Journal of Computer Applications
计算机应用,2014,34(1):276 -280
ISSN 1001-9081
CODEN JYIIDU
2014-01-10
http://www.joca.cn

文章编号:1001-9081(2014)01-0276-05
doi:10.11772/j.issn.1001-9081.2014.01.0276

基于 UML 模型的系统级测试用例生成方法

冯秋燕*
(河南财经政法大学 图书馆,郑州 450000)
(*通信作者电子邮箱 fqy_03@126.com)

摘　要:采用基于 UML 模型的软件测试方法,主要整合用例图与顺序图进行系统级的软件测试。首先提出用例执行图(UEG)的生成算法、顺序执行图(SEG)的生成算法,及基于 UEG 和 SEG,生成系统测试图(STG)的算法;其次,根据制定的三层次准则,遍历 UEG、SEG、STG 生成测试用例,主要解决交互错、场景错、用例执行错和用例依赖错等问题。最后,经实例分析和实验验证,该方法可以基于用例图和顺序图进行系统级的软件测试。

关键词:UML 模型;软件测试;用例图;顺序图;测试用例
中图分类号:TP311.5　　**文献标志码**:A

System-level test case generating method based on UML model

FENG Qiuyan*
(*Library, Henan University of Economics and Law, Zhengzhou Henan* 450000, *China*)

Abstract: With the software testing methods based on Unified Model Language (UML) models, use case diagrams and sequence diagrams were integrated for system testing. Firstly, three algorithms were proposed including the algorithm for generating Use case diagram Execution Graph (UEG), the algorithm for generating Sequence diagram Execution Graph (SEG) and the algorithm for generating System Testing Graph (STG) based on UEG and SEG. Then, the UEG, SEG and STG were traversed to generate test cases for system-level testing based on specified three-level criteria, mainly to detect interaction, scenario, use case and use case dependency faults. Finally, the experimental validation shows that the solution can do system-level software testing based on use case diagram and sequence diagram.

Key words: UML model; software testing; use case diagram; sequence diagram; test case

0　引言

针对不同的研究目的、被测软件的具体特征,研究人员提出了状态机模型[1]、马尔可夫链模型[2]、UML 模型[3]等。随着基于 UML 模型的广泛应用,基于 UML 模型的软件测试逐渐成为软件测试的发展趋势和主流。文献[4]提出了 TOTEM 系统测试方法(Testing Object oriented systems with the unified Modeling language system testing methodology),该方法分析用例之间的顺序依赖关系进而生成测试用例,其本质是一种预先明确初始状态的半自动化场景覆盖方法;文献[5-7]基于 UML 交互模型(顺序图(Sequence Diagram,SD)和协作图)主要检测对象间的交互错;其中,文献[5]仅针对实时系统;文献[6]采用类别划分法使用手动方式,基于顺序图做组件间的交互测试;文献[7]基于 UML 用例图(Use case Diagram,UD)与消息序列图做集成测试,检测被测系统组件间的交互错。文献[8]首先构造映射系统架构的图,然后遍历该图进而采用类别划分法手动的生成测试用例。

本文将被测系统转换为系统测试图,即首先将用例图转换为用例执行图(Use case diagram Execution Graph,UEG)、顺序图转换为顺序执行图(Sequence diagram Execution Graph,SEG),整合 UEG 与 SEG 为系统测试图(System Testing Graph,STG);同时,析取类图、数据词典等的相关约束信息并将其加入 STG 中。然后基于特定的三层次覆盖准则和错误模型,遍历 UEG、SEG 与 STG,得到系统级测试用例。

1　相关概念

定义 1　用例执行图(UEG)。$G_{UEG} = \langle V_{UEG}, \Sigma_{UEG}, q0_{UEG}, F_{UEG} \rangle$,其中:

$V_{UEG} = U \cup A, U = \{U_1, U_2, \cdots, U_n\}$ 是一个有限的节点集合,$U_i(1 \leqslant i \leqslant n)$ 表示一个用例;$A = \{A_1, A_2, \cdots, A_m\}$ 是一个有限的节点集合,$A_j(1 \leqslant j \leqslant m)$ 表示一个参与者。

$\Sigma_{UEG} = AU \cup UD, AU = \{A \times U\} \cup \{U \times A\}$ 是参与者 $A_j(A_j \in A)$ 与用例 $U_i(U_i \in U)$ 之间的关系。

