基于UML活动图测试场景的优先级判断

谢棠棠[1,2] 李 俊[3] 张为群[1,2]
(西南大学计算机与信息科学学院 重庆 400715)[1]
(重庆市智能软件与软件工程重点实验室 重庆 400715)[2] (西南师范大学出版社 重庆 400715)[3]

摘 要 由于UML模型在设计过程中产生,作为测试模型不需要额外的开支,因此以UML作为测试模型的研究很多。目前的研究方法大多数采用灰盒测试方法[1]对各种模型的条件、节点、方法等进行遍历,得到测试场景。然而在实际中,通过这些方法从UML模型中会得到很多的测试场景,目的就是要对找出的灰盒测试用例进行优先级判断,找出较为重要的测试用例,这些测试用例能为实际的测试过程提供着重测试的依据。

关键词 UML,测试场景,优先级判断,活动图

Priority Judgment Based on the Test Scenario from UML Activity Diagram

XIE Tang-tang[1,2] LI Jun[3] ZHANG Wei-qun[1,2]
(College of Computer and Information Science, Southwest China University, Chongqing 400715, China)[1]
(Chongqing Intelligent Software and Software Engineering Laboratory, Chongqing 400715, China)[2]
(Southwest China Normal University Press, Chongqing 400715, China)[3]

Abstract Because UML model can be generated in the process of design, and it doesn't need additional expense as a test model, so many people intend to use the UML model as a test model. The current research methods go though all the conditions, nodes and methods of the UML model by using the grey-box-testing theory. In fact, too much test scenarios can be generated from the UML model by these methods. This paper is to find the important test scenarios by the priority judgment of all the test scenarios, which can provide a basis of stressed test in the actual testing process.

Keywords UML, Test scenario, Priority judgment, Activity diagram

1 引言

目前工业界普遍采用UML作为设计标准,而基于UML作为模型进行测试的方法也在不断发展,在测试的过程中,对于很多问题在进行单独测试时并不能发现,而必须在某一个测试场景中测试才能够找出这个错误,所以对软件的测试也必须在测试场景中进行。在以前的研究[2,3]中以UML活动图作为测试模型进行测试,并完成测试场景提取实验,然而在对复杂的UML活动图进行测试场景提取可能会得到大量的测试场景,这些测试场景并非同样重要,可以通过一些算法或需求的重要性来确定其优先级,有了这些优先级可以选择测试重点,对重要的测试场景优先测试、重点测试,正如Ross Collard在"Use Case Testing"一文中说:"测试用例的前10%到15%可以发现75%到90%的重要缺陷"。而本研究的重点就是对UML活动图的测试场景进行优先级判断,找出其中较为重要的测试场景,通过这些优先的场景对系统进行重点测试,可以找出软件中更多的、更为重要的错误,让软件更加健壮。

2 UML活动图优先级值确定准则

在以往对UML活动图的研究[2,3]中论述过它作为测试模型的优越性,利用灰盒测试理论和图的遍历算法,加上对其定义的测试准则,可以对活动图的场景进行提取,并完成了提取实验,实验证明这是能够实现的。面对这些测试场景我们现在的工作是要确定其优先级顺序,在这里必须扩展活动图,对每个事件加入优先级值这个变量,让每个结点的入度等于这个优先级值变量。

在UML活动图中对每个事件加入了优先级值,对于每个测试用例就有一个优先级值总和,这个总和的大小就是衡量这个测试场景优先级的标准,得到的优先级值大的场景要么是该场景中有些事件特别重要,要么是该场景比较复杂容易出错。如果这些事件在实际中重复次数多或者需求较为重要,我们就让它的优先级高,因为我们使用软件最多的地方是测试的重点,必须保证常用的、重要的功能更少出错;如果该场景包含的事件数量多,这种情况往往是流程较为复杂,是容易出错的场景,这里也应该重点测试。下面首先要解决的问题就是如何确定每个事件的优先级值。

到稿日期:2008-07-22 返修日期:2008-10-06 本文受重庆市西南师范大学青年基金项目(SWNUQ2005020, SWNUQ2005011),重庆市信息产业发展资金(200611004),重庆市自然科学基金(CSTC,2006BA2003),重庆市西南师范大学高新技术培育基金(XSGX09)资助。

