基于 UML 的软件 Markov 链使用模型构造研究*

颜 炯, 王 戟+, 陈火旺

(国防科学技术大学 计算机学院,湖南 长沙 410073)

Deriving Software Markov Chain Usage Model from UML Models

YAN Jiong, WANG Ji+, CHEN Huo-Wang

(School of Computer Science, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: Phn: +86-731-4573637, Fax: +86-731-4575802, E-mail: jiwang@nudt.edu.cn, http://www.nudt.edu.cn

Received 2004-06-23; Accepted 2005-03-11

Yan J, Wang J, Chen HW. Deriving software Markov chain usage model from UML models. *Journal of Software*, 2005,16(8):1386−1394. DOI: 10.1360/jos161386

Abstract: Software statistical testing is concerned with testing software systems based on their usage models. In the context of UML(unified modeling language)-based development, it is desired that the usage models can be derived from the UML analysis artifacts. This paper presents a method that derives the software Markov chain usage models from the reasonably annotated UML artifacts. The method utilizes the annotated use case diagrams, the annotated sequence diagrams, and the use case execution sequence relations. These annotations and the use case execution sequence relations are called statistical testing constraints. The method presents an algorithm that derives the Markov chain usage models from the UML artifacts and the statistical testing constraints. The framework of the support tool, UMGen, is also presented. The usage models can be used to generate software statistical test cases and estimate the software reliability.

Key words: UML (unified modeling language); statistical testing; Markov chain usage model; software reliability

摘 要: 软件统计测试要求基于软件使用模型产生测试例对软件系统进行测试,并根据测试结果评价软件可靠性,是高可靠软件测试的重要组成部分.由于统一建模语言(unified modeling language,简称 UML)已经成为事实上的面向对象标准建模语言,因此,从软件 UML 模型构造软件使用模型就成为面向对象软件统计测试的关键.为此,定义了加入统计测试约束的 UML 用例图、序列图以及用例执行顺序关系,为基于 UML 的软件统计测试提供了一个形式化描述基础.在此基础上,给出一个从软件 UML 模型构造软件 Markov 链使用模型的算法,并给出了自动化支持工具 UMGen 的类图结构,基于一个卫星控制系统,说明了所提出方法的有效性.

关键词: 统一建模语言;统计测试; Markov 链使用模型;软件可靠性

* Supported by the National Natural Science Foundation of China under Grant Nos.90104007, 60233020 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant Nos.2001AA113202, 2001AA113190 (国家高技术研究发展计划(863)); the Huo Ying Dong Education Foundation under Grant No.71064 (霍英东教育基金), the Program for New Century Excellent Talents in University (新世纪优秀人才支持计划)

作者简介: 颜炯(1971－),男,湖南衡南人,博士生,主要研究领域为软件测试,软件可靠性;王戟(1969－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为高可信软件技术,Agent 软件方法学;陈火旺(1936－),男,教授,博士生导师,中国工程院院士,主要研究领域为软件工程,人工智能.

中图法分类号: TP311 文献标识码: A

随着软件控制系统的广泛应用,社会对软件依赖程度不断提高.航空航天、武器装备和医疗设备等系统中的软件结构非常复杂,软件失效往往会造成严重后果.1996 年,由于软件失效导致欧洲航天局 Ariana 5 型火箭发射失败后,软件可靠性得到了学术界和工业界的进一步重视.

软件测试是软件开发的重要步骤和软件质量保证的重要环节,软件统计测试则是高可靠软件测试的重要内容.软件统计测试与传统的软件测试不同,它要求基于软件使用模型产生测试例对软件系统进行测试.软件统计测试也称为软件可靠性测试,因为根据统计测试结果可以估计被测试软件的可靠性.净室软件开发方法[1]用 Markov 链描述软件使用模型,根据 Markov 链使用模型产生统计测试例对软件进行软件可靠性确认和验证.作为净室方法的关键技术之一,统计测试在高可靠软件的质量确认和验证过程中起到了关键作用.因此,软件统计测试得到了广泛深入的研究[2,3].

