一个基于UML协作图的集成测试用例生成方法

王林章,李宣东,郑国梁
(南京大学计算机科学与技术系,江苏南京210093)

摘 要: UML协作图描述了系统的一个协作过程中参与对象之间的结构关系和交互行为,确认它们是否被正确实现是集成测试的工作.本文提出了一个基于UML协作图生成集成测试用例的方法,将表示设计的协作图作为测试模型,首先通过遍历每条消息的直接后继识别协作图中的表示用例实现的所有可能的场景路径,然后在遍历每条场景路径的过程中获取相应协作执行的路径条件、参数变量和预期方法调用序列,最后使用范畴-划分方法确定场景路径上的输入、输出、环境条件的合理组合作为覆盖该场景路径的测试用例,用于测试一个协作场景路径上的交互行为.该方法,集成了白盒方法和黑盒方法,在覆盖所有的测试需求的前提下,生成的测试用例较少.

关键词: 测试用例生成; 集成测试; UML协作图; 场景路径
中图分类号: TP311.5 **文献标识码**: A **文章编号**: 0372-2112(2004)08-1290-07

An Approach to Generate Integration Test Cases Based on UML Collaboration Diagrams

WANG Lin-zhang, LI Xuan-dong, ZHENG Guo-liang
(*CS Department, Nanjing University, Nanjing, Jiangsu* 210093, *China*)

Abstract: UML collaboration diagrams represent the structure relationship and interactive behavior of the objects involving in a collaboration of the software system, whether they are correctly implemented or not could be validated by integration testing. An approach is proposed to generate integration test cases based on UML collaboration diagrams. It takes a collaboration diagram as the test model, it identifies all the *scenario* paths in the diagram which represents use case realization by traversing the direct successors of each message. It selects and traverses each *scenario* path to get the method call sequence, path condition and parameters. It applies category partition method to generate rational combination of input parameters, environmental conditions, as well as the corresponding output and method call sequence, to form a test case for each *scenario* path. This method, combines white-box and black-box test method to generate fewer test cases to test the gray-box behavior, as well as to cover all the integration requirements.

Key words: test cases generation; integration testing; UML collaboration diagram; scenario path

1 引言

面向对象技术和统一建模语言(UML)在软件工程发展进程中具有里程碑的意义,自提出后,先后成为研究热点,并且迅速在工业界得到广泛的应用,他们对开发高质量软件起了很大的促进作用,但仍不能保证软件零缺陷,还给软件测试带来新的挑战[1~3].

软件的复杂度随着软件规模的不断增大而增大,面向对象系统开发过程中克服软件复杂性的思想是将软件系统划分不同粒度的基本单元分别实现,如类、组件、子系统等,然后通过集成这些单元来构造系统.单元之间通过交互实现系统的行为,所以对交互过程的建模也是设计阶段重点关注的内容,UML协作图[4]描述了上述交互,表示了行为的集成.在结构和行为集成的每一个环节都可能引入错误,导致软件中存在缺陷,要发现并排除这些缺陷,集成测试非常重要[5].在面向对象语境下,传统的集成测试方法不能直接使用[6],而且用UML模型描述系统给信息的提取带来了新的问题.

测试工作的核心主要是生成测试用例,在基于UML的面向对象软件系统中,分析模型、设计模型是软件在其生存期不同阶段的变体,是生成最终程序的基础,也是生成测试的信息来源.当然他们在每一阶段的测试中都起相应的作用,特别地,分析模型是生成系统测试的测试用例的基础和系统测试的检验依据,设计模型是生成集成测试用例和集成测试的检验依据.设计模型包含规约和程序结构的信息,同时也描述了

收稿日期:2003-07-28;修回日期:2004-03-20
基金项目:863项目(No.2002AA116090);自然科学基金项目(No.60207036,60233020);973项目(No.2002CB312001)

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

系统的相应功能片断的行为,因此可以结合白盒测试和黑盒测试方法,从设计模型生成集成测试用例.在设计阶段结束后,利用成为基线的设计模型生成所需的测试用例,可以在代码阶段结束后便可以开始测试工作,便于合理组织测试资源,而且对设计模型进行分析的同时也能发现设计本身的缺陷,以便及时排除,以防缺陷随着软件开发过程的进展而被放大.我们希望能够研究出仅从 UML 设计模型图自动生成测试用例的方法,能够实现部分自动化而不增加用户额外的工作量,这样的测试方法容易被已经使用 UML 的工业界采用[7,8].

