基于交互纵览图的集成测试方法与实现*)

谢棠棠[1,2] 张为群[1,2] 李 俊[3]
(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 UML2.0的标准中新增加了交互纵览图,交互纵览图作为测试模型可以解决交互图测试模型缺乏流程控制这一关键问题和活动图测试模型不便于表述交互的过程。本文针对交互纵览图更具有描述系统集成性这一特点,设计了一种对交互纵览图拆分、组合,从而覆盖消息的集成测试方法。

关键词 UML,UML交互纵览图,集成测试

An Integrated Testing Method and Experimentation Based on Interaction Overview Diagram

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

Abstract UML2.0 standard adds the interaction overview diagram. This diagram as a test model resolves the interaction diagram which doesn't express the characteristic of flow control and avoids the limitation which activity diagram is short of describing the Interaction. Focusing on the integrated feature of the interaction overview diagram test model, this paper deals with integrated testing through splitting and assembling the interaction overview diagram to get message sequence of test case.

Keywords UML, Interaction overview diagram, Integrated testing

1 引言

基于UML开发的软件系统在分析、设计阶段中有各种模型图,这些模型图中包含了大量软件分析、设计信息,而这些信息不仅是软件实现的依据,也是软件测试的重要依据[1]。但随着软件的发展,UML标准也在不断完善和更新当中。UML1.0标准在描述顺序图等模型时缺乏对控制流的体现,UML2.0的标准中就新增加了交互纵览图(interaction overview diagram)[2]。这种模型关注并提高了对控制流的概览,其中控制流的每个节点都可以是一个交互图,这就更便于描述集成场景。利用交互纵览图进行集成测试,则可以大大提高集成测试的质量。

本文针对交互纵览图中不同消息之间的复杂联系和层次,设计了一种通过对交互纵览图拆分、组合,从而对交互纵览图测试模型进行消息方法遍历的集成测试。

2 交互纵览图测试模型

交互纵览图测试模型可以解决交互图测试模型缺乏流程控制这一关键点,即没有对循环、递归或条件序列提供良好的表示,跟踪交互图测试模型中的流会产生视觉上的混乱[3]。

交互纵览图测试模型将活动图测试模型和交互图测试模型相结合,将交互图测试模型以帧的形式代替活动图测试模型的节点,加入到活动图测试模型当中。由于交互纵览图描述了活动流程,也描述了对象之间的动态交互关系,这一特点用在测试上,就非常适合集成测试。本文主要研究基于交互纵览图测试模型的测试方法。

2.1 交互纵览图的形式化描述

定义1 UML交互图P可以表示为一个四元组:$P=(O,M,M_1,M_F)$。其中,O是SD中非空有穷的对象集合。

$M=\{m_1,m_2,m_3,\cdots,m_n\}$是$SD$上描述的有穷消息集合,每个消息定义为:$m_i\in guard\times m\_name\times paramenter\_list\times return\_value\times sender\times peceiver\times type$,其中,$guard$为消息发送的卫式条件;$m_name$为消息名;$paramenter_list$和$return_value$分别为消息的参数和返回值;$sender\in O$是$M$的发送者,$peceiver\in O$是$M$的接收者;$M$的类型$type\in\{syncall,asyncall,send,return\}$。$syncall$代表方法的同步调用;$asyncall$表示方法的异步调用;$send$表示信号的发送;$return$表示方法同步调用后的返回。

$\rightarrow$是一个表示M元素之间的时间先后的偏序关系。

$M_1,M_F\subseteq M$分别为交互图所表示的场景的所有可能的起始事件集合和终止事件集合;

定义2 UML交互纵览图层次结构可表示为一个三元组:$S=(P,f,sort)$,其中,

(1)P是非空交互图集。

(2)$f:P\rightarrow 2^P$,是求当前交互图的子交互图的函数,刻画交互图层次结构中的父子结点关系,体现交互图之间的顺序

*)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)联合资助。谢棠棠 助教,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;李 俊 助教,研究方向为软件工程、软件测试等。

