一种软件测试需求建模及测试用例生成方法

杨 波[1)] 吴 际[1)] 徐 珞[1),2)] 毕 考[1)] 刘 超[1)]

[1)](北京航空航天大学计算机学院 北京 100191)

[2)](华北计算机研究所 北京 100083)

摘 要 测试在需求分析阶段就开始介入,不仅能帮助开发人员更有效地完善需求,也能让测试人员设计出更贴近需求的测试.同时当需求进行了更改之后,测试人员也能及时和准确地了解需求的变化、更改测试需求.利用模型驱动的思想,对测试需求的概念进行阐述.定义了一种测试需求的元模型和测试需求建模方法,利用测试需求建模方法,可以得到测试需求模型,从而得到相应的测试目标,生成所对应的测试用例.并给出了对 SIP 协议进行测试的实验分析,在其中体现了利用测试需求建模得出的测试需求模型的作用.

关键词 需求工程;软件需求;软件测试;测试需求;模型;测试用例

中图法分类号 TP301 **DOI 号** 10.3724/SP.J.1016.2014.00522

An Approach of Modeling Software Testing Requirements and Generating Test Case

YANG Bo[1)] WU Ji[1)] XU Luo[1),2)] BI Kao[1)] LIU Chao[1)]

[1)](*School of Computer Science and Engineering*, *Beihang University*, *Beijing* 100191)

[2)](*North China Institute of Computer Technology*, *Beijing* 100083)

Abstract Getting testers involved in both requirements analysis and testing can play an important role in three ways. To begin with, developers can improve the quantity of the software requirements, apart from that, testers facilitate the requirements of testing easily, and one more thing, testers can know the requirements timely and accurately after the software requirements are modified. We introduce the concept of testing requirements by making use of the idea of model-driven, and then present an approach of testing requirements modeling. We can get test cases according to the test requirements model which corresponding to the test objectives. We conducted a case study on sip protocol use this approach. The preliminary results show that our approach is feasible and effective.

Keywords requirements engineering; software requirements; software testing; testing requirements; model; test case

1 引 言

软件的需求与测试之间存在很紧密的联系,很长时间以来,需求和测试被认为存在先后的顺序关系.在进行软件需求分析阶段,测试被认为还没有开始.随着软件规模的扩大,这个观点得到了纠正.Dorothy[1]指出,在软件的需求分析阶段,测试就开

收稿日期:2012-07-20;最终修改稿收到日期:2013-09-20.本课题得到国家"八六三"高技术研究发展计划项目基金(2009AA01Z145)资助.**杨 波**,男,1981 年生,博士研究生,主要研究方向为软件测试、数据挖掘.E-mail: yangbo@sei.buaa.edu.cn.**吴 际**,男,1973 年生,博士,副教授,主要研究方向为软件测试、软件可靠性分析.**徐 珞**,男,1976 年生,博士,高级工程师,主要研究方向为软件测试.**毕 考**,男,1986 年生,硕士,主要研究方向为软件测试.**刘 超**,男,1958 年生,教授,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域为软件工程、软件测试.

始介入的话，能够使得软件的需求分析更加详尽.

软件系统的需求是软件系统设计与实现的基础，需求质量会影响架构师设计出的架构、程序员写出代码，并且也会影响测试人员进行测试.随着软件系统规模和复杂性的增加，软件测试的任务也变得越来越复杂与繁重.为了减少测试过程的盲目性，提高测试过程的效率，在需求分析的开始阶段，就对测试任务进行明确的分析，对测试目标进行有效的组织，并进行准确的记录显得尤为重要.正如需求分析是软件系统设计与实现的基础，软件测试也应该以准确而详尽的测试需求分析为基础.

在尽可能早的时候就开始进行的软件测试行为有以下几个特点：

(1)软件的测试在被测系统的需求分析之时，就应该开始进行，在开发者与用户进行沟通之后，双方根据系统的需要，以及测试的需要，制定出相关的需求之后，测试的需求就可以通过被测系统规格说明或被测系统模型建立.这样在软件系统生命周期的早期，测试人员便可着手设计系统的测试计划、分析系统测试任务的规模并对合适的资源进行安排.整个从需求开始阶段就进行的测试，不仅对于完善需求有很大的帮助，同时对整个测试过程来说也有着十分重要的意义.

(2)在理解清楚软件需求的基础之下，能够对软件的测试规划产生积极的影响[2].如果对软件的需求理解有误，那么将会导致对测试的需求理解错误，使测试过程产生偏差，要修正测试并重新实施可能会造成大量的人力、时间与资金的浪费.

(3)软件的需求有时会发生变化，这在很多软件项目当中非常常见，而往往需求的变化会带来测试的变化，从软件的需求分析阶段就开始进行软件测试的规划与设计，能够在需求变化时，准确地找到测试当中可能变化的内容，从而快速地进行相应的补充或者调整.

(4)软件的需求和软件的测试之间没有明显的先后之分，需求工程师和测试工程师之间可以相互合作，共同完善需求，这对于需求很重要，对测试也很重要.

(5)在软件开发过程中，软件缺陷越早被发现，改正它的代价就越小.根据美国国家标准技术研究所(NIST)估计，在发布后修正缺陷的代价是在单元测试阶段修正的 30 倍.而 IBM 的估计更是高达上百倍.因此，遵循尽早开展测试的原则，不仅可以提高软件质量，同时也能够降低开发成本.

(6)“重用”是软件开发中的一个重要问题[3-5].这种“重用”的思想对测试也产生了影响.一方面，由于被测系统中存在着重用的功能模块，因此势必引发了对重用相应的测试用例的诉求；另一方面，大量的已有测试制品(如测试设计、代码、自动化工具等)对于测试系统开发而言也是一笔宝贵的财富，重用这些制品势必可以提高新系统的开发效率.

因此在测试活动变得越来越复杂的今天，需要测试需求作为测试活动的基础，用于描述测试目标，精确定义需要测试的内容，并指导整个测试过程的进行.

在不同的文献与测试工具中，对于测试需求有不同的理解和描述方式.IBM 公司 Rational Robot① 利用自然语言来描述测试需求，并以层次结构的方式进行组织，以作为整个测试过程的基础.欧洲电信标准化协会(ETSI)对一致性测试过程定义了一套方法学[6]：测试人员首先基于被测系统需求规格说明或协议标准文档等设计测试套结构(Test Suite Structure，TSS)与测试目标(Test Purposes，TP)，然后通过测试目标 TP 得到抽象测试用例(Abstract Test Cases，ATC)，通过测试套结构与测试目标(TSS&TP)得到抽象测试套(Abstract Test Suite，ATS)，最后通过抽象测试用例 ATC 得到可执行测试用例，通过抽象测试套 ATS 得到可执行测试套.其中测试套结构与测试目标(TSS&TP)起到测试需求的作用，对测试目标进行分组并描述，作为后续测试活动的前提与基础.

测试目标的概念在一致性测试领域得到了广泛的接受，除了欧洲电信标准化协会定义了 TPLan 语言[7]，用来对测试目标进行半形式化的描述之外.Jard 等人[8]将 TP 定义为一种包含接受和拒绝两个特殊状态的输入输出的标记迁移系统，并且通过形式化测试目标，来指导测试用例自动生成.Tretmans 等人[9]也研究了测试目标描述和自动测试生成问题.

另外，conformiq② 使用模型标注的方式来表达测试需求，在利用 conformiq 来进行测试设计时，需要先使用 UML2.0 的状态机模型对 SUT 进行描述，接着再利用专门的语言来对状态机模型进行标注，标记出那些必须要覆盖的状态和迁移.这些标注都在此后的测试生成阶段，成为一个个对应的测试目标，这样在生成测试用例的过程当中，conformiq

① http://www.rational.com/products.
② http://www.conformiq.com/products/conformiq-designer

可以找到相应的测试用例来覆盖这些目标.

考虑到测试需求在不同的文献和工具当中的理解和定义并不相同,对测试的需求的描述方式也不一样.而测试的需求在整个测试过程当中相当重要,为此提出一种测试需求模型,来指导测试人员开展后续测试活动、方便测试人员与设计人员之间的沟通,还可在之后介绍的模型驱动测试的过程中对测试系统模型的生成提供必要信息.

模型驱动测试使用测试模型表示测试架构和测试行为,使用图形化的方式创建测试模型.图形化的测试模型便于测试人员理解和修改.这样将对基于特定语言的难于理解的测试脚本的维护转化成对通用直观的测试用例模型的维护,从而降低了测试人员的工作量,提高了工作效率.模型驱动测试可以实现从模型到测试脚本的自动转化,从而保持模型和测试脚本的同步.当应用程序发生变动时,测试人员只需要对测试模型进行修改,并通过测试模型自动生成测试脚本.

模型驱动测试可以有效地减少测试这一软件开发过程中初始阶段的工作量,并提高测试的覆盖率[10-11].目前,测试占据软件开发项目的 30%~70%时间和资源的花费.模型驱动测试的这种新的方法和相关的工具集将提高软件开发者和测试者工作效率,并且在保持软件质量高标准的同时,减少产品投入市场的时间.

测试需求模型的引入是模型驱动测试方法学的完善和补充,在系统设计模型向测试设计模型的转换过程中,测试需求模型提供必要的信息,对转换过程起指导作用;同时,通过测试目标生成测试用例,避免了基于测试过程中通过某种覆盖准则生成测试用例的盲目性,实现了生成的测试用例向相应的被测系统需求的向上追溯,使得测试人员更好的对整个测试过程进行评估.

测试需求模型的应用场景如图 1 所示.

首先通过测试需求元模型及其相关约束定义测试需求建模语言,这是整个测试需求模型的基础;然后基于测试需求建模语言的定义构造可视化的建模环境,用以对测试需求进行建模以及对测试需求模型语义正确性的自动化验证;同时,定义测试需求描述语言,可以实现由测试需求模型到测试需求描述脚本的自动转换;最后,由测试需求模型可以自动生成抽象测试用例.

针对软件的需求和软件的测试之间存在的关系,提出了一种软件测试需求的定义,并且给出了测

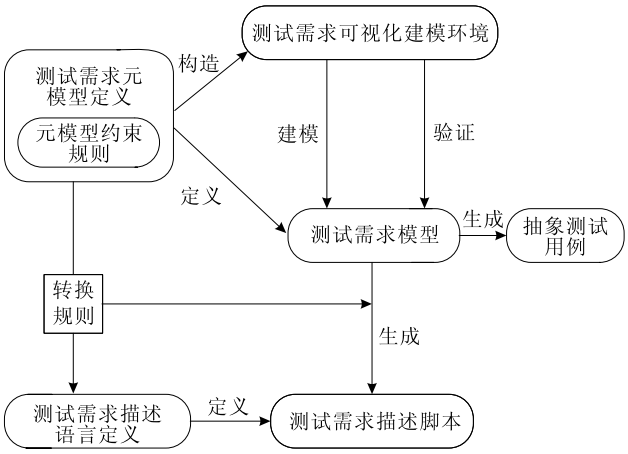

图 1 测试需求模型的应用场景图

试需求建模的方法,利用模型驱动的思想,对软件的测试进行设计.