2　协作图与集成测试

2.1　协作图的语法和语义

UML 协作图就是用来表示一组通过交互来实现某些行为的对象,可以用来按交互中的角色及其关系对一个用例的特定的实现场景进行建模.它描述了特定行为的参与对象的静态结构和动态交互.动态交互部分是一个消息集合,用协作实现过程中的消息流定义系统行为方面的内容[9,10].

本部分介绍一个客户通过 ATM 上验证用户卡和 PIN 有效性、建立会话的用例片断[5]的实例级协作图,如图 1 所示.协作图上存在条件消息说明考虑到许多不同的用例场景,涉及到多个对象之间的交互,其中每一个场景都对应协作图上的一条执行路径,这正是测试所要尝试的.图 2 表示了在这个协作中参与对象相应的类图.

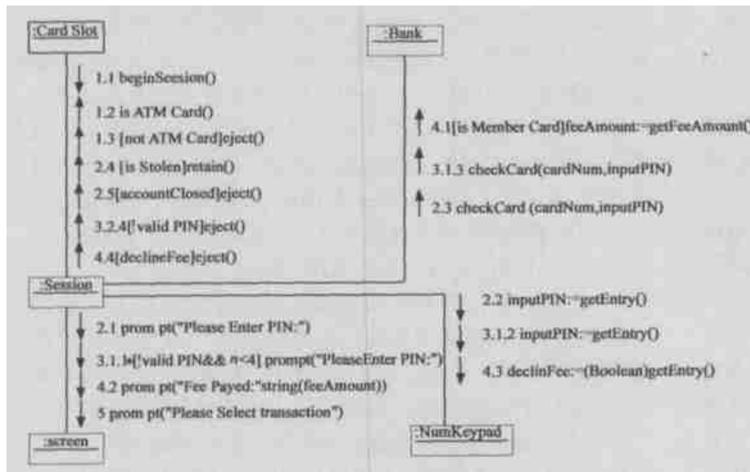

图 1　ATM 建立一次会话实例的协作图

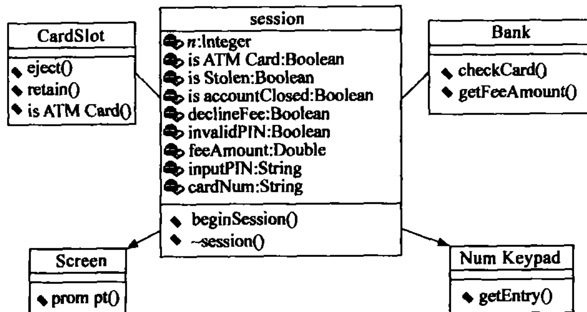

图 2　图 1 ATM 用户验证的类图

一个描述用例实现的实例级协作图上用于建模的元模型包括:类元角色名、关联角色名、对象符号、链接符号、消息符号、链上的约束等.对象框能够表示参与协作中交互的对象,链表示这些对象之间的结构关系,消息的箭头形状表示协作对象间的通信模式和对象的执行特征,消息标签表示对一个对象的操作调用的允许顺序的实例、操作的语义.在实际的建模活动中有些元素并不在模型上反映,例如图 1 的协作图只表示了协作中参与的对象之间的消息,这是表示对象交互的最小元素集.本文关注协作图表示的动态行为的测试,前面提到协作图上表示交互的主要元素是消息,下面详细介绍消息.

协作图中的消息用带有消息标签的箭头表示,附在连接发送者和接受者的链上,链用于访问目标对象,箭头沿着链指向接受者,一个链上可以有多个消息,沿着相同或不同方向传递,消息的相关顺序由消息标签中的顺序号表示.按照 UML1.5[6]中提出消息箭头有三种形状表示对象之间的通信方式和控制流类型:实线实心箭头表示同步消息,用于表示过程调用和嵌套控制流,每次调用增加一个嵌套层次;刺状箭头表示平面控制流,是异步通信,无嵌套,表示前驱-后继关系,指向下一步骤;虚线刺状箭头表示从过程调用返回.图 1 中的消息都是带有刺状箭头的实线,表示前驱/后继关系.消息标签说明发送的消息、发送的条件、由消息激活的方法、相应的参量和返回值,以及交互中的消息顺序,包括调用嵌套、迭代、分支、并行和同步,完全格式消息标签的语法文献[10]中有详细的定义.其语法如下:[predecessor] [guardcondition] [sequence-expression] [return-value-list] : = message-name (argument) [predecessor]表示前驱,在协作中前驱是用逗号隔开的顺序号列表,并后跟“/”;[guardcondition]表示线程同步的卫式条件;[sequence-expression]表示顺序项列表,每一项表示交互中的一个嵌套层次,如果所有的控制并行则无嵌套;整数表示过程调用中的相邻高层中的消息的顺序,同一整数项中的不同消息在嵌套层是顺序相关的,同一前缀的消息顺序号构成序列.循环表示条件或迭代执行,表示根据条件执行零个或多个消息;[return-value-list]在有返回值的情况下表示消息返回值的名称,名称之间用逗号隔开,可以作为后续消息的参量;message-name 表示目标对象中引发的事件名称,可以用不同的方式实现,如操作调用;argument 是消息引发的方法的参量列表.