$UDep = \{U \times U\}$ 是两个用例 U_l、$U_k(U_l, U_k \in U)$ 之间的依赖关系。

$q0_{UEG} \in A$ 是用例图开始节点集合即源参与者节点集合,满足 $q0_{UEG} \times U \in \{A \times U\}$。

$F_{UEG} \in A$ 是用例图终止节点集合即终止参与者集合,满足 $U \times F_{UEG} \in \{U \times A\}$。

定义 2　顺序执行图(SEG)。$G_{SEG} = \langle V_{SEG}, \Sigma_{SEG}, q0_{SEG}, F_{SEG} \rangle$ 是一个有向图,其中:

$V_{SEG} = \{V_1, V_2, \cdots, V_n\}$ 是消息节点的集合,$V_j = \langle preC, trigger\_condition, M_i, [c(\lambda)], postC \rangle$ 是根据消息 M_i 建立的节点。其中:$preC$ 是 M_i 的前置约束;$trigger\_condition$ 是消息 M_i

收稿日期:2013-06-27;**修回日期**:2013-08-25。
作者简介:冯秋燕(1988-),女,河南新乡人,助理工程师,硕士研究生,主要研究方向:现代软件工程技术、数据挖掘。

的触发条件;*postC* 是 M_i 的后置约束;$c(\lambda)$ 是一个可选项,表示 M_i 的监护条件。

Σ_{SEG} 是边的集合,$Pre(V_j)$ 表示节点 V_j 的前驱,$Suc(V_j)$ 表示节点 V_j 的后继,$(Pre(V_j),V_j) \in \Sigma_{SEG}$,$(V_j,Suc(V_j)) \in \Sigma_{SEG}$。

$q0_{SEG}$ 是 SEG 的唯一入口节点。

F_{SEG} 是 SEG 的终止节点集合。

定义 3　系统测试图(STG)。$G_{STG} = \langle V, \Sigma, q0, F\rangle$,其中:

$V = V_{UEG} \cup V_{SEG}$ 是 STG 中所有节点的集合;

$\Sigma = \Sigma_{UEG} \cup \Sigma_{SEG} \cup \Sigma_G$,$\Sigma_G = (V_{UEG} \times q0_{SEG})$ 是从 UEG 到 SEG 的边即表示从 UEG 中的用例节点到与其相关的 SEG 间的关联;

$q0 = q0_{UEG}$ 是 STG 的开始节点集合;

$F = F_{UEG} \cup F_{SEG}$ 是 STG 的终止节点集合。

定义 4　测试用例(T)。系统测试用例 $T = \langle ID, Input, OutExpect, preCon, postCon\rangle$,其中:*ID* 是唯一标识;*Input* 是测试用例的触发初始输入;*OutExpect* 是期盼输出结果;*preCon* 是系统所处的开始状态;*postCon* 是系统所处的后置状态。

2　系统测试图 STG 的生成

本文在生成 STG 的过程中,主要做以下工作:1)由用例图生成用例执行图 UEG;2)由顺序图生成顺序执行图 SEG;3)整合 UEG 与 SEG 生成 STG。

2.1　用例执行图 UEG 的生成

根据定义 1,分析以下情况:1)若用例图无参与者作为终止节点,则 F_{UEG} = null;2)若一个参与者既属于 $q0_{UEG}$ 又属于 F_{UEG},则将其分别记入 $q0_{UEG}$ 和 F_{UEG} 中。下面给出用例执行图 UEG 的生成算法:

算法 1　用例执行图 UEG 生成算法 Transform_UDToUEG。

输入　用例图 UD;

输出　用例执行图 UEG。

1)对用例图中的每个参与者建立参与者节点 A_i;对用例图中的每个用例建立用例节点 U_j。

2)对所有节点 A_i 进行分析,标记出 $q0_{UEG}$ 和 F_{UEG}。

3)对每个 A_j:若 $A_j \in q0_{UEG}$,则建立 A_j 到与 A_j 相关的 U_m 的边;若 $A_j \in F_{UEG}$,则建立与 A_j 相关的 U_m 到 A_j 的边。