面向对象方法已经成为当前主流软件开发方法,但目前软件开发仍普遍使用传统方法进行软件测试.现有的面向对象软件测试研究多数集中于基于状态图的类和类簇测试,而且实际应用很少[4].由于建立正确的软件模型是面向对象软件开发成功的关键,因此,1997 年,对象管理组织(object management group,简称 OMG)采纳统一建模语言(unified modeling language,简称 UML)[5]作为面向对象标准建模语言,使 UML 成为软件工业中占据主导地位的对象建模语言.研究基于 UML 模型的软件测试不仅可以支持软件测试,而且有利于把测试工作提前到软件开发周期的早期进行.因此,基于 UML 的软件测试得到了广泛研究.例如,Briand 研究了基于 UML 模型生成系统测试需求[6].Regnell 等人把使用模型分为状态模型和层次模型两部分,研究了从用例产生使用模型,并将其应用于交换机控制软件测试[7],但是这种使用模型比较复杂且难以自动生成.Riebisch 等人提出一种方法,可以从 UML 用例和状态图产生描述系统行为的模型,并基于此模型产生测试例[8],这种方法需要比较完备的软件设计模型.此外,由于 UML 模型都没有为多数软件测试提供足够的信息,往往无法自动生成测试例,因此,必须对软件 UML 模型进行必要的可测试性扩展[4].为了支持软件统计测试和软件可靠性评估,8 个欧洲工业厂商和大学合作开展了 MaTeLo(Markov test logic)项目[9],其主要目的是开发基于 Markov 链使用模型的测试例自动产生技术及支持工具,目前研究成果主要是从 UML 场景产生软件 Markov 链使用模型,但是无法自动生成 Markov 链的迁移概率,并且未考虑场景执行关系,因此难以直接应用于软件系统的统计测试.

针对高可靠软件统计测试,我们研究了从软件 UML 模型构造软件 Markov 链使用模型,提出了一个从软件 UML 模型构造软件 Markov 链使用模型的方法,并基于一个卫星控制系统的例子,说明了方法的有效性.

本文第 1 节给出加入统计测试约束的 UML 用例图和序列图的形式定义,并以一个卫星控制系统软件为例进行说明.第 2 节研究从加入统计测试约束的 UML 模型构造软件 Markov 链使用模型的方法.第 3 节简要说明基于 Markov 链使用模型的软件统计测试例产生方法,并给出了使用模型自动生成工具的类图结构.最后是对工作的总结和展望.

1 加入统计测试约束的 UML 模型形式定义

系统分析设计过程中得到的软件 UML 模型一般包括用例图(use case diagram)及用例描述、完成各个用例功能的序列图(sequence diagram)集合以及应用领域类的类图(class diagram)[10].

1.1 软件行为的场景描述

基于 UML 的软件开发使用场景和用例获取及描述需求.用例给出了软件系统级功能描述.一个用例的场景集合描述了完成用例功能过程中的软件总体行为,每个场景描述了完成用例功能过程中软件对象之间以及软件对象与环境之间一次特定的交互过程.目前,对待场景有两种基本的观点:面向设计的观点认为场景描述了软件的高级设计,用状态机定义场景的语义[11];面向需求的观点则认为场景描述了系统的功能规范[12],并基于偏序关系定义场景语义.UML 使用序列图描述场景,因此下文中不加区别地使用这两个术语.

1.2 UML模型的形式定义

本文首先基于偏序关系给出加入统计测试约束的 UML 用例和序列图的形式定义.序列图包括两个方向,水平方向是参与交互的对象实例(instance),简称实例;垂直方向是实例的生命线(lifeline),生命线之间的水平箭头表示一个消息,其方向指明了消息的发送方和接收方,消息按照发生时间顺序从上到下排列.

定义 **1**. 序列图 m 中参与交互的实例定义为

$$inst_m=\{i_1,i_2,\ldots,i_n\}.$$

定义 **2**. 一个实例的生命线上发送或者接收消息的位置称为位点(location).每个实例的生命线上都有一个有限离散位点集合.令 $loc_{m,i}$ 为序列图 m 中实例 i 的位点集合,dom_m 为序列图 m 的位点集合:

$$loc_{m,i}=\{0,\ldots,l_{m,i}\};$$

$$dom_m=\{\langle i,l,x\rangle|i\in inst_m,l\in loc_{m,i},\text{如果 } l \text{ 是消息发送位点,则 } x=!,\text{否则 } x=?\}.$$

定义 **3**. 每个消息用一个消息名或者操作名进行标记,并可以附加参数,每个消息用一个消息序号作为唯一标识.这样的一个消息是集合 *MessageIds* 的一个元素:

$$MessageIds\subseteq MessageNames\times ParameterLists\times N.$$

定义 **4**. 如果一个消息调用的操作产生一个返回值,则发送返回值的消息在序列图中表示为与原消息有相同消息序号的返回消息.每个返回消息在集合 *ReturnMessageIds* 的一个元素:

$$ReturnMessageIds\subseteq MessageIds\times ReturnValue.$$

定义 **5**. 集合 *Messages* 包含了一个序列图 m 中的所有消息:

$$Messages=MessageIds\cup ReturnMessageIds.$$

定义 **6**. 序列图中的每个消息与图中两个位点关联,即消息发送点和消息接收点.函数 msg_m 规范了序列图 m 的位点和消息之间的关系:

