山东大学

硕士学位论文

基于UML活动图模型的测试用例生成方法的研究

姓名：粘新育

申请学位级别：硕士

专业：计算机技术

指导教师：王晓琳;高波

20070405

摘　要

测试用例的设计与生成是软件测试的重点和难点之所在，其本质是如何依据一种以适当方式描述的软件规格说明来设计和生成有效的测试用例。近年来，随着面向对象技术的成熟和广泛应用，基于统一建模语言 UML 的软件测试方法成为研究的热点，并取得了不少重要的研究成果。

本课题旨在针对一般大型、复杂软件所共有的交互性特征，重点研究基于 UML 活动图模型的测试用例设计与生成方法，并实现了一个与 UML 建模工具 Rational Rose 相集成的软件测试用例设计与辅助生成工具。

本文首先对 UML 以及基于模型的测试方法做了简单介绍，指出 UML 模型用于指导测试的优势所在，并分析了 UML 各种模型及其可测试性和测试策略。进而指出活动图模型不仅是进行业务需求分析和系统设计的有力工具，同时也是系统测试的重要依据。

基于上述分析，本文重点研究了基于 UML 活动图模型的测试用例设计与生成方法。对测试用例、测试场景等相关概念和技术进行了介绍，给出了基于 UML 活动图模型生成测试用例的总体策略，包括基于活动图模型控制流结构的测试场景生成和针对活动的输入量的测试数据生成。在测试场景生成部分，本文针对活动图模型的结构化问题提出了对象流处理方法及并发模块的实例化方法；在测试数据生成中，则针对测试数据的描述与生成组合问题，为活动图模型定义了测试剖面，用于描述活动图模型中活动结点的输入输出等测试相关信息，并提出了改进的轮转法以实现测试数据的组合。

最后，本课题实现了基于 UML 活动图模型的测试用例自动生成工具，为测试人员提供了测试剖面定义、测试大纲及测试用例生成功能，并提供了一个管理和使用测试大纲与测试用例的平台。

关键词： 软件测试；测试用例自动生成；UML 模型；测试场景；测试剖面

ABSTRACT

Test case design and generation is one key and challenging problem in software testing technology, which essence is how to develop effective test cases based on software specifications. Recently with the maturity and popularity of OO technology, testing based on UML has become an active topic, with some significant results achieved.

Aimed to the interactive characteristics of most large complex software systems, this research is devoted to the study of test case design and generation method based on UML activity model. Also a supporting tool is implemented which integrated with Rational Rose.

First of all, this paper gives a simple introduction of UML and model-based testing technology, points out advantages of applying UML models to direct testing, then analyzes all kinds of UML models, their testability and test strategy. And point out UML activity diagram is not only a powerful model in requirement analysis and design, it can also be an important basis for software testing.

Based on above analysis, this paper is focused on the approach of test case generation based on UML activity model. Related definition and technology are brought out such as test case and test scenario, then the test case generation strategy is put forward, which consists of the generation of scenarios based on the control flows of activity model and the generation of basic test data for the input of activities. In test scenario generation part, the methods of dealing with object flow and synchronized modules are presented in order to get a structuralized model. In the test data generation part, test profile is defined in order to describe test related information such as input/output data. An improved cycling method is also designed to solve the test data combination problem.

Finally, the design and implementation of the UML-activity-model-based test case generation tool is introduced. This tool provides the functionality of test profile definition, automated test outline and test case generation. A platform is also provided for management

of test outlines and test cases.

Key Words: software testing; automated test case generation; UML model; test scenario; test profile

原创性声明和关于论文使用授权的说明

原 创 性 声 明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的科研成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律责任由本人承担。

论文作者签名：耿新育　　日　期：2007.4.5

关于学位论文使用授权的声明

本人完全了解山东大学有关保留、使用学位论文的规定，同意学校保留或向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅；本人授权山东大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或其他复制手段保存论文和汇编本学位论文。

(保密论文在解密后应遵守此规定)

论文作者签名：耿新育　导师签名：[illegible]　日　期：2007.4.5

第1章 绪 论

随着系统规模和复杂性的日益增加，人们对软件测试越来越重视。据统计，通常软件开发组织将30%—40%的精力花费在软件测试上；而对于生命悠关的软件（如飞机控制和核反应堆），测试所花的时间和费用往往是其它软件工程活动时间之和的三到五倍。为了尽量减少花费在软件产品测试上的时间和精力，人们开始投入到软件测试方法与测试工具的研究上。而软件测试领域中的一个关键的同时也是极为困难的问题就是如何设计和生成有效的测试用例。

1.1 课题背景及研究意义

1.1.1 软件测试方法介绍

目前，有关软件测试方法和测试用例生成策略的研究已有许多重要成果，包括黑盒方法（又称功能测试、基于规格说明的测试）和白盒方法（又称结构测试、基于源代码的测试）。

结构测试虽然是一种非常有效的测试手段，可以直接捕获程序的执行情况，并自动记录和报告测试的覆盖率。但还是存在如下一些问题：（1）它仅仅关注程序本身的结构，很难保证对软件系统问题域的覆盖充分性。（2）由于软件系统实际执行路径的数目通常是天文数字，因此，结构测试一般仅限于单元测试。

与结构测试相比，黑盒测试方法则从系统或模块的外部要求和特性出发，从各种不同的角度对其进行全方位的测试。因而，功能测试的显著特点是：（1）与结构测试相比，考虑的因素更多、更全面，因而是保证软件测试质量必不可少的技术手段。（2）直接针对系统的各项功能，避免了单纯追求程序结构上的覆盖率，特别是面向对象软件，加强测试的针对性可以减少大量与应用本身无关的冗余测试。（3）适用于从单一模块直到完整系统的任意级别的测试[1]。

黑盒测试的依据是软件系统的规格说明（即编码前的分析设计模型/文档）。与基于源程序的白盒测试技术相比，它还具有如下优点[2]：（1）抽象程度好，不涉及实现细节，

因此可以很好地指导功能测试。(2) 可以用于直接针对需求和设计的检查和验证，从而尽早发现分析和设计阶段的问题，检查出基于源代码测试无法发现的错误。(3) 可以尽早进行测试计划制定和测试用例生成的工作，使得测试过程与设计实现过程实现并行，从而提高开发效率。(4) 能够获得预期输出结果，这也是基于源代码的测试无法做到的。(5) 规格说明用于生成最终程序，并生成测试用例，因此测试与实现独立。

但是，功能测试的最主要问题是：(1) 系统需求和分析阶段的规格说明一般采取自然语言编写，形式化程度低，规范性差，内容涉及面广泛，难于给出一个系统化、自动化的测试方法，也难于判定测试的充分性和完整性。(2) 而已有的一些形式化规格说明语言，如 Z 语言[3]，虽然自身具有精确、简洁、无二义性等特点，由于形式化程度太高，对开发人员来说难以理解和掌握，因此也难以广泛应用。

为此，寻找一种统一的、使用广泛而又相对形式化程度较高的规格说明描述语言，是功能测试急需解决的问题。目前，UML 作为 OMG 的标准建模语言，已被业界广泛采用，并有大量成熟的可视化建模工具用于从软件需求分析到设计实现部署的各阶段，从而在广泛性、形式化和自动化方面为问题的解决提供了一个契机。

1.1.2 软件测试的重点与难点

所谓软件测试，就是根据软件开发各阶段的规格说明和程序的内部结构而精心设计一批测试用例（即输入数据及其预期的输出结果），并利用这些测试用例去运行程序，以发现程序错误的过程[4]。由以上定义可知，软件测试的关键问题之一就是要合理地设计一个有限的测试用例集合，以最大概率地代表整个测试用例空间。

测试用例的选择与软件本身的特性和特点有着密切的关系。比如，对于一个典型的批处理程序，由于在运行过程中不再有任何来自程序以外的操作和数据输入，所以其运行结果完全由程序的初始环境设置、参数和输入数据决定。假设一个批处理程序有 N 个输入变量，则其测试用例空间完全可以用一个 N 维向量空间表示。而对于一个交互式程序，测试用例空间并不是一个简单的 N 维数值空间，而是对应于系统交互过程的、由输入控制点及输入指令或数据等要素组成的复杂的多元组序列 [1]。