4)对每个 U_l,建立 U_l 与其相关的 U_k 间的有向边,算法终止。

算法 1 中,第 2)步主要建立初始参与者节点集合 $q0_{UEG}$ 与终止参与者节点集合 F_{UEG};第 3)步主要建立参与者 A_i 与其相关的用例 U_m 间的关系;第 4)步主要建立用例依赖关系,若存在从 U_l 到 U_k 的边,则说明 U_k 依赖于 U_l;算法 1 的时间复杂度为 $O(n)$;

2.2　顺序执行图 SEG 的生成

根据定义 2,作以下分析:1)SEG 的开始节点 $q0_{SEG}$ 用于存放顺序图中第一条消息的触发初始输入;2)SEG 的终止节点 F_{SEG} 用于存放顺序图中所有结束状态情况即顺序图中每种场景的最后一条消息的后置状态。下面给出用例执行图 SEG 的生成算法:

算法 2　用例执行图 SEG 生成算法 Transform_SDToSEG。

输入　顺序图 SD;

输出　顺序执行图 SEG。

1)对顺序图中的每个消息建立节点 V_i;

2)解析顺序图 XML 模型[9],找出第一个消息节点 V_0 和终止消息节点集合 $FV = \{V_f \mid 1 \leqslant f \leqslant n\}$;建立初态节点 $q0_{SEG} = V_0.preC$;对 $V_f \in FV$,建立终态节点集合即 $F_{SEG} = F_{SEG} \cup \{V_f.postC\}$;

3)按照顺序图中的场景,建立顺序图中各节点间的边 Σ_{SEG},算法终止。

算法 2 中,第 1)步对顺序图中的消息建立节点,同一条消息的多次调用只建立一个节点;第 2)步是在 SEG 图中建立顺序图中所描述的所有场景;算法 2 的时间复杂度为 $O(n)$。

2.3　系统测试图 STG 的生成

根据定义 3,做以下分析:1)STG 的开始节点 $q0$ 为 UEG 的开始节点 $q0_{UEG}$;2)STG 的终止节点 F 为 UEG 的终止节点 F_{UEG} 与 SEG 终止节点 F_{SEG} 的集合。下面给出用例执行图 STG 的生成算法:

算法 3　系统测试图 STG 生成算法 Com_UEGSEGToSTG。

输入　用例执行图 UEG 和顺序执行图 SEG;

输出　系统测试图 STG。

1)建立 STG 的节点 $V = V_{UEG} \cup V_{SEG}$。

2)建立 STG 的开始节点 $q0 = q0_{UEG}$。

3)建立 STG 的终止节点 $F = F_{UEG} \cup F_{SEG}$。

4)对 V_{UEG} 中的所有节点建立其与 $q0_{SEG}$ 间的有向边 Σ_G;建立 STG 的边 $\Sigma = \Sigma_{UEG} \cup \Sigma_{SEG} \cup \Sigma_G$;算法终止。

算法 3 中,第 1)步中 STG 的节点为 UEG、SEG 中的节点集合;第 4)步中 STG 的边为 UEG、SEG 中的边以及 V_{UEG} 与 $q0_{SEG}$ 之间边 Σ_G 的集合,算法 3 的时间复杂度为 $O(n)$。

3　测试用例的生成

在做系统级测试的过程中,本文规定三层次遍历准则,以得到相对完善的测试用例:第一个层次为遍历 UEG,覆盖所有路径,由此生成的测试用例可以检测初始化错;第二个层次为遍历 UEG 中与用例节点相关的 SEG,生成测试用例可以检测交互错[10];第三个层次为从 STG 的所有路径中识别出用例依赖关系,由此生成的测试用例可以检测用例依赖错。为此,本文给出以下三个覆盖准则即三层次覆盖准则。

覆盖准则 1(C1)　覆盖所有用例的准则:给定一个测试集合 T 和一个用例图,T 必须使每个用例执行至少一次。

覆盖准则 2(C2)　覆盖顺序图中所有消息路径的准则:给定一个测试集合 T 和一个顺序图,T 必须使顺序图中的每条消息路径执行至少一次。

覆盖准则 3(C3)　覆盖用例图中所有用例依赖关系的准则:给定一个测试集合 T 和一个用例图,T 必须覆盖用例图中所有用例依赖关系。

下面依据三层次准则分 3 节介绍。

3.1　遍历 UEG 并生成测试用例集合 T_{UEG}

每个用例都是通过对象之间的交互、协作实现的,用例的成功执行需要满足以下条件:1)在开始执行时,用例所涉及

的每个对象均处于预置的正确状态;2)在给定不同的输入数据与系统状态时,用例应能给出所对应的正确执行结果;为满足以上条件与覆盖准则 1(C1),本文给出如下算法:

算法 4 UEG 测试用例生成算法 TestSetGeneration_UEG。

输入 用例执行图 UEG;

输出 测试用例 T_{UEG}。

1)对所有的用例 $U_i \in G_{UEG}$,查找所有的参与对象即 O_i = FindObjects(U_i)。

2)对所有的参与对象 $O_{ij} \in O_i$,查找 O_{ij} 的所有属性即 M = FindAttributes(O_{ij}),查找 M 的输入域即 In = IdentifyInputDomain(M) 和输出域即 Out = IdentifyOutputDomain(I, M);根据输入值 $in \in In$ 与输出值 $out \in Out$ 生成测试用例即 t = SelectTestCase(in,out);将 t 加入测试用例集合 T_{UEG} 中,即 $T_{UEG} = T_{UEG} \cup t$。

3)从 UEG 中,查找 U_i 到相关的 SEG 间的关联,并标识关联 α 即 $\alpha = U_i \rightarrow G_{SEG}$;调用算法 5 即 T_1 = TestSetGeneration_SEG(α);将 T_1 加入测试用例集合 T_{UEG} 中,即 $T_{UEG} = T_{UEG} \cup T_1$,算法终止。

算法 3 中,第 2)步主要检测初始化错即给定每个对象一个特定的状态是否能得到正确的输出结果;第 3)步主要检测用例执行错,用例是由对象间的交互实现的,由顺序图进行描述,故需要查找与用例相关的顺序图。

3.2 遍历 SEG 并生成测试用例 T_{SEG}

顺序图描述不同对象间的交互,经常会发生以下交互错[11]:对特定消息反馈错误的信息、消息被不正确的或尚未初始化的对象接收、错误的消息被传送给正确的对象、传参错、不正确的输出等;一个顺序图通常描述多种操作场景,每个场景对应不同的消息调用顺序,对给定场景经常会出现下述错误[10]:消息没有按照预置的消息顺序发生、场景的异常终止等。

为检测以上错误,根据覆盖准则 2(C2),本文对 SEG 进行遍历,根据 SEG 中的所有路径,生成测试用例 T_{SEG},给出算法 5 如下(其中 n_j 标识当前所访问的节点):

算法 5 SEG 测试用例生成算法 TestSetGeneration_SEG。

输入 顺序执行图 SEG;

输出 测试用例 T_{SEG}。

1)深度遍历 SEG 中的所有路径即 P = EnumerateAllPaths(G_{SEG})。

2)对每条路径 $P_i \in P$,从节点 Start 出发即 n_j = Start;查找 n_j 的前置约束条件即 $preC_i = n_j.$ getPreCond();t_i = null;对路径 $P_i \in P$ 中的每个节点 n_j,提取与 n_j 相关的消息即 $e_i = n_j.$ getMessage();确定 e_i 的测试用例即 $t_{ij} = \langle n_j.preC,\ n_j.Input,\ n_j.Output,\ n_j.[c(\lambda)],\ n_j.postC\rangle$;将 t_{ij} 按序加入 t_i 中,即 $t_i = t_i \cup t_{ij}$。

3)将 t_i 按序加入 T_{SEG} 中,即 $T_{SEG} = T_{SEG} \cup t_i$。

算法 5 中,第 2)步主要针对 SEG 中的每条路径即顺序图中的每个特定场景,建立该场景的测试用例 t_i,该用例可检测交互错和场景错;算法 5 的时间复杂度为 $O(n^2)$。

3.3 遍历 STG 并生成测试用例 T_{STG}

用例之间有依赖关系的,系统应能正确地根据依赖关系执行。系统执行图 STG 描述了参与者与用例以及用例间的依赖关系,为得到系统级的测试用例,本文给出遍历 STG 并生成测试用例 T_{STG} 的算法如下:

算法 6 STG 测试用例生成算法 TestSetGeneration_STG。

输入 系统执行图 STG;

输出 测试用例 T_{STG}。

1)初始化 T_{STG} = null;深度遍历 STG 中的所有路径即 $Paths$ = EnumerateAllPaths(G_{STG})。

2)遍历每条路径 $Pth_i \in Paths$,若 Pth_i 存在用例依赖关系,则标识依赖的用例

$UDep$ = FindAllUseCaseDependency(Pth_i)

