例的输入参数已被选出作为候选属性,每条测试用例对应的活动图中的简单路径作为分类依据,则可基于经典的分类树生成算法(ID3 算法)通过自顶向下构造决策树来对样本进行学习[12]。

ID3 算法采用求信息增益的方法从候选属性中选择具有最大信息增益的属性作为树节点要测试的属性。由于每个学习样本是一条测试用例,故需要根据测试用例集中的数据进行属性输入域的划分,以方便信息增益的计算。对于集合类型属性,划分的目标是将属性输入域划分成一系列互不相交的子集;对于区间类型属性,则将输入域划分成一个区间序列,序列中的区间按照从小到大的顺序排列(对于区间(a,b)和(c,d),设 $a<b, c<d$,若 $b<=c$,则认为(a,b)小于(c,d)),且任两个区间没有交集。选出当前节点所要划分的属性后,将该属性标记到本节点,根据属性输入域的划分对每个子域(一个子集合或一个区间)生成一个分枝,并将本节点的测试用例集进行相应的划分,对应每个分枝生成相应的测试用例集。如此,则可以构建出分类树。

4.2　基于分类树产生测试用例

本文的主要思想是通过分类树来分析测试用例的输入域,获得和活动图简单路径相关的输入域信息,从而指导新的测试用例的产生。因此,基于分类树的随机测试用例生成方法通过对分类树的遍历产生一组新的测试用例集。

沿着树根从最左边的子节点向下遍历,访问到叶节点时,即获得从树根到当前叶节点的一条根路径 δ,同时也获得了该路径上的属性-值对的列表,标记为$\langle(A_0, v_0),(A_1, v_1)\cdots(A_k, v_k)\rangle$,其中 $v_i(0\leqslant i\leqslant k)$ 代表属性 A_i 值的一个子域(集合或区间)。这表明:对于当前叶节点上所标记的简单路径 σ,已运行的覆盖 σ 的一些测试用例的属性值落在了根路径 δ 的属性-值对列表中。即使当前属性-值对列表包含了全部属性,由于不能保证满足根路径 δ 上属性-值对列表的所有测试用例都会覆盖当前叶节点标记的简单路径 σ,因此有必要基于以叶节点为终点的每条根路径上的属性-值对列表产生一个新的测试用例。对于该列表中的每一个属性 A_i,从输入域 v_i 中随机选择一个值作为新的测试用例中的属性 A_i 上的值;对于不在该列表中的属性,则从该属性的初始输入域上取随机值。当访问了一个内部节点(设其标记属性为 A)的所有子节点以后,设其所有分枝的输入域总和(并集或区间值序列)为 v,则所有已产生的匹配这条根路径的测试用例在 A 上的值都落在区域 v 内。如果 v 未覆盖属性 A 的整个输入域,则新的测试用例应该从属性 A 的未被覆盖的输入域中取值。设以该内部节点为终点的根路径上的属性-值对列表为 L,如果属性 A 是集合类型,v' 表示其未被覆盖的输入域,则新产生一条测试用例,设置如下:属性 A 的值从 v' 中随机产生,列表 L 中除 A 以外的属性从表中标记的输入域中随机取值,其他的属性则从整个输入域中取值;如果属性 A 是区间类型,设当前内部节点各分枝上的区间值序列为$\langle v^1, v^2, \cdots, v^s\rangle$($s$

是当前内部节点的子节点数)，对应于 A 的初始输入域区间的补集也是一个区间序列，可表示为 $\langle v^1', v^2', \cdots, v^t' \rangle$ $(0 \leqslant t \leqslant s+1)$，则对于该序列中的每个区间 v^i' $(1 \leqslant i \leqslant t)$，生成一个新的测试用例，设置如下：属性 A 的值从 v^i' 中随机产生，表 L 中除 A 以外的属性从表中标记的输入域中随机取值，其他的属性则从整个输入域中取值。这样，当遍历完树中所有的节点后，就会产生一组新的测试用例集。