充分性和有效性是测试用例生成时需要重点考虑的问题。测试用例的充分性一般可以通过测试覆盖准则来衡量，测试覆盖准则可以指导测试用例的选择，避免测试的盲目

性，保证软件测试的充分性。而有效性则是指如何以最小的代价获得尽可能好的测试用例。目前，纯手工选择测试用例使得软件测试的成本居高不下，因此测试用例自动生成方法的研究就具有重要意义。

1.1.3 统一建模语言 UML

1997 年，OMG 组织发布了统一建模语言 UML，同时业界推出了众多 UML 建模工具，例如 Rational 公司的 Rational Rose、TogetherSoft 公司的 Together、北航软件所开发的 UML_Designer 等。目前，UML 已被众多软件企业和开发者用于从软件需求分析到设计实现部署的各阶段。据专家预测，在未来 15—20 年内，UML 将是软件开发的支柱技术。

UML 融和了多种方法的成果，定义良好、功能强大、普遍适用。UML 严格地定义了对象元模型的语义，提供了获得对象结构和行为的表示法。同时，UML 支持在软件开发的各个阶段、从不同的抽象层次对系统各方面的相关信息进行建模。这些特点，使得以模型为驱动的软件开发成为可能。以 RUP 为代表的现代软件开发方法，都提倡围绕模型来实施各项软件开发工作。

作为一种标准建模语言，UML 对软件开发各阶段提供了强有力的支持。从另外一个方面来看，UML 被业界广泛采用的事实以及它的规范性（即形式化），也使得 UML 模型能够用于指导测试，验证模型与代码的一致性，检查软件产品是否符合需求和设计。

1.1.4 软件测试与软件开发过程

软件测试不仅仅是测试用例的实际执行活动，并不是在项目编码完成后才开始进行。实际上，软件测试是软件过程的一个核心工作流，贯穿于整个软件开发过程始末。软件测试过程包含了测试计划的制定、测试大纲的编写、测试用例的设计与生成、测试的实施、测试结果与问题的分析和报告、以及软件测试的管理等工作。

图 1-1 给出了软件测试过程的基本模型 POCERM[5]。该模型具有计划性、平行性、完整性、可重用性、独立性、周期性、可管理性等特点。

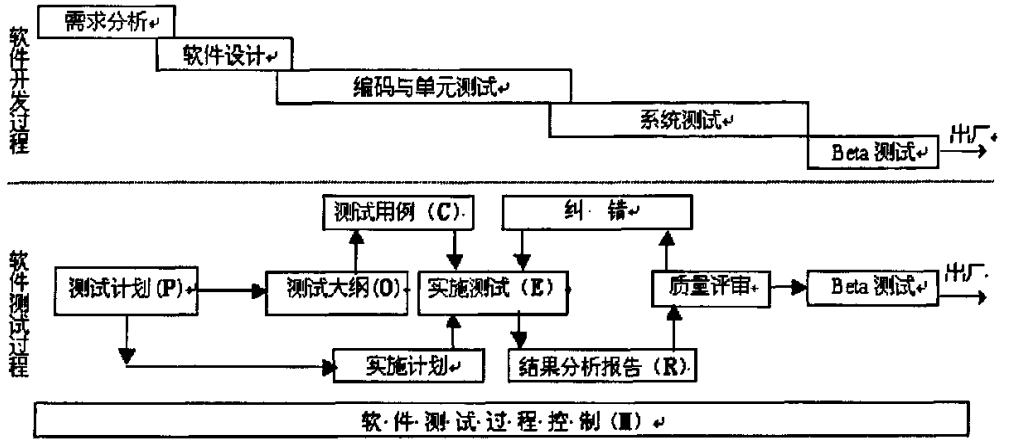


图 1-1 软件测试过程模型 POCERM

POCERM 模型中的测试过程并行于软件开发过程。测试过程和软件开发过程相互影响，相互制约。一方面，测试过程受控于开发过程。开发过程中的各个子过程的进度安排、进展状况，以及中间产品质量的优劣和粒度的大小，都直接影响着测试过程中间产品的可靠性、代表性和有效性。另一方面，测试过程也影响着开发过程各个阶段的运作。反馈的问题不仅会延迟开发的进度，同时也直接反映着系统质量和性能的优劣。

例如，软件需求分析和设计阶段产品的质量，将直接影响到测试大纲和测试用例的质量。而对测试大纲和测试用例的人工动态走查，则可以有效地发现系统实现中存在的问题，帮助系统改进相关的文档、程序代码和注释。

1.1.5 研究意义

根据上面的分析，可以得出下面的结论：

（1） 黑盒测试方法对软件测试而言具有不可替代的重要意义。

（2） 由于自然语言描述的规格说明规范性差，难于支持测试的自动化；而 Z 语言等形式化规格说明又因为难于理解和掌握，而难以广泛应用。因此黑盒测试目前急待解决的问题就是，寻找一种统一的、使用广泛而又相对形式化程度较高的规格说明描述语言。

（3） UML 以其被广泛采用的事实和形式化特性，成为当前黑盒测试关注的热点。

（4） 测试用例的设计与生成是软件测试的重点和难点所在。而测试覆盖准则的制定以及测试用例的自动化支持则是通常必须考虑的问题。

（5） 软件测试是一个贯穿于整个软件开发活动的核心工作流。软件测试过程与软件开发过程相互依赖，相互影响。

为此，本课题研究了基于 UML 模型的测试用例生成方法（重点放在 UML 活动图模型上），并实现了相应的自动化支持工具。该工具支持从 UML 需求分析和设计阶段的模型自动生成测试用例，同时还提供了与测试过程控制平台集成的测试大纲和测试用例库，以方便其使用和管理。该工具不仅能大大减少测试用例生成的代价，帮助软件开发与测试过程尽可能并行，同时还有助于保证软件开发各阶段的一致性。

1.2 国内外研究现状

目前，在测试用例自动生成方法和基于模型的测试技术上，国内外都进行了大量的研究。

在测试用例自动生成方面，Tsai 等提出了从关系代数查询表示的规格说明中自动生成测试用例的方法[6]，Weyuker 等提出了基于布尔规格说明的测试数据自动生成方法[7]。此外，基于 Z 语言的测试用例生成技术[3]也已经非常成熟。但这些测试用例生成方法，因为形式化程度太高而难于得到广泛的应用。

在基于模型的测试技术，特别是基于 UML 模型的测试技术方面，下面几项研究具有较大的影响：

◆ A.J.Offutt 等提出了基于 UML 状态图模型的测试用例生成方法[8]。该研究的主要贡献在于：（1）提出了基于 UML 状态图的四种测试覆盖准则：迁移覆盖、全判定覆盖、迁移对覆盖、全路径覆盖。（2）开发了一个验证性的测试用例生成工具 UMLTest，同时就经典的 Cruise Control 例子进行了测试。这是第一个基于 UML 模型自动生成测试用例的工具。（3）评价了上述四种测试覆盖准则。

◆ Chris Rudram 的论文[9]中研究了基于 UML 活动图模型的测试用例生成方法。Chris 通过扩充 UML 活动图的语法和语义，提出了形式化的活动图(FAD，Formal Activity Diagram)。在形式化活动图中，图元的划分更细致和具体，其语义也更丰富。在 FAD 中，活动分为用户动作、系统动作、输出动作、复合活动，而且这些动作都有自己严格的语义。例如：用户动作表示用户的输入动作，其中包括输入变量名及其值域范围、输入端口以及输入的约束条件；系统动作则表示系统对变量的处理，通过 BNF 来表示；输出

动作则通过输出断口和输出变量来表示。元素语义的细化这一思想对于测试用例的设计与生成非常有用。

但由于 Chris 的研究基于实时系统，因此他所提出的测试用例生成方法并不完全适用于交互式系统。此外，Chris 的研究仅仅对基于活动图模型的测试用例生成从理论上做了一个初步探索，没有考虑活动图模型中的复杂元素，包括循环、并发等情况。另外，我们可以看出，Chris Rudram 的方法中活动图转化成形式化活动图（FAD）需要手工完成。

◆ UML 的形式化及模型的验证。UML 是一个半形式化的语言，因此在实际应用中可能产生一些问题，包括模型二义性、不完整性等。为此，Evans 等人提出使用 Z、Z++来进一步形式化 UML [10]。Evans 等目前正在进行项目 Precise UML（pUML）的研究[11]，致力于开发定义良好的 UML。在模型验证方面，Lilius 和 Paltor 开发了一个 UML 模型验证工具 vUML，用于检查 UML 状态图。