对 $UDep$ 中的所有用例 $UDep_i \in UDep$,对每个 $UDep_{ij} \in UDep_i$,查找 $UDep_{ij}$ 中所涉及的所有参与者到用例的关联,即 AUD = FindActorToUC($UDep_{ij}$);对 AUD 中的每个关联 $au_k \in AUD$,查找 au_k 的输入域即 InD = GetInputDomain(au_k) 和输出域即 $OutD$ = GetOutputDomain(au_k);构成测试用例即 t = SelectTestCase(InD,$OutD$, $UDep_{ij}.PreC$, $UDep_{ij}.PostC$);将 t 加入 T_{STG} 中,即 $T_{STG} = T_{STG} \cup t$,算法终止。

算法 6 中,第 2)步实质是一个二重循环,算法时间度为 $O(n^3)$,主要解决用例图中的用例依赖关系,解决用例依赖错。

4 实例分析

本文方法的具体处理过程为:赋予用例图 UD 与顺序图 SD 的 XMI 元模型适当的 Schema 约束得到等价的 XML 模型[12],解析 XML 模型并识别其中的消息信息[13],同时解析 OCL 约束文件并提取所需信息,采用本文算法将 UD 转换为 UEG、将 SD 转换为 SEG、整合 UEG 与 SEG 为 STG,进而遍历 UEG、SEG、STG 得到测试用例,将元模型进行实现,并采用依赖注入的方法[11],用先前生成的测试用例发现实现过程中出现的错误。以在线购买系统为例,说明本文方法的执行过程。图 1 为在线购买系统的一个用例,图 2 为其中的密码认证顺序图。

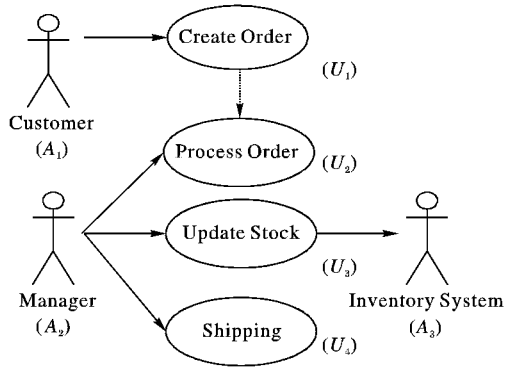

图 1 购买系统用例

对上述在线购买系统实例,生成测试用例的过程及结果如下:

1)根据定义 1 和算法 1,得到与用例图 1 对应的 UEG,如图 3 所示;由图 3 可看出参与者与用例及用例与用例间的关系。

2)根据定义 2 和算法 2,得到与顺序图 1 对应的 SEG,如

图 4 所示。为便于理解,在图 2 中,本文将消息用 $m_i(1 \leqslant i \leqslant 8)$ 标注,对同一条消息的多次调用只标注一次,如图 4 所示,在生成 SEG 时,对同一条消息的多次调用也只建立一个消息节点。

图 4 中,SEG 中,始发节点 $StateX = V_1.preC$,$StateY = \{V_2.postC, V_4.postC, V_5.postC, V_9.postC, V_{10}.postC\}$,定义 2 中,$q0_{SEG} = StateX$,$F_{SEG} = StateY$;

3)根据定义 3、算法 3、图 3 和图 4,得到对应的 STG;密码认证过程只是在线购买系统的一小部分,只和用例 A_1 相关且相关性为 α,故只截取该部分的 STG,如图 5 所示。