第 2 节提出了测试需求的定义和测试需求模型,并且给予详细描述;第 3 节主要讲述如何从测试需求模型当中的描述模型生成测试用例;第 4 节给出使用本文方法进行的实验,并对实验结果进行分析;第 5 节给出相关工作的介绍;最后在第 6 节对本文工作进行总结和展望.

2 测试需求

2.1 测试需求定义

本文提出的测试需求的定义,参考了 IEEE 软件工程术语表[12]中对软件需求的定义.

测试需求的定义如下.

(1)为了达到用户的测试目标,测试系统需要满足的条件或能力.

(2)为了满足合同、测试标准、规范或其他强制性要求,测试系统需要具备的条件和能力.

(3)对于(1)或(2)中条件或能力的清晰的、无二义性的表述.

测试需求的好坏与整个测试过程紧密相关,对于一个良好的测试需求,我们希望它能够具备以下 6 个特征:

(1)完整性.测试需求当中的每一项内容都必须是描述清楚的,以便测试人员在测试时获得所需要的必要的信息.

(2)正确性.正确反应了测试任务和用户的要求.

(3)可追溯性.从测试需求可向上回溯到系统需求和设计,向下追踪到测试用例.

(4)清晰性.测试需求描述是清晰的、无二义性的,为了保证测试需求的清晰性,测试需求建模语言

必须具备尽可能严格的语法和语义定义.

(5)一致性. 测试需求中各部分内容的描述是一致的,不存在相互矛盾的地方,在建模元素之间的关联关系和约束关系要进行明确的定义.

(6)可行性. 每一项测试需求在已有的条件下都是可以测试、可以实施的.

2.2　测试需求模型

随着软件系统的复杂程度越来越高,如何有效地对软件系统进行测试成为了重点关注的问题. 模型驱动测试[8]是一种新颖而具有发展前景的自动化测试方法,它为测试人员提供了简便有效的方式,从而来实现高效自动化的测试. 这种新的方法及其工具集不仅能够提高测试人员的工作效率,并且能够减少产品投入市场的时间.

测试需求模型需要能够对测试需求进行可视化、无二义性的描述,因此该模型需要描述哪些方面是需要被测试的:模型需要具备严格的语法和语义,将测试需求和被测系统的功能以及性能需求对应;该模型还需要是能够保证测试是完整的,这样可以通过模型检查的技术,来保证模型各部分具有一致性;测试需求模型能够指导测试的设计与测试用例的生成,这样使得测试需求具有可行性;测试需求模型还能建立起被测系统需求到测试设计的可追溯的关系. 在接下来,将用测试需求的元模型对测试需求模型进行详细的描述.

2.2.1　测试需求元模型

随着 UML 与 MDA 的兴起和流行,模型已经成为软件开发的核心制品,而模型重要性的提升使得建模语言以及定义建模语言的元模型逐渐成为软件开发中的一个核心要素. 软件开发往往涉及多个领域,而不同的领域往往需要不同的建模语言及建模工具. 但是,手工地为不同的建模语言开发建模工具代价高昂. 元建模技术[13]是解决这个问题的方法之一,通过元建模,可以根据领域需要定制合适的元模型以定义领域建模语言,进而自动生成支持该建模语言的建模工具. 大量的工程实践表明,与领域建模以及 MDA 相结合,元建模可以大幅度地提高软件开发效率.

测试需求元模型作为测试需求建模的基础,定义了测试需求模型中的各种核心概念以及核心概念之间的关系与依赖. 本文提出的测试需求模型的元模型如图 2 所示.

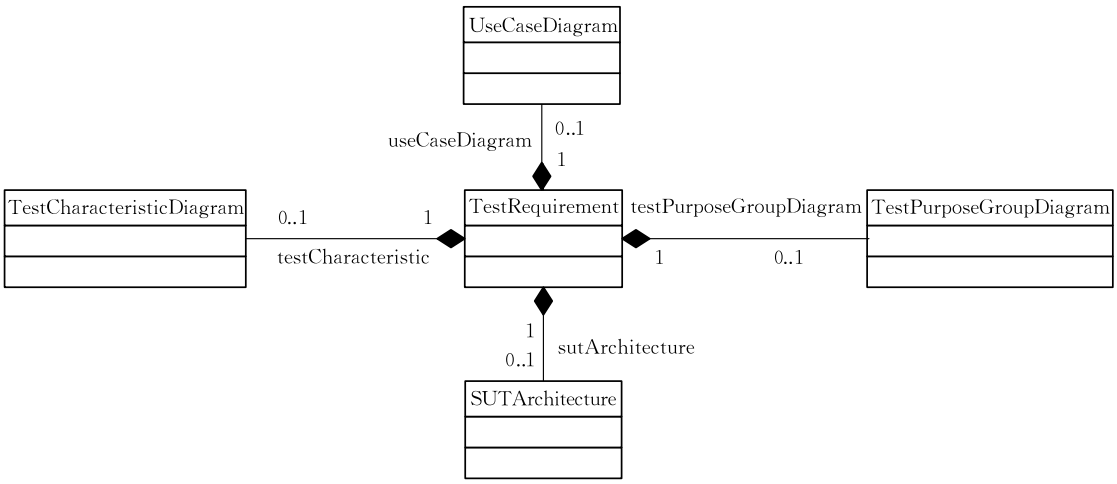

图 2　测试需求元模型

测试需求元模型主要分为 4 个部分:测试特征元模型(图 2 中的 TestCharacteristicDiagram)、基于被测系统用例的测试需求描述元模型(图 2 中的 UseCaseDiagram)、基于被测系统构件的测试需求描述元模型(图 2 中的 SUTArchitecture)以及测试目标分组及描述元模型(图 2 中 TestPurposeGroupDiagram). 它们分别从不同的方面对测试需求进行描述. 这 4 个部分相对独立但又相互关联,对于完整的测试需求模型而言缺一不可.

2.2.2　测试特征元模型

测试特征的概念参考了 OMG 对服务质量特征的定义. 测试特征①(图 3 中的 TestCharacteristic)是一组可定量表达的特性,独立于其所度量的具体元素. 测试特征可以是对测试过程中被测系统行为特征的描述,如响应时间等,也可以用于描述测试系统自身的特征,如测试系统需模拟并发用户的数目以及测试数据选择策略等. 测试需求建模语言的设计使得建模人员可以将测试特征针对领域进行扩

展,使得测试特征的核心内容不至于过于庞大,同时也可满足不同领域测试人员的需要.

测试特征的维度(图 3 中的 Dimension)是对测试特征描述进行量化表达的视角与方法.如当测试特征为某一系统函数的响应时间时,测试人员选择的度量方式可以是函数一次执行的延迟、所有执行的平均延迟或是延迟时间的方差.测试特征可能需要若干不同维度的量化值,如被测系统可靠性这一测试特征,需要修复时间、失效时间等度量方式.单一的量化值无法全面的对测试特征进行描述.因此测试特征维度的引入是十分必要的.

另外一个与测试特征密切相关的概念是测试约束.测试约束在测试目标分组与描述模型中用于对相应的测试目标进行约束.

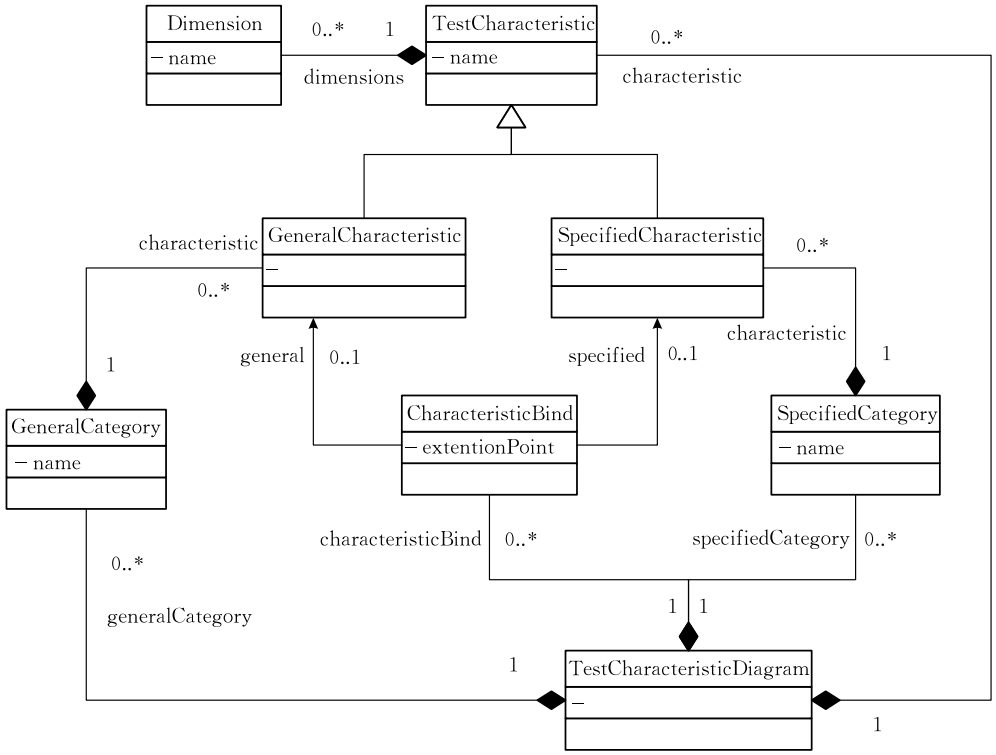

图 3 测试特征元模型

2.2.3 测试目标分组及描述元模型

测试目标的规格说明需要具有一定的逻辑结构[7].在本文中采用分组的方式对测试目标进行组织.如可以通过区分测试目标(图 4 中的 TestPurpose)为功能性测试还是非功能性测试、测试目标涉及哪个功能模块等方式将测试目标进行分组.对测试分组(图 4 中的 TestGroup)的划分可以有不同的粒度,每个分组可以有自己的子分组,直至具体到相应的测试目标.