消息上的条件增加了消息的表达能力,也增加的消息复杂度,消息上的条件有几种形式:在一个新的调用层次开始,如图 1 中消息标签 4.1[isMemberCard]feeAmount: = get-feeAmount()所示:4.1 表示一个新的调用序列,条件成真时该层次的消息顺序发送,否则都不发送;在顺序发送的消息上,例如消息 1.3 中,[notATMCard]条件表示二值条件,条件成真时其后的消息 eject()执行一次,否则不执行,对其他消息没有影响;在一个表示迭代的嵌套调用消息开始,如消息 3.1.1 表示了一个迭代执行多个消息的情况,根据 validPIN 和 n 的取值可能发送 1 或 2 次消息.

这些协作图上的模型元素所表示的软件系统的不同方面

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

的信息,都是系统的本质信息,需要在软件从设计到实现过程中必须保持的,因而软件实现中的行为的本质方面都会在设计中表示,所以可以从设计表示中获取行为方面的信息,用于确认软件实现中是否正确实现.

2.2 场景路径

一个表示用例实现的协作图实现了用例的不同事件流,包括正常流和异常流,每个事件流称为一个场景,这些场景都被相应的协作图实现.一个协作图所表示的协作正是从一个外部事件触发开始,在参与协作的对象之间完成一系列的交互,最后正确的协作结束时都导致一个外部输出,这也是一个线程执行的路径.要找到这些路径,一般的方法是将协作图转换为流图,然后在流图上使用图遍历的方法达到路径覆盖从而得到所有的路径.本文为了避免复杂流图的转换和不可行路径的遍历开销,试图直接从协作图上获取这些路径.我们从 Paul C Jorgenson[5]定义的原子系统功能(ASF)和方法/消息路径(MM-path)得到启发,本文定义了一个场景路径(scenario-path):

定义 场景路径(scenario-path,s-path)是协作图上从一个没有前驱消息的消息开始,沿着消息的偏序执行序列,到达一个没有后继的消息结束的由消息相连的方法执行序列.

由于面向对象软件的事件驱动特性,软件的执行由事件开始,反映场景执行的场景路径由一个外部事件触发的消息(没有前驱消息)开始,表示路径入口,通过消息传递访问协作图中参与一个协作场景实现交互的对象的方法序列,达到一个引发端口输出事件的方法且该方法自己不再发出新的消息(没有后继消息)结束,到达路径出口,这时系统处于静止状态,等待另一个系统级端口输入事件开始进一步的处理.场景路径表示了协作图中的一个场景的线程执行的完整踪迹,也是参与协作的对象之间的交互的踪迹,消息的传递激活对象方法的执行是对象之间控制流的转移,而路径中通过消息激活的方法调用的参数定义和使用、对象的创建、使用和撤销明显表示了一条在控制流上执行的数据流.

由于协作图在表示用例实现时,协作图上的场景路径是从第一条入口消息开始到所有能触发端口输出事件或没有后继消息的出口消息之间的所有可能的消息流.要获取场景路径上的消息流,可以通过消息之间的偏序关系以及决定消息流分支和循环的条件来实现.回顾协作图上消息及其顺序号的定义,整数表示过程调用中相邻高层中的消息顺序,同一整数项下的不同消息是表示一个前驱-后继式的平面控制流,嵌套的消息顺序号表示过程的调用控制流.消息顺序号隐含地表示了消息之间的偏序关系,因此可以在协作图上提根据消息的发送条件及其顺序号来跟踪场景路径.协作图上的分支和循环导致了许多可能的路径,极端的情况下路径的数目可能到达一个庞大的数字,要达到协作图上的路径覆盖,采用简单的方法达到判定/循环覆盖.先将循环消息看作条件消息,在遍历场景路径的过程中,访问一条消息后,先找出该消息可能的直接后继,然后采用深度优先的方法选择一个直接后继继续遍历,当到达一个没有后继的消息时,就完成一个场景路径的遍历,回溯到前一消息的下一个直接后继,继续遍历,只到所有消息的所有直接后继都已被访问,则完成了所有的场景路径的遍历.这样实现每条消息至少遍历一次,每个条件分支都至少遍历了一次,也避免了路径的穷尽覆盖.然后处理协作图上的循环,一般是给出绕过循环、循环一次、循环最多次以达到循环路径的覆盖.对于存在分支和循环的场景路径,例如图 1 所示的协作图上有 7 个条件判断和一个最大次数为 2 的循环,可能的路径有 $2*2*2*2*3*2*2*2=384$ 条,但由于有 5 个条件判定各有一个分支直接导致场景路径结束,所以实际的场景路径应为 $1+1+1+1+1+3*2+2=13$ 条,这 13 条场景路径对应相应软件功能的 13 个执行场景.