$$msg_m:dom_m\rightarrow P(Messages).$$

定义 **7**. 函数 *source* 和 *target* 定义了消息的源实例和目的实例:

$$source,target:Messages\rightarrow inst_m.$$

对$\forall\langle i,l,x\rangle\in dom_m$和$\forall m_id\in Messages$,上述函数定义为

$$source:\mu\rightarrow i,\text{if } \mu=m\_id\in msg_m(\langle i,l,!\rangle),$$

$$target:\mu\rightarrow i,\text{if } \mu=m\_id\in msg_m(\langle i,l,?\rangle).$$

以下定义中,假定 x,x'和 $x''\in\{!,?\}$.

定义 **8**. 序列图 m 的位点之间的偏序关系$\leq\subseteq dom_m\times dom_m$ 定义为

(1) $\langle i,l,x\rangle\leq\langle i,l+1,x'\rangle$;

(2) $m\_id\in msg_m(\langle i,l,!\rangle)\wedge m\_id\in msg_m(\langle i',l',?\rangle)\Rightarrow\langle i,l,!\rangle\leq\langle i',l',?\rangle$;

(3) 由(1)和(2)确定的传递闭包.

函数 msg_m 在以下约束下保证一致性:

(1) 每个消息在其所属的序列图中是唯一的.

$\forall\langle i,l,x\rangle,\langle i',l',x\rangle\in dom_m$:if $m\_id\in msg_m(\langle i,l,x\rangle)\wedge m\_id\in msg_m(\langle i',l',x\rangle)\Rightarrow\langle i,l,x\rangle=\langle i',l',x\rangle$.

(2) 每个位点可以发出多个消息,或者收到一个消息,但不能在收到一个消息的同时发出另一个消息.

$\forall\langle i,l,x\rangle\in dom_m,m\_id\in msg_m(\langle i,l,x\rangle)$:if $card\ (msg_m(\langle i,l,x\rangle))>1\Rightarrow x=!$.

(3) 返回消息必然把返回值返回到发送消息的实例.

$\forall\langle i,l,?\rangle\in dom_m$,if $v\in ReturnValue\wedge\langle m\_id,v\rangle\in msg_m(\langle i,l,?\rangle)\Rightarrow$

$\exists\langle i,l',!\rangle\in dom_m$,s.t.$m\_id\in msg_m(\langle i,l',!\rangle)\wedge\langle i,l',!\rangle\leq\langle i,l,?\rangle$.

(4) 一个消息的返回消息不能多于一个.

$\forall\langle m\_id,v\rangle,\langle m\_id',v'\rangle\in ReturnMessageIds$:$\exists\langle i,l,x\rangle,\langle i',l',x\rangle\in dom_m$:

if$(\langle m_id,v\rangle\in msg_m(\langle i,l,x\rangle)\wedge\langle m_id',v'\rangle\in msg_m(\langle i',l',x\rangle)\wedge m_id=m_id')\Rightarrow\langle i,l,x\rangle=\langle i',l',x\rangle\wedge v=v'$.

定义**9**. $\prec$是序列图m中消息之间的序关系,$\prec \subseteq Messages \times Messages$,并满足:

$\forall m\_id, m\_id' \in Messages, \exists \langle i,l,x\rangle \in dom_m$ s.t. $msg_m(\langle i,l,x\rangle)=m\_id \wedge \exists \langle i',l',x'\rangle \in dom_m$ s.t. $msg_m(\langle i',l',x'\rangle)=m\_id'$:if $\langle i,l,x\rangle \le \langle i',l',x'\rangle \Rightarrow m\_id \prec m\_id'$.

本文假定在一个序列图m中,被测试对象集合与外界交互的消息满足全序关系.

在每个用例执行之前,软件必须满足特定的前置条件.在完成用例功能的过程中,执行不同的场景会使软件进入不同状态,决定了下一个可以执行的用例,因此,本文用场景执行后置条件来规范用例执行后置条件.场景执行后置条件和用例执行前置条件可以用对象约束语言(object constraint language,简称OCL)来描述.

定义**10**. 加入了统计测试约束的序列图m定义为元组:

$$m=(inst_m, dom_m, msg_m, post_m, pf_m),$$

$inst_m, dom_m, msg_m$定义如前,$post_m$是序列图m的执行后置条件,pf_m是序列图m在相关用例中的执行概率.

定义**11**. 用例定义为元组:

$$UC=(pre_{uc}, SDSet_{uc}),$$

其中,$SDSet_{uc}$是用例中所有序列图的集合,pre_{uc}是用例执行前置条件.