依赖，$f(P)$定义交互图 P 的子交互图。令 $f^{+}=\bigcup_{i\geq 1} f^{i}$ 为关系 f 的传递闭包，则对 $p\in P$，$f+(P)$为 P 不包括自身的所有子孙的集合。

(3) *sort*: P→{ACTION, OR, AND, PSEUDO={init, fork, join, branch, final}}，是交互图结点类型映射。sort 作为交互图结点类型映射函数，其映射关系为：

①若 $f(p)=\varnothing \wedge sort(p)=$ACTION，则 p 为 ACTION 交互图，其执行过程不可中断；

②若 $f(p)\neq\varnothing \wedge sort(p)=$OR，则 p 为 OR 交互图，其执行过程可被中断。p 的子交互图集中的交互图之间不可并行触发，可以顺序触发或条件触发(按监护条件)，交互图之间为 OR 关系；

③若 $f(p)\neq\varnothing \wedge sort(p)=$AND，则 p 为 AND 交互图，其执行过程可被中断。p 的子交互图集中的交互图可以并行触发，交互图之间为 AND 关系；

④若 $f(p)=\varnothing \wedge sort(p)\in$PSEUDO，则 p 为伪交互图；伪交互图包括初始(init)、分叉(fork)、汇合(join)、分支(branch)、终止(final)。

定义 3 交互纵览图 D 可以表示为一个四元组：$D=(P, T, C, F)$[4]，其中，

(1)P 是非空有限交互图集；

(2)T 是非空有限变迁集；

(3)$F\subseteq(P\times T)\cup(T\times P)$为交互图与变迁之间的流关系；

(4)$C(t)$是变迁 t 的条件表达式。

定义 4 $D=(P, T, C, F)$是一个交互纵览图。如果 $X=P\cup T$，对任意 $x\in X$，$x=\{y\in P|(y,x)\in F\}$称为 x 的前集，$x=\{z\in P|(x,z)\in F\}$称为 x 的后集。

定义 5 $D=(P, T, C, F)$是一个交互纵览图。D 中的当前交互图集 v 是 P 的一个任意子集。对于任意变迁 $t\in T$，如果$'t\subseteq v$并且 $C(t)$的值为真，则变迁 t 对于用例集 v 是可触发的变迁集，用 enable(v)表示 v 中所有可触发的变迁集。

定义 6 $D=(P, T, C, F)$ 是一个交互纵览图。变迁 $t\in T$ 能从交互图集 v 触发的条件：$t\in$enable(v)，$(v-'t)\cap t'=\varnothing$，则变迁 $t\in T$ 能从当前交互图集 v 触发记为：$v[t>$。在当前交互图集 v，变迁 t 将 v 变成 v'，记为：$v[t>v'$，新状态 $v'=(v-'t)\cup t'$。

2.2 对交互图之间的联系进行描述

交互图之间的业务流程是一个复杂的动态变化过程，它描述了交互图之间的流转换，其中包含多种不同的变迁方式。在交互纵览图中可以使用如下概念对交互图间的联系进行描述。

定义 7(顺序变迁) $\exists t_1, t_2\in T$，若 $v[t_1>\wedge\neg v[t_2>\wedge v'[t_2>\wedge v[t_1>v'$，则变迁 t_1 和 t_2 有顺序关系，为一组顺序变迁。

定义 8(并发变迁) $\exists t_1, t_2\in T$，若 $v[\{t_1, t_2\}>$，则变迁 t_1 和 t_2 有并发关系，为一组并发变迁。其中 $v[\{t_1, t_2\}>$表示变迁 t_1 和 t_2 在 v 下可同时发生。

性质：分叉变迁和汇合变迁一定是一组并发变迁。

定义 9(分叉变迁) $\forall p\in P\wedge sort(p)=$for k，若 $\exists t_1, t_2, \cdots, t_n\in T, t'_1=\{p\}, t_i=\{p\}, v[\{t_2, \cdots, t_n\}>$，则称 t_i 为 t_1 的一组分叉变迁，其中 $2\leqslant i\leqslant n$。