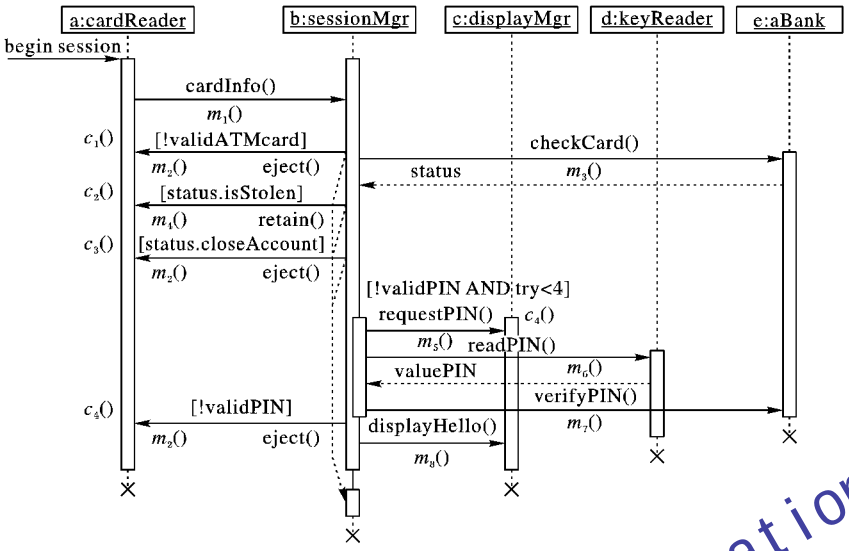

图 2　密码认证顺序图

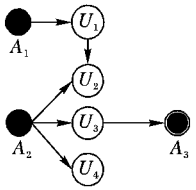

图 3　图 1 对应的 UEG

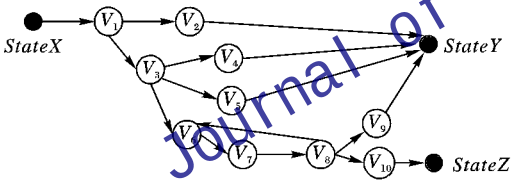

图 4　图 2 对应的 SEG

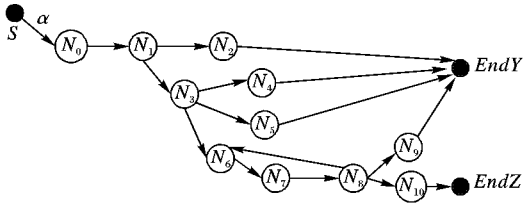

图 5　图 3 和图 4 所对应的 STG

在图 5 中,根据定义 3 和图 4,STG 的开始节点 S 与图 4 中 A_1 对应、N_0 与图 4 中 $StateX$ 对应、N_1 至 N_{10} 分别与图 4 中 V_1 至 V_{10} 对应、$EndY$ 和 $EndZ$ 分别与图 4 中 $StateY$ 和 $StateZ$ 对应,即 $S = A_1$,终止节点集合 $F = \{EndY, EndZ\}$,图 4 中的开始节点 $StateX$ 演变为 STG 中的普通节点 N_0,S 与 N_0 的相关性为 α。

4)根据算法 4 与图 3,UEG 中的测试用例如下:

$T^{(1)}_{UEG} = \langle 1, U_1.In, U_1.Out, U_1.In, U_1.Out \rangle$

$T^{(2)}_{UEG} = \langle 2, U_2.In, U_2.Out, U_2.In, U_2.Out \rangle$

$T^{(3)}_{UEG} = \langle 3, U_3.In, U_3.Out, U_3.In, U_3.Out \rangle$

$T^{(4)}_{UEG} = \langle 4, U_4.In, U_4.Out, U_4.In, U_4.Out \rangle$

$T^{(5)}_{UEG} = \langle 5, A_1.In, U_1.Out, A_1.In, U_1.Out \rangle$

$T^{(6)}_{UEG} = \langle 6, A_2.In, U_2.Out, A_2.In, U_2.Out \rangle$

$T^{(7)}_{UEG} = \langle 7, A_2.In, U_3.Out, A_2.In, U_3.Out \rangle$

$T^{(8)}_{UEG} = \langle 8, A_2.In, U_4.Out, A_2.In, U_4.Out \rangle$

$T^{(9)}_{UEG} = \langle 9, A_3.In, U_3.Out, A_3.In, U_3.Out \rangle$

其中:$T^{(1)}_{UEG}$、$T^{(2)}_{UEG}$、$T^{(3)}_{UEG}$ 分别对用例 $U1$、$U2$、$U3$ 进行测试;$T^{(4)}_{UEG}$、$T^{(5)}_{UEG}$、$T^{(6)}_{UEG}$、$T^{(7)}_{UEG}$、$T^{(8)}_{UEG}$、$T^{(9)}_{UEG}$ 分别对关联 $A_1 \rightarrow U_1$、$A_2 \rightarrow U_2$、$A_2 \rightarrow U_3$、$A_2 \rightarrow U_4$、$U_3 \rightarrow A_3$ 进行测试。