在协作图上遍历生成场景路径的同时,很容易基于路径覆盖准则获取相应场景执行过程中的控制流和数据流、覆盖路径的条件,然后可以确定每一路径所需要的输入和状态条件,当满足所有路径条件时线程就会沿着该路径执行,这正是定义集成测试用例所需要的信息,因此被看作基于协作图的集成测试的基础.下面分析协作图表示的协作在实现中通过参与者之间的集成完成时可能发生的故障,这样便于研究针对检错为目的的测试.

2.3 基于协作图的协作集成测试模式

协作图上描述的消息的错误实现可能导致协作中表现的行为发生偏差,从而使最终实现与设计不一致,偏离软件规约规定的功能.例如由于消息名的编码错误、错误的参数、不正确的参数值、不正确的或缺少输出以及非预期的运行时绑定导致的错误方法调用;消息前驱约束实现错误导致发送者对象发送违反接受者对象前提条件的消息或者发送者对象发送违反接受者对象的顺序约束的消息;消息的发送者或接受者对象错误导致错误的对象和消息的绑定;参与者缺少功能或特征导致错误的对象引起正确的异常、正确的对象产生错误的异常.所以链上的消息箭头和约束、消息标签、对象等协作图上的元素未按设计正确实现会导致最终的软件与设计不一致,如果协作图的实现中存在错误,肯定在协作图至少一条路径上,要定位该错误,在未知该错误在那条路径上时,只能通过协作图上选择所有可能的执行路径,并导出能够跟踪这些路径的测试用例,并用它们来执行软件,来确定错误在哪条路径上,从而可以调试软件,改正错误,以保证实现与设计一致.

这些可能发生的错误都是在参与协作的对象之间交互时发生,也就是系统在通过不同对象的方法之间集成实现系统的行为时发生,这种错误通常通过集成测试来发现.在面向对象的语境中我们可以通过基于线程或基于使用的测试技术测试对象的交互.用协作图描述的交互行为在集成测试时,协作集成是最佳的测试模式[5],通过在协作中参与的组件来决定集成测试所有参与的对象.基于协作的集成测试需要检测协作的参与者之间的接口和交互,通过协作组织集成.每次处理一个协作,直到协作图上的所有协作都被测试,也即要求系统中每个对象之间的消息已经被至少测试一次时集成测试完成.

协作图是对象、组件、子系统、系统范围测试需求的良好来源,对系统的一个有限片断来说,提供了实现的抽象视图,它通常是过多或过少细节之间的一个良好的折中,可以用于

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

在不同抽象级别和粒度级别建立软件系统的模型. 由于协作图是描述的对象之间的交互,而系统的功能正是通过这些交互实现的,所以要考虑将设计描述的协作图作为生成集成测试的测试用例的测试模型. 协作图上包括了对象间传递的消息及其顺序,这正是设计级的控制流和数据流信息,以往数据流和控制流只能从程序源码中分析得到. 数据流和控制流对生成测试有很大的作用,所以我们利用协作图生成测试用例,就是要从协作图上提取出相应的数据流和控制流信息,利用传统的数据流、控制流生成测试用例的方法,生成可用于集成测试的测试用例. 所以本文使用作为系统设计描述的协作图作为测试行为的测试模型,避免重新构造测试模型或者进行模型转换.