1.3 本文的主要工作

本课题重点研究了基于 UML 活动图模型的测试用例设计与生成方法，主要研究内容如下：

1．分析了 UML 各种模型用于指导测试的可行性和测试策略，进而指出 UML 活动图模型不仅是进行业务需求分析和系统设计的有力工具，同时也是系统测试的重要依据。

2．提出了基于 UML 活动图模型生成测试用例的测试覆盖准则、总体策略。总体策略由测试场景生成与测试数据生成两条主线组成。在测试场景生成中，针对活动图模型的结构化问题提出了对象流的识别与处理方法、并发模块的识别与实例化方法；在测试数据生成中，则针对测试数据的描述与生成组合问题，为活动图模型定义了测试剖面，用于描述活动图模型中活动结点的输入输出等测试相关信息，同时结合传统的边界值分析和等价类划分的方法，给出了基本的测试数据生成方法，并提出了一种改进的轮转法以实现测试数据的组合。

3．实现了一个与 Rational Rose 相集成的测试用例设计与生成工具，为测试人员提供了测试剖面定义、测试大纲与测试用例生成功能，并提供了一个管理和使用测试大纲

与测试用例的平台。

1.4 本文的组织与安排

本文组织如下：

第一章（即本章）介绍本课题的研究背景，国内外研究现状，以及本课题的研究意义、研究目标和研究内容等。

第二章介绍基于模型的测试用例生成技术。从模型的定义开始，介绍了用于指导测试的常用模型，基于模型的测试过程，以及基于模型测试技术的优缺点。最后分析了UML模型用于指导测试的优势所在。

第三章分析UML各模型的可测试性、各自的适用范围和测试策略。本章重点研究了UML用例图、UML状态图和UML活动图模型。通过分析发现UML活动图模型具有描述业务过程、工作流和并发过程的能力，不仅在软件需求分析和设计中不可缺少，同时也是测试用例的设计与生成的重要依据。

第四章详细阐述基于UML活动图模型的测试用例生成方法和技术。本章首先给出了活动图模型的形式化描述，并对基于活动图模型的测试用例给出了明确的定义，然后，针对活动图模型描述能力的不足，定义了活动图测试剖面，并详细描述了测试用例生成方法，包括活动图的结构化方法、活动图的测试场景生成方法、测试数据生成与组合方法。

第五章介绍本系统的总体设计与实现。本章先从总体上给出了系统的逻辑结构和实现机制，然后介绍测试用例生成系统，包括设计与实现中的关键算法及实例介绍。

最后是课题工作的总结与展望。概括全文，总结研究成果和所做的工作，并对进一步的工作方向进行了展望。

第 2 章 基于模型的测试技术

随着模型在软件设计和开发中越来越普遍的应用（包括 UML），基于模型的测试也越来越受到人们的关注。模型描述了系统的功能以及系统可能的行为，使得测试以一种清晰有序的方式进行。本章将对基于模型的软件测试技术进行基本的介绍。

2.1 模型的定义

所谓模型，是对一个系统的抽象，它从某个特定的视角、在某个特定的抽象层次上对所建模的系统进行描述。人们常常在正式构造系统之前，首先建立一个简化的模型，以便更透彻地了解其本质，抓住问题的要害。正如 James Rumbaugh 所言，“建模意在把握系统最为本质的部分”。模型是理解、设计、测试和维护系统的基础。就软件而言，我们可以给出如下定义：

模型（Model）：简单的说，模型是对软件行为的一个描述。软件行为可以通过系统接收的输入序列、动作、条件、输出逻辑、模块间的数据流等来描述。模型需要尽可能形式化，并与实际系统相符合，从而使得模型具有可重用性、可共享性和精确性。

模型不但抓住了所研究系统最本质的关系而且比所研究系统更易于开发和分析。模型具有四要素：主题（Subject）、论点/理论（point of view/ theory）、表示（representation）、技术（technique）。首先模型必须有定义完好的主题，在测试中我们感兴趣的是能够帮助我们生成有效测试用例的被测系统模型。其次，一个模型必须以一定的理论为基础，该理论能够指导识别相关的问题及信息，并根据模型生成测试所需的信息，例如有穷自动机理论、图论、Markov 链等。此外，一个建模技术必须具有某一特定的表示方法，很多软件测试模型都用图来表示。最后，模型是复杂的人工制品，因此建模技术的好坏与建模者有着密切的关系。

现实中存在大量的模型，他们从不同的侧面描述了软件行为。例如：控制流、数据流、程序依赖图通过分析源代码的结构来表现系统实现的行为。决策表、状态机则用于描述系统的外部行为，即人们所说的黑盒行为。当我们说起基于模型的测试时，我们倾向于认为这里的模型是指黑盒模型。

2.2 测试中常用模型介绍

测试中常用的模型包括：有穷状态机（FSM），状态图（Statechart），Markov 链,UML 模型等。下面分别进行介绍。

有穷状态机（FSM）与状态图（Statechart）：有穷状态机有两种表示形式：状态转移矩阵、状态转移图。早在软件工程产生之前，有穷自动机的相关理论就已经非常成熟了。在计算机硬件部件的设计和测试中，有穷状态机的使用已经成为一种标准。[12]较早提出将有穷状态机模型用于软件组件的设计和测试。目前，FSM 在协议一致性测试中是必不可少的。基于 FSM 的测试用例生成方法包括[13]：W 方法、Wp 方法（Partial W-method）、迁移遍历方法(Transition Tour)、DS(Distinguishing Sequence method)方法、UIO（Unique-Input-Output）方法等。但有穷自动机存在的一个问题是，复杂软件将产生大的有穷自动机，相应地创建和维护的费用就会比较高。

状态图是有穷状态机的扩展，用于解决复杂的大型交互式系统建模问题。状态图由 Harel 在 1987 年提出，与有穷状态机相比，状态图新增加了如下几个特性：（1）层次化特性和并发状态。（2）时间约束、动作、事件、广播机制等。

Markov 链：Markov 链是一个随机模型，它被用于描述软件的使用。Markov 链很象有穷自动机，可以认为是一种概率性的有穷自动机。Markov 链不仅用于生成测试用例，还用于收集和分析失效数据，从而对软件的各种质量参数进行评估和度量（如可靠性、平均失效时间等）[14]。

UML 模型： UML 模型综合了多种模型的优点，可以从不同的视角描述系统，具有很强的描述和管理能力。目前，UML 已被众多软件企业和开发者所采用。UML 融和了多种方法的成果，具有定义良好、功能强大、普遍适用的优点。这些特点，为基于模型的测试提供了非常好的契机。目前，基于 UML 模型的测试方法研究已经展开，主要集中在动态模型方面，包括状态图、活动图以及用例图等。