用例之间可能存在⟨⟨extend⟩⟩和⟨⟨include⟩⟩关系,同时还存在执行顺序关系,它说明了软件支持的业务过程[4].统计测试必须考虑用例执行顺序关系,因为它反映了软件的使用方式,并可能触发不同的失效[6].本文假设软件运行时,在同一时刻只能执行一个用例.

定义 **12**. UML 用例执行顺序关系定义为一个六元组 $UCExecRelation=(UCSet, s_{start}, s_{end}, V, \Omega, seq)$,其中:

- $UCSet$ 是软件用例集合;
- s_{start} 是软件执行初始结点,s_{end} 是软件执行终止结点;
- $V=\{s_{start}, s_{end}\} \cup UCSet$,$V$ 是结点集合;
- Ω是执行某个用例前软件状态和执行某个场景后软件状态的集合;
- seq 是一个函数.$seq: V \times \Omega \times (0,1] \to V$,规范了结点之间的迁移关系.如果 $seq(v, \omega_j, p_j)=v'$,那么$\Sigma p_j=1$.

定义**13**. 加入了统计测试约束的软件UML模型定义为

$$system=(actors, UCSet, UCExecRelation),$$

其中$actors$是软件使用者集合,$UCSet$是软件用例集合,$UCExecRelation$是用例执行顺序关系.

1.3 卫星控制系统

本文以一个通信卫星嵌入式控制软件系统(satellite control system,简称 SCS)为例进行说明[1].一个卫星通信系统由地面控制系统(ground control system,简称 GCS)、卫星、上行站(uplink site,简称 UL)和下行站(downlink site,简称 DL)4 部分组成,图 1 是简化功能的 SCS 用例图.SCS 的使用过程是:GCS 首先执行用例 Connect,连接成功时,SCS 状态变为 Connected,可以执行 Transmit 用例;连接失败时,SCS 状态为 Idle.执行 Transmit 用例时,UL 向 SCS 发送数据包,SCS 将所有数据包转发到 DL.SCS 成功转发所有数据包后,执行 Close 用例,通知 UL,DL 和 GCS 关闭连接,SCS 的状态变为 Idle.图 2 描述了 SCS 用例执行顺序关系,未标明迁移概率的有向边的迁移概率均为 1.

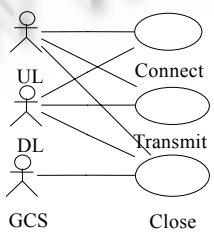

Fig.1 Use case diagram of SCS

图 1 SCS 用例图

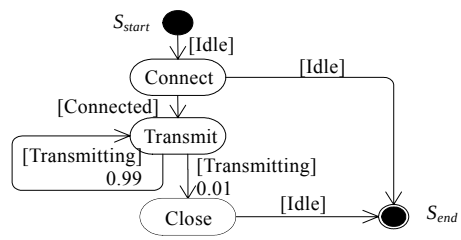

Fig.2 SCS use cases execution sequential relations

图 2 SCS 的用例执行顺序关系

SCS 定义为 *SCS*=(*actors*,*UCSet*,*UCExecRelation*),其中 *actors*=(*GCS*,*DL*,*UL*),而 *UCSet*=(*Connect*,*Transmit*,*Close*),图 2 描述了 *UCExecRelation*.

各个用例定义为

Connect=(⟨*Idle*⟩,⟨$m_{Connect,1}$,$m_{Connect,2}$,$m_{Connect,3}$⟩),

Transmit=(⟨*Transmitting* or *Connected*⟩,⟨$m_{Transmit,1}$,$m_{Transmit,2}$⟩),

Close=(⟨*Transmitting*⟩,⟨$m_{Close,1}$⟩).

图 3~图 8 给出了描述 SCS 各个场景的序列图,表 1 给出了各场景在相关用例中的执行概率和执行后置条件.

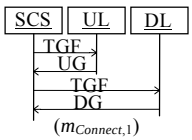

Fig.3 Scenario 1 of Connect

图 3 Connect 用例的场景 1

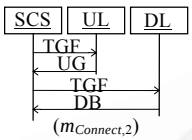

Fig.4 Scenario 2 of Connect

图 4 Connect 用例的场景 2

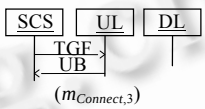

Fig.5 Scenario 3 of Connect

图 5 Connect 用例的场景 3

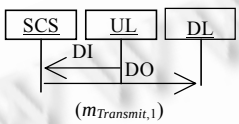

Fig.6 Scenario 1 of Transmit

图 6 Transmit 用例的场景 1

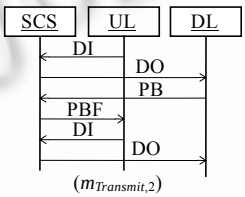

Fig.7 Scenario 2 of Transmit

图 7 Transmit 用例的场景 2

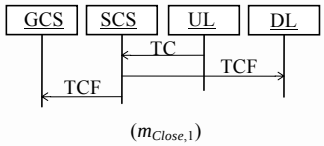

Fig.8 Scenario of Close

图 8 Close 用例的场景

Table 1 Execution probability and postcondition of each scenario in its associated use case