4.3 测试执行和分类树的更新

基于分类树已经产生了一组测试用例集，但是不能保证这些测试用例全部都是合理的，如 4.1 节中所说明的，需要用过滤器来选出合理的一组测试用例。

当每条测试用例运行结束后，获取它对模型的覆盖信息，如果满足了模型覆盖率的要求，则整个过程结束，接下来从已运行的全部测试用例中选择出满足模型覆盖率要求的测试用例集；否则，则根据本条测试用例及其覆盖信息更新当前的分类树：从根节点开始访问分类树，每当访问到一个节点时，首先将本条测试用例加入到该节点的测试用例集中，以便该节点的更新操作。如果当前节点是内部节点(标记属性 A)，则判断本测试用例中属性 A 的值是否落在当前节点的某个分枝子输入域中。若找到这样的分枝，则递归访问该分枝下的子节点。否则，以该内部节点为根的子树将基于该节点中保存的测试用例集而重新被构建。如果当前节点是叶节点且新测试用例对应的类别与叶节点所标记的类别不一致，该叶节点将被重新构建。

5 实例研究

为了说明本方法的有效性，我们开发了一个模拟股票交易系统并对其进行实例研究。该系统模型主要参考文献[14]，用 Java 语言实现，包括 40 个类和 305 个方法。描述系统工作流程的活动图如图 4 所示。

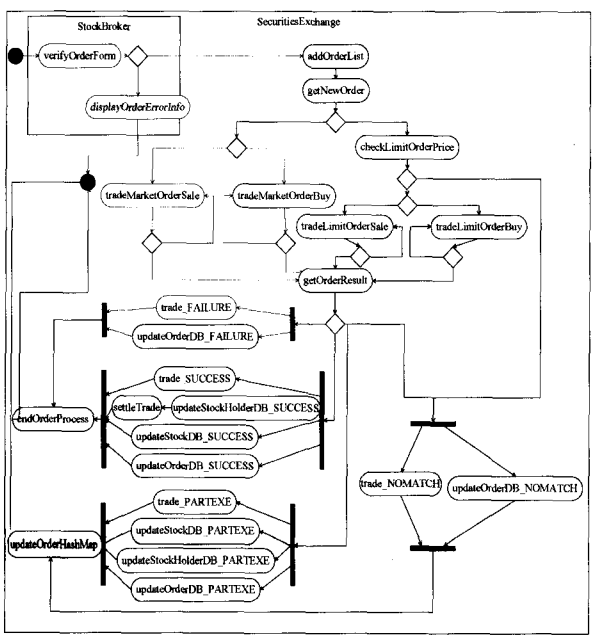

图 4 模拟股票交易系统的活动图

系统主要功能是执行股票订单，完成股票交易。其主要功能模块分为股票经纪人和股票交易所两部分：股票经纪人接受客户的股票交易订单并对其验证。如果订单无效，如客户帐号或交易的股票不存在，则当前订单无效，被终止。否则，该订单将被提交给股票交易所处理。股票交易所按照订单类型(市场订单或限制订单)和交易类别(买入或卖出)进行相应处理。执行订单是一个匹配过程，即将当前订单和数据库中已有的待处理的订单依次比较，有合适的则进行交易：对于市场订单，只需从已有订单中找出相匹配的订单，按当前市场价格交易；对于限制订单，必须以不低于限制价格卖出或不高于限制价格买入，故首先检查它的限制价格在当前是否有效。如果有效，则按照限制价格或更好的价格交易，否则，该订单暂时不处理，设其状态为“未交易”。对于任何订单，其交易结果可能为下面 4 种之一：“交易失败”、“交易成功”、“部分交易”和“未交易”。订单执行结束后，股票交易所将启动多线程并发处理业务，最后订单处理结束。

表 1 实验结果表

	RT		CT-RT	
	测试用例数	耗时(s)	测试用例数	耗时(s)
1	2301	1624	452	568
2	749	584	894	809
3	765	549	622	735
4	2338	1721	596	854
5	2078	1643	761	933
6	1959	1690	641	598
7	306	340	984	935
8	2295	2614	606	609
9	469	241	572	965
10	666	416	384	650
平均值	1393	1144	652	766