2.3 基于模型测试的基本过程

基于模型的测试过程由下列活动组成 [15]：

1．了解被测系统，根据测试目标确定被测组件及其功能特性。

2．选择合适的模型。选择模型时，要考虑系统特征、工具支持等方面的因素。

3．建立模型。

4．根据模型生成测试用例集。

5．运行测试用例集。

6．收集测试结果，对测试结果进行评价。

7．利用测试结果，评估测试用例集的价值、度量软件的质量特性等。

图 2-1 给出了基于模型测试的基本过程图示：

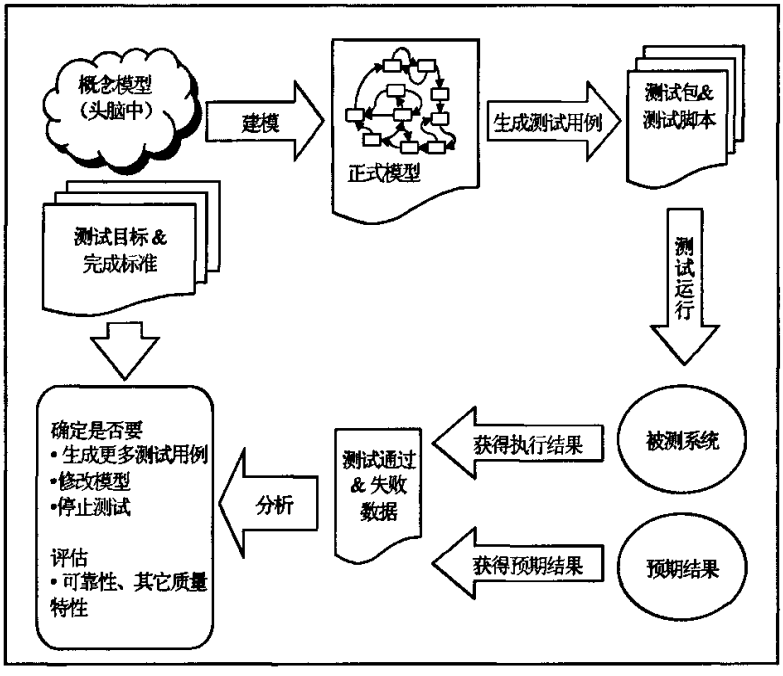


图 2-1 基于模型的测试过程

2.4 基于模型测试的优缺点及可能存在的问题

基于模型测试的优点

（1） MBT 技术可以用于各种不同系统的测试，包括图形用户界面、嵌入式系统等。

（2） 模型具有很好的抽象性，表现了被测系统的关键元素，而去除了非关键元素。

（3） 模型具有可重用性，可在项目组的不同人员中共享。

（4） 很多模型都具有成熟和雄厚的理论基础，这样就使得基于模型的软件测试活

动（包括测试用例的自动生成等）变得容易。例如：基于有穷自动机的测试用例自动生成可以采用图论算法，而 Markov 链中则可以采用随机过程理论来度量软件的可靠性。

（5） 根据模型还可能获得对应输入的预期输出结果。

基于模型测试的难点及所关注的问题

（1） 基于模型的测试要求测试者具备一定的技能，他们必须熟悉所使用的模型及相关原理。同时还需要熟悉相关工具、脚本和编程语言。

（2） 在基于状态的模型中，存在状态空间爆炸的问题，这不仅是建模的难点，同时也对测试产生了一定的障碍和影响。此外，对于一个大型的系统而言，模型的创建、维护与管理也是一个值得关注的问题。

（3） 制定模型的测试覆盖准则是基于模型测试的一个重要问题。

（4） 测试自动化。

2.5 UML 模型在测试方面的优势

分析了基于模型测试的优点和难点之后，我们认为 UML 模型在指导测试方面有如下优点[16]：

（1） 通用性：UML 作为标准建模语言，具有广泛的适用性，目前已被软件开发界广泛采用，支持从软件需求分析到设计实现部署的各阶段。并有大量商业工具支持。

（2） 形式化：UML 模型具有严格的定义，提供了获得对象结构和行为的表示方法。这种形式化特性使得测试信息的提取和自动化变得容易。

（3） 强大的描述能力：UML 模型集众家之长，具有强大的描述能力。UML 包括一系列视图和模型，他们从不同的层次和角度描述了软件系统的结构、行为以及软件的使用。

（4） 强大的管理能力：UML 模型通过提供不同层次的视图和包机制等，具备了强大的管理能力，解决了模型维护和管理的问题。同时通过分层等方法在一定程度上解决了状态空间爆炸的问题。

（5） 可重用性：UML 模型支持在软件开发的各阶段、从不同的抽象层次对系统

各方面的相关信息进行建模。这些模型不仅可以用于软件开发阶段，还可以用于指导测试，因此避免了专门为测试构造模型，实现了软件分析和设计阶段制品的重用，同时也将测试活动与开发过程集成起来。

（6） 可迭代性：可以尽早开始测试活动（包括测试计划的制定、测试大纲与测试用例的设计等），并随着设计活动的细化不断细化生成的测试制品。这样，软件开发与测试开发可以并行进行，并在整个测试过程中进行持续测试活动。众所周知，越早开始测试越好。

基于以上原因，我们认为 UML 不仅是软件开发的重要工具，同时也是指导测试的重要模型。

2.6 本章小结

本章首先对模型进行了定义，然后列举了测试中常用的模型，并给出了基于模型测试的过程，以及基于模型测试的优缺点。本章最后还分析了 UML 模型的优点，并指出 UML 可用于指导软件测试。

第 3 章 UML 各模型的可测试性与测试策略分析

上一章我们分析了 UML 模型用于指导测试的优势所在。本章首先对 UML 进行介绍，然后在国内外已有研究的基础上，分析 UML 各模型的可测试性，并指出各模型适用的范围及相应的测试策略。

3.1 UML 简介

一、 UML 的内容

统一建模语言 UML 是面向对象技术与可视化建模技术发展的里程碑，是当今成功地进行软件开发所不可缺少的关键环节。UML 是对软件密集型系统中的人工制品进行可视化表示、详细刻画、构造和存档的标准语言[17]，它融合了多种方法的成果，定义良好、功能强大、普遍适用，适用于以面向对象技术描述的任何类型系统，具有很宽的应用领域；而且 UML 适用于系统开发的不同阶段，从需求规格描述直至系统维护。

为了刻画软件开发的各个方面，UML 提供了九种图，包括：类图、对象图、用例图、交互图（包括顺序图和合作图）、状态图、活动图、构件图、配置图。UML 通过四层元模型体系结构来定义，并将 UML 分成三个逻辑子包：基础设施包、行为元素包、模型管理包。基础设施包给出了系统静态结构的基本架构，包括：类图、对象图、组件图、配置图。行为元素包用于支持系统动态行为建模，包括：用例图、交互图（包括顺序图和合作图）、活动图、状态图。总而言之，所有这些不同类型的 UML 图为软件系统提供了描述静态结构、动态行为和物理配置的强大能力。

二、 UML 与软件过程

软件开发过程是将用户需求转化为软件系统所需要的一系列活动[18]。它规定了为达到一定的目标，谁在什么时候应该做什么，以及如何做等内容。

UML 是一种建模语言，一种符号体系，而不是一种方法。从理论上讲，任何方法都应由建模语言和建模过程两个部分组成。其中建模语言提供了这种方法中用于表示设计的符号（通常是图形符号），建模过程则描述进行设计所需要遵循的步骤。UML 是与

过程无关的，适用于各种软件开发过程。

美国 Rational 公司在综合多种软件开发方法和业界多家著名公司经验的基础上，在 1998 年推出了 Rational 统一过程（Rational Unified Process，简称 RUP）。该过程是有效使用 UML 的指南，是迄今为止比较理想的软件开发过程。RUP 具有如下特点：用例驱动、以体系结构为中心、迭代、增量地进行软件开发。

RUP 将软件生命周期分为多个周期（cycle），每一个周期工作在产品的新一代上。周期进一步被分为四个连续的阶段（phase）：初始阶段、细化阶段、构造阶段和移交阶段。图 3-1 给出了 RUP 软件开发过程：

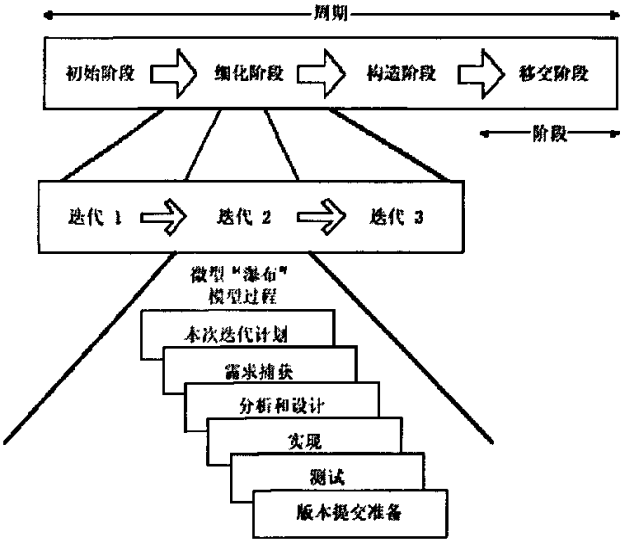


图 3-1 RUP 软件开发过程

尽管测试并不是 UML 的一个组成部分，但它在 RUP 过程模型中不可缺少，同时软件测试 POCERM 模型也适用于 RUP 的每一个迭代阶段。目前 UML 已经成为软件系统建模的工业标准，而在众多软件开发组织中，测试的费用通常占到系统开发的 30%－40%甚至更多。基于这样的实事，我们认为，研究基于 UML 模型的软件测试可行性及测试方法，并提供相应的自动化支持工具，具有非常重要的意义。

为了研究如何基于 UML 模型进行软件测试，我们将注意力集中在 UML 的行为元素包中（即动态模型）。因为软件测试活动大多是为了发现软件系统运行过程中出现的缺陷，通常这些缺陷都是动态（与行为相关）的。但这些动态模型通常还需要基础设施包中的静态模型信息进行补充。