谢棠棠 讲师,研究方向为软件工程、软件测试等;李 俊 讲师,研究方向为软件工程、软件测试等;张为群 教授,研究方向为软件工程及人工智能技术等。

准则一：通过需求分析，根据实际业务的重要性确定事件的优先级。这种方法可以通过活动图直接查看需求的重点，同时也是测试的重点。为了方便表示和对优先级的确定，不妨把需求分为9级，最高9级，最低1级，依次表示不同的需求重要性，1到9在这里也是我们的测试优先级值，当然也可以根据具体情况分成更多或更少的级次，但方法可以类推。

准则二：对于在需求分析中没有加入优先级值的活动图，我们可以通过路径的覆盖次数决定优先级值。本人在如何获得测试场景的论文中已经介绍过可以通过路径覆盖等准则获取测试场景，在这个过程中我们可以多做一件事，就是每经过一个事件，就对它的优先级值加1，当遍历结束后，每个事件的优先级值就得到了。

3 测试场景优先级的确定方法

在得到每个事件优先级值之后，我们可以通过多种方法来确定测试场景的优先级，大体可以分为在测试场景提取过程中确定优先级和先提取所有测试场景再对测试场景排序的方式确定优先级这两种方法。

3.1 在测试场景提取过程中确定优先级

①提取过程的推导及原理证明

一个活动图可以抽象成一个有向图 $G=(V,E)$，然后通过动态划规算法确定各种测试场景的优先级。

首先这个活动图被划分成 $K>2$ 个不相交的集合 V_i，$1\leqslant i\leqslant K$，其中 V_1 和 V_k 分别只有一个接点 s（开始结点）和 t（结束结点），图中所有事件的入度即边〈u,v〉均具有如下性质：若 $u\in V_i$，则 $v\in V_{i+1}$，$1\leqslant i<k-1$，且每个边〈u,v〉均有优先级值 $c(u,v)$。从 s 到 t 的一条路径的优先级值是各个边优先级值 $c(u,v)$ 的总和，现在就是要求得这个图 s 到 t 的最大优先级值，求得之后每个边优先级值 $c(u,v)-1$，再求得次优先级的值，依次类推，直到最大优先级值为0。

对于每一条由 s 到 t 的路径，可以把它看成在 $k-2$ 个阶段中作出的某个决策序列的相应结果，第 i 次决策就是确定 V_{i+1} 中的哪个结点在这条路径上，$1\leqslant i\leqslant K-2$。下面证明最优性原理对活动图成立。

假设 s，V_2，V_3，…，V_{k-1}，t 是一条由 s 到 t 的最大优先级值路径，还假定从开始结点 s 开始，已作出了到结点 V_2 的决策，因此 V_2 就是初始决策所产生的状态。如果把 V_2 看成是原问题的一个子问题的初始状态，解这个子问题就是找出一条由 V_2 到 t 的最大优先级值路径，这条路径显然是 V_2，V_3，…，V_{k-1}，t。如若不然，设 V_2，Q_3，…，Q_{k-1}，t 是一条由 V_2 到 t 的更大优先级值路径，则 s，V_2，Q_3，…，Q_{k-1}，t 是一条比路径 s，V_2，V_3，…，V_{k-1}，t 更短的由 s 到 t 的路径。与假设矛盾，故最优性原理成立，我们可以用动态划规方法来找出优先级场景。

用动态规划求活动图最大优先级值路径的决策序列可以表示如下：设 S_0 是开始结点，假定需要作 n 次决策 X_i，$1\leqslant i\leqslant n$。设 $X_i=\{r_{i,1},r_{1,2},\cdots,r_1,p_2\}$ 是 X_i 的可能决策值的集合，而 $S_{1,1}$ 是在选取决策值 $r_{1,j1}$ 以后所产生的状态，$1\leqslant j_1\leqslant p_1$。又设 $T_{1,j1}$ 是相应于状态 $S_{1,j1}$ 的最优决策序列。那么，相应于 S_0 的最优决策序列就是 $\{r_{1,j1},T_{1,j1}\mid 1\leqslant j_1\leqslant p_1\}$ 中最优的序列，记为 $\underset{1\leqslant j_1\leqslant p_1}{OPT}\{r_{1,j1}T_{1,j1}\}=r_1,T_1$。如果已作了 $k-1$ 次决策，$1\leqslant k-1\leqslant n$。设 X_1，…，X_n 的最优决策值是 r_1，…，r_{k-1}，它们所产生的状态为 S_1，…，S_{k-1}。又设 $X_k=\{r_{k,1},\cdots,r_{k,pk}\}$ 是 X_k 的可能值的集合。$S_{k,jk}$ 是选取决策值 $r_{k,jk}$ 后所产生的状态，$1\leqslant j_k\leqslant p_k$。$T_{k,jk}$ 是相应于 $S_{k,jk}$ 的最优决策序列。因此，相应于 S_{k-1} 的最优决策序列是 $\underset{1\leqslant j_1\leqslant p_1}{OPT}\{r_1,_{j1}T_{1,j1}\}=r_1,T_1$。于是相应的最优决策序列为 $r_1,\cdots,r_{k-1},r_k,T_k$。