系统利用数据库来保存信息，包括客户表(已注册客户信息)、股票表(当前股票信息)、股票持有表(当前客户持有股票信息)、订单表(已提交的股票订单信息)。系统的测试输入是一份股票订单，例如(客户帐号：00001；股票号：0001；交易数额：100 股；订单类型：限制订单；交易类型：卖出；限制价：5 元)。实验中共有 10 组实验数据，每组实验用 RT 方法和 CT-RT 方法分别产生测试用例，直到覆盖活动图中全部 18 条简单路径。在 CT-RT 方法中，我们首先运用 RT 方法随机产生 100 条测试用例，然后对这些测试用例学习构建分类树。实验结果如表 1 所示为覆盖全部 18 条简单路径。虽然 CT-RT 不能保证每次都比 RT 优越(如 RT 方法产生的最小测试用例的数量是 306，而 CT-RT 方法是 384)，而且考虑到分类树的生成、遍历和更新，CT-RT 平均每条测试用例产生时间要大于 RT。但是，总体看来，达到同样的简单路径的覆盖率，CT-RT 产生的测试用例数比 RT 要少得多，总体耗时也少。实验表明了针对活动图的简单路径覆盖，CT-RT 方法提高了 RT 方法的效率。

结束语 本文针对基于模型的随机测试方法中产生冗余测试用例的问题，提出了一种基于分类树的随机测试用例产生方法。本方法主要思想是通过对已运行测试用例的学习构建分类树，利用分类树划分测试用例的输入域，从而指导新的测试用例的生成，为满足一定的模型覆盖率而避免产生冗余的测试用例。实例研究表明，在一定条件下，该方法比完全随机的方法可以大大减少随机产生的测试用例数量。

本文通过分类树模型来对测试用例的输入域进行动态划分和分析。同样，也可以运用一些其他的分类模型，如贝叶斯

(下转第 285 页)

服务提供者与客户的耦合	存在紧耦合，客户桩(stub)需要由客户应用进行编译	存在紧耦合，客户桩(stub)需要由客户应用进行编译或者导入	无紧耦合，客户桩(stub)可以在运行时下载到客户
通知客户获得所需服务的机制	没有可用的分布式事务通知机制	没有可用的分布式事务通知机制	使用 Java 事件通知模型
区别本地和远程地址空间	没有显式区别本地对象和远程对象，需要在本地建立远程对象的对象引用	没有显式区别本地对象和远程对象，需要在本地建立远程对象接口的专用指针	区别本地和远程对象。使用3种技术来处理远程交互：1)建立对远程对象的引用；2)非远程对象可以串行化并以传值的方式传递；3)简单数据类型也可以利用传值的方式传递
编程语言及基础设施相关			
何种编程语言	多种编程语言	多种编程语言	Java
应用组件之间的交互	所有远程对象使用 CORBA 接口定义语言(CORBA-IDL)进行发布	所有远程对象使用 COM 接口定义语言(COM-IDL)进行发布	所有远程对象使用 Java 远程接口发布，并实现 Java 远程接口
请求协议	依赖于通用 ORB 协议(GIOP)通用 Internet 跨 ORB 协议(IIOP)	依赖于对象远程过程通信协议(ORPC)	依赖于 Java 远程方法协议(Java Remote method Protocol，JRMP)
操作系统	具有平台无关性	Microsoft Windows 系列	具有平台无关性，但需要安装 Java 虚拟机(JVM)
应用的网络方面	ORB 负责处理数据的调用和格式转换	ORPC 负责处理数据和调用格式的转换	不需要特别数据/调用格式的转换，因为是在 JVM-JVM 之间进行，利用 Java 串行化技术通信

结束语 SOA 的设计目标是以服务为基础，通过服务的交互来实现系统动态、松耦合集成，极大地降低了复杂性与成本。但目前为止没有文献对实现它的方法进行明确的论述，造成人们在实现它时总是无从下手。本文在深入分析了 SOA 实现方法机理的基础上，提出各自的优点和不足之处；并将主要方法作横向比较研究。Jini 是一种基于 Java 的分布式系统，具有许多其它分布式模型没有的优点，是如今使用最多的分布式计算编程模型。但在如今多种模型并存的现实情况下，Jini 不可能完全代替 CORBA，DCOM，RMI，基于 UDDI 的 Web Service 等分布式计算体系，所以应该努力解决这些分布式计算体系之间共存并且相互操作的问题，这样不仅能够充分利用不同的分布式计算体系的优点，还可以巧妙地避免彼此的缺点。而且从应用的层次上来看，用户并不希望被绑定在一家公司的产品上，而希望有更多的选择余地，所以，下一步的研究应着重于解决这些分布式系统之间互操作的方法，它不仅有理论上的意义，还具有现实的意义。