3.2 模型的可测试性定义

模型可测试性是指，仅用模型中给出的信息，设计和实现一定的算法来产生就绪运

行的测试用例的难易程度。可测试的模型应当既支持手工生成测试用例又支持自动生成测试用例。

一个可测试的模型需要满足以下要求[19]：

（1） 它应该完整而准确地反映被测系统，并描述了要测试的所有功能特性。

（2） 它是对细节的抽象，这些细节会使得测试成本过分高昂。

（3） 它保留被测系统中有助于发现错误和验证系统一致性的关键细节。

（4） 对状态模型而言，它描述了所有的事件、动作和状态，并对各状态进行了明确定义。

由上述定义可知，UML 模型的可测试性是指，模型是否包含足够的信息，以自动或手工生成测试用例。在制定 UML 时，可测试性并没有考虑在内。UML 实际上是一种半形式化语言，它的高度灵活性，使得模型中可以存在二义性、不完整性和不一致等问题，因此开发者使用 UML 的方式决定了 UML 模型的可测试性。

如果一个模型是可测试的，那么可以称该模型是测试就绪的。

3.3 UML 各模型的可测试性与测试策略分析

3.3.1 用例图分析

目前，UML 的用例图已经被广泛用于捕获需求，并通过文本描述、交互图或者活动图进行细化，作为客户与需求分析人员、系统开发人员之间的交流介质。在面向对象开发中，用例是系统级需求规格说明的最重要形式。“用例的集合即是系统的全部功能”[20]。

用例将软件开发过程的各个活动紧密关联起来。现代软件开发是一个用例驱动的过程。开发过程中的各个活动，从需求捕获到设计、实现、测试，都是用例驱动的。而测试，最根本的就是要验证系统能否满足用户的需求，即用例集合。

用例图描述了用户的需求，因此可用于指导系统级的功能测试。

在可测试性方面，UML 用例图模型还存在下列问题[19]：

（1） 没有定义用例中所涉及到的输入输出变量以及它们之间的关系。用例表示了系统与外部动作者之间的对话。这种对话包括输入输出信息、用户操作等。缺乏输入输出信息，则生成的测试用例必然是不完整的。

（2） 没有描述各用例的使用频度、重要程度、安全级别等信息。这些信息，虽然对软件开发过程并不是非常重要，但对于软件测试却意义重大，特别是对软件测试资源的分配和软件测试进度的安排。

（3） 没有描述用例之间的顺序依赖关系和约束关系。一般的系统有上百个用例，这些用例之间存在顺序依赖和约束关系，这种约束正是系统业务逻辑的直接体现和结果。也就是说，用例并不能以任意的次序执行，某些用例只有在其它一些用例执行之后才能进行。在基于用例图进行测试时，用例之间的这种约束关系必须被明确刻画出来。

（4） 用例是系统功能的高层表示，但用例图仅仅描述了系统功能的框架以及使用者与系统功能之间的交互关系，而用例的很多具体信息并不能在用例图中体现出来。这些信息包括：用例的简单描述、用例的前置后置条件、用例中的事件流(包括正常情况、异常情况)等信息。如果缺乏这些信息，那么生成的测试用例也仅仅是一个框架，是一个需要细化的测试大纲，而不是一个可以立即执行的测试用例。

上述问题影响了 UML 用例图模型的可测试性。

3.3.2 状态图分析

UML 状态图来源于 David Harel 的 Statechart，它支持并发、层次化、事件等特性。UML 状态图用于描述一个特定对象的所有可能状态以及由于各种事件的发生而引起状态之间的转移。大多数 OO 技术都使用状态图来表示单个对象在其生命周期中的行为。

因此，状态图通常用于类级测试，而系统是由多个相互合作的对象组成的，而单个状态图中无法表达多个对象之间的交互合作信息，因此要进行更高层的测试，必须对单个分离的状态图进行一定组合，提出相应的组合规则。

UML 状态图本质上是一个扩展的有穷自动机，它与传统自动机的区别在于，UML 状态图支持并发、层次化、事件等特性。正如第二章中所言，FSM 的相关理论非常成熟，基于 FSM 以及状态图的测试技术自 90 年代也取得了相当的成果，并在电信、协议设计等领域形成了自己的标准。因此基于 UML 状态图的测试可以借鉴这些成熟的技术。

在可测试性方面，UML 状态图模型还存在下列问题：

（1） 状态图模型的正确性、完整性：在 UML 状态图模型用于生成测试用例之前，它应该是完整的、正确的。UML 的灵活性使得模型中可能存在不完整性和二义性。

（2） 状态图的结构化：状态图中存在复合状态（包括并发复合状态和顺序复合状态）。如果不进行处理，则难以将 FSM 的理论用于 UML 状态图模型的测试上。

（3） UML 状态图中缺乏对状态本身的定义和必要的测试约束信息。如果不对状态进行精确定义，则无法对结果状态进行检查。

上述问题影响了 UML 状态图模型的可测试性。

3.3.3 交互图分析

交互图常常用来描述一个用例的行为，显示该用例中所涉及的对象和这些对象之间的消息传递关系。它主要用于描述对象之间的动态合作关系以及合作过程中的行为次序。交互图有两种形式，即顺序图和合作图，他们分别从不同的侧面来描述对象之间的交互关系。顺序图用来描述对象之间的动态交互关系，着重体现对象间消息传递的时间顺序。而合作图则更着重于对象之间的静态连接关系[21][22]。

总之，如果要描述一个用例中几个对象协同工作的行为，交互图是一种有力的工具。合作图可以用于表示测试用例的一个具体场景。

在可测试性方面，UML 交互模型存在下列问题：

（1） 用交互图来表示复杂控制非常困难，例如选择、循环等。

（2） 交互图仅仅表示了用例的一个具体场景，通常需要多个场景才能够完全覆盖一个用例。

对于上述问题，我们认为，单独使用交互图难以生成高质量的测试用例。测试时，通常将交互图模型和用例模型相结合，为基于用例模型的测试提供具体的场景信息。

3.3.4 类图分析

类图技术是面向对象方法的核心技术。在面向对象的建模技术中，类图描述了系统中的类及相互之间的各种关系。类图模型中包含的信息包括：属性、操作、关联、聚集与组成、泛化、约束规则（OCL）、构造型、可见性等，这使得类图具有很强的表达能

力。

类图可以用于软件开发的各阶段。在项目的不同开发阶段，应当使用不同的观点来画类图。如果处于分析阶段，应画概念层类图；当开始着手软件设计时，应画说明层类图；当考察某个特定的实现技术时，则应画实现层类图。同时，画类图时，不要一开始就陷入实现细节，而应该把精力放在关键的领域[23]。

类图模型在指导测试时具有如下作用：

（1） 为动态模型补充和提供有用的信息，以提高动态模型的可测试性。

（2） 类图模型同样可以用于指导类级和类簇级测试。

[24]给出了一种使用 UML 类图指导测试的方法，其主要思路是将 UML 类图转化成类依赖关系图（CDG: Class Dependency Diagram），然后基于 CDG 生成测试用例。

3.3.5 活动图分析

UML 活动图的技术思想主要来源于 Jim Odell 的事件图、SDL 状态建模技术和 Petri 网技术。这些技术主要用于描述工作流和并行过程的行为[23]。活动图模型所具有的描述系统工作流程和并行活动的能力，使得它成为描述系统业务过程的重要工具。UML 活动图可以用于记录单个操作或方法的逻辑，单个用例、或者单个业务流程的逻辑。

活动图模型的使用者包括客户、测试小组以及软件开发人员等。对于他们而言，活动图模型是系统功能的可视化蓝图，是理解系统行为的重要工具。

具体的说，UML 活动图模型可用于以下几个方面：

（1） 细化用例，描述系统功能性行为。用例是捕获和表现用户需求的关键技术，为了进一步细化用例，就需要获得系统执行的基本流程、异常流程、以及其它可能的流程，这些流程正是系统处理客户请求时的主要和次要执行场景。我们可以采用标准的用例模板来描述这些信息，但是这种方式在用例场景复杂的情况下难于理解，而且在测试时也难于自动化。而 UML 活动图模型则可以很直观和方便地描述这些流程。

（2） 描述用例之间的顺序依赖关系。这是一种概念层活动图，其中活动结点都是用例。在本章“用例图分析”一节中，我们指出，使用概念层活动图可以解决用例间约束描述的问题。由（1）（2）我们不难看出，活动图对于用例图