从上面最优决策序列的推导，我们可以从开始结点开始，以逐步向下递推的方式列出求下一阶段决策值的递推关系式，即根据 X_{i+1}，…，X_n 的那些最优决策序列来列出求取 X_i 决策值的关系式，再回溯求解这些关系式得出最优决策序列。

所求得的最大优先级值为：

$$COST(i,j)=\max_{\substack{1\in V_{i-1}\\ \langle i,j\rangle\in E}}\{c(1,j)+cost(i-1,1)\}$$

②算法实现

```
Procedure PATH(E,k,n,p)
//按结点的顺序给结点编号，有 n 个节点的活动图。E 是事件集，c(i,j)是边〈i,j〉的优先级值。P(1,k)是最大成本路径
real COST(n),
integer D(n-1),P(k),r,j,k,n
COST(1)←0
For j←2 to n do //计算 COST(i)，设 r 有〈r,j〉∈E,且使 COST(r)+c(r,j)取最大值
    COST(j) ←COST(r)+c(r,j)
    D(j) ←r
Repeat
//找出最大值路径
p(1) ←1;p(k) ←n
for j←k-1 o 2 by - 1 do //找路径上的第 j 个接点
    p(j) ←D(p(i+1))
    repeat
END PATH
```

3.2 在测试场景提取后确定优先级

基于提取后测试场景，我们对每个测试场景的每个事件的优先级值进行求和，得到测试场景的优先级值为 $\sum_i(1\leqslant i\leqslant n)$，然后我们可以对 $\sum_I$ 进行排序算法得到测试场景的优先级序列 X_k，具体算法如下：

```
Procedure PATH(c,n,p)
// n 是测试场景数量，c(i,j)是事件〈i,j〉的优先级值。P 是各条测试场景的事件数量
for i←1 to n do //从第一个场景到最后一个场景，依次求出∑i(1≤i≤n)
  for j←1 to pi do
    ∑i←∑i+c(i,j)
  repeat
repeat
//对∑I 进行排序
for z←1 to n do
  for y←1 to z do
    Xz←∑y≥∑z
Repeat
Repeat
END PATH
```

4 模拟试验

本文对上述的优先级判断作了实验模拟，试验证明上述设想可以实现。本试验仍然选用 Borland 公司推出的 UML

建模工具 ModelMaker 为设计和运行平台，通过调用它的 ToolsApi 接口模块控制活动图的一切元素。

本文以一个简单的从自动售货机买物品的片段为例子试验，对得到的所有场景排序得到场景的优先级，如图 1 所示。

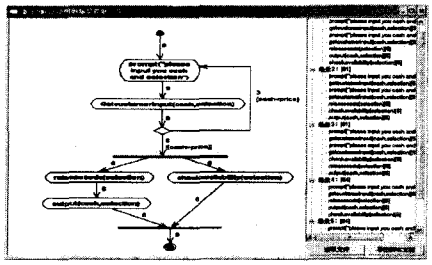


图 1

从这个实验中我们可以找到优先级高的测试场景，同时在场景中也可以看到各个消息的优先级。

结束语 本文通过对 UML 活动图测试场景进行优先级判断，从而利用这些优先级来指导我们的测试工作，当看到优先级高的测试场景就应该花大力气着重测试，这样就更能发现系统的关键问题，保障系统的健壮性，而本文的进一步工作是确定测试的充分性，测试需要到达什么级别的优先级我们就认为系统测试可以接受了。

参 考 文 献

[1] 李秋英，刘斌，阮镰. 灰盒测试方法在软件可靠性测试中的应用[J]. 航空学报，2002，23(5)：455-458

[2] 谢棠棠，张为群. 一种基于 UML 模型的系统测试方法[J]. 西南师范大学学报：自然科学版，2005，30(2)：258-263

[3] 谢棠棠. 基于 UML 模型的测试场景生成研究与工具实现[D]. 重庆：西南师范大学，2005

[4] Wang Linzhang，Yuan Jiesong，Yu Xiaofeng，et al. Generating Test Cases from UML Activity Diagrams Based on Gray-Box Method[C]//11thAsia-pacific Software Engineering Conference (APSEC'04). Washington：IEEE Computer Socrety，2004：284-291