2.1 中分析了协作图中描述的信息,这也是作为测试模型的协作图中包含的对生成测试用例有用的信息,这些信息是规约在转变成实现时必须保持的,这些信息也是测试时要确认在软件实现中是否正确保持的设计信息,因此这就是测试工作第一步要获取的测试需求. 测试需求的正确实现要求也就是测试规约,它规定了能让软件执行并正确反应测试需求的测试用例. 如果我们能够用足够的测试用例执行了程序,并证明协作图上所有测试需求都在软件中正确保持了,就可以说测试充分了,本文主要关注协作图表示的系统的动态交互行为,而协作图上用于表示交互行为的主要是消息及其偏序序列,所以本文研究的测试方法的充分性准则是协作图上所有的消息及其偏序关系都被测试用例覆盖.

一个场景路径上由消息激活的方法序列表示了要测试的行为,路径上对象间的消息传递描述了要实现相应的功能对象间必要的交互,协作图上场景路径的覆盖能够满足充分性准则,这样就可以把问题转换到满足协作集成测试需求的场景路径的分析和处理上,第 3 部分详细描述从协作图生成测试用例的方法.

3　基于协作图生成集成测试用例的方法

3.1　研究假定

为了有针对性地解决从协作图生成测试用例的问题,本文假定:(1) 协作图描述的协作与用例图描述的规约是一致的. 模型本身的验证是通过非形式化的复审和形式化的模型检验方法进行的,已超出本文的研究范围;如果协作图上的场景路径集不能覆盖所有消息,则说明协作图本身有错误;(2) 系统中的对象都是自行开发的,不包括第三方组件对象,文中的对象可以是不同粒度的对象(类的实例、类簇、组件、子系统、系统),也就是说我们在分析任意对象或类时,在必要的情况下都可以获取其规约和内部详细设计信息;(3) 只研究针对检错进行的动态交互行为的测试;(4) 本文为了明确解决问题,假定消息类型只有普通消息、条件消息、循环消息,而且只存在顺序循环,不存在嵌套循环.

3.2　方法概述

基于协作图的集成测试(Collaboration diagram-based Integration Testing ,CIT)方法集合了传统的白盒测试方法和黑盒测试方法,用于测试协作图中参与协作的对象之间通过消息的交互,对每个协作图处理一次,得到相应的测试用例集. 首先分析协作图,根据消息的顺序号和消息的条件,找到每一条消息的直接后继消息,然后根据场景路径的定义,使用深度优先方法遍历消息及其直接后继直至到达无直接后继的消息从而生成场景路径,然后回溯到没有被访问的直接后继,重复上述方法找到所有的场景路径;在访问消息获取场景路径的同时,获取该路径的方法调用序列、参数和路径条件,将这些集成测试的关键因素用范畴-划分方法定义为方法序列、环境条件、系统输入、系统输出等范畴,结合该协作片断的用例规约和类图中的定义生成这些范畴的可能选择,然后结合路径约束条件在这些范畴的划分中确定选择项的合理组合,这样我们就等到了该场景路径完整的测试用例,包括外界输入、交互输入、预期方法调用序列、后条件、预期输出. 对协作图中的所有场景路径都构造了测试用例,就形成了协作集成测试用例集. 这样在实际执行集成测试时不但可以直接观察到系统级的输入作用下协作实现过程中的实际输出,还能够通过动态插装方法在代码中加入不影响软件功能的观察代码,使测试人员能够观察到实际协作执行时的方法调用序列和数据流的定义和使用,然后通过比较最终系统实际执行时输出与预期输出的一致性决定该协作实现的功能是否正确,通过比较应该发生的方法调用序列和实际执行时观察到的方法调用序列是否一致,确定协作表示的交互行为是否正确.

本方法可以以增量的方法进行,最终生成的测试用例的具体程度,与相应 UML 模型图的设计精化程度相关,因为协作图描述的场景的详细程度与测试用例的表达能力直接相关.

3.3　UML 协作图生成测试用例的算法描述

CIT方法能够从协作图规约文档直接生成测试用例,主要描述分析协作图规约文档提取集成测试需求信息并生成消息后继表的算法 UMLspecificationparser(),然后为每个消息确定其直接后继消息的 findsuccessor()算法,根据消息后继表遍历所有场景路径获取测试用例规约信息的 spathgenerator()算法,以及从测试用例规约使用范畴划分方法确定测试用例输入值、预期输出值、预期输出行为的算法 testcasegenerator(). UMLspecificationparser()算法从协作图从获取集成测试需求信息是通过对 UML 协作图规约的文本文件(如 Rational Rose 的 MDL 文件)进行分析完成的[11],在本文的工具框架中实现相应的功能,本文对分析算法不作详细描述. 本文用邻接表定义消息后继列表,将分析结果信息按消息顺序号升序记录到 messagelist 的表头中,以便下一步处理. 本文定义消息后继列表如下:

```
messageitem{
    mid:string;// 消息顺序号
    mguardcondition:string;// 消息卫式条件表达式
    mlabel:string;// 消息标签
    msender,mreciever:object;// 消息发送者对象,消息接收者对象
    mmethod:method;// 消息激活的方法
    mtype:[flat flow,procdure call,call return];// 根据消息的箭头及线型划分的消息类型
    link:pointer of msuccessor;
```