模型而言至关重要。

（3） 描述复杂的业务过程。活动图模型具有描述并行过程的能力，这使得它可以方便地对业务过程和工作流建模。这是理解、构造、测试和维护系统的重要模型。

（4） UML 活动图模型甚至可以用于描述复杂的计算型算法。

总之，活动图模型所具有的描述系统工作流程和并行活动的能力，使得活动图模型成为系统测试的重要依据。此外，UML 活动图模型可以针对系统的不同层次建模，从系统级、子系统级到类级，因此活动图模型也可以指导不同层次的测试，包括系统级的功能测试、集成测试以及对象类的单元测试。

在可测试性方面，UML 活动图模型存在下列问题：

（1） UML 活动图模型中存在着复杂的非结构化和并发特征。这将加大测试的难度。

（2） UML 活动图模型中仅仅刻画了活动之间的控制流关系，而活动的输入输出变量以及其它与测试密切相关的信息并没有描述，如果不补充这些信息，那么生成的测试用例将是不完整的。

上述问题影响了 UML 活动图模型的可测试性，针对这些问题，我们给出如下方法以扩充其可测试性，详细的论述将在本论文第四章给出：

（1） 识别出活动图模型中的对象流、并发模块、循环，并进行相应的处理。

（2） 定义 UML 活动图的测试剖面，用以描述活动相关的输入输出变量、活动的前置条件后置条件不变式以及测试覆盖准则等信息。

基于 UML 活动图模型的测试覆盖准则以及测试用例生成方法也将在本文第四章给予详细介绍。

现代软件开发过程都是用例驱动的，包括测试。通过前面的分析可知，活动图模型对于用例而言具有不可替代的意义，同时也是描述和理解系统流程和功能行为的重要手段。所以，研究基于 UML 模型的测试方法（特别是系统级功能测试），活动图模型的测试方法研究是非常根本和重要的部分。

3.4 本章小结

本章首先简单介绍了 UML，然后在国内外已有工作的基础上，详细分析了 UML 各模型用于指导测试的可行性以及相应的测试策略，包括用例图、状态图、交互图、类图以及活动图。通过分析我们发现，UML 活动图模型是基于 UML 模型测试（尤其是系统级功能测试）应研究的根本和重点问题。

第 4 章 基于 UML 活动图模型的测试用例生成方法

上一章指出了 UML 活动图模型是系统测试（尤其是系统级功能测试）的重要依据。本章将详细论述基于 UML 活动图模型的测试用例设计与生成方法。

我们首先分析 UML 活动图模型及其适用范围，介绍测试用例、测试场景等相关概念和技术，然后给出了基于 UML 活动图模型生成测试用例的总体策略，包括基于活动图模型控制流结构的测试场景生成和针对活动输入量的测试数据生成。在测试场景生成部分，我们针对活动图模型的结构化问题提出了对象流处理方法及并发模块的实例化方法；在测试数据生成中，我们针对测试数据的描述与生成组合问题，为活动图模型定义了测试剖面，用于描述活动图模型中活动结点的输入输出等测试相关信息，并提出了改进的轮转法以实现测试数据的组合。

4.1 UML 活动图模型分析

4.1.1 活动图适用范围分析

要研究基于活动图模型的测试方法，我们需要首先了解活动图模型的适用范围。

UML 活动图在系统分析和设计中特别有用，包括从初始阶段、细化阶段到构造的整个过程。活动图主要用于软件系统动态建模，在下列情况下可以使用 UML 活动图模型：

（1） 分析和细化用例。用例图模型从使用者的角度来描述系统功能。但是，仅仅通过用例图并不能描述用例中的事件序列，特别是当用例中存在多个可能的执行场景时（包括基本场景、异常场景等）。这时，就可以使用活动图来清晰地记录特定用例中的活动执行顺序。在 Rational Rose 中，活动图可以直接依附于一个用例。

（2） 理解和建模业务过程和工作流，处理多线程应用。由于活动图模型具有描述系统工作流程和并发行为的能力，因此，适合于建模业务过程和工作流。这是理解、构造、测试和维护系统的重要模型。

（3） 描述复杂算法。

实际上，UML 活动图还特别适用于描述软件的外部交互操作行为。总之，UML 活动图描述系统工作流程和并行活动的能力，不仅使它成为构造和理解系统不可缺少的工具，同时也成为软件测试特别是功能测试的重要依据。UML 活动图模型适用于软件系统测试和集成测试，同时也适用于针对单元模块的功能测试或结构测试。

4.1.2 活动图模型元素分析

本文从测试的观点给出了活动图模型的一种简明的形式化定义，并据此研究了活动图模型的结构化和测试场景生成的基本方法。

活动图模型可表示为：

$$AD = (A_D, T_D) \quad (1)$$

A_D 为结点集合，定义为：

$$A_D = \{Initial_D, Final_D, Activity_D, ComponentActivity_D, Fork_D, Branch_D, Join_D, Merge_D, SignalSender_D, SignalReceiver_D, Object_D\} \quad (2)$$

其中 $Initial_D$、$Final_D$、$Activity_D$、$ComponentActivity_D$、$Fork_D$、$Branch_D$、$Join_D$、$Merge_D$、$SignalSender_D$、$SignalReceiver_D$、$Object_D$ 分别表示初始结点、终止结点、基本活动结点、组合活动结点、并发分支结点、条件分支结点、并发汇聚结点、条件汇聚结点、信号发送结点、信号接受结点以及对象结点。

T_D 为活动图中的迁移边集合，定义为：

$$T_D = \{ ControlFlow_D, SignalFlow_D , ObjectFlow_D\} \quad (3)$$

其中

$$ControlFlow_D: (A_D, FlowLabel_D) \rightarrow A_D \quad (4)$$

$$SignalFlow_D: (SignalSender_D, FlowLabel_D) \rightarrow SignalReceiver_D \quad (5)$$

$$ObjectFlow_D: (A_D, FlowLabel_D) \rightarrow Object_D, \text{ or } (Object_D, FlowLabel_D) \rightarrow A_D \quad (6)$$

$$FlowLabel_D = (Events_D, Conditions_D, Actions_D) \quad (7)$$

其中 $ControlFlow_D$, $SignalFlow_D$, $ObjectFlow_D$ 分别表示控制流、信号流以及对象流；$FlowLabel_D$ 表示迁移标识。

所谓测试场景就是活动图中一条具体的执行路径。在一个活动图 AD 中，设

$$p = (a_1', \ldots, a_n') \quad \text{其中 } a_i' = (t_i, a_i),\ t_i = a_{i-1} \rightarrow a_i \ (i = 2, \ldots, n) \quad (8)$$

则称 p 是相对于活动图所代表系统（子系统）的从结点 a_1 到结点 a_n 的经过迁移边 t_2，…，t_n 的一个测试场景。

4.2 相关概念与技术

4.2.1 测试用例

一般而言，软件测试用例是针对软件的某些有预期质量要求的特性指标（如功能、性能、可靠性等方面的要求）和软件自身特性（规格说明中规定的功能、软件的内部结构和外部环境特性等），为达到或满足预定的软件测试目标而设计和生成的测试数据、操作序列和与之对应的预期输出结果的集合。不言而喻，对于测试而言程序输入数据的选择是关键。对于数据处理型的应用，测试数据主要是由一些基本类型的数值组合而成，但是对于交互式软件而言，操作序列无疑也是测试用例的重要组成部分[25]。

UML 活动图具有描述复杂控制流程和并行活动的能力，适合于对复杂的计算流程和工作流程进行建模。实际上，UML 活动图特别适用于描述软件的外部交互操作行为。因此，它成为软件集成测试、系统测试的重要依据，同时也适用于针对单元模块的功能测试或结构测试。

基于 UML 活动图模型的测试用例主要由两个部分组成，即测试场景与测试数据，可表示为：

TC_{AD} =(Scenario, Data)

Scenario 表示针对活动图的一个测试场景，由活动图中一系列相关的活动组成（包括活动的迁移信息、触发事件等），对应于活动图中一条典型的执行路径。Data 表示测试数据，是指对应于特定测试场景的输入信息（包括各种类型的数据以及用户操作等）。此外，基于 UML 活动图模型的测试用例中还应包括相应的预期输出结果、以及用于验证输出结果的前置条件/后置条件/不变式等信息等。