表 **1** 各场景在相关用例中执行概率和执行后置条件

	Scenario 1		Scenario 2		Scenario 3	
	pf_1	$post_1$	pf_2	$post_2$	pf_3	$post_3$
Connect	0.8	Connected	0.1	Idle	0.1	Idle
Transmit	0.8	Transmitting	0.2	Transmitting	-	-
Close	1	Idle	-	-	-	-

2 从 UML 模型构造 Markov 链使用模型

本节提出一个从软件 UML 模型构造软件 Markov 链使用模型的方法.该方法需要使用加入统计测试约束的软件 UML 模型.同时,并给出了从软件 UML 模型构造软件 Markov 链使用模型的算法.

2.1 基于UML的软件统计测试的基本步骤

基于 UML 的软件统计测试一般包括以下步骤:

(1) 为 UML 模型加入统计测试约束.

(2) 从加入统计测试约束的软件 UML 模型构造软件 Markov 链使用模型.

(3) 根据软件 Markov 链使用模型产生测试例对软件进行统计测试,并根据测试结果估计软件可靠性.

定义 **14**. 软件 Markov 链使用模型定义为一个五元组⟨S,Γ,δ,q_0,F⟩,其中

- S 是软件执行的状态集合;
- Γ是迁移标记集合,每个迁移标记 $t\in\Gamma$是一个三元组⟨*TransitionType*,*Name*,*pf*⟩,其中,*TransitionType*∈{event, action}规范了迁移类型,*Name* 是迁移名称,*pf* 是迁移概率;
- δ是一个函数.δ:$S\times\Gamma\to S$,规范了软件执行过程中软件状态间的迁移关系;

• q_0 是初始状态,即软件执行前所处的状态;

• F 是软件终止执行时所进入的终止状态集合.

软件的状态 $s \in S$ 定义为一个三元组⟨$State_{num}$,Label,Notation⟩,其中

• $State_{num}$ 是状态的编号;

• Label∈{"*Initial*","*Final*"},分别用于标记初始状态 q_0 和集合 F 中的状态;

• Notation 是状态附加信息,例如场景执行后置条件.

2.2 为UML模型加入统计测试约束

从 UML 模型构造 Markov 链使用模型,需要为 UML 模型加入必要的统计测试约束,包括:

(1) 用例执行前置条件和用例中每个场景执行后置条件,可以用 OCL 进行规范;

(2) 每个场景在相关用例中的执行概率;

(3) 用例执行顺序关系.

上述约束是在系统分析设计过程中加入的,在不损害分析设计方便性的前提下,这些约束信息有利于深入理解系统,具有较好的可操作性.

2.3 构造用例的使用模型

构造能够描述一个用例所有场景执行过程和执行概率的 Markov 链使用模型的算法如下:

算法 **1**. 构造用例的 Markov 链使用模型.

Step 1. 为用例的第 1 个场景生成 Markov 链,步骤如下:

1. 创建一个 Markov 链,其初始状态为 S_0,并令 *CurrentState* 指向 S_0,*pf* 为场景执行概率;
2. 根据消息发生顺序,对场景中被测试对象与外界交互的所有消息 *m* 进行以下处理:
   (1) 如果消息 *m* 是被测试对象收到的消息,那么令迁移 *t*=⟨event,*m*,*pf*⟩;否则令 *t*=⟨action,*m*,*pf*⟩;
   (2) 插入一个新状态 *S*,并用 *t* 连接 *CurrentState* 和 *S*,然后令 *CurrentState* 指向 *S*;
3. 令 *CurrentState* 的 Label 为"*Final*",Notation 为当前场景的执行后置条件.