对于每一个测试目标,都需要对其进行相应的描述.测试目标是从执行某一特定的测试场景或者路径,或是验证特定需求的特性角度对测试用例的一个精确目标的描述.测试目标比测试用例更加抽象,更加简洁,更加清晰并独立于被测系统的设计与实现.本文对测试目标描述的定义参考了 UML 顺序图,采用类顺序图的形式来实现对测试目标的描述.在本文中,测试目标实际上是测试人员所关注的测试系统与被测系统交互的消息序列,测试需求描述中的发送消息(图 4 中的 SendMessage)与接收消息(图 4 中的 ReceiveMessage)表示测试人员在本测试目标中所关注的消息序列,希望测试用例可以对其进行覆盖.而拒绝消息(图 4 中的 RefuseReceiveMessage)表示测试人员所不关心的消息,即不希望测试用例执行时测试系统会与被测系统就此消息进行交互.同时,在测试需求描述中可以引入测试约束模型元素用于对测试目标进行约束.

测试约束对与之相关联的测试特征(图 3 中的 TestCharacteristic)的取值进行了约束.测试约束有两种类型(图 4 中的 ConstraintConnectionType),其中 OFFERED 类型为测试系统自身需要满足的约束,测试系统需满足此约束条件才能正确完成测试工作,如测试系统模拟的并发用户数目必须达到某一值,或测试系统提供的数据必须约束在某一特定区间之内.而 REQUIRED 类型的测试约束为被测系统的行为需满足的约束,如被测系统响应时间需小于某一确定值.

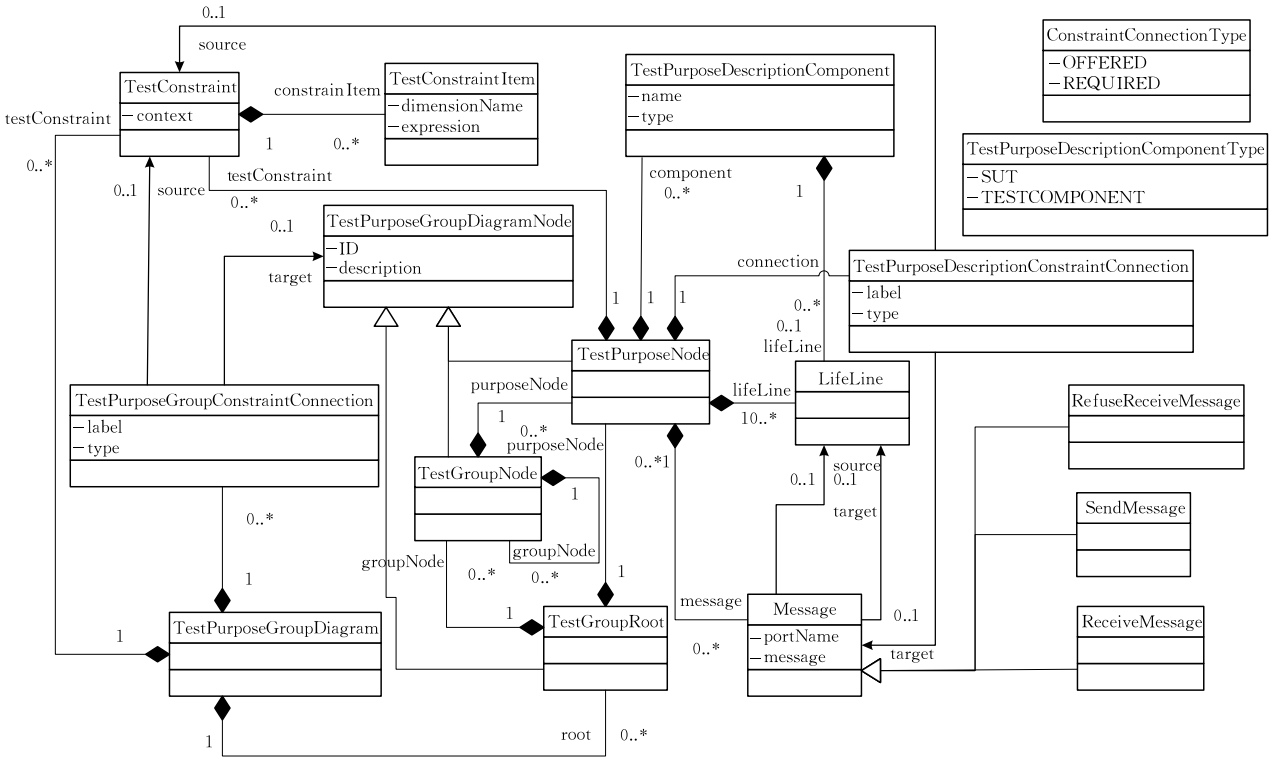

图 4　测试目标分组及描述元模型

2.2.4　基于被测系统构件的测试需求描述元模型

基于被测系统构件的测试需求描述模型通过对被测系统相应的构件图进行标注，来描述测试需求中的被测软件实体 EUT 与测试端口. 对测试人员关注的构件(图 5 中 Component)进行标注(图 5 中 TestObjectEUT)表示此构件为被测软件实体，在后续的测试过程中需要对其进行相应测试；定义测试系统的端口(图 5 中 TestObjectPort)用于标注相应的被测系统接口(图 5 中 Interface)，并定义允许的输入消息类型与输出消息类型.

2.2.5　基于被测系统用例的测试需求描述元模型

用例图是一种基于场景的可视化表示方法，用来描述和论证系统大粒度的行为模式及其连接方式，它为高级的体系结构设计提供了系统行为框架并从体系结构的角度给系统行为赋予了一定的特征，测试人员利用它可以从较高的抽象层次来理解

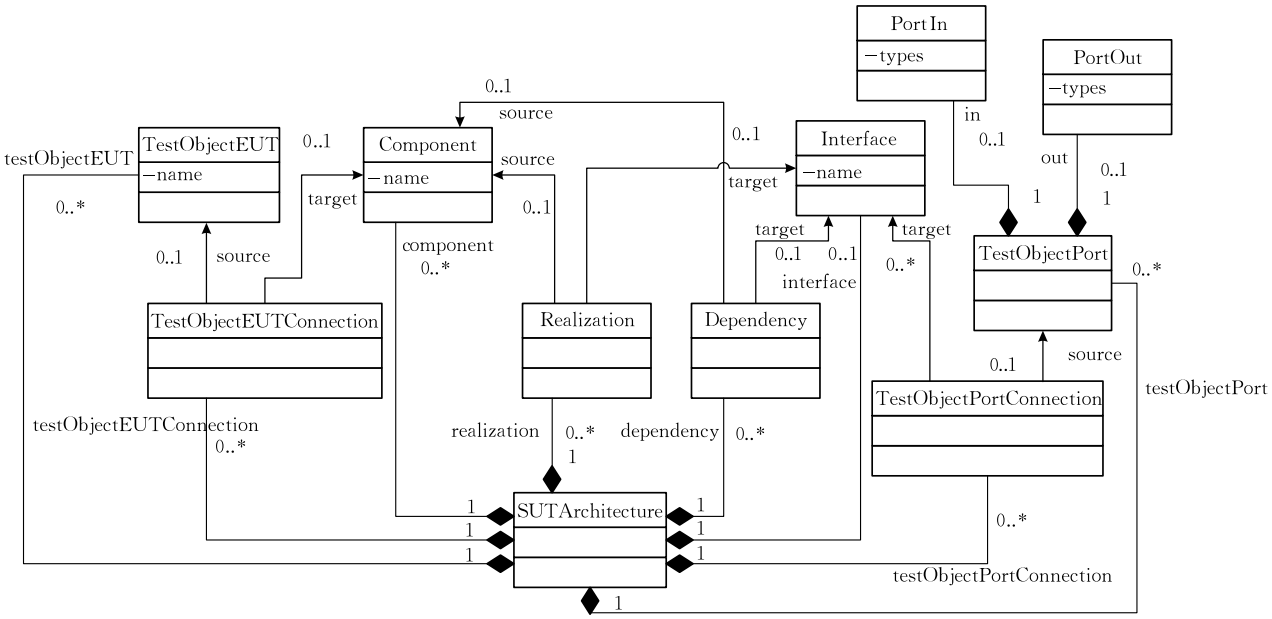

图 5　基于被测系统构件的测试需求描述元模型

系统的行为.从用户的观点出发对系统建立模型是用例要完成的任务,一组用例就是从用户的角度出发对如何使用系统的描述.因此用例为测试提供了良好的基础.

基于被测系统用例的测试需求描述模型以被测系统的用例图为基础,如图 6 所示.根据测试目标分组及描述模型中定义的测试目标对相应的用例(图 6 中的 UseCase)进行标注(图 6 中的 TestPurposeForUseCase),向测试人员指明对此用例的测试对应于哪一测试目标.通过测试目标生成相应的测试用例集之后,测试人员便可使用此测试用例集对用例进行相应测试.

2.3 元模型 OCL 约束

对测试需求元模型进行定义之后,还需要使用 OCL[14-15] 对测试需求元模型进行相应约束,以保证测试人员建立的测试需求模型在语义上的正确性.

对测试需求元模型的约束按模型元素所在的不同组成部分进行了分类,以基于用例的测试需求描述元模型与测试特征元模型的 OCL 约束为例.该约束如表 1 所示.

表 1 为基于用例的测试需求描述元模型与测试特征元模型的 OCL 约束,主要包括对用例的测试需求标注的相应约束(约束 1)以及对用例及参与者的相应约束(约束 2、3).