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

```
}
msuccessor{
    mid:string;// 后继消息顺序号
    mscondition:string;// 选择该后继的条件表达式
    next:pointer of msuccesor
}
messagelist:messageitem[n];
```

findsuccessor()算法为消息生成直接后继列表并记录到消息的后继链表中,描述如下:

```
findsuccessor(messagelist){ //找到所有消息的所有可能后继和每个后继条件
    //输入:消息后继表 messagelist
    //输出:消息后继表 messagelist
    //出口准则:所有消息都处理完
for i:=1 to n do {
    if i= =n 或者 messagelist[i]为端口输出事件触发消息
      messagelist[i].link=nil;i:=i+1;//最后消息或触发端口输出事件的消息无后继消息
    else{
      if messagelist[i+1]是普通消息
        则 messagelist[i]的直接后继为 messagelist[i+1];
      else{
        k:=1;
        do while messagelist[i+k]是条件消息{
            messagelist[i+k]是 messagelist[i]的直接后继,且条件为 messagelist[i+k]的条件取真
            if messagelist[i+k]是循环或嵌套层次的第一个消息{
                t:=该层次的消息数;messagelist[i+k+t]是 messagelist[i]的直接后继;
                条件为 messagelist[i+k]的条件取假;k:=k+t;}
                //条件取假时循环或嵌套内的消息都不发送
        else{
            messagelist [i+k+1]是 messagelist [i]的直接后继;
            条件为 messagelist [i+1]的条件取假;k:=k+1;//不发送
        }endif
      }enddo
    endif
    i:=i+1;
  }endif
}endfor;
}endfindsuccessor;
```

spathgenerator()通过访问消息后继表中消息及其直接后继生成场景路径,在遍历场景路径时同时记录路径上相应的控制流和数据流信息,执行完毕得到一个场景路径集 spath 相应的控制流数据流测试规约.其算法描述如下:

```
s-pathgenerator(m:msuccessorlist){
    //输入:messagelist;(消息后继表)
    //输出:spath;(场景路径集)
spath{
    sid:integer;//场景路径顺序号
    pathcondition:list of gardcondition;//场景路径的实现条件
    mlist:list of message;//场景路径上的消息流列表
    parameters:list of variable;//场景路径上定义和使用的变量列表
    userinputparameters:list of variable;//场景路径上外界输入参数列表
    methodsequence:{objectname,methodname,next};场景路径上由消息激活的方法序列
}[n];
i,j:integer;
i:=1;j:=1;
Sid:=j;//场景路径序号
//递归访问消息及其直接后继
访问消息 m,在消息标签中取出并记录返回值、参数、调用的对象方法;
记录消息条件到路径条件中;
if m 的后继消息为空
{    一条场景路径结束;j:=j+1;return;}//回溯到 m[i]的直接前驱继续
else{
    k:=1;
    while m.link< >nil do{// m 的后继消息列表不空
        successor:=m.link;
        spathgenerator(successor) //深度优先遍历该消息及其后继消息
        m.link:=successor->next;//下一后继消息;
    }endwhile
}endif;
}endfind;
```

testcasegenerator()选定遍历一个场景路径过程中生成的控制流和数据流,定义相关范畴,生成测试用例的规约.消息激活的方法序列正是我们要测试的对象间的交互,是软件执行时表现出来的对象之间的控制流传递行为,是对象之间通过消息交互的结果,定义为交互范畴 Interaction Category;输入参数列表为测试该场景时外界的输入,定义为输入参数范畴 input parameter Category;我们通过对参与协作对象的类图的分析,以及用例描述,可以获取我们所研究的交互行为定义、输入参数的定义、其他影响消息执行的变量的定义、系统环境条件的定义等,从而可以获取这些系统环境条件、输入参数、方法行为的可能选择,然后结合.利用范畴-划分思想,用相应场景路径的路径条件和加在消息上的约束、以及系统本身的属性约束来限定这些选择,排除无意义和有冲突的值,然后通过这些不同的范畴的不同选择的可能组合作为一个场景路径的测试用例,这样每个场景路径生成一个集成测试用例.具体算法描述如下:

```
Testcasegenerator(){
    //输入:spath[n]
    //输出:tcsuite[n]{envirnmentpara,input(inputpara,paravaliue),expectoutput,postcondition}
i:integer;
for i:=1 to n do {
    tcsuite[i].postcondition:=spath[i].pathcondition;
    tcsuite[i].environmentpara:={满足路径条件的环境条件}
    tcsuite[i].input.inputpara=spath[i].userinputsparamenters;
    选择对本场景路径条件有影响的参数确定其可能的取值;
```

