文章编号:1000 - 582X(2003)08 - 0043 - 05

基于 UML 的面向对象软件测试框架*

林　宏，曾　一
(重庆大学 计算机科学与工程学院，重庆　400044)

摘　要:统一建模语言(UML)是一组面向对象分析和设计的形式化表达语言，基于 UML，统一软件开发过程(USDP)提出了软件开发的过程方法。目前，基于 UML 的软件测试研究中，很少有研究将开发过程和测试过程结合的系统研究。笔者提出了一个基于 UML 的面向对象软件测试框架，该框架有效地结合了 UML 和统一软件开发过程。所提出的测试框架将软件测试分为系统测试、类族测试和类测试，其中，系统测试根据 use - case 和系统序列图生成测试用例，类族测试根据集成的状态图生成测试用例，类测试根据类的状态图生成测试用例。介绍了 UML 与软件测试的层次关系，并讨论了各类测试用例的生成方法。

关键词:统一建模语言；状态图；测试
中图分类号:TP311.5　　**文献标识码**:A

统一建模语言(Unified Modeling Language，UML)是使用面向对象概念进行软件系统建模的一组表示法，它已被国际标准化组织吸收为软件建模领域的国际标准[1]。基于 UML 的设计和开发过程越来越受到广泛的关注，这也不断地增强和扩展了 UML 的设计和开发过程。基于 UML 的测试过程却很少受到关注。软件测试是软件开发中极为重要的过程，在很多软件开发组织，测试在整个软件开发过程中所占的比例约为 40%。测试的发展将直接关系到软件产品的质量，同时，这也是 UML 软件开发过程不断完善的有效保证。

目前，基于 UML 的软件测试的讨论有很多，但系统的讨论测试框架的研究却较少，而且没有系统化。笔者从软件工程的角度出发，结合 UML 和统一软件开发过程，提出了具有实际工程应用意义的面向对象软件测试框架。

1　UML 模型与测试层次

软件测试是软件开发过程的重要组成部分，软件测试的框架和测试方法也与开发过程密切相关。本文中介绍的软件开发过程是基于 UML 的统一软件开发过程(USDP)。统一软件过程将开发过程划分为初始(inception)、细化(elaboration)、构造(construction)和移交(transition)[2]。

在面向对象软件测试中，测试层次划分为系统测试、类族测试和类测试[3]。其中，系统测试侧重于检测系统功能是否达到用户需求，它基于 use case 或其他功能性需求生成测试用例，其测试用例在 USDP 中的初始阶段生成；类族测试的重点是测试类之间的方法调用，它基于集成的状态图生成测试用例，其测试用例在初始阶段生成；类测试的重点在于提高测试的代码覆盖率，它基于类的状态图生成测试用例，测试用例在构造阶段生成。图 1 给出了 UML 开发过程和测试过程的层次关系。

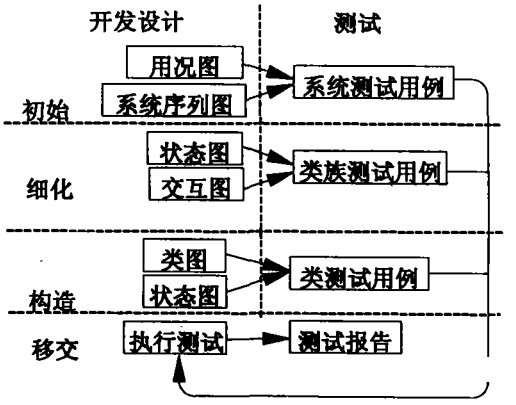

图 1　UML 和测试过程的层次关系

* 收稿日期:2003 - 03 - 01
作者简介:林宏(1975 -)，男，四川省江油市人，重庆大学硕士研究生，主要从事面向对象技术和面向对象软件测试的研究。

2 基于 Use - Case 的系统测试

一个软件系统由一系列提供给用户的服务组成,这些服务由若干对象实现。用户使用服务的方式是向目标系统发送请求,并接受目标系统的处理返回。目标系统内部通过若干对象的相互协作响应用户的处理请求。图 2 阐明了基于 Use - Case 的测试用例构造模型。