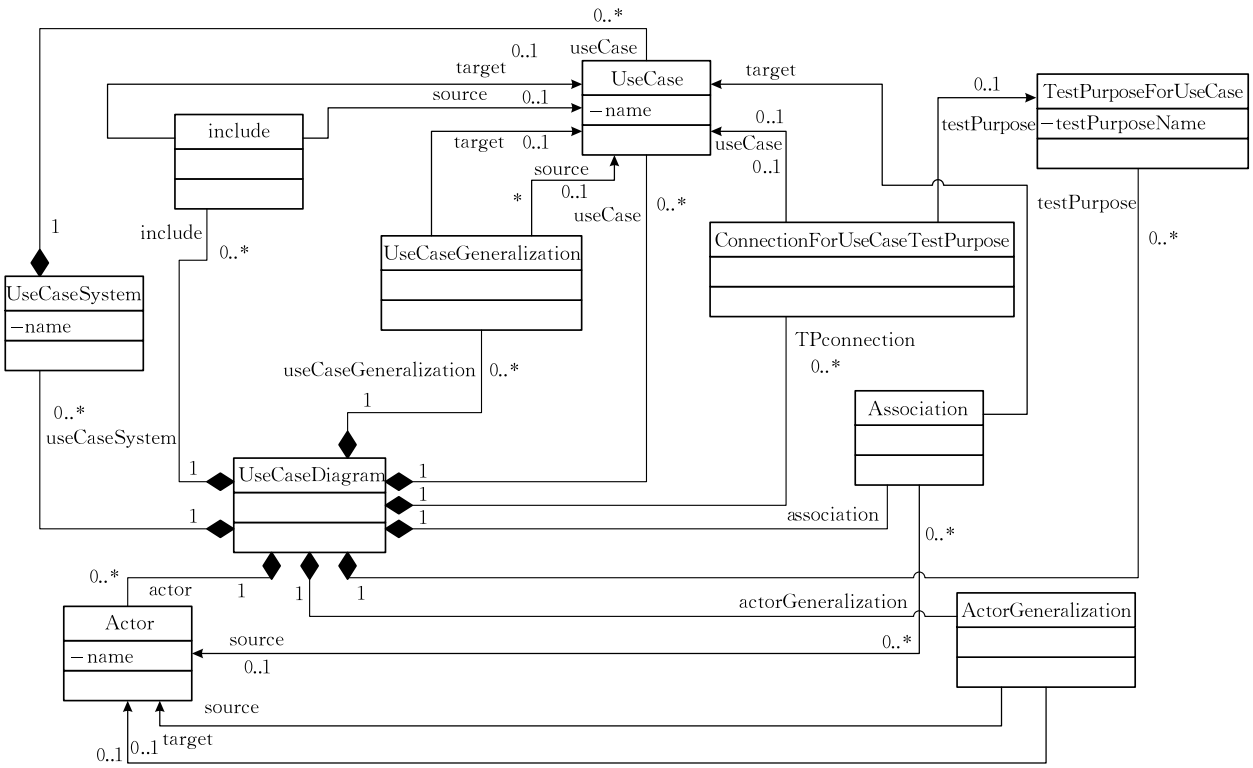

图 6 基于被测系统用例的测试需求描述元模型

表 1 基于用例的测试需求描述元模型的约束与测试特征元模型的约束

约束 1:对用例进行标注的测试目标必须在测试目标分组及描述模型中已定义	context TestPurposeGroupDiagram inv:TestPurposeNode.id ->includesAll(TestRequirement .UseCaseDiagram.TestPurposeForUseCase .testPurposeName)
约束 2:用例参与者需与相应用例相关联	context Actor inv: UseCase->notEmpty()
约束 3:用例不能重名	context UseCase inv:UseCase.allInstances()->forAll(uc1,uc2 | uc1<>uc2 implies uc1.name<>uc2.name)
约束 4:测试目标标注必须对相应用例进行标注	context TestPurposeForUseCase inv: UseCase->notEmpty()
约束 5:测试特征不能重名	context TestCharacteristics inv:TestCharacteristics.allInstances()->forAll(tc1,tc2 | tc1<>tc2 implies tc1.name<>tc2.name)

2.4 模型验证

虽然模型的语法符合要求,但可能会因为测试人员的疏忽或测试需求分析本身的错误,导致模型在语义上存在错误.例如用于标注某个用例的测试

目标未在测试目标分组与描述模型中定义，或是不同的测试目标拥有相同的 ID 等. 如果存在这些错误，测试人员很难人工的去对测试需求模型元素进行逐个检查，这样不仅时间代价过大，效果也极不理想. 因此，测试需求模型语义检查的自动化是十分必要的.

通过设置测试需求模型的语义约束规则. 当使用 OCL 进行约束时，以元模型中的元类为上下文(context)的每个不变式(invariant)在这个元类的实例的整个生命周期中必须为真，这些不变式是模型中模型元素必须满足的约束. OCL 语言给出了对测试需求元模型的无二义性的约束，其每一约束都对应于对测试需求模型的一条约束规则. 通过这样约束的规则，可以对模型进行验证.

2.5　测试需求描述脚本

测试需求描述脚本以简练的形式描述并保存了对测试需求的描述，测试需求建模方法可通过测试需求模型自动生成测试需求描述脚本. 在需要的情况下，测试人员可基于被测系统模型通过测试需求描述脚本还原测试需求模型.

为了规范生成测试需求描述脚本，定义了一种测试需求描述语言，用于测试人员编写测试需求描述脚本，作为对测试需求的文本描述形式. 相较于测试需求模型，测试需求描述脚本剥离了被测系统模型中与测试需求无关的部分，如被测系统各个不需要被测试的构件与接口的关系、各用例与参与者之间的关系等. 测试需求描述脚本以简练的形式描述并保存了对测试需求的描述，测试需求建模平台可通过测试需求模型自动生成测试需求描述脚本. 在需要的情况下，测试人员可基于被测系统模型通过测试需求描述脚本还原测试需求模型.

3　基于测试需求模型的测试用例生成

测试需求模型对测试需求进行了直观且无二义性的描述，作为整个测试活动的基础，能对后续测试过程进行有效的指导. 通过测试需求模型与被测系统模型生成测试用例，所生成的测试用例对应于测试需求模型中相应的测试目标，通过此测试目标可进一步追溯到作为测试目标设计依据的被测系统需求，从而实现了从系统需求到测试用例的向下追踪与从测试用例向系统需求的向上回溯.

3.1　输入输出的标记迁移系统

测试用例通过被测系统模型与测试需求模型中的测试目标描述模型生成. 首先需将被测系统的状态图模型(在经过需求分析阶段，经过多次的与用户沟通，可以借助相关的建模工具得到). 转换为标记迁移系统. 标记迁移系统是一种在计算机辅助设计和验证中得到广泛使用的形式模型，其结点代表系统的状态或配置，边代表在行为发生时状态或配置之间的转移.

本文对标记迁移系统进行适当的扩展，以区分输入行为与输出行为，称其为输入输出标记迁移系统. 它是一个四元组，定义如下：

$$M=(Q^M,A^M,\rightarrow_M,q_0^M)$$

其中，Q^M是状态的有穷非空集合，$q_0^M\subseteq Q^M$是初始状态. A^M是行为字母表，它被划分为两个子：$A^M=A_I^M\bigcup A_O^M$. 其中 A_I^M是输入字母表，A_O^M是输出字母表. $\rightarrow_M\subseteq Q^M\times A^M\times Q^M$为状态转移函数.

为方便描述，以字符?开头的行为表示输入行为，以字符!开头的行为表示输出行为，如?a 表示输入行为 a，即 $a\subseteq A_O^M$；!b 表示输出行为 b，即 $b\subseteq A_O^M$.

3.2　测试用例生成

为了得到测试用例，需要将被测系统的状态图模型转换为标记迁移系统

S(Specification)：$S=(Q^S,A^S,\rightarrow_S,q_0^S)$.

其行为字母表 $A^S=A_I^S\bigcup A_O^S$，A_I^S与 A_O^S分别为被测系统的输入与输出.

根据测试系统的状态图，利用标记迁移系统 S 生成算法，可以得到对应的标记迁移系统，在标记迁移系统 S 生成算法中，首先通过 ConstructFrom(SpecificationNode node)方法以被测系统状态图模型的开始状态为起点对状态图模型进行遍历，将其结构信息保存在标记迁移系统中，然后将标记迁移系统中同时拥有输入输出消息的转移进行分裂，并插入新建立的中间状态节点. 生成算法如下所示.

算法 1.　标记迁移系统 S 生成.

输入：被测系统状态图 *Statechart*

输出：元素类型为 *SpecificationNode* 的顺序容器 SList

BEGIN

得到 *Statechart* 的 *BeginState* 状态；

构造 *BeginState* 的相应 *SpecificationNode* 对象 *BeginSN* 并添加到容器 SList；

调用方法 *ConstructFrom*(*BeginSN*)；

FOR SList 中的每个元素的每个转移信息(*SpecificationNodeAndEdge* 对象)

IF 转移信息对应的 *Statechart* 中的 *Transition* 同时拥有输入与输出消息

THEN 将此转移信息分裂为两条转移，分别对应于输入消息与输出消息，设置

它们的 *message* 属性与 *messageType* 属性，并创建新的 *SpecificationNode*

作为这两个转移的中间节点；

ELSE

设置相应 *message* 属性与 *messageType* 属性；

END IF

END FOR

END

例如：图 7 显示了某被测系统的状态图与标记迁移系统 S.

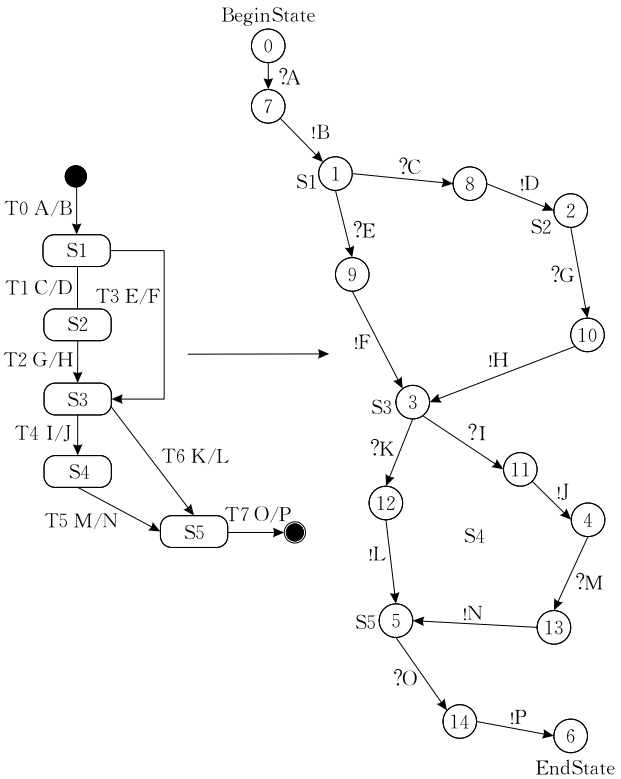

图 7 被测系统状态图与标记迁移系统 S

测试需求模型中的测试目标描述模型同样需要转化为相应的标记迁移系统 TP(Test Purpose)，以作为测试选择的指导规则，标记迁移系统 TP 的定义如下：

$$TP = (Q^{TP}, A^{TP}, \rightarrow_{TP}, q_0^{TP})$$

其拥有两种结束状态：

$Accept^{TP} \subseteq Q^{TP}$，$Refuse^{TP} \subseteq Q^{TP}$. 它的行为字母表与标记迁移系统 S 的行为字母表相同，即 $A^{TP} = A^S$. $Accept^{TP}$ 表示状态被测试目标所接受，如果达到此状态，则测试判定成功，测试通过；$Refuse^{TP}$ 表示状态在此测试目标下不被关注，测试判定未定义.

例如：图 8 左侧为图 7 所示的测试需求模型中的测试需求描述模型视图. 在测试需求描述中，由被测系统向测试系统反馈的消息若使用虚线图符，表示此测试目标拒绝此消息，即此测试目标不关注到达此消息的消息序列. 图 8 中右边的图形为左侧模型生成的测试目标 TP，需要强调的是，在测试目标中定义的消息序列仅仅限定了它们的先后次序，并不要求它们是连续的，在系统的实际行为中它们之间可存在其它消息或消息序列，由此也体现了测试目标的抽象性.