5)根据算法 5 与图 4,本文首先列出 SEG 中的每条测试路径如下所示:

$P_1: \langle StateX, V_1(m_1,a,b), V_2(m_2,b,a) \mid c_1, StateY \rangle$

$P_2: \langle StateX, V_1(m_1,a,b), V_3(m_3,b,e), V_4(m_4,b,a), StateY \rangle$

$P_3: \langle StateX, V_1(m_1,a,b), V_3(m_3,b,e), V_5(m_2,b,a) \mid c_3, StateY \rangle$

$P_4: \langle StateX, V_1(m_1,a,b), V_3(m_3,b,e), V_6(m_5,b,c) \mid c_4^*, V_7(m_6,b,d) \mid c4^*, V_8(m7,b,e) \mid c4^*, V_9(m_2,b,a) \mid c5, StateY \rangle$

$P_5: \langle StateX, V_1(m_1,a,b), V_3(m_3,b,e), V_6(m_5,b,c) \mid c_4^*, V_7(m_6,b,d) \mid c4^*, V_8(m_7,b,e) \mid c_4^*, V_{10}(m_8,b,c), StateY \rangle$

依据每条测试路径即可得到测试用例,其中“*”表示该消息节点为循环中的消息节点。这里,以 P_1、P_2 为例说明 SEG 的测试用例,由 P_1、P_2 得到的测试用例分别为:

$T^{(1)}_{SEG} = \langle 1, StateX, StateY, StateX, StateY \rangle$

$T^{(2)}_{SEG} = \langle 2, StateX, StateY, StateX, StateY \rangle$

其中:$T^{(1)}_{SEG}$ 是针对路径 $StateX < V_1 < V_2 < StateY$ 进行的测试;$T^{(2)}_{SEG}$ 是针对路径 $StateX < V_1 < V_3 < V_4 < StateY$ 进行的测试。

6)根据算法 6 与图 5,本文首先列出 STG 中的每条测试

路径如下所示：

Pth_1:〈S,N_0,N_1,N_2,$EndY$〉

Pth_2:〈S,N_0,N_1,N_3,N_4,$EndY$〉

Pth_3:〈S,N_0,N_1,N_3,N_5,$EndY$〉

Pth_4:〈S,N_0,N_1,N_3,N_6^*,N_7^*,N_8^*,N_9,$EndY$〉

Pth_5:〈S,N_0,N_1,N_3,N_6^*,N_7^*,N_8^*,N_{10},$EndY$〉

依据每条测试路径即可得到测试用例，如由 Pth_1、Pth_2 得到的测试用例分别为：

$T_{STG}^{(1)}$ = 〈1,S,$EndY$, S,$EndY$〉

$T_{STG}^{(2)}$ = 〈2, S,$EndY$, S,$EndY$〉

其中：$T_{STG}^{(1)}$ 是针对路径 $S < N_0 < N_1 < N_2 < EndY$ 进行的测试；$T_{STG}^{(2)}$ 是针对路径 $S < N_0 < N_1 < N_3 < N_4 < EndY$ 进行的测试。

5 实验设计和分析

本文实验选择了 BSWBN、CBMS 开放性源码系统的 UML 模型作为实验对象，为了更好地检验本文测试方法对系统的测试效果，故对源程序作了标注。本文与文献[3]方法作对比说明。文献[3]与本文方法的实验结果在本文测试用例数、所测类方法数、挖掘错误数、错误检测率四个方面的数据对比，如表 1 所示。

表 1 文献[3]方法与本文方法的实验结果数据对比

模型	测试用例数		被测方法数		挖掘错误数		错误检测率/%	
	文献[3]方法	本文方法	文献[3]方法	本文方法	文献[3]方法	本文方法	文献[3]方法	本文方法
BSWBN	273	304	734	857	113	141	70	88
CBMS	186	242	512	732	53	81	81	90

从表 1 可看出：本文的测试用例数、被测类方法数多于文献[3]方法，这是因为本文进行系统级的测试是从三个层次分别进行的，遵循三层次准则。对 BSWBN 植入 160 个错误、CBMS 系统植入 90 个错误，用上述测试用例挖掘相关错误。本文的挖掘错误数、错误检测率均大于文献[3]方法，可见，满足本文所述三层次覆盖准则的测试用例更有效，因为本文所检测的系统错误主要为交互错、场景错、用例错、用例依赖错等，故而有所局限。