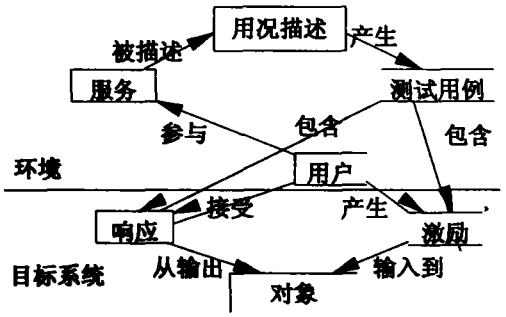

图 2 基于 Use - Case 的测试用例构造模型

一个系统包含一系列的用况,其中一个或几个用况组合起来为客户提供服务,下面给出自动咖啡出售机的用况和服务(见表 1),描述用况的用况图如图 3 所示。

表 1 自动咖啡出售机的用况和服务实例

Use Cases	Actors	Services
购买咖啡	客户	购买咖啡
初始	管理员	机器控制
停止出售	管理员	机器控制

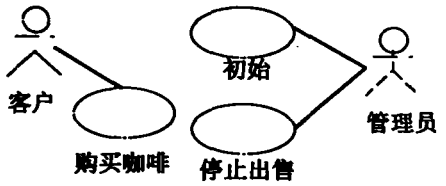

图 3 自动咖啡出售机的用况图

对用况的用户描述往往采用系统序列图(或序列表),它将系统看做一个整体,并从用户的角度描述用况的处理过程,以购买咖啡用况为例,给出其处理过程序列表(见表 2)。

表 2 购买咖啡用况的处理过程

过程描述	步骤	参与者活动	系统活动
	1	投币(Money)	
	2		储值加一(m: = m + 1)
	3	取咖啡(Coffee)	
	4		出咖啡(m: = m - 1)
	5	结束	
扩展描述	3a	如果投币,转到 2	
	4a	如果 m < 1,转到 5	
	5a	如果取咖啡,转到 4	

系统测试是对用况的测试,测试用例的生成过程如下:

1) 标记出用况描述中的所有参与者活动和系统活动;

2) 为参与者活动和系统活动的主过程和每一种扩展描述过程生成测试用例。

表 3 给出购买咖啡用况的测试用例。

表 3 购买咖啡用况的测试用例

测试输入	覆盖处理过程	预期结果
Money, Coffee	主过程 1,2,3,4,5	得到一杯咖啡
Money, Money, Coffee, Coffee	扩展过程 1,23a,2,3,4,5a,4,5	得到两杯咖啡
Money, Coffee, Coffee	扩展过程 1,2,3,4,5a,4a,5	得到一杯咖啡

3 基于 State Diagram 的类族测试

State Diagram 是 UML 中对系统的动态行为进行建模的表示方法,它包括对反应型对象的行为建模[4]。State Diagram 展现了对象在生命周期内可能处于的状态以及在这些状态间转换的激发条件,它在本质上采用了状态图(Statechart)的概念[5]。状态图中包含基本状态(basic state)和复合状态(composite state),复合状态分为或状态(or - state)和与状态(and - state)。或状态的子状态之间是相互排斥的或关系,如图 4 中的 CVM 就是一个或状态,它的子状态 OFF 和 ON 之间是相互排斥的关系。与状态的子状态之间是并发的非排斥关系,如图 4 中的 ON 状态就是一个与状态,当处于 ON 状态时,就意味着同时处于 COFFEE 和 MONEY 两个子状态。

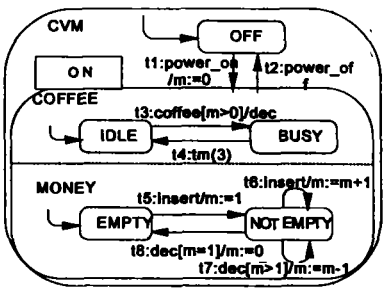

图 4 自动咖啡出售机状态图

根据状态图的基本语义,状态图中存在状态的层次和并发,同时状态之间存在广播通信。为了消除这些特性对测试的影响,我们首先将状态图转换成扩展有限状态机(EFSM),然后在扩展有限状态机的基础上生成类族测试用例。图 5 阐明了基于状态图的类族测试用例构造模型。

Korson 给出 UML 状态图转换成 EFSM 需要的定义[6],下面给出 UML 状态图中的这些定义,其中 states

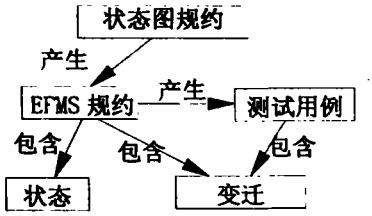

图 5　基于状态图的类族测试用例构造模

和 trans 表示 UML 状态图中的有限状态集合和有限变迁集合。

一个变迁 t 表示为：event[guard]/action send。其中 event 表示激发变迁 t 的事件，记为 event(t)，guard 表示变迁 t 必须满足的逻辑条件，记为 guard(t)，action 表示变迁 t 执行的一系列操作，记为 action(t)，send 表示变迁 t 产生的一系列事件，记为 send(t)。source(t) 和 target(t) 表示变迁 t 的源状态和目标状态集合。