从图 8 中可以看出，当测试需求描述中的测试系统向被测系统发送输入消息，如果得到的输出是消息 H，测试目标则会拒绝此消息. 如果在一定的时限之内没有收到消息为 H 的输出，测试系统将会向被测系统发送输入 0，如果在之后收到被测系统的输出消息为 P，测试目标则会接受此消息.

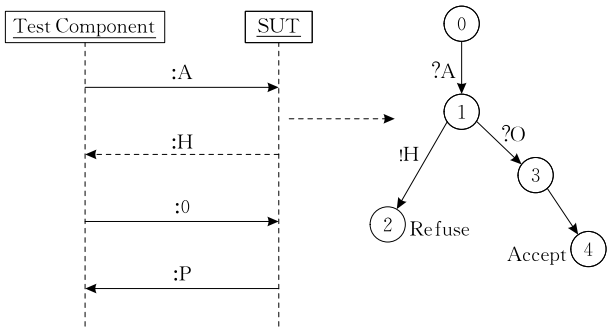

图 8 测试目标描述与标记迁移系统 TP

在分别生成标记迁移系统 S 与标记迁移系统 TP 后，需要将它们进行同步为新的标记迁移系统，以得到标记迁移系统 S 中被标记迁移系统 TP 所接受的行为以及被标记迁移系统 TP 所拒绝的行为. 标记迁移系统 S 与测试目标的 TP 的同步 $S \times TP$ 是一个输入输出标记迁移系统 $SP = (Q^{SP}, A^{SP}, \rightarrow_{SP}, q_0^{SP})$，SP 有两个不相交的状态集合：$Accept^{SP} \subseteq Q^{SP}$，$Refuse^{SP} \subseteq Q^{SP}$.

标记迁移系统 SP 标识出了标记迁移系统 S 中被标记迁移系统 TP 接受的行为(如果被测系统行为这些行为序列一致，则测试通过)与被标记迁移系统 TP 拒绝的行为(表示测试目标不关心被测系统的这些行为). 接下来需要通过选择标记迁移系统 SP 中被测试目标接受的行为来得到相应的测试用例. 一个测试目标可能对应很多个测试用例，首先通过标记迁移系统 SP 生成包含对应测试目标所有测试用例信息的标记迁移系统 ATC(All Test Cases)，$ATC = (Q^{ATC}, A^{ATC}, \rightarrow_{ATC}, q_0^{ATC})$，ATC 拥有两个结束状态：$Pass \subseteq Q^{ATC}$，$Fail \subseteq Q^{ATC}$，它们分别代表对测试的不同判定.

由图 7 的标记迁移系统 S 与图 8 的标记迁移系统 TP，可以得到标记迁移系统 SP 和 ATC，如图 9

所示.

图 9 左侧的标记迁移系统 SP 由图 7 中的标记迁移系统 S 与图 8 中的标记迁移系统 TP 生成. 下面给出标记迁移系统 SP 的生成算法.

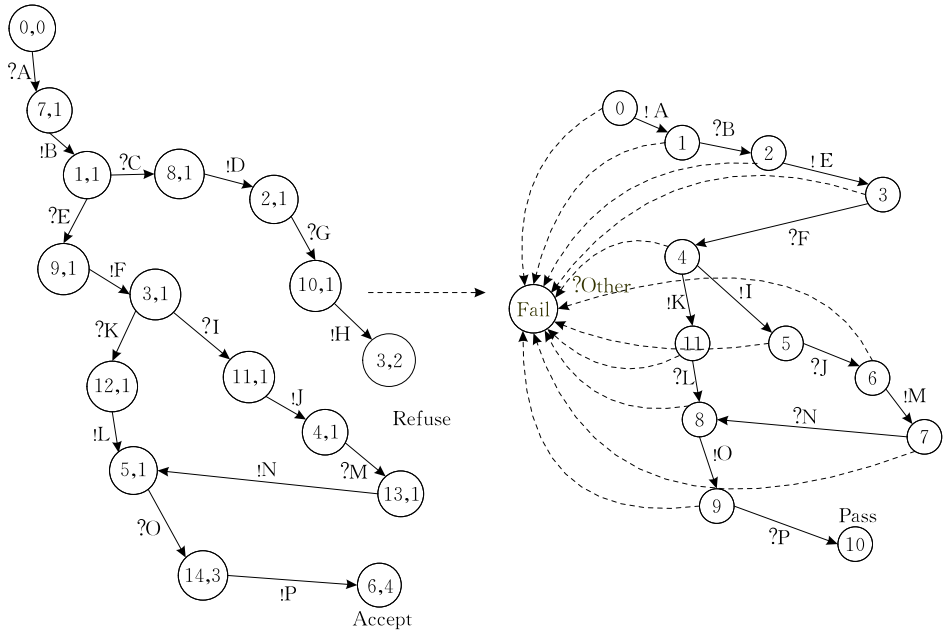

图 9　标记迁移系统 SP 与 ATC

算法 2.　标记迁移系统 SP 生成.

输入：标记迁移系统 S，标记迁移系统 TP

输出：元素类型为 *SPNode* 的顺序容器 *SPList*

BEGIN

由标记迁移系统 S 与标记迁移系统 TP 的初始节点创建 *SPNode* 类型的对象 init；

为 *SPList* 添加对象 init，置 init 的相应 *tag* 值为 0，表示未被处理；

FOR *SPList* 中每个 *tag* 为 0 的元素 *element*

通过 *element* 的 *S_id* 得到标记迁移系统 S 中相应状态节点 Snode；

通过 *element* 的 *TP_id* 得到标记迁移系统 TP 中相应状态节点 TPnode；

FOR *Snode* 的每个后继转移 *Sne*

创建 *SPNode* 类型的对象 child；

IF *TPnode* 有后继转移 *TPne*，且其消息与 *Sne* 上的消息相同

THEN 由 *TPne* 对应的后继状态的 *id* 与 *Sne* 对应的后继状态的 *id* 初始化

SPNode 类型的对象 child；

ELSE

由 *TPne* 对应的后继状态的 *id* 与 *element* 的 *TP_id* 初始化 *SPNode* 类型对象 *child*，即新对象的 *TP_id* 值不变；

ENDIF

在 *element* 的 children 容器中中加入 SPNode 对象 child，child 的 parent 容器中加入 SPNode 对象 element；

置 child 的 verdict 值与 TP 中相应状态节点 verdict 值相同；

IF SPList 中没有 child

THEN 将 child 加入 SPList；

置 child 相应 *tag* 为 1；

ENDIF

ENDFOR

ENDFOR

从 SPList 中的 Accept 状态节点开始，对标记迁移系统进行深度优先搜索（搜索前继状态），置可达的状态节点的 is_RA 值为 true，其它为 false；

END

最后，通过 ATC 生成相应的测试用例集. 首先生成标记迁移系统形式的测试用例 TC：$TC=(Q^{TC}, A^{TC}, \rightarrow_{TC}, q_0^{TC})$. 其拥有两种结束状态：$Pass \subseteq Q^{TC}$，$Fail \subseteq Q^{TC}$，它们分别代表对测试结果的不同判定. 行为字母表 $A^{TC}=A_I^{TC} \cup A_O^{TC}$，其中 $A_O^{TC} \subseteq A_I^S$，即测试用例的输出必须为被测系统的允许输入；$A_I^{TC} \subseteq A_O^S$，即测试用例的输入只能为被测系统的输出.

图 10 展示了通过图 9 中的 ATC 所生成的测试用例集. 以初始节点为起点，搜索可到达 Pass 状态节点的所有路径，其中每一路径与路径中状态节点到节点的转移为一个测试用例，由此得到此测试目标对应的测试用例集.

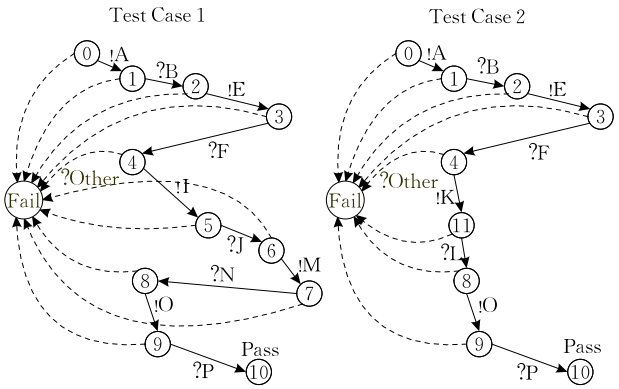

图 10 测试用例标记迁移系统

4 实验分析

4.1 实验方案设计

目前人们使用的网络主要有两种，一种是语音网络，一种是数据网络. 语音网(例如 PSTN)使用电路交换，质量和流量控制等被嵌入网络中. PSTN 网可以保证语音质量，但不能有效地使用网络资源，同时也很难扩展新的业务. 数据网(例如 Internet)使用分组交换，质量和流量控制等则在终端系统中实现，因而更加灵活，更加容易扩展，也更容易实现新的业务. 近年来由于数据网的快速发展，出现了 IP 电话. IP 电话有两套独立的协议，H. 323 以及 SIP. H. 323 协议是由 ITU 提出的，同时也是实现最多的协议，被众多产品支持. SIP 协议是由 IETF 提出的，提出时间相对晚一些，然而由于它具有很好的可扩展性，互操作性，协议本身也更简单一些，SIP 正变得更加流行.

实验选取了 SIP 协议建立会话并进行释放的一个简单流程，这个流程如下.

1. 用户代理 A 通过代理服务器向用户代理 B 发送包含有会话请求的 INVITE 消息；

2. 用户代理 B 根据 INVITE 消息中携带的会话描述符信息，通过代理服务器向用户代理 A 回送一个“180 振铃”作为应答；

3. 若用户代理 B 的用户接受呼叫，则用户代理 B 返回一个“200 OK”应答；

4. 用户代理 A 收到“200 OK”应答后发出一个 ACK 请求，完成一个三次握手过程，会话开始；

5. 当用户代理 A 的用户希望结束会话时，通过用户代理 A 直接向用户代理 B 发送一个请求；

6. 用户代理 B 收到 BYE 请求后结束会话，并为这个 BYE 请求返回一个“200 OK”应答.

通过 SIP 的这个工作流程，进行如下实验：

1. 建立被测系统的测试需求模型；

2. 在测试需求模型中人为插入语义故障，模型验证对模型进行自动语义验证并报告相应错误；

3. 通过测试需求模型生成相应测试需求描述脚本；

4. 生成相应测试目标描述对应的测试用例.