6 结语

本文将 UML 模型转换为可测试的模型，提出用例执行图 UEG、顺序执行图 SEG、系统测试图 STG 的生成算法及对 UEG、SEG、STG 的遍历算法，据此，生成系统级测试用例。该方法不需要对 UML 模型做任何修改或手工干预便可直接得到测试用例，有望与现有测试工具结合。

由于本文考虑模型和错误类型的有限性，只能对软件进行阶段性的固定错误类型的测试。下一步的工作在保证测试用例有效性的前提下，考虑 UML2.0 其他模型特性、增加检错类型，简化测试步骤，得到一个基于 UML 模型的更加实用、高效的测试方法。

参考文献：

[1] ZHENG J, FAN D J, HUANF Z Q, *et al*. Conformance checking of component-based systems for scenario-based specifications [C]// ICSESS 2011: Proceedings of the 2011 IEEE 2nd International Conference on Software Engineering and Service Science. Piscataway, NJ: IEEE Press, 2011:5 - 9.

[2] AVRITZER A, de SILVA S E, LEAO E, *et al*. Automated generation of test cases using a performability model [J]. Software IET, 2011,5(2):113 - 119.

[3] ARITRA B , SUDIPTO G . Test input generation using UML sequence and state machines models [C]// ICST 2009: Proceedings of the 2009 International Conference on Software Testing Verification and Validation. Colorado: ICST, 2009:121 - 130.

[4] BRIAND L, LABICHE Y. A UML-based approach to system testing [J]. Journal of Software and Systems Modeling, 2002,33(16):10 - 42.

[5] LETTRARI M, KLOSE J. Scenario-based monitoring and testing of real time UML models [C]// Proceedings of the 4th International Conference Toronto, LNCS 2185. Berlin: Springer-Verlag, 2011: 312 - 328.

[6] BASANIERI F, BERTOLINO A, MARCHETTI E. The cow suit approach to planning and deriving test suites in UML projects [C]// Proceedings of the Fifth International Conference on the UML. Berlin: Springer-Verlag, 2008: 383 - 397.

[7] SARMA M, DEBASISH K D, MALL R B. Automatic test case generation from UML sequence diagrams[C]// ADCOM: Proceedings of the 2007 International Conference on Advanced Computing and Communications. Guwahati: ADCOM, 2007:60 - 65.

[8] SAMUEL P, JOSEPH A. Test sequence generation from UML sequence diagrams [C]// SNPD: Proceedings of the Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, Parallel/Distributed Computing. Washington, DC: SNPD, 2008: 879 - 887.

[9] KO J W, SIM S H, SONG Y J. Test based model transformation framework for mobile application [C]// ICISA: Proceedings of the 2011 International Conference on Information Science and Applications. Jeju Island: ICISA, 2011:1 - 7.

[10] LIU J H , CHEN C . Complete path coverage testing based on change[J]. Journal of Computer Applications, 2012,32(11):3075 - 3081.(刘继华,陈策.基于变迁的完全路径覆盖测试[J].计算机应用,2012,32(11):3075 - 3081.)

[11] NGUYEN D P, LUU C T, TRUONG A H, *et al*. Verifying implementation of UML sequence diagrams using Java PathFinder[C]// KSE: Proceedings of the 2010 Second International Conference on Knowledge and Systems Engineering. Hanoi: KSE, 2010: 194 - 200.

[12] CHAI Y M, FENG Q Y, WANG L M. Research on methods for generating test cases of inter-classes interaction based on UML models and OCL constraints[J]. Acta Electronic Sinica, 2013, 41(6): 1242 - 1248.(柴玉梅,冯秋燕,王黎明.基于 UML 模型和 OCL 约束的类间交互测试用例生成方法研究[J].电子学报,2013,41(6):1242 - 1248.)

[13] HE H, CHENG C L, ZHANG Z Y, *et al*. Efficient and universal testing method of user interface based on SilkTest and XML[J]. Journal of Computer Applications, 2013,33(1):258 - 261.(何浩,程春玲,张征宇,等.基于 SilkTest 和 XML 的通用高效的用户界面测试方法[J].计算机应用,2013,33(1):258 - 261.)