Step 2. 集成其他场景生成用例的 Markov 链,此时,需要对其他所有场景进行以下处理:

1. 令 *CurrentState* 指向 S_0,*pf* 为当前场景执行概率,并令 *postC* 为场景执行后置条件;
2. 根据消息发生顺序,对场景中被测试对象与外界交互的所有消息 *m* 进行以下处理:
   (1) 如果消息 *m* 是被测试对象收到的消息,那么令迁移 *t*=⟨event,*m*,*pf*⟩;否则令 *t*=⟨action,*m*,*pf*⟩;
   (2) 如果 *CurrentState* 的 Label 为"*Final*",那么
      ① 插入一个新状态 *S* 并将 *CurrentState* 的 Label 和 Notation 复制到 *S*;
      ② 清除 *CurrentState* 的 Label 和 Notation,令 *pf* ′为 *CurrentState* 的入迁移的概率之和减去 *pf*;
      ③ 用迁移⟨event,*null*,*pf*′⟩连接 *CurrentState* 和 *S*;
   (3) 如果 *CurrentState* 的一个出迁移 *tr* 与迁移 *t* 除 *pf* 外均相同,那么将 *pf* 累积到 *tr* 的迁移概率,并令 *CurrentState* 指向 *tr* 的目的状态;否则插入一个新状态 *S*,并用 *t* 连接 *CurrentState* 和 *S*,并令 *CurrentState* 指向 *S*;
3. 如果 Markov 链中存在一个终止状态,其 Notation 为 *postC*,那么将 *CurrentState* 与此状态进行合并;否则令 *CurrentState* 的 Label 为"*Final*",Notation 为 *postC*;

Step 3. 对各个状态的出迁移的迁移概率进行范化,使其和为 1.

2.4 构造软件使用模型

集成用例使用模型可以构造软件使用模型,这个过程需要使用用例执行顺序关系.

算法 **2**. 集成用例 Markov 链构造软件使用模型.

Step 1. 插入一个状态 q_0 并令 q_0 的 Label 为"*Initial*";

Step 2. 对每个用例 *TargetUC* 进行以下处理:

如果在 UCExecRelation 中, TargetUC 是 s_{start} 的后继,那么

(1) 令 pf 为 UCExecRelation 中 s_{start} 到 TargetUC 的迁移概率;

(2) 令 S_{in} 为 TargetUC 的 Markov 链的初始状态;

(3) 用迁移⟨event,*null*,*pf*⟩连接 q_0 和 S_{in};

Step 3. 对每个用例 *SourceUC* 的 Markov 链的每个终止状态 S_{out} 进行以下处理:

1. 检查每个用例 *TargetUC* 的前置条件,如果 S_{out} 的 Notation 蕴含 *TargetUC* 的前置条件,那么
   如果 *UCExecRelation* 包含一个函数 *seq*(*SourceUC*,Notation of S_{out},*pf*)=*TargetUC*,那么
   (1) 令 S_{in} 为 *TargetUC* 的 Markov 链的初始状态;
   (2) 用迁移⟨event,*null*,*pf*⟩连接 S_{out} 和 S_{in},并清除 S_{out} 的 Label;
2. 如果 *UCExecRelation* 包含一个函数 *seq*(*SourceUC*,Notation of S_{out},*pf*)=s_{end},并且 $pf\neq 1$ 那么
   (1) 插入一个状态 *s*,令 *s* 的 Label 为"*Final*",并清除 S_{out} 的 Label;
   (2) 用迁移⟨event,*null*,*pf*⟩连接 S_{out} 和 *s*.

2.5 小 结

上述内容详细讨论了从软件 UML 模型构造 Markov 链使用模型的方法,为在基于 UML 的软件开发过程早期获得统计测试的 Markov 链使用模型提供了一个完整的方法.

在保证提供必要统计测试信息的前提下,应用本文方法得到的 Markov 链存在部分状态和迁移冗余,但是不影响生成统计测试例和评价统计测试结果.如果对 UML 序列图中每个消息加入前置条件和后置条件,那么可以生成比较精确的 Markov 链使用模型,但是加入约束条件的工作量比较大.本文方法把场景在用例中的执行概率作为该场景中所有消息的执行概率,并由此生成 Markov 链迁移概率.因为 Markov 链迁移粒度太小,即使存在类似系统,统计每个迁移的执行概率也非常困难.因此,从相对容易获得的场景执行概率获取 Markov 链迁移概率,可以有效地解决 Markov 链使用模型的迁移概率获取问题.

图 9 是根据本文方法生成的 SCS 的 Markov 链使用模型.其中状态 q_0 的 Label 为"*Initial*",而状态 1 和状态 2 的 Label 为"*Final*",Notation 为"Idle".