状态 s 包含另一状态 s'，则称 s 为 s 的子状态，s 的子状态集合记为 $\rho(s)$。对于一个或状态 s，进入该状态后将进入的第一个子状态 s 称为 s 的缺省子状态。状态集 $S \subseteq$ states，状态 $s \in S$，如果 S 中的所有状态 s 都满足 $\rho(s') \subseteq S$，并且 S 是满足该条件的最小集合，则称 S 是状态 s 的最小完全子状态集，记为 $\rho^*(s)$。如果 $s_1 \in \rho^*(s)$，则称 s_1 是 s 的子孙，s 是 s_1 的祖先。如果 $s_1 \in \rho^*(s)$ 并且 $s_1 \notin s$，则称 s_1 是 s 的严格子孙，s 是 s_1 的严格祖先。在图 4 中，ρ(COFFEE) = {IDLE, BUSY}，ρ^*(COFFEE) = {COFFEE, IDLE, BUSY}。

状态集合 C $\subseteq$ *states* 满足下列条件则称为一个构造：①C 包含根状态；②对于每一个与状态 s，或者 s 和 s 的所有子状态都包含在 C 中，或者都不包含在 C 中；③对于每一个或状态 s，或者 s 和 s 的某一子状态都包含在 C 中，或者都不包含在 C 中。状态集合 $S \subseteq$ states，如果构造 C 包含 S，C 中的任一或状态 s 都不是 S 的严格祖先，并且 s 的缺省子状态都在 C 中，那么称 C 为 S 的完全构造，记为 complete(S)。在图 4 中，{CVM, ON, COFFEE, IDLE, MONEY, EMPTY} 是一个构造；complete({CVM, ON}) = {CVM, ON, COFFEE, IDLE, MONEY, EMPTY}。

如果或状态 s 是 source(t) $\cup$ target(t) 中所有状态的严格祖先，并且所有满足这种条件的状态都是 s 的祖先，则称状态 s 是 t 的 scope，记为 scope(t)。如果 2 个变迁的 scope 存在祖先子孙关系，则称这两个变迁相互冲突。在图 4 中，scope(t_3) = COFFEE。

$t \in$ trans，状态 s 满足 source(t) $\rho^*(s)$ 和 target(t) $\notin \rho^*(s)$，并且所有满足该条件的状态都是 s 的子孙，则称 s 是 t 的最大离开状态，记为 gds (s)，DS(t) 表示状态集合 ρ^*(gds (s))。$t \in$ trans，状态 s 满足 source (t) $\rho^*(s)$ 和 target(t) $\rho^*(s)$，并且所有满足这种条件的状态都是 s 的子孙，则称 s 是 t 的最大到达状态，记为 gas (s)，$AS(t)$ 表示状态集合 ρ^*(gas (s)) $\cap$ complete(target(t))。在图 4 中，gds(t_1) = OFF；$DS(t_1)$ = OFF；gas(t_1) = ON；$AS(t_1)$ = {ON, COFFEE, IDLE, MONEY, EMPTY}。

基于上面的定义，下面给出 UML 状态图到 EFSM 的转换规则。

$T \subseteq$ Ttrans，如果 T 中的任一变迁 t 满足 source(t) $\subseteq C$，event(t) = e，并且 T 中的任意两个变迁不相互冲突，则称 T 对 C 关于 e 有效。如果对 C 关于 e 有效而不包含在 T 中的变迁都与 T 中的某一变迁相互冲突，则称 T 是对 C 关于 e 有效的最大变迁集。

EFSM 表示为一个三元组 (GStates, C_0, GTrans)，GStates 是全局状态集合，其中的每一个全局状态都是一个构造；C_0 是初始全局状态；GTrans 是全局变迁。一个全局变迁 gt 是一个五元组 (C, e, g, a, C')，C 和 $C' \in G$states，$T \subseteq$ trans 满足：