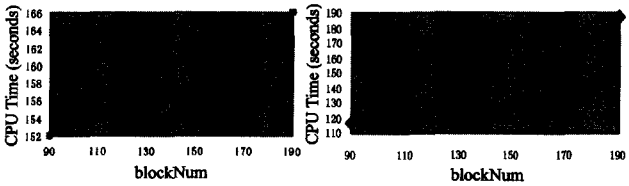
[5] Schmuller J. UML 基础、案例与应用(第三版)[M]. 李虎，赵龙刚，等译. 北京：人民邮电出版社，2004

[6] Binder R V. 面向对象系统的测试[M]. 华庆一，王斌君，陈莉，译. 北京：人民邮电出版社，2001

(上接第 155 页)

维有本地封闭立方体所包含的封闭单元数从 5600 万递减到 5100 万，相应的文件大小从 994MB 递减到 912MB。由该图趋势可以大致估计，当分块数为 1 时，产生的全局封闭立方要大得多，这进一步验证了引理 1。

然而，尽管封闭立方体存储空间会随着分块数增加而减小，相应的预计算时间和查询时间会增加，其中查询时间增幅较大。如图 6 所示，当数据分块数从 90 递增到 190 时，预计算时间增幅 14s，查询时间增幅 71s。因此，用户可以根据实际应用需求，通过调整数据分块的个数，在空间和时间上选择一个最佳点。



(a) 不同数据分块数下的预计算时间 (p=11)　(b) 不同数据分块数下的查询时间 (p=11)

图 6

结束语 企业和单位积累大量的数据亟待高性能的联机分析和处理，本文提出了一种在 PC 集群下的分布式处理的解决方案，即基于 MapReduce 框架的封闭立方体分布式计算和查询。实验表明，通过融合 MapReduce 框架和封闭立方体这两种技术的优势，实现的系统具有更大的数据压缩率，能够短时间内处理千万级数据，具有较好的扩展性，并且能通过 MapReduce 框架自动进行系统容错、负载平衡等处理。

在下一步工作中，我们将考虑怎样使用 MapReduce 框架实现数据立方体的增量维护，并研究和实现一个在 MapReduce 框架之上的分布式数据仓库原型系统。

参 考 文 献

[1] Gray J，Chaudhuri S，Bosworth A，et al. Data Cube：A Relational Aggregation Operator Generalizing Group-By，Cross-Tab，and Sub-Totals [J]. Data Mining and Knowledge Discovery，1997，1(1)：29-53

[2] Sismanis Y，Deligiannakis A，Roussopoulos N，et al. Dwarf：Shrinking the PetaCube [C]//SIGMOD. 2002：464-475

[3] Lakshmanan L V S，Pei J，Han J W. Quotient Cubes：How to Summarize the Semantics of a Data Cube [C]// VLDB. 2002：778-789

[4] Lakshmanan L V S，Pei J，Zhao Y. QCTrees：An Efficient Summary Structure for Semantic OLAP [C] // SIGMOD. 2003：64-75

[5] Beyer K，Ramakrishnan R. Bottom-Up Computation of Sparse and Iceberg CUBEs [C] // SIGMOD. 1999

[6] Xin D，Shao Z，Han J W，et al. C-Cubing：Efficient Computation of Closed Cubes by Aggregation-based Checking [C]// ICDE. 2006：4

[7] 李盛恩，王珊. 封闭数据立方体技术研究 [J]. 软件学报，2004，15(8)：1165-1171

[8] Ng R T，Wagner A，Yin Y. Iceberg-cube Computation with PC Clusters [C]//SIGMOD. 2001，30(2)：25-36

[9] Frank D，Todd E，Andrew R. The cgmCUBE project：Optimizing parallel data cube generation for ROLAP [J]. Distributed and Parallel Databases，2006，19(1)：29-62

[10] Chen Y，Dehne F，Eavis T. Parallel ROLAP Data Cube Construction on Shared-nothing Multiprocessors [J]. Distributed and Parallel Databases，2004，15(3)：219-236

[11] Dean J，Ghemawat S. MapReduce：Simplified Data Processing on Large Clusters[C]// 6th Symposium on Operating Systems Design and Implementation. 2004

[12] Apache. Hadoop[EB/OL]. http：//lucene.apache.org/hadoop/，2007