定义 10(汇合变迁) $\forall p\in P\wedge sort(p)=join$，若 $\exists t_1, t_2, \cdots, t_n\in T, t'_i=\{p\}, 't_n=\{p\}, v[\{t_1, \cdots, t_{n-1}\}>$则称 t_i 为 t_n 的一组汇合变迁，其中 $1\leqslant i\leqslant n-1$。

定义 11(分支变迁) $\forall p\in P\wedge sort(p)=branch$，若 $\exists t_1, t_2\cdots, t_n\in T, t'_1=\{p\}, 't_i=\{p\}$，则称 t_i 为 t_1 的一组分支变迁，其中 $2\leqslant i\leqslant n$。

2.3 交互纵览图作为测试模型的可行性

首先，交互纵览图具有图的特性，可以采用图的遍历算法对图每个边或节点遍历。其次，交互纵览图有一个开始节点和结束节点，可以用作遍历的开始和结束。每个交互帧可看作一个节点，每个消息可以看作是图的一个边，图中的各种条件用于作为图遍历的选择标准。由此，可对图中各种可能发生的路径进行遍历，这种路径恰好就是我们业务的流程路径。

3 拆分、组合、覆盖准则

本文沿用传统的白盒测试的思想，针对交互纵览图的特性提出了一种拆分、覆盖、组合的遍历准则，其准则如下。

准则 1 针对交互纵览图的每个帧节点用一个节点(我们称为标准节点)$N1\cdots Nn$ 代替，将原有的一个交互纵览图变成一个活动图和多个交互图。

准则 2 对每个拆分子图采用如下覆盖准则，得到测试场景：

准则 2.1：对子顺序图按从上到下的顺序和跳转条件原则遍历每一个消息；

准则 2.2：对子协作图按起点不重复和跳转条件原则遍历每个消息；

准则 2.3：对子活动图按起止节点路径覆盖完全和跳转条件原则遍历每个消息。

准则 3 以活动图的测试场景为主线，将每个标准节点用各个替代的子图场景替代，重新组合成多个完整的场景。

4 交互纵览图的测试算法

针对上面讲的几种测试准则，我们可以设计以下算法，实现对整个交互纵览图的遍历，所得到的测试场景正是在进行集成测试时关注的重点。

4.1 拆分算法

对整个交互纵览图节点进行编历，把帧节点用标准节点代替，把帧节点变成一个子图。

```
Procedure split(diagram:Immdiagram)
 Begin
   For I:=1 to n do //n 表示节点数
     If node[i]. type="Frame" then
     Begin
     Diagram[i]=Node[i];
      Node[i]. Type="Standard";
     End;
End;
```

4.2 遍历算法

拆分后得到一个活动图和多个顺序图或协作图。遍历时针对各个图不同的特点进行遍历。

4.2.1 活动图遍历

由于活动图具有分叉、循环、并发等情况，而且每个活动图有开始和结束节点，整个图的遍历必须对上面三种情况区别对待，提取出测试场景。

```
Procedure PathActivity(diagram:Immdiagram);
begin
GetNode(diagram, node);//获取图的节点和转换
While node[i]〈〉Endnote do //如果当前节点不是尾节点
        begin
        Record(node[i]);//记录当前节点
        if node[i]is Parellizat then //遇到并发，并发部分节点必须
        全部执行
        begin
```

```
        SearchnextParellizat (Parell);//获取并发的配对节点
        Get(node[i],Parell);//获取并发路径
        RecordParell(node);//记录并发节点的路径
        Connect(node[i], bingfa); //连接并发的节点
        end;
        If node[i]is branch then //遇到分叉
          Search(node[i]. NextNode2); //遇到分叉递归另一个节点
        If node[i]is circle then //遇到循环,让循环节点只执行一次
          begin
          cut(node[i]. NextArrow2); //断开循环
          Search(node[i]. NextNode2); //递归循环的一个节点
          end;
        Search(node[i]. NextNode1);//递归下一个节点
        End;
SortActivity(node); //对节点序列进行排序
ScenceActivity(node); //获取场景顺序
ScriptActivity(node);//获取活动图的脚本信息,包含消息的说明或限制,以及发出者
end;
```