基于 UML 活动图模型的测试用例集 TS 是满足一定测试覆盖准则的测试用例的集合，可以表示为：

$TS = \{TC_{AD}\} = AD(CA)$

其中，CA 表示活动图的测试覆盖准则(test Coverage criteria on Activity diagram)。测

试用例集是活动图 AD 的一个函数。

4.2.2 测试大纲

软件测试大纲是软件测试的依据。它明确详细地规定了在一次测试中对系统的每一项功能或特性所必须完成的基本测试项目和测试完成的标准。无论是自动测试还是手动测试，都需要满足测试大纲的要求。

实际上，测试大纲是从测试的角度对被测对象的功能和各种特性的细化和展开。其中针对系统功能的测试大纲是基于测试人员对系统需求规格说明书中有关系统功能定义的理解，将其逐一细化展开后编制而成的。因此，测试大纲不仅是软件开发后期测试的依据，而且在系统的需求分析阶段也是质量保证的重要文档和依据。

4.2.3 场景技术

场景就是顺序化的、确定化的系统执行轨迹。场景技术在软件生命周期中有着广泛的应用，在需求分析阶段，场景可用来捕获需求和系统的功能；在软件设计阶段，场景是软件体系结构建模的主要依据，也是软件主要行为的集中体现。

同样，我们认为场景技术对于测试人员来说也很重要。因为现实中进行系统测试时，必须全面考虑系统的用户使用时会出现的种种情况，包括正常和非正常情况。虽然软件的黑盒测试在软件的接口处进行，不考虑程序的内部逻辑结构特性，但实施系统测试时，程序运行的上下文环境（或者说“历史”）总会影响系统的执行轨迹，而最终影响执行结果。因此，我们有必要在测试时把描述系统执行过程的场景作为一个参考，以利于保证测试的全面可靠。经过分析，可以总结出场景共有以下三点作用：

（1） 验证模型设计是否满足需求。需求捕获中最大的问题是软件人员与用户之间的沟通障碍，设计的模型不能准确描述用户的要求或者与初始的想法有些偏离，根据模型自动生成的场景可以反过来验证模型的正确性，保持整个软件开发过程的一致性。

（2） 为软件走查提供依据。参加走查的评审人员要在很短时间内读懂软件设计模型（如 UML 模型）是有困难的，而场景是反映系统的期望运行方式，是从用户的角度来看的，易于被他人理解，从而易于有的放矢的提出问题，减少了走查时的随机性。

（3） 为功能测试用例的生成打下基础。在大型复杂软件的测试实践中，特别是对于分布式和交互式系统的测试，测试用例通常包括一个特定的测试场景和与之相对应的一组输入数据（包括操作指令、输入数值和初始化设置值等）两个部分，所以场景的生成是必须的。

4.3 基于活动图模型生成测试用例的总体框架

根据前面的定义，基于活动图模型的测试用例由测试场景和测试数据两部分组成。

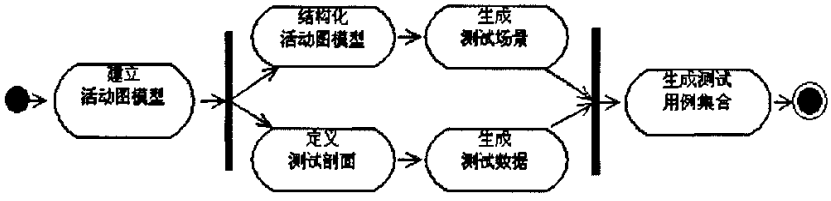


图 4-1 基于 UML 活动图模型的测试用例生成总体框架

因此，针对活动图模型的测试用例生成应当包含如下三个部分，如图 4-1 所示。

一是活动图模型的结构化和测试场景生成。由于活动图中存在着非结构化和并发特性以及对象流、消息流等因素，因此要首先完成活动图模型的结构化以消除这些因素。在结构化的活动图模型基础上，根据一定的测试覆盖准则和测试策略，生成满足要求的测试场景集合。

二是测试剖面说明和基本测试数据生成。实际上，针对活动图模型的测试不仅需要一组满足一定覆盖准则的测试场景，同时还需要针对各个活动中的输入输出设计和生成一组适当的数据实例集合（包括用户操作），作为备选的测试数据。

三是通过对测试场景和基本测试数据进行适当的组合生成一组所需的测试用例集合。

然而，由于活动图本身主要表示活动的控制流程，对于输入数据和其它测试要求没有提供规范的描述手段，因此我们需要一种途径来描述在各个活动节点上的输入数据空间和测试数据生成约束条件等。这正是定义测试剖面的目的。

通过定义测试剖面，基于活动图的测试模型就建立起来了，它由结构化的活动图模型和测试剖面两个部分组成。关于测试剖面的定义，请参看 4.5.1 节。

4.4 测试覆盖准则设计

测试覆盖准则是软件测试的一个关键问题。测试覆盖准则是施加于一组测试用例之上的一条或多条规则。测试覆盖准则可以减少测试的盲目性，保证测试的充分性，帮助测试人员制定测试策略，生成测试用例以发现尽可能多的错误，并决定何时停止测试。

[1]提出了基于程序交互执行流程图 PIEF 的四条功能测试覆盖准则，因为我们采用的 UML 活动图与程序交互执行流程图 PIEF 从顺序关系上看非常相似，也是由控制点和转移边组成，所以以下四条覆盖准则同样适用于活动图，但是在第 2 条覆盖准则的基础上增加了信号流覆盖准则。

准则 1(控制点覆盖准则) 图上的所有控制点至少要执行过 1 次。对应于活动图，即是各种不同类型的结点都要覆盖。

准则 2(转移边覆盖准则) 图上的所有转移边至少要执行过 1 次。在活动图里，转移边具体是指控制流、对象流和信号流。活动图中的对象是前一个活动的输出，后一个活动的输入，对象到达某个状态后，必须由下一个活动接收进行下一步的处理，类似于处理流程，所以转化为控制流比较适合。生成场景时把对象流转换成控制流，至少执行一次（即在场景中包含一次）就可以达到测试目的。而信号流，则更加复杂，信号的接收一般是一个反复等待、查询的过程，信号的发送与接收并没有必然的流程关系。

信号流分为同步与异步两种，同步信号流是一个双向的信号发送接收流，发送方必须接到接受方发回的应答信号后才能继续下一步操作，是一个握手的过程，所以有可能在此阻塞。异步信号流只是一个单向的消息，发送方在发送信号后不考虑接收方的情况，而可以直接向下执行，接收方有可能接到，也有可能接收不到信号。针对上述分析，单纯的转移边覆盖准则不能完全覆盖信号流的各种情况，所以本文专门提出了信号流覆盖准则。

信号流覆盖准则，即生成场景时，至少要包含三种可能场景：

一种是信号流发生场景，即发送活动与接收活动同时出现，发送活动达到发送信号的各种条件，且接收活动接收到了信号并作出相应的反应；

一种是发送方不发送信号的场景，发送方未达到发送信号条件或者触发了其他活动而没有发送信号；

一种是发送方发送信号而接收方接收不到场景，例如网络中断是最常见的原因。这时我们关心发送方与接收方的后续操作，特别是同步信号流的发送方是一直等待，还是超时重发；异步信号流的接收方没收到信号会有什么不同的操作流程。这些是设计人员最容易忽略的，正是测试者最应该测试的，因此这种场景比上两种更为重要。

准则 3(逻辑路径覆盖准则) 图上的所有路径至少要执行过 1 次。在运用此项测试准则时，必须对图中的循环作一定的限定，比如限定只考核两种情况，即不进入循环体和进入循环体 1 次或 1 次以上。经此简化后的路径称作逻辑路径或基本路径。

准则 4(代表值覆盖准则) 对于每一个控制点以及控制点上的每一个控制量，依据一定的测试策略选定一个有限的输入值集合，称作代表值集合，则每个控制点的代表值集合中的每一个值至少要被测试过 1 次。

4.5 基于活动图模型生成测试用例的方法

本节将对基于活动图模型生成测试用例的具体方法进行介绍。包括：定义测试剖面、结构化活动图模型、基于活动图模型的测试场景生成、测试数据生成与组合四个部分。

4.5.1 定义测试剖面

UML 是一种半形式化的建模语言，在 UML 活动图模型中，控制流信息得到了明确的表达，但是却没有明确地规定应当如何描述活动中所涉及的各种输入、输出，以及相关的约束条件。然而这种描述对于测试数据的设计和生成则是最基本的和必需的。为此，我们引入了针对活动的测试剖面（Test Profile for Activity）概念，用以描述活动的输入输出数据和相关的测试约束条件。