4.2 实验结果以及分析

本文基于 Eclipse3. 5、GMF2. 2 版本开发了测试需求模型可视化系统. 型编辑器包括菜单视图、导航视图、属性视图、编辑器视图和工具栏. 基于 GMF 开发的模型编辑器的编辑器视图中的图形显示对应两个 xml 格式文件，提供了对模型元素语义和模型元素图形化表示的存储功能.

1. 测试需求建模

实验根据 SIP 协议标准[16]建立相应的测试需求模型. 测试特征分为通用测试特征(general characteristics)与测试人员为对 SIP 测试所特化的测试特征(SIP specified characteristics). 通用测试特征 Sequence 的维度 consecutive 用于表示在测试目标中定义的消息序列是否需要在其相应的测试用例中连续出现；通用测试特征 Load 的维度 concurrency 用于表示测试中的并发数；通用测试特征 ResponseTime的维度 duration 表示相应的时间间隔，而测试特征 SIP_ResponseTime 特化了 ResponseTime，表示测试中相应的时间间隔(维度 duration)以毫秒(ms)为单位.

根据上面的流程，可以得到测试特征模型视图. 如图 11 所示.

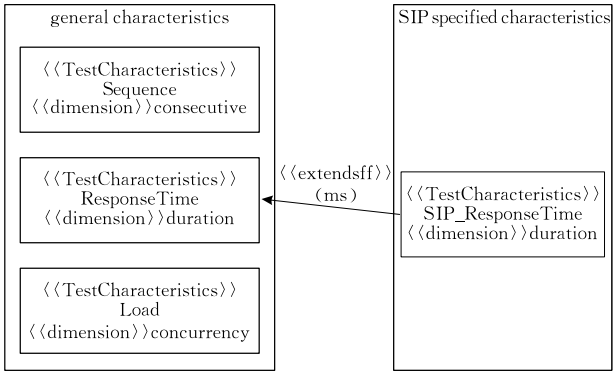

图 11 测试特征模型视图

利用测试需求模型可视化系统，建立测试目标分组及描述模型视图. 此视图以树形结构展示测试目标分组. 与根节点相关联的矩形内为对相应测试分组的描述. 对于测试分组节点与测试目标节点可进行相应的测试约束标注，如图中的测试约束上下文为 Load，表示此测试约束对应的测试特征的名字；测试约束项限定了对应测试特征维度的取值，要

求测试系统的并发数在 190 与 200 之间，由于是对测试系统的约束，因此约束类型为 OFFERED. 测试目标分组树以测试目标节点为叶节点. 测试目标节点 TP-2 所显示的测试目标描述模型如图 12 所示.

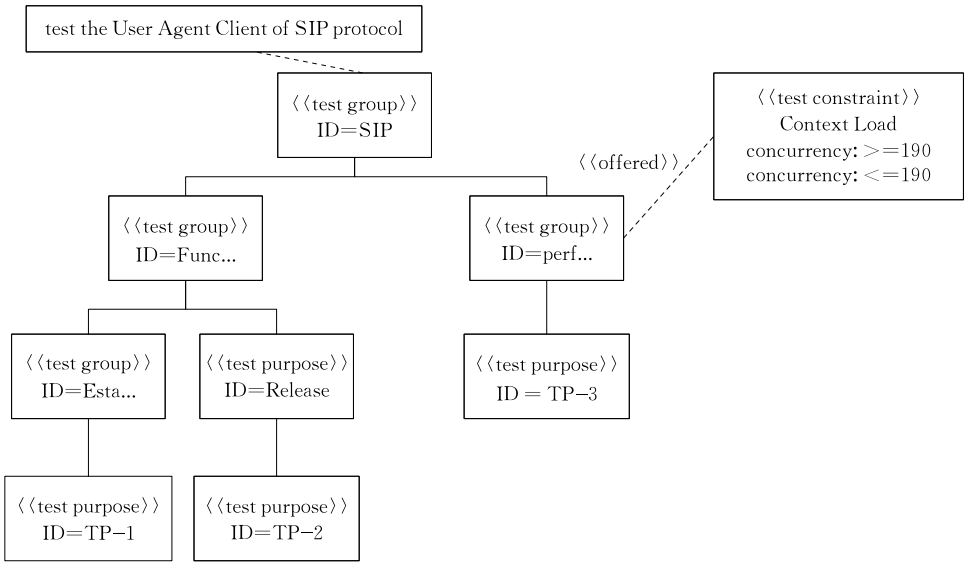

图 12　测试目标分组及描述模型视图

图 13 为测试目标节点 TP-2 对应的测试目标描述模型视图，用于描述在此测试目标中所关注的测试消息序列，作为指导测试人员设计测试用例或是自动生成测试用例的基础. 此测试目标主要关注 SIP 协议中通话结束后的释放过程：期望测试系统模拟远端用户代理通过 SIP 端口向被测系统（本地用户代理）发送消息 BYE，若被测系统通过 SIP 端口返回消息 200_for_BYE 则测试通过，在这两条消息中可以存在其他的消息序列，并不要求它们是连续的.

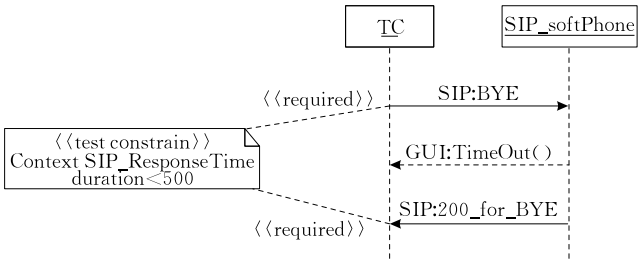

图 13　TP-2 的测试目标描述模型视图

在图 13 中对发送消息 BYE 与返回消息 200_for_BYE 进行了相应的测试约束标注，测试约束对应于测试特征模型中的测试特征 SIP_ResponseTime，约束项对测试特征的维度 duration 的取值进行了约束，表示这两条消息之间的时间间隔需小于 500 ms，这是对被测系统响应时间的要求，因此测试约束类型为 REQUIRED.

基于用例的测试需求描述模型如图 14 所示. 参与者分别为客户代理（User Agent）与代理服务器（SIP Proxy Server）. 用例图中只对 3 个用例进行了建模，其中用例 CallAsCaller 表示本地用户代理通过代理服务器向远端用户代理发起对话，与其关联的矩形为测试需求标注，表示此用例对应的测试目标为 TP-1；用例 RemoteHangUp 表示远端用户代理主动结束对话，对应测试目标为 TP-2；用例 UserHangUp 表示发起对话的本地用户代理结束对话，对应测试目标为 TP-3.

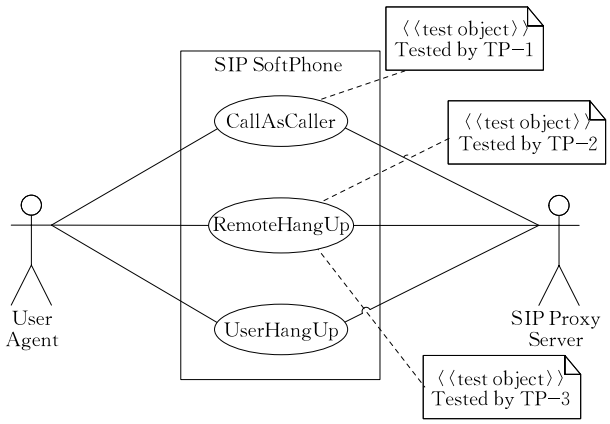

图 14　基于用例的测试需求描述模型视图

对 SIP 协议建立的测试需求模型中基于被测系统构件的测试需求描述模型视图如图 15 所示. 构造型为〈〈Component〉〉的矩形为被测系统构件，构造型为〈〈interface〉〉的矩形为被测系统接口. 对被测系统构件的标注表示此构件为被测软件实体 EUT，下图表示仅需对 Main_Control 构件（对应于客户代理）与 SIP_Protocol 构件（对应于 SIP 协议的具体

实现)进行测试,而并不关注 SIP 协议所处理的具体多媒体会话及相应的实时传输协议(RTP),因此 Media_Manager 构件与 RTP_Protocol 构件未被标注.对被测系统接口的标注表示测试系统通过相应的端口与此接口通信,并定义了端口允许通过的消息类型.下图中 GUI 接口允许的输入消息类型为用户操作 UserOp 类型,允许的输出类型为显示消息 PromptMsg 类型;SIP 接口允许的输入、输出消息类型均为 SIP 消息类型 SipMsg.

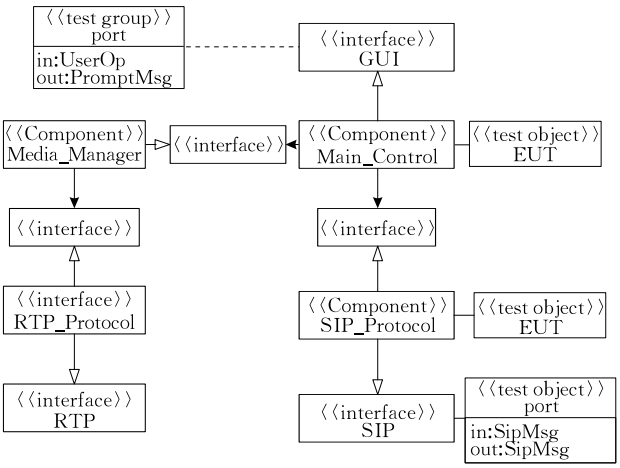

图 15 基于被测系统构件的测试需求描述模型视图

2. 在测试需求模型中人为插入语义故障,模型验证对模型进行自动语义验证并报告相应错误.

测试需求模型中人为插入若干有代表性的语义故障以检测测试验证能否自动识别.为此,对所建立的测试需求模型元素进行了如下更改以插入故障:

(1)删除测试目标分组及描述模型中的 TP-3 测试目标节点;使测试需求模型不满足测试目标分组与描述元模型的约束集合中的约束 1 以及表 1 中的约束 1;

(2)将基于被测系统构件的测试需求描述模型中的 TP-2 测试目标节点的测试目标描述模型中 SIP_ResponseTime 约束的 duration 维度的 Dimension Name 属性更改为"delaying";使测试需求模型不满足描述元模型的约束集合中约束 8;

(3)将测试特征模型中测试特征 Load 的 name 属性更改为"Sequence",与测试特征 Sequence 的 name 属性相同;使测试需求模型不满足描述元模型的约束集合中的约束 7 与表 1 中的约束 5;

(4)删除基于被测系统构件的测试需求描述模型中的 Main_Control 组件的测试需求标注;使测试需求模型不满足基于用例的测试需求描述元模型的约束与测试特征元模型的约束集合中的约束 6;

(5)删除基于被测系统构件的测试需求描述模型中的 GUI 接口的测试需求标注中的 PortOut;使测试需求模型不满足基于用例的测试需求描述元模型的约束与测试特征元模型的约束集合中的约束 1;

(6)更改基于被测系统构件的测试需求描述模型中的 SIP 接口 name 属性为"NetPort";使测试需求模型不满足测试目标分组与描述元模型的约束集合中的约束 6;

插入上述典型的语义故障之后,在测试需求模型可视化系统对测试需求模型进行自动验证,显示的错误信息图 16 所示的.其中 error1 由上面的更改 3 引起;error2 由更改 5 引起;error3 与 error8 由更改 4 引起;error4 与 error5 由更改 6 引起;error6 由更改 2 引起;error7 与 error9 由更改 1 引起.