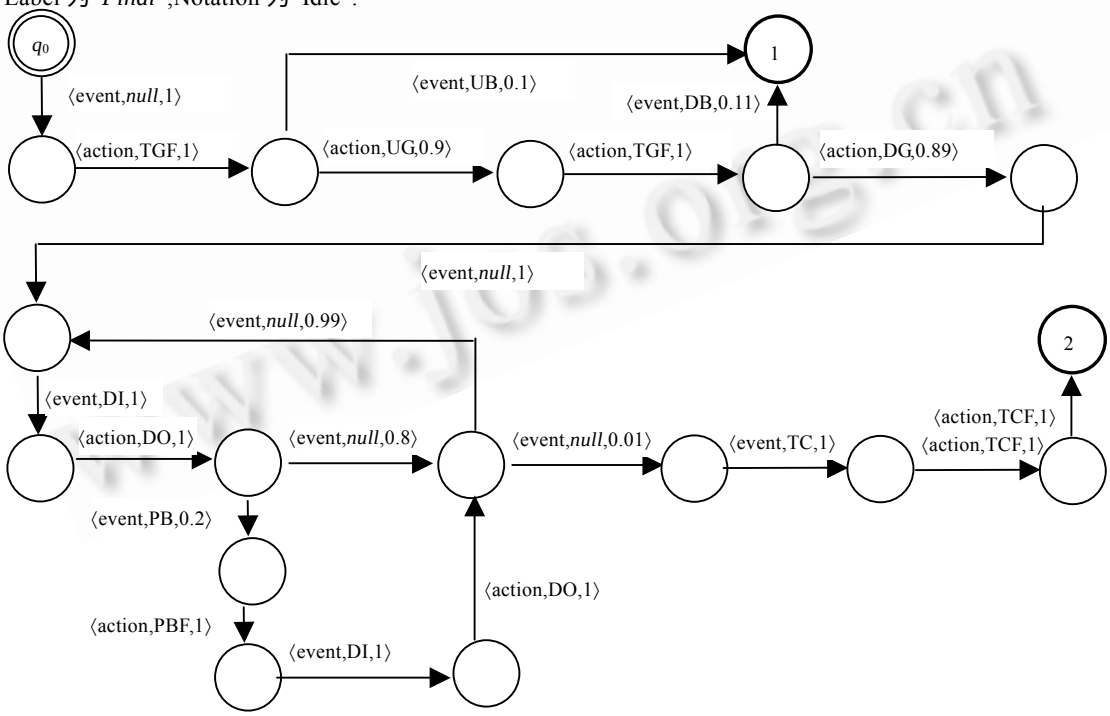

Fig.9 Markov chain usage model of SCS

图 9 SCS 的 Markov 链使用模型

3 统计测试例产生与使用模型自动生成工具

3.1 统计测试例产生

Markov 链使用模型中每个状态的出迁移都代表被测试软件的一个激励或一个响应.产生统计测试例时,要求根据迁移边的迁移概率,从 Label 为“*Initial*”的状态到 Label 为“*Final*”的状态遍历使用模型的迁移边,去除遍历得到的迁移标记序列中标记名为 *null* 的激励,最终得到的标记序列就是被测试软件一次执行过程中收到的激励和发出的响应组成的序列,一个测试序列就是这样一个激励-响应序列.通过给一个测试序列中的变量赋值就得到一个测试例,对测试序列中的变量赋值可以根据变量值域的分布进行选择.

以图 9 所示的使用模型为例,以下测试序列表示一个转发了 3 个数据包的通信过程,其中一个数据包在转发过程中发生损坏并且重新转发成功:

[TGF].(UG).[TGF].(DG).(DI).[DO].(DI).[DO].(PB).[PBF].(DI).[DO].(DI).[DO].(TC).[TCF].[TCF].

而以下测试序列表示一个由于连接失败导致通信终止的过程:

[TGF].(UG).[TGF].(DB).

其中,(∗)和[∗]分别表示被测试软件收到的激励和发出的响应.测试例执行之前,序列中的激励被转化为测试脚本模拟外界环境对软件系统的输入,响应可以作为部分测试失效辨识.高可靠软件的统计测试要求根据使用模型不断产生测试例对软件进行测试,当软件 Markov 链使用模型的相关数学特征达到覆盖准则要求,并且软件可靠性指标也达到要求时,测试终止[13].

3.2 使用模型自动生成工具

为了支持统计测试自动化,我们开发了支持工具 UMGen.工具支持为 UML 模型增加统计测试约束,并支持从 UML 模型构造 Markov 链使用模型.图 10 是 UMGen 的类图框架.其中,XMIParser 通过分析 UML 模型的 XMI 格式文件访问 UML 模型信息,UMLParser 通过 UML 建模工具提供的 API 访问 UML 模型信息,STCnstrts 是 UML 模型的统计测试约束,包括用例和场景约束 UCCnstr 以及用例执行顺序关系 UCExecRelation.UC2MC 构造 UML 用例的 Markov 链,MCIntegrator 根据用例执行顺序关系集成用例的 Markov 链构造软件使用模型.UserInterfaces 负责收集用户的输入信息并提供相关信息反馈.