1) T 是对 C 关于 e 有效的最大变迁集；

2) $C' = (C - \cup_{t \in T} DS(t)) \cup (\cup_{t \in T} AS(t))$；

3) $g = \cup_{t \in T}$ guard(t)；

4) $a = \cup_{t \in T}$ action(t)。

按照以上转换规则，图 4 中的 CVM 状态图模型转换成 EFSM 模型如图 6 所示。

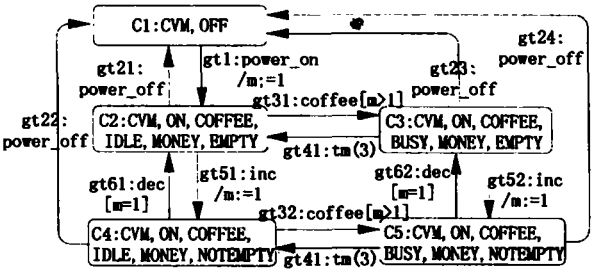

图 6　自动咖啡出售机 EFSM

在 EFSM 的基础上，采用基于控制流的状态覆盖策略，即生成的测试路径将覆盖程序所有可能的状态。下面给出生成的测试树如图 7 所示。观察图 7 发现，路径 gt1, gt31 是一条不可执行路径。应当指出路径的可执行问题是不可判定问题，为了覆盖状态 C3，我们为其重新选择路径，其最终生成的测试用例如表 4 所示：

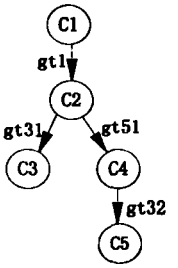

图 7　基于状态覆盖的测试树

表4 类族测试的测试用例

测试路径	覆盖状态	预期结果
gt_1, gt_{31}, gt_{32}	C_1, C_2, C_4, C_5	得到1杯咖啡
gt_1, gt_{31}	C_1, C_2	未得到咖啡
$gt_1, gt_{31}, gt_{32}, gt_{62}, gt_{23}$	C_1, C_2, C_4, C_5, C_3	得到咖啡

4 基于 State Diagram 的类测试

在类测试中,关注对象内的状态和状态间的转换条件,并力求达到较好的代码覆盖率。此时,可把每一个类的 state diagram 看成一个独立的有限状态机,该状态机描述了该类的代码的 def - use 关系,根据 def - use 关系生成类和方法的基于数据流的测试用例。

基于数据流的方法中,具有代表性的是 Rapps 和 Weyuker 覆盖标准族[7-8],它根据程序中变量的 def - use 关系,提出了一系列具有偏序关系的覆盖准则。在此,选用该标准族中的 all - du - paths 标准,下面给出相关定义。

一个变量 x 在变迁 t 中被赋值,则称变迁 t 定义了 x,记为 $x \in \mathrm{def}(t)$;一个变量 x 在变迁 t 中被用于赋值,则称变迁 t 有关于 x 的 c - use,记为 $x \in \mathrm{c-use}(t)$;一个变量 x 在变迁 t 中出现在判断条件里,则称变迁 t 有关于 x 的 p - use,记为 $x \in \mathrm{p-use}(t)$。

如果一条路径 $t_1, \ldots t_j, x \in \mathrm{def}(t)$,路径中除 t_1 和 t_j 外没有任何变迁定义了 x,该路径中不包括循环,而且满足下列两个条件之一:①$x \in \mathrm{c-use}(t_j)$,②$x \in \mathrm{p-use}(t_j)$,则称该路径为一条 du - path。

all - du - paths 标准:对于每一变迁 t 的每一个定义变量 x,如果一个测试路径集包含了所有的关于 x 的 du - path,则称该测试路径集满足 all - du - paths 标准。

Money 对象的 state diagram 如图8所示,表5列出该对象的变量定义使用集(见表5)。

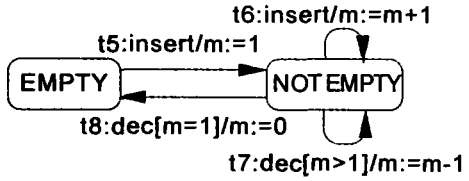

图8 Money 对象的 State Diagram

表5 Money 对象的变量定义使用集

变量	define	c - use	p - use
m	t_5, t_6, t_7, t_8	t_6, t_7	t_7, t_8

以变迁 t_5 为例,关于 t_5 的所有 du - path 为:t_5, t_6, t_5, t_7, t_5, t_8。

在为所有变迁生成的 du - path 中,将舍弃被其他测试路径覆盖的路径,对于不完整路径,将为其增加前缀和后续序列构成完整路径。同时,还将舍弃不可执行路径。下面给出满足 all - du - path 覆盖标准的测试用例集合。