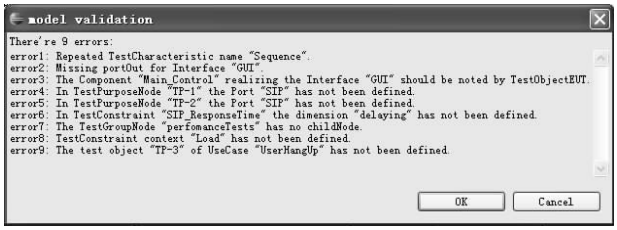

图 16 模型语义错误信息

3. 通过测试需求模型生成相应测试需求描述脚本.

4. 生成相应测试目标描述对应的测试用例.

选取建立的测试需求模型内的测试目标 TP-2 来生成其相应的测试用例集.图 17 为通过测试需求模型可视化系统建立的 SIP 状态图模型,表示客户代理的状态转移过程,作为生成测试用例所需的被测系统模型.图中除了会话建立与释放的过程,还包括取消(CANCEL)会话的过程,如状态图中的转移 T7、T12、T13 以及 T14.

被测系统状态图模型中转移的描述对应着相应转移的触发消息以及在转移过程中被测系统的反馈消息.如图 17 所示,SIP 从"Init"状态到"Calling"状态的转移所对应的描述为"T1 GUI:invite/SIP:INVITE",其中"T1"表示此转移对应的序号为 1;"GUI:invite"为此转移的触发消息,冒号前的 GUI 表示被测系统接收消息的接口,冒号之后的 invite 表示触发消息的具体内容;斜杠"/"之后的内容为被测系统在此转移过程中的反馈消息,冒号前的 SIP 表示被测系统反馈消息的接口,冒号后的 INVITE 为客户代理发送消息的具体内容.

当本地客户代理向目标地址发送 INVITE 请求之后,由 Init 状态进入 Calling 状态,这时若对端

直接返回 200_for_INVITE(SIP 应答,状态码 200,表示成功),则客户代理进入 Ready 状态会话开始;若对端返回 183_for_INVITE(SIP 应答,状态码 183,表示振铃),则客户代理进入 Ringing 状态并等待对端的同意消息,直至收到 200_for_INVITE,则进入 Ready 状态开始会话. 如果希望会话结束,会出现两种情况:一种是本地客户代理请求结束会话,另一种是对端客户代理请求结束会话. 而测试目标 TP-2 所关注的是对端客户代理请求结束对话的情况.

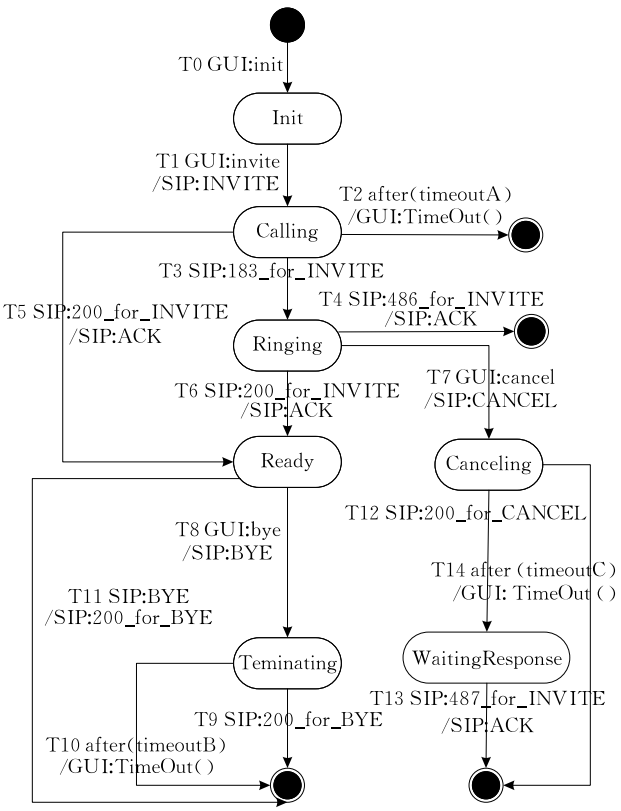

图 17　SIP 状态图

在测试需求模型可视化系统上 TP-2 对应的测试目标描述模型界面,并选择图 17 中的状态图作为被测系统模型,进行此测试目标对应的测试用例集的自动生成. 通过测试目标 TP-2 生成的测试用例集可以在测试需求模型可视化系统上直接展示,如图 18 所示.

测试目标 TP-2 关注远端客户代理请求释放对话的过程. 此测试目标生成了两个测试用例(见附录 3):TestCase1 对应于图 16 中 SIP 状态图的状态转移序列 BeginState $\xrightarrow{T0}$ Init $\xrightarrow{T1}$ Calling $\xrightarrow{T5}$ Ready $\xrightarrow{T11}$ EndState;TestCase2 对应于 SIP 状态图的状态转移序列 BeginState $\xrightarrow{T0}$ Init $\xrightarrow{T1}$ Calling $\xrightarrow{T3}$ Ringing $\xrightarrow{T6}$ Ready $\xrightarrow{T11}$ EndState. 这两个测试用例

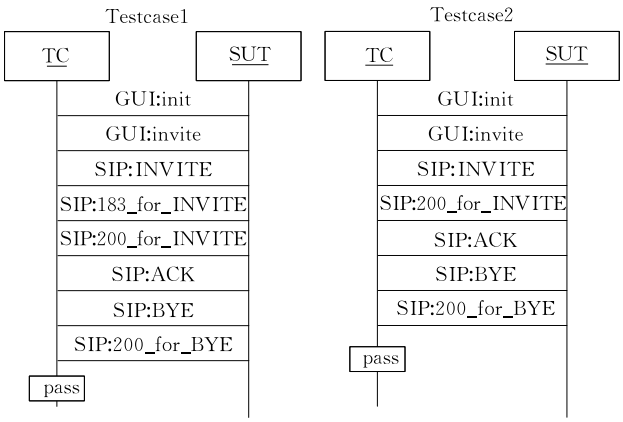

图 18　可视化测试用例

均用于测试 TP-2 所关注的会话释放过程. 而测试目标 TP-2 的设计基于被测系统的需求规格说明中的相应需求,从而实现了从系统需求到测试用例的向下追踪与从测试用例向系统需求的向上回溯,当测试活动中某一测试用例判定不通过时,可明确的向测试人员指出系统实现与规格说明中不一致的具体需求.

生成测试用例过程的中间形式,包括测试目标描述模型生成的标记迁移系统 TP、被测系统模型生成的标记迁移系统 S 以及标记迁移系统 SP、ATC 以及存储测试用例信息的标记迁移系统 TC,它们的具体信息在生成之后被打印出来,通过这些信息可以完全还原相应的标记迁移系统.

5　相关工作讨论

5.1　模型驱动测试

模型驱动架构(MDA)是 OMG 提出的方法学,MDA 强调整个系统开发过程由对软件系统的建模行为驱动,完成系统需求分析、架构设计、构建、测试、部署运行和维护工作. 模型驱动架构不仅可以应用到平台相关系统、平台独立系统和系统代码生成这些抽象层次的系统建模,而且也可以用到测试建模.

MDA 基于 UML 以及其它工业标准,支持软件设计和模型的可视化、存储和交换[17,21]. 和传统 UML 模型相比,MDA 能够创建出机器可读和高度抽象的模型,这种模型通过转换(Transformation)技术可自动转换为代码、脚本、数据库定义以及各种平台部署描述,使建模语言不仅仅用作分析设计语言,更可用作为一种高级编程语言.

模型驱动测试是一种新颖而具有发展前景的自动化测试方法,它为测试人员提供了简便有效的方

式,从而来实现高效自动化的测试.这种新的方法及其工具集不仅能够提高测试人员的工作效率,并且能够减少产品投入市场的时间[27-28].

Dai 等人[18-19]提出了一种模型驱动的测试方法.该方法以 U2TP① 测试建模语言为基础,提出了一个模型驱动的测试过程(如图 19 所示).其中,包括多种模型之间的转化,涉及平台无关系统设计模型(PIM)、平台相关系统设计模型(PSM)、平台无关测试设计模型(PIT)、平台相关测试设计模型(PST)、系统代码、测试代码等.

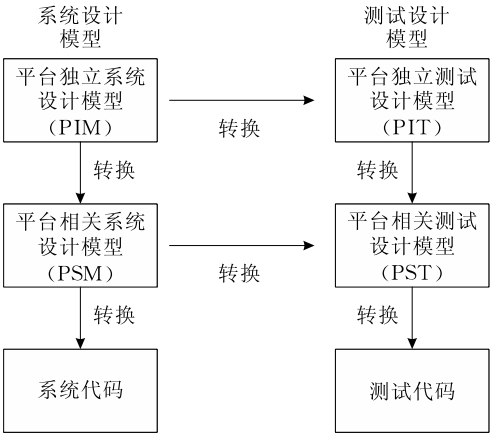

图 19 模型驱动开发与模型驱动测试

Baker 等人[20]在传统的 V 模型[22]的基础上提出模型驱动测试模型.在该模型中,单元测试、集成测试、系统测试等各个测试阶段都有一个对应的建模活动,所建立的模型可转换为 TTCN-3 或 JUnit 代码,进而得以自动化执行.

在平台独立测试设计模型向测试代码转换的过程的研究中[23],主要是在参考 U2TP 标准定义测试系统元模型以用于对测试系统进行建模[16,24-25],Bi 等人[25]并基于 eclipse 上的 GMF(Graphical Modeling Framework,图形建模框架)框架构造了可视化的测试系统建模工具,然后以测试系统元模型与 TTCN-3 元模型的映射关系为基础,实现了测试系统模型向 TTCN-3 代码的转换.

本文的工作是在 Bi 等人[25]的基础上来进行的,和之前的工作一样,本文的工作也参考了 U2TP 规范定义.

5.2 基于模型的测试

基于模型的软件测试属于基于规范的软件测试范畴,其特点是:在产生测试例和进行测试结果评价时,都是根据被测系统的模型及其派生模型进行的.基于模型的测试最初应用于硬件测试,广泛应用于电信交换系统测试,目前在软件测试中得到了一定应用,并在学术界和工业界得到了越来越多的重视.基于模型的测试实践为模型驱动的软件测试方法奠定了基础.