4.2.2 顺序图遍历

由于顺序图的纵向表示时间,我们可以根据消息的纵向位置判断消息的先后,形成消息队列[5]:

```
Procedure PathSequence(diagram:Immdiagram);
var
Association:array[0…n]of string;
begin
GetAssociation(diagram,Association); //获取图的消息
SortSequence(Association); //对消息序列进行排序
ScenceSequence(Association); //获取场景顺序
ScriptSequence(Association); //获取顺序图的脚本信息,包含消息的说明或限制,以及发出者
end;
```

4.2.3 协作图遍历

由于协作图的消息队列自带执行顺序,我们就可以对消息进行排序,找出消息的执行序列:

```
Procedure PathCollaboration(diagram:Immdiagram);
var
Event:array[0…n]of string;
begin
GetEvent(diagram, Event); //获取图的事件
SortCollaboration(Event); //对消息序列进行排序
ScenceCollaboration(Event); //获取场景顺序
ScriptCollaboration(Event); //获取协作图的脚本信息,包含消息的说明或限制,以及发出者
end;
```

4.3 组合算法

通过上面的遍历算法得到多条测试路径,以活动图为主线,把活动图中的每个标准节点用相应的子图路径来替代,得到完整的路径。

```
Procedure Assemble (diagram:Immdiagram);
Begin
GetActivePath(diagram,path[0]); //获取活动图的路径
For I:=1 to n do
    If diagram[i]<>Null then
    Begin
    Getpath(diagram[i], path[i]); //获每个子图的路径
    AssembleToActive(path[0],path[i]); //合并每个子图到活动图当中
    end;
End;
```

5 模拟实现

ModelMaker 是 Borland 公司推出的 UML 建模工具,常被认为是一个 UML 图形工具或是 Delphi Case 工具。然而,它比一般的图形工具和 Case 工具要快得多。同时,它还可以自动生成一些智能式的代码,具有良好的可扩展性、支持 UML 图、逆向生成与分解的双向代码管理等等。ModelMaker9.0 版本开始对 UML2.0 支持,同时由于它的 ToolsApi 模块提供了对 UML 图操作的接口,我们可以利用 RoolsApi 对 UML 图进行随意操作,所以本实验选择在 ModelMaker9.0 环境中进行。考虑到篇幅,本文仅以活动图结合顺序图为例进行实验。