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

tcsuite[i]. input. paravalue = {符合路径条件的输入参数可能的取值};

取出 spath[i]的方法激活序列,分析每一个方法的定义,确定其可能的行为,选择符合场景路径条件的行为,并将方法序列的执行过程的输出序列存到 tcsuite[i]. expectoutput 中;

$i := i + 1$;

}endfor;

}endtestcasegenerator;

我们可以用同样方法为属于同一个用例的其他协作图构造了测试用例,则该用例的集成测试集构造完成.然后,可以用增量方式为组件、子系统、系统构造完整的集成测试用例集,而且可以用自动的方式进行.

3.3 支撑工具结构设计

为了支撑上述测试用例生成方法的自动化,我们设计一个基于 UML 设计模型图的集成测试用例的自动生成工具 UMLTGF,我们仍然使用 UML 为 UMLTGF 建模,本文给出 UMLTGF 的类图,如图 3.相应的功能描述如下:

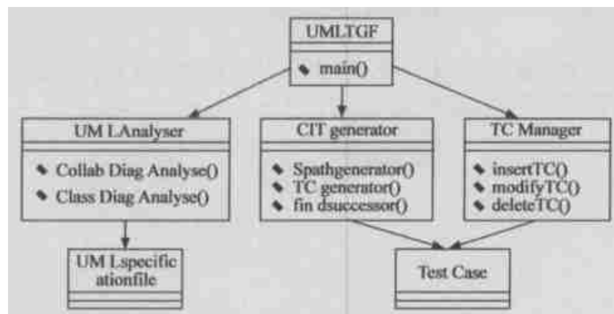

图 3 UMLTGF 类图

(1) UML 模型分析器.能够直接读取 UML 协作图规约的文本文件(如 MDL 文件),进行解析,并获取对象角色、对象的属性和方法,链、约束,消息流和消息标签,存入相应的中间表,以便测试用例生成器使用;(2) 协作集成测试用例生成器.根据消息的顺序号和条件,跟踪消息流,生成每条消息的直接后继,然后通过遍历消息及其直接后继的方法生成协作图上每个用例场景对应的协作的场景路径,同时可以获取场景执行的控制流和数据流,并用范畴-划分方法生成相应场景路径对应的集成测试用例.输入为从协作图中提取的信息中间表,输出为生成的测试用例集合;(3) 测试用例管理.为使得生成的测试用例集可复用、可维护、无冗余,对每个测试用例设计了增加、修改、删除操作,用来在测试用例生成过程中管理这些测试用例集.

4 相关研究和比较

基于 UML 设计模型生成测试用例的研究近年来有不少成果[12~15],文[12~14]需要从设计模型中提取信息重构测试模型,然后用常规方法生成测试,都研究在组件集成层次上的交互;文[15]初步探索了直接使用设计模型生成测试,但只研究的静态检查方法,没有能够给出生成用于动态测试的方法;有的方法还处于理论探讨阶段,有的已经实现了概念证明工具(Proof of concept tools).

对于面向对象软件系统的集成测试,最充分的测试集应该是覆盖到所有的事件序列,但覆盖粒度的不同,如消息序列、事件序列,所需的测试代价不同,本文方法考虑到测试的成本,只覆盖了所有的消息序列,能够测试系统行为的所有可能场景,是测试用例细节过多或过少的一种折中选择,这样测试用例的数目较少,易于实现.本文的方法完全基于 UML,直接利用协作图设计模型,避免了重构测试模型或模型转换的开销,只在方法实现步骤的最后用范畴 - 划分方法确定最终的测试用例值时需要用户交互干预,其他部分便于实现自动化.但是假设的前提和约束太多,还不能够达到实用的程度,对方法提供自动支持的工具还是框架设计,还需进一步深入研究.