基于模型的软件测试可以根据软件行为模型和结构模型生成测试用例[26-27],当前软件规模庞大也使基于程序的测试十分困难,而基于模型的软件测试方法不仅可以有效地提高测试效率,提高测试例生成的自动化程度,也有利于评价测试结果.

在基于模型的测试中,采用某种覆盖准则来指导测试用例的生成,并决定何时来终止测试.覆盖准则是指测试需要满足或覆盖软件的哪些具体元素.

对于基于模型的测试目前从设计模型到测试模型自动转换的研究成果不多,更多的研究集中在通过设计模型直接生成测试用例或测试代码.其优势在于快速生成测试代码;不足之处在于缺少对测试的描述,很难复用、演化以及根据用户需求定制测试代码.

5.3 基于测试系统模型的 TTCN-3 代码生成

本文的工作是在 Bi 等人[25]的基础上来进行的,和之前的工作一样,本文的工作也参考了 U2TP 规范定义.Bi 等人在参考 U2TP 标准定义测试系统元模型,提出了测试系统元模型,以作为对测试系统进行建模的基础.然后基于测试系统元模型采用 eclipse 平台上的 GMF 框架构建了可视化的测试系统模型编辑器.为实现测试系统模型向 TTCN-3 代码的转换,并且参考了 TTCN-3 元模型[28],定义出测试系统元模型向 TTCN-3 元模型的映射关系.

Bi 等人的研究主要是关注测试模型向测试代码的转换.我们提出的是基于被测系统模型来指导生成测试需求模型,将测试需求和被测系统的功能以及性能需求对应,这样以指导测试人员开展后续测试活动、方便测试人员与设计人员之间的沟通,还可在模型驱动测试的过程中对测试系统模型的生成提供必要信息,具有重要指导意义.

6 总结与展望

本文参考模型驱动测试的思想,定义了一种用于描述测试需求的可视化建模语言,用以对测试需求进行直观且无二义性的描述,作为测试活动的基础,对指导后续测试活动的开展具有重要意义.同时

① www.omg.org/technology/documents/modeling_spec_catalog.htm

本文开发了相应的测试需求建模可视化系统，使得测试人员可对测试需求进行建模、分析与验证；并通过被测系统模型与测试需求模型中相应测试目标生成测试用例，展现了测试需求模型以及测试需求建模方法在模型驱动测试方法学中的意义.

本文对测试需求的概念以及测试需求模型描述的内容进行了分析，并为系统设计模型向测试系统模型的转换提供信息.本文考虑到通用性，对于各个不同特定领域的测试需求描述可能不够完整，仍有自己的局限性.

另外，本文对测试需求的研究工作还没有在工业化的软件系统的开发过程中得到应用与实践，因此未来也需要在这个方向进行努力.

致　谢　北京航空航天大学软件工程研究所模型驱动测试小组成员之前所做了大量工作，特别是已经毕业了的陈辛夷等同学付出了许多劳动，在此表示感谢.最后要感谢百忙之中评阅论文的各位专家！

参　考　文　献

[1] Dorothy Graham. REQUIREMENTS: Requirements and testing: Seven missing-link myths. IEEE Software, 2002, 19(5): 15-17

[2] Liu Chao. Program interactive execution flow chart and its coverage test criteria. Journal of Software, 1998, 9(6): 458-463(in Chinese)
(刘超. 程序交互执行流程图及其测试覆盖准则. 软件学报, 1998, 9(6): 458-463)

[3] Jin Zhi. Ontology-based on system automatic acquisition. Chinese Journal of Computers, 2000, 23(5): 486-493(in Chinese)
(金芝. 基于本体的需求自动获取. 计算机学报, 2000, 23(5): 486-493)

[4] Lu Ru-Qian, Jin Zhi, Chen Gang. Ontology-oriented requirements analysis. Journal of Software, 2000, 11(8): 1009-1017(in Chinese)
(陆汝钤，金芝，陈刚. 面向本体的需求分析. 软件学报, 2000, 11(8): 1009-1017)

[5] Qiu Shuo, Wu Ji, Jin Mao-Zhong. TTCN-3 performance testing based on test suite reuse. Computer Science, 2008, 35(11): 117-122, 151(in Chinese)
(邱硕，吴际，金茂忠. 基于测试集复用的 TTCN-3 性能测试. 计算机科学, 2008, 35(11,专刊): 117-122, 151)

[6] ETS 300 406. Protocol and profile conformance testing specifications: Standardization methodology. Sophia-Antipolis, France: European Telecommunications Standards Institute (ETSI), 1995-04

[7] Stephan Schulz, Anthony Wiles. TPLan—a notation for expressing test purposes//Proceedings of the Testing of Software and Communicating Systems. Tallin, Estonia, 2007: 292-304

[8] Jard C, Jeron T. TGV: Theory, principles and algorithms: A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems. International Journal on Software Tools for Technology Transfer (STTT), 2005, 7(4): 297-315

[9] Tretmans J, Brinksma E. TorX: Automated model based testing. In: Hartman A, Dussa-Zieger K, eds//Proceedings of the 1st European Conference on Model-Driven Software Engineering. Nurnburg, Germany, 2003: 13-25

[10] Myers G J. The Art of Software Testing. New York, USA: John Wiley & Sons, 1979

[11] Zhen Ru-Dai. Model-driven testing with UML 2.0// Proceedings of the 2nd European Workshop on Model Driven Architecture (MDA). Canterbury, UK, 2004: 179-187

[12] IEEE Standard 610.12-2002. IEEE Standard Glossary of Software Engineering Terminology. New York, USA: IEEE Press, 2002

[13] Budinsky, Frank, eds. Eclipse modeling framework: A developer's guide. Boston, USA: Addison-Wesley Professional, 2004

[14] Warmer J, Kleppe A. Object constraint language, the getting your models ready for MDA. Boston: Addison-Wesley Longman Publishing Co., 2003

[15] Vaziri M, Jackson D. Some shortcomings of OCL, the object constraint language of UML//Proceedings of the 34th International Conference on Technology of Object-Oriented Languages and Systems. Santa Barbara, USA, 2000: 555-562

[16] Rosenberg J, Schulzrinne H, Camarillo G. SIP: Session initiation protocol, Reston, USA: Internet Engineering Task Force, RFC 3261, 2002

[17] France R B, Ghosh S, Dinh-Trong T, et al. Model-driven development using UML 2.0: Promises and pitfalls. IEEE Computer, 2006, 39(2): 59-66.

[18] Zander J, Dai Z R, Schieferdecker I, et al. From U2TP models to executable tests with TTCN-3-an approach to model driven testing//Proceedings of the Testing of Communicating Systems. Montreal, Canada, 2005: 289-303

[19] Dai Z R, Grabowski J, Neukirchen H, et al. From design to test with UML//Proceedings of the Testing of Communicating Systems. Oxford, UK, 2004: 33-49

[20] Baker P. Model-driven testing: Using the UML testing profile. Berlin, Germany: Springer-Verlag, 2009

[21] Pastor O, España S, Panach J I, et al. Model-driven development. Informatik-Spektrum, 2008, 31(5): 394-407

[22] Rook P. Controlling software projects. Software Engineering Journal, 1986, 1(1): 7-16

[23] Deng Xiong, Chang Chuang-Ye, Wu Ji, et al. Research on model driven EJB test. Chinese Journal of Electronics, 2006, 34(12A): 2467-2472(in Chinese)
(邓雄, 常创业, 吴际等. 模型驱动的 EJB 构件测试建模研究. 电子学报, 2006, 34(B12): 2467-2472)

[24] Wang Lin-Zhang, Li Xuan-Dong. Rearch on model-driven software testing. Computer Science, 2005, 32(10): 230-235 (in Chinese)
(王林章, 李宣东. 模型驱动的软件测试研究. 计算机科学, 2005, 32(10): 230-235)

[25] Bi Kao, Chen Xi-Yi, Wu Ji. The research for generating TTCN-3 code from test system model. Computer Science, 2010, 37(11): 59-65(in Chinese)
(毕考, 陈辛夷, 吴际. 基于测试系统模型的 TTCN-3 代码生成研究. 计算机科学, 2010, 37(11): 59-65)

[26] Broekman E, Notenboom E. Testing Embedded Software. Boston, USA: Addison-Wesley, 2003

[27] Chow T S. Testing software design modeled by finite-state machines. IEEE Transactions on Software Engineering, 1978 (3): 178-187

[28] Schieferdecker I, Din G. A meta-model for TTCN-3// Proceedings of the Applying Formal Methods: Testing, Performance, and M/E-Commerce. Toledo, Spain, 2004: 366-379

YANG Bo, born in 1981, Ph. D. candidate. His main research interests focus on software testing and data mining.

WU Ji, born in 1973, Ph. D., associate professor. His main research interests include software testing and software reliability analysis.

XU Luo, born in 1976, Ph. D., senior engineer. His main research interests focus on software testing.

BI Kao, born in 1986, M. S. graduate. His main research interests focus on software testing.

LIU Chao, born in 1958, Ph. D., professor, Ph. D. supervisor, senior member of China Computer Federation. His main research interests include software engineering and software testing.

Background

Model-driven Testing is a promising approach for the automation of software testing. This approach provides the simple and effective way to finish the highly effective testing for tester. This new approach will enable software developers and testers to become far more productive and reduce the time-to-market, while maintaining high standards of software quality. Model-driven testing describes all kinds of test elements, such as static architecture, dynamic behavior and test data. Model-driven testing takes the modeling as core. Testers can model the system under test and design test cases with testing requirements. Model-driven testing can support the automatic generation of test script from the test models.

This work is supported by the Chinese National 863 Target-oriented project (Grant No. 2009AA01Z145). The goal of this project is to investigate mode-driven testing method and develop supporting tools for Web application testing. Software Engineering Institute of Beihang University takes charge of this project and divides the research into three parts, including the core platform, the test modeling and generation platform, and the test execution platform.

The test requirement model can intuitively, clear and unambiguously describes the test requirement, and is an important part of Model-Driven Testing methodology. In the conversion process from the system design model to the test design model, the test requirement model can provide the necessary information, guide the conversion process. At the same time, generating test cases through the test requirements can implement test cases back up to the appropriate system requirement, which make the testers evaluate the entire testing process better.

This paper proposes a model driven method for getting the concept of the test requirements and the content of the test requirements model describes, and defines the semantic constraints based on the test requirements meta-model, and provide the automatic semantic validation for the test requirement model, and then presents an approach of testing requirements modeling. We can get test case according to the test requirements model which corresponding to the test objectives. For test objective corresponds to test case, software requirements can trace down to test cases, and test cases can trace up to software requirements.