本实验是以自动售货机出售苏打水为例,从交互纵览图中提取测试用例,如图 1。

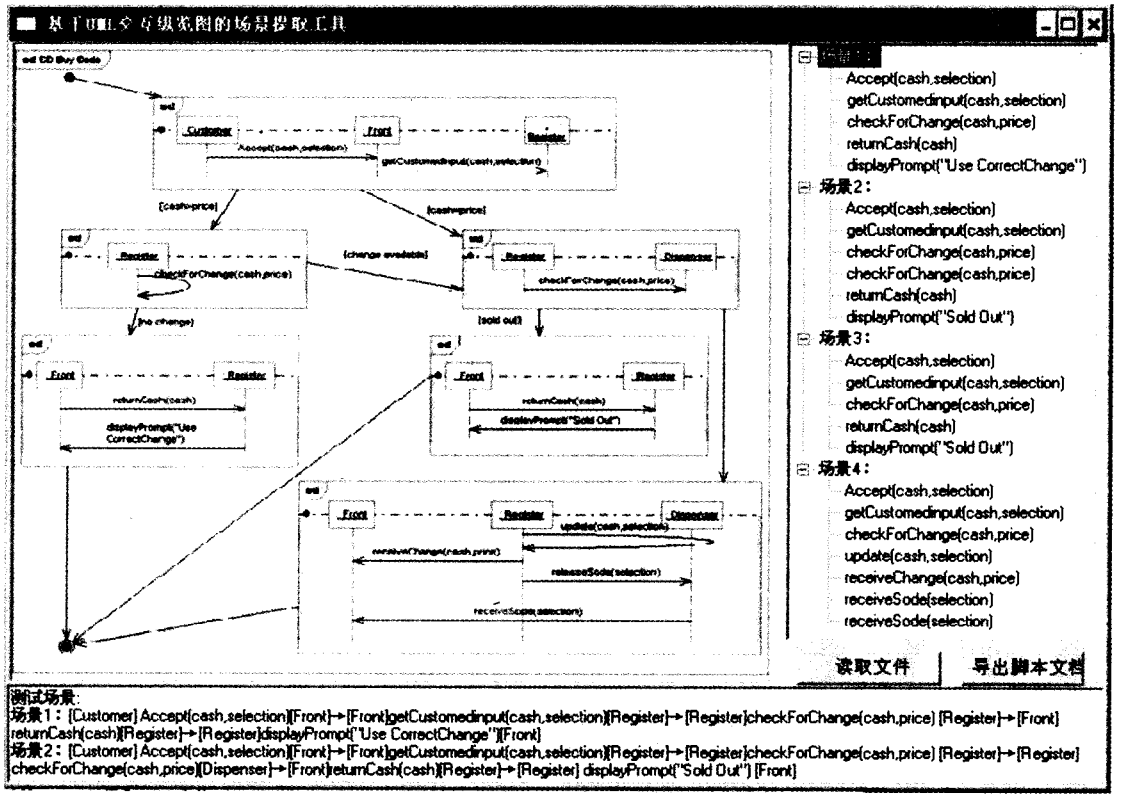

图 1 自动售货机出售苏打水测试场景

从上面试验得到如下场景:

场景 1:[Customer]Accept(cash,selection)[Front]→[Front]getCustomedinput(cash,selection)

(下转第 290 页)

序号相同，则启动它所包装的线程进行实际的执行，完成后将锁序号加1，这样系统中的线程就只能按照测试线索的顺序获得锁来执行直到结束。将该辅助对象和包装对象加入系统，此时系统的测试步骤如下：

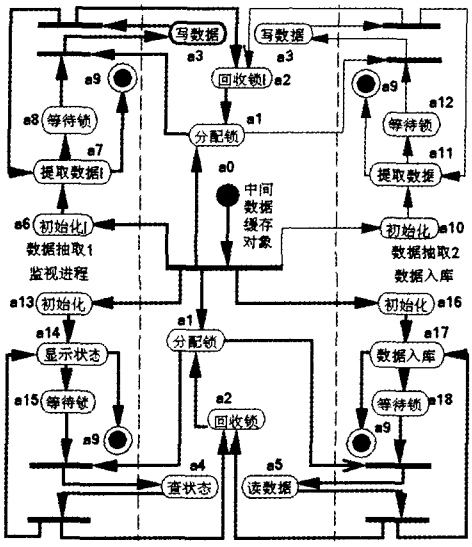

图5　线索 s1 示意图

1）启动系统的包装对象；

2）启动辅助线程对象，锁的初始分配序号为1；

3）系统开始正常执行，直到结束；

4）检查系统执行结果，给出测试结果。

5.3　测试结果

很容易验证测试线索 *s1* 和 *s2* 满足前面提出的三个测试策略。按照测试线索 *s1* 和 *s2* 执行测试，找出了原软件中的一个并发执行错误。检查发现数据读入对象读完数据但还未设置好状态时就释放了锁，这时数据监控对象读取了数据状态，导致数据监控对象显示错误的状态，即图5上的动作 *a3* 出错(粗线条表示)。在本实例中，按照测试线索 *s1* 或 *s2* 都能发现该并发错误，说明本文生成的测试线索具有较好的代表性。虽然该数据采集软件在实际运行过程中又发现了一些错误，但是经分析它们均不是由于并发而导致的错误。这个实例在一定的程度上表明本建模方法和测试算法对于查找软件中的并发错误有一定效果和帮助。