5 总结和将来的工作

本文提出了一个直接基于 UML 协作图,采用协作集成测试模式生成集成测试用例的方法,用基于线程执行的方法识别协作图中的场景路径,遍历场景路径获取交互中的参数变量和方法调用序列,最后使用范畴-划分方法找到场景路径上的变量、方法、输入、输出、环境条件的合理组合作为覆盖该场景路径的测试用例.

尽管 UML 在工业界和研究领域用得相当广泛,对基于 UML 生成测试用例的研究还存在许多不足.缺少整体的、系统的解决方法,还没有能够把待测试系统的所有能够为测试用例生成提供信息的模型图综合利用,形成一个系统的测试用例生成方法,这也正是我们将来的研究目标:第一,提出一种与面向对象软件开发过程集成的测试过程,测试用例应以增量、系统的、与项目成本和进度相适应的可管理的数目的方法产生,利用业务模型生成用户验收测试的测试用例,利用 UML 分析模型生成系统测试的测试用例,利用 UML 设计模型生成集成测试的测试用例,利用实现模型生成单元测试的测试用例,能够合理计划测试资源,并能够对软件开发过程起到过程改进的作用;第二,针对 UML 单个模型图,研究其在为不同层次测试生成测试用例的方法,提出相应可行的测试准则和评估方法,尽量能够直接使用 UML 模型文档;第三,针对某一层次测试的测试用例生成时,研究综合利用待测试系统的各种模型图生成该测试层次的测试用例的方法;第四,为上述方法提供自动的工具支持,并希望能够与主流建模工具、测试工具集成.

参考文献:

[1] Imran Bashir, Amrit L Goel. Testing Object-Oriented Software: Life Cycle Solution[M]. New York: Springer-Verlag, Inc, 1999.

[2] David C Kung, Pei Hsia, Jerry Gao. Testing Object-Oriented Software [C]. USA, IEEE Computer Society, 1999.

[3] Beizer. Black-Box Testing: Techniques for Functional Testing of Software and Systems[M]. New York: John Wiley&Sons, Inc, 1995.

[4] UML Specification 1.5[S]. available at http://www.omg.org/uml, March 2004.

[5] Paul C Jorgrnsen. Software Testing: A Craftsman's Approach[M]. CRC

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

Press ,Inc ,1995.
[6] Robert V Binder. Testing Object-Oriented System: Models , Patterns , and Tools[M]. Addison-Wesley ,2000.
[7] Grade Booch ,James Rumbaugh , Ivar Jacobson. The Unified Software Development Process[M]. Addison-Wesley ,2001.
[8] Philippe Kruchten. The Rational Unified Process-An Introduction[M]. 2nd edition ,Addison-Wesley ,Reading ,MA ,2000.
[9] Grade Booch ,James Rumbaugh , Ivar Jacobson. The Unified Modeling Language User Guide[M]. Addison-Wesley ,2001.
[10] Grade Booch ,James Rumbaugh , Ivar Jacobson. The Unified Modeling Language Reference Manual[M]. Addison-Wesley ,2001.
[11] A J Offutt ,A Abdurazik. Generating tests from UML specifications[A]. Proc. 2nd International Conference on the Unified Modeling Language (UML '99) [C]. Fort Collins ,CO ,October ,1999. 416 - 429.
[12] Hartmann J , Imoberdof C ,Meisenger M. UML-based integration testing [A]. ISSTA 2000 Conference Proceeding[C]. Portland ,Oregon ,22 - 25 August 2000. 60 - 70.
[13] Byoungju Choi , Hoijin Yoon ,Jin - Ok Jeon. A UML-based test model for component integration test[A]. Workshop on Software Architecture and Component[C]. Japan ,1999. 12. 63 - 70.
[14] Basanieri F ,Bertolino A. A practical approach to UML-based derivation of integration tests[A]. Proceeding of QWE2000[C]. Bruxelles ,3T , November 20 - 24 ,2000.
[15] A J Offutt ,A Abdurazik. Using UML collaboration diagrams for static checking and test generation [A]. Proc. 3rd International Conference on the Unified Modeling Language (UML '00) [C]. York ,UK, October ,2000. 383 - 395.

作者简介:

王林章 男 ,1973 年生 ,江苏建湖人 ,博士生 ,主要研究软件工程、软件测试、模型驱动的测试. Email :wanglz @seg. nju. edu. cn.

李宣东 男 ,1963 年生 ,湖南邵阳人 ,博士 ,教授 ,博士生导师 ,主要研究软件工程、形式化方法、模型检验.

郑国梁 男 ,1937 年生 ,浙江桐乡人 ,教授 ,博士生导师 ,主要研究软件工程、形式化方法.

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net