参 考 文 献

[1] Wu Jie. Distributed System Design. 高传善，等译. 北京机械工业出版社，2001：5-15

[2] 李兵，陈鸣. CORBA 对分布式管理体系结构的支持. 解放军理工大学学报：自然科学版，2001(4)：39-43

[3] 徐泽华，王耀南，罗瑞琼. 基于 DCOM 的分布式计算机控制系统的设计. 工业控制计算机，2001(6)：7-10

[4] 程炜，杨宗凯，乐春晖. 基于 Web Service 的一种分布式体系结构. 计算机应用研究，2002(6)：105-107

[5] Kumaran S. Jini 技术指南. 林琪，欧阳宇，等译[M]. 北京机械工业出版社，2003：10-20

[6] Eckel B. Thinking in Java [M]. 2nd edition，Revision 12，2000：50-65

(上接第 266 页)

信念网络[12]等来学习测试用例。因此，未来工作主要是多种分类学习模型的应用以及进一步的实例研究。

参 考 文 献

[1] Hamlet R. Random testing. Encyclopedia of Software Engineering. John Wiley and Sons，1994

[2] Rumbaugh J，Jacobson I，Booch G. The Unified Modeling Language User Guide. Boston：Addison-Wesley，2001

[3] Wang L，Yuan J，Yu X，et al. Generating Test Cases from UML Activity Diagram Based on Gray-Box Method // Proc. of APSEC'04. IEEE Computer Society，New Jersey，2004：284-291

[4] Chen T，Poon P，Tang S，et al. Identification of Categories and Choices in Activity Diagrams // Proc. of the 5th International Conference on Quality Software (QSIC'05). IEEE Computer Society，2005：55-63

[5] Chen M，Qiu X，Xu W，et al. UML Activity Diagram-based Automatic Test Case Generation for Java Programs. The Computer Journal，2007，doi：10.1093/comjnl/bxm057

[6] Chen T Y，Huang D H，Zhou Z Q. Adaptive random testing through iterative partitioning // Proc. of the 11th International Conference on Reliable Software Technologies，LNCS 4006. Berlin Heidelberg：Springer-Verlag，2006：155-166

[7] Ciupa L，Leitner A，Oriol M，et al. Object distance and its application to adaptive random testing of object-oriented programs // Proc. of the 1st International workshop on Random Testing (RT'06). ACM Press，2006：55-63

[8] Grochtmann M，Wegener J，Grimm K. Test case design using classification trees and the classification-tree editor CTE // Proc. of the 8th International Software Quality Week (QW'95). Software Research Institute，San Francisco，CA，1995

[9] Singh H，Conrad M，Sadeghipour S. Test Case Design Based on Z and the Classification-Tree Method // Proc. of the 1st International Conference on Formal Engineering Methods (ICFEM'97). IEEE Computer Society，1997：81-90

[10] Krupp A，Mueller W. Classification Trees for Random Tests and Functional Coverage // Proc. of the Conference on Design，Automation and Test in Europe，EDAA. 2006：1031-1032

[11] McMinn P. Search-based Software Test Data Generation：A Survey. Software Testing，Verification and Reliability，2004，14(2)：105-156

[12] Han J，Kamber M. Data Mining：Concepts and Techniques. Morgan Kaufmann Publishers，2001：279-296

[13] Kolman B，Busby R，Ross S. Discrete Mathematical Structures. Fourth Edition. Prentice Hall，Inc.，2001

[14] Blaha M，Rumbaugh J. Object-oriented Modeling and Design with UML. Second Edition. Pearson Education，Inc.，2005