结论和展望　本文在分析并发控制原理的基础上，给出了并发软件的三个基本测试策略，这三个策略覆盖了进程同步、多进程并发和关键操作等并发控制的关键点，对测试并发软件具有较好的指导性。接着本文提出了利用活动图来对系统的并发控制流建模的方法，在分析并发测试的难点之后提出了一种测试线索集的生成算法，并证明了该算法生成的测试线索集能有效地避免测试过程中进程对象的组合爆炸问题，提高了其实际应用价值。而且按照该测试线索集测试软件，可以满足本文所提出的三个并发软件测试策略，说明本算法产生的测试线索集对发现软件中的并发错误具有较好的指导意义。文中最后还通过对比得出本方法在适应性、易用性和稳定性方面优于 Kim 等人提出的基于状态图的测试方法，并结合实际软件给出了应用该方法的样例，验证了其可行性和有效性。

本文在介绍测试方法时主要考虑的是单个关键资源对象和进程对象只访问单个关键操作的情况。在实际的并发软件中，关键资源对象可能有多个，进程对象也可能要访问多个关键操作，在本文的基础上，进一步的工作将要考虑这种情况下测试线索的生成方法和测试方法。

参 考 文 献

1　Kim S-K, Wildman L, Duke R. A UML Approach to the Generation of Test Sequences for Java-based Concurrent Systems. In: Software Engineering Conference, 2005. Proceedings. 2005 Australian 29 March-1 April 2005. 100～109

2　孙钟秀，韩杰，谢立，等. 基于有限状态进程的事件约束定义[J]. 软件学报，2002，13(11)：2163～2166

3　曾一，赵炜，张利武. 基于 UML 活动图生成功能测试线索的方法[J]. 计算机工程与设计，2006(12)

4　赵炜. 基于 UML 提取系统测试线索方法的研究[C]：[硕士论文]. 重庆大学计算机学院，2006. 04

(上接第285页)

[Register]→[Register] checkForChange (cash, price) [Register]→[Front] returnCash(cash)

[Register] → [Register] displayPrompt ("Use CorrectChange")[Front]

场景 2：[Customer] Accept(cash, selection)[Front]→[Front]getCustomedinput(cash,selection)

[Register]→[Register] checkForChange (cash, price) [Register]→[Register] checkForChange (cash, price) [Dispenser]→[Register] returnCash (cash) [Register]→[Register]displayPrompt("Sold Out")[Front]

场景 3：[Customer] Accept(cash, selection)[Front]→[Front]getCustomedinput(cash,selection)

[Register]→[Register] checkForChange (cash, price) [Dispenser]→[Front]returnCash(cash)

[Register]→[Register] displayPrompt ("Sold Out") [Front]

场景 4：[Customer] Accept (cash, selection) [Front] [Front]getCustomedinput(cash,selection)

[Register]→[Register] checkForChange (cash, price) [Dispenser][Register]update (cash, selection)[Register]→[Register] receiveChange (cash,price)[Front]→[Register]releaseSoda(selection)[Dispenser]→[Dispenser] receiveSode(selection)[Front]

结束语　本文在分析多种测试模型的基础上，找到更为适合集成测试的测试模型并结合传统的白盒测试方法，总结出一种基于拆分、覆盖、重组的集成测试方法，在选择测试模型和改进方法方面较以前提高。

参 考 文 献

1　Binder R V. Testing Object-oriented System: Models, Patterns, and Tools. Addison-Wesley, 2000

2　UML Specification 2.0. http://www.uml.org, 2007

3　Booch G, Rumbaugh J, Jacobson I. The Unified Modeling Language User Guide. Addison—Wesley, 2001

4　袁洁松，王林章，李宣东，等. UMLTGF：一个基于灰盒方法从 UML 活动图生成测试用例的工具. 计算机研究与发展，2006，43(1)：46～53

5　Lund M S, StØlen K. Deriving tests from UML 2.0 sequence diagrams with neg and assert[c]. In: Proceedings of A the 2006 International Workshop on Automation of Software test, May 2006. 22～28