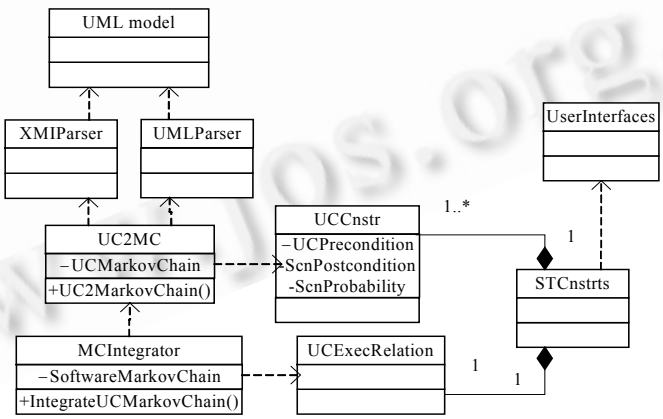

Fig.10 Class diagram framework of automation tool UMGen

图10 Markov链使用模型自动生成工具UMGen的类图框架

4 结论与未来工作

本文详细讨论了从软件 UML 模型构造软件 Markov 链使用模型,提出了一个从基于 UML 的软件开发早期制品生成软件 Markov 链使用模型的完整方法.该方法基于 UML 的用例图和序列图,并要求加入统计测试约束.

在此基础上给出一种算法,可以自动生成软件 Markov 链使用模型.本文最后给出了支持工具的结构.

本文工作的创新包括:通过规范用例中场景的执行概率,提供了一种获取 Markov 链使用模型迁移概率的简便方法;同时,规范了用例执行前置条件和场景执行后置条件,并规范了用例执行顺序关系.以上规范的内容作为统计测试约束,用于构造 Markov 链使用模型.

我们在上述约束的基础上,给出了从 UML 模型构造 Markov 链使用模型的算法.

今后的工作包括:从带时间约束的 UML 模型生成软件 Markov 链使用模型,这是实时软件统计测试的关键;结合使用其他 UML 模型信息,如类图的信息,为在相关值域选择具体测试数据提供依据.

References:

[1] Prowell SJ, Trammell CJ, Linger RC, Poore JH. Cleanroom Software Engineering: Technology and Process. Addison-Wesley, 1999.

[2] Yan J, Wang J, Chen HW. Automatic generation of Markov chain usage models from real-time software UML models. In: Ehrich HD, Schewe KD, eds. Proc. of the 4th Int'l Conf. on Quality Software. Los Alamitos: IEEE Computer Society Press, 2004. 22−31.

[3] Feng H, Xu XS, Wang J. The determination of resemblance between the usage chain and the test chain in statistical testing. Computer Engineering & Science, 2003,25(1):17−19 (in Chinese with English abstract).

[4] Binder RV. Testing Object-Oriented Systems: Models, Patterns and Tools. Addison Wesley Longman, Inc., 1999.

[5] Rumbaugh J, Jacobson I, Booch G. The Unified Modeling Language Reference Manual. Addison Wesley Longman, Inc., 1999.

[6] Briand L, Labiche Y. A UML-based approach to system testing. Software and System Modeling, 2002,1(1):10−42.

[7] Regnell B, Runeson P, Wohlin C. Towards integration of use case modeling and usage-based testing. Journal of Systems and Software, 2000,50(2):117−130.

[8] Riebisch M, Philippow I, Götze M. UML-Based statistical test case generation. In: Aksit M, Mezini M, Unland R, eds. Proc. of the Int'l Conf. NODe 2002. LNCS 2591, Heidelberg: Springer-Verlag, 2003. 394−411.

[9] Beyer M, Dulz W, Zhen F. Automated TTCN-3 test case generation by means of UML sequence diagrams and Markov chains. In: Proc. of the 12th Asian Test Symp. Los Alamitos: IEEE Computer Society, 2003. 102−106.

[10] Douglass BP. Real-Time UML: Developing Efficient Objects for Embedded Systems. 2nd ed., Pearson Education, Inc., 2000.

[11] Damm W, Harel D. LSCs: Breathing life into message sequence charts. In: Ciancarini P, Fantechi A, Gorrieri R, eds. Proc. of the 3rd Int'l Conf. on Formal Methods for Open Object-Based Distributed Systems. Kluwer Acdemic Publisher, 1999. 293−312.

[12] Alur R, Etessami K, Yannakakis M. Inference of message sequence charts. IEEE Trans. on Software Engineering, 2003,29(7): 623−633.

[13] Whittaker JA, Thomason MG. A Markov chain model for statistical software testing. IEEE Trans. on Software Engineering, 1994, 20(10):812−824.

附中文参考文献:

[3] 冯华,徐锡山,王戟.统计测试中测试链与使用链的相似性判别.计算机工程与科学,2003,25(1):17−19.