表6 Money 对象类测试的测试用例

测试路径	是否可执行	预期结果
t_5, t_6, t_8	是	得到一杯咖啡
t_5, t_6, t_7, t_8	是	得到两杯咖啡
t_5, t_7, t_8	否	无
t_5, t_8	是	得到一杯咖啡

5 结 论

UML 给出了面向对象的分析、设计和建模模型,它使得面向对象的程序开发更加规范和高效。笔者基于 UML 的分析和设计方法,提出了基于此的面向对象软件的测试框架,这对于软件工程和测试工程都具有一定的理论意义和工程价值。该测试框架将测试过程分为3个过程:系统测试、类族测试和类测试,生成这些测试用例的阶段分别对应于统一软件过程中的初始、细化和构造。在测试用例的生成方法中,它们分别基于 UML 中的各种分析图形。

笔者侧重于提出一个基于 UML 的面向对象软件测试框架,并介绍了该框架中各种测试用例的生成方法,但对测试用例生成方法的效率和覆盖程度并未做深入讨论,这需要在进一步的工作中深入研究。其中,在类测试阶段,测试用例的代码覆盖率和自动化程度要求较高,本文中采用了 all - du - path 覆盖标准,但是这会生成大量的不可执行路径,最终造成代码覆盖率和自动化程度的下降。在进一步的工作中,将综合考虑控制流和数据流特性,并结合启发式算法,以提高代码的覆盖率和生成测试用例的自动化程度。

参考文献:

[1] Craig Larman. UML 和模式应用:面向对象分析与设计导论[M]. 北京:机械工业出版社, 2001.

[2] IVAR J, GRADY B, JAMES R. 统一软件开发过程[M]. 北京:机械工业出版社, 2001.

[3] JOHN D M, TIMOTHY D K. Integrated Object - oriented Testing and Development Processes[J]. Communications of the ACM. 1994,37(9):59 - 77.

[4] GRADY B, JAMES R, IVAR J. The Unified Modeling Language User Guide [M]. Chicago: Addison Wesley, 1999.

[5] ROBERT V B. 面向对象系统的测试[M]. 北京:机械工业出版社, 2001.

[6] KIM Y G, HONG H S, BAE D H. Test Cases Generation from UML State Diagrams[J]. IEE Proceeding on Software. 1999,46(4):187 - 192.

[7] SANDRA P, WEYUKER E J. Selecting Software Test Data Using Data Flow Information[J]. IEEE Transactions on Software Engineering. 1985,11(4):367 - 490.

[8] PHYLLIS G F, WEYUKER E J. An Applicable Family of Data Flow Testing Criteria[J]. IEEE Transactions on Software Engineering. 1988,14(10):1 483 - 1 490.

A Test Framework of Object – Oriented Software Based on UML

LIN Hong, ZENG Yi

(College of Computer Science, Chongqing University, Chongqing 400044, China)

Abstract: Unified Modeling Language (UML) is a set of language which is used in object – oriented analysis and design. Based on UML, Unified Software Development Process (USDP) presents a method to be used in the whole process of software development. For the moment, among the study of UML – based testing, few of them focus on the combination of the development process and testing process. This paper proposes a UML – based object – oriented software testing framework, which effectively combine UML and USDP. The test framework divides the test process into system testing, cluster testing and class testing. We generate system test cases from use – case diagrams and system sequence diagrams, and we produce class integration test cases from integrated state diagrams. Then, class test cases are generated from class state diagrams. This paper demonstrates the relationship between UML and the test hierarchy. Meanwhile, test cases generation methods are also discussed.

Key words: unified modeling language; statechart; testing

(编辑　吕赛英)

(上接第34页)

Product Configuration Management Based on Supply Chain and Capital Flow

SHI Wei-ren, KANG Jing, PENG Shi-qiang

(College of automation, Chongqing University, Chongqing 400044, China)

Abstract: The product configuration management plays an important role in the production of enterprises. Traditional product configuration management only modifies the component of production. It can not reveal production costs and due dates. This paper puts forward an improved product configuration management based on supply chain and capital flow so as to find different manufacture strategy which take the client's different requirement into account. This kind of product configuration management can also forecast production costs and due date. It improves on the product configuration management in existence, which can well meet the requirements of competition.

Key words: product configuration management; supply chain; capital flow; due dates; cost

(编辑　张小强)