测试剖面是指被测软件在执行到某个特定状态时，其自身应当满足的、或者外界应当提供的某种测试条件。从测试用例设计与生成的角度讲，我们更关心的是外界（如用户）的输入条件，例如，当程序执行到某个交互活动时，或者在某种特定的测试场景下，备选测试数据应满足的约束条件。

对应于活动图的层次结构，测试剖面被分为两种类型。一种是针对整个活动图的，称为“活动图测试剖面”；另一种是针对单个活动的，称为“关于活动的测试剖面”。每个活动图都有一个活动图测试剖面说明，以及多个关于活动的测试剖面说明。

一般而言，针对活动图模型的测试规格说明应当包含由用户定义的测试相关信息，

为测试用例的生成提供必要的说明。对应于活动模型的“层次型”结构特点，活动图模型的测试规格说明应当包含针对各个活动图的测试剖面说明，而活动图的测试剖面说明又包含一组针对活动的测试剖面说明。

活动图测试剖面主要描述整个活动图的测试相关信息，包括测试覆盖准则、测试策略以及对并发活动的测试约束等。关于活动的测试剖面则描述该活动的相关测试信息，包括该活动所涉及的输入（包括操作）输出数据描述、前置条件、后置条件、不变式（主要用于检查测试结果），以及执行频度权值等。

这种层次化的测试剖面结构与活动图模型之间的关系可以用图 4-2 来表示。

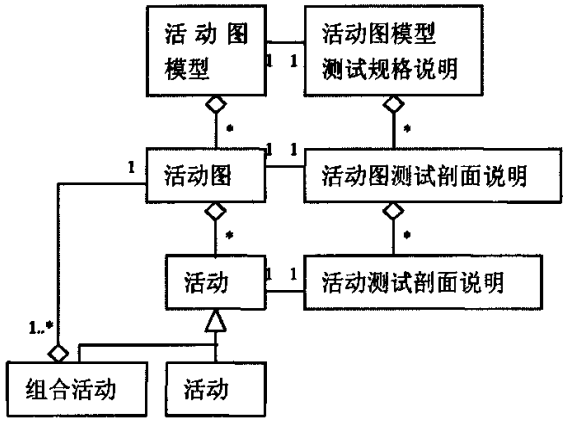


图 4-2 UML 测试剖面与活动图模型的类关系图

4.5.2 结构化活动图模型的方法

如前所言，活动图中存在对象流、并发模块以及循环等特征，这些复杂的非结构化特征增加了测试用例生成的难度。为此，本文给出了相应的处理方法。

对象流处理

在 UML 活动图中，对象所表示的是一个活动的输入或输出[17]，对象流表示的是一种数据流关系。为此，我们需要将对象流转化为活动的输入输出信息，并将活动之间通过对象流表达的隐式控制流显式化。

首先将 4.1.2 节中关于对象流的形式化定义进一步修改和细化：

$DataFlow_D=\{DataFlowIn_D, DataFlowOut_D\}$

$DataFlowIn_D$：(A_D，$FlowInLabel_D$)→$Object_D$

$DataFlowOut_D$：($Object_D$，$FlowOutLabel_D$)→ A_D

对于一个包含对象流的活动图 AD，对象流的处理规则如下：

对每个 $Object_D$ ∈AD，执行下列过程：

（1） 获得与该 $Object_D$ 相关的 $DataFlow_D$ 集合。

（2） 对集合中的每一对 $DataFlowIn_D$：($A1_D$，$FlowInLabel_D$)→$Object_D$ 和 $DataFlowOut_D$：($Object_D$，$FlowOutLabel_D$)→ $A2_D$，增加一个从活动 $A1_D$ 到活动 $A2_D$ 的迁移 $InformationFlow_D$:($A1_D$,$FlowLabel_D$)→$A2_D$，其中 $FlowLabel_D$ 为 $FlowInLabel_D$＋$FlowOutLabel_D$ 。

（3） 对于 $DataFlowIn_D$，将 $Object_D$ 作为 A_D 的输出；对于 $DataFlowOut_D$，将 $Object_D$ 作为 A_D 的输入。

（4） 删除与该 $Object_D$ 相关的 $DataFlow_D$ 集合。

在测试中，针对对象流的测试主要是检查和验证活动开始之前和结束之后对象状态是否正确。因此，在输出和接受该对象为输入的活动中应当分别在出口条件和入口条件中说明针对该对象的约束条件。

循环的处理

循环是大多数软件算法实现的重要部分和软件使用的常见过程。但是，在软件测试中却很少注意到他们。循环可以分为四种：简单循环、串接循环、嵌套循环和不规则循环[25]。如图 4-3 所示：

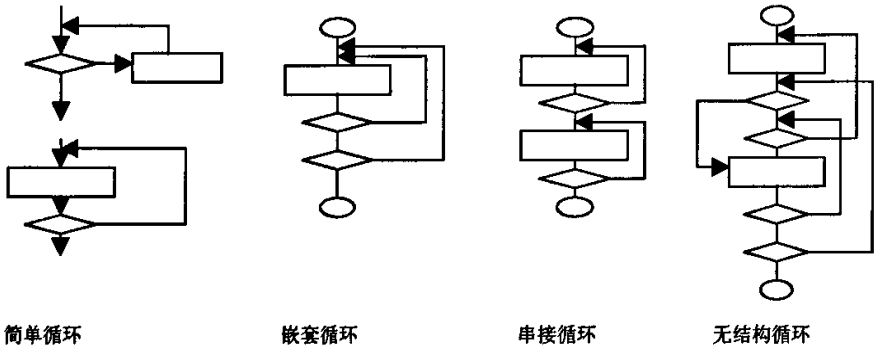


图 4-3 循环类型

对于简单循环，可以采取下列测试用例集（其中 n 是允许通过循环的最大次数）：

（1） 整个跳过循环。

（2） 只有一次通过循环。

（3） 两次通过循环。

（4） m 次通过循环。

（5） n-1, n, n+1 次通过循环。

对于嵌套循环，如果要将简单循环的测试方法用于嵌套循环，可能的测试次数就会随着嵌套层数成几何级数增加，这将导致不实际的测试数目。为此本文提出了一种减少测试数的方法：

（1） 从最内层循环开始，将其它循环设置为最小值。

（2） 对最内层循环使用简单循环测试，而使外层循环的迭代数为最小值。

（3） 由内到外构造下一个循环的测试，但其他的外层循环为最小值，并使其他的嵌套循环为典型值。

（4） 继续直到测试完所有的循环。

对于串接循环，如果串接循环彼此独立，可以使用嵌套循环的策略测试串接循环。如果循环不独立，推荐由嵌套循环的方法进行测试。

对于不规则循环，我们要求尽可能将这类循环重新设计为结构化的循环结构。

并发模块的识别

图 4-4 是一个包含并发模块的活动图模型。

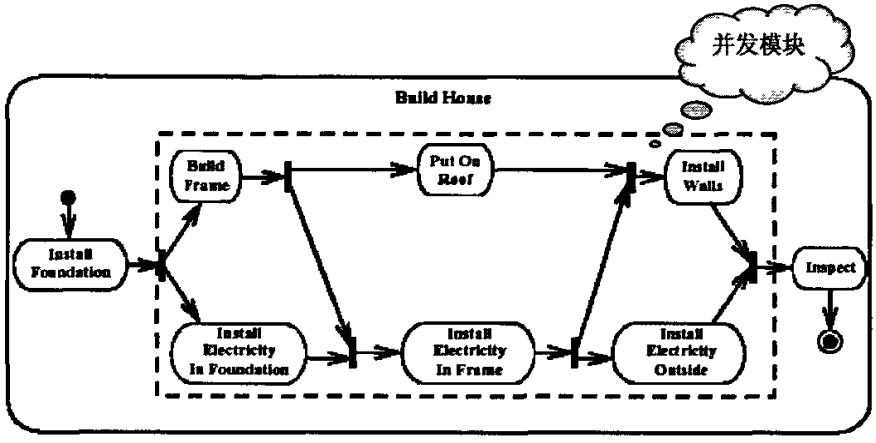


图 4-4 带有并发模块的活动图

在识别活动图模型的并发模块时，我们根据 UML 活动图模型中并发模块的特点，通过修改图论中的宽度优先搜索算法来实现。

首先分析并发模块的特点。并发模块中包含了并发 fork 和并发 join 结点。并发 fork 结点有一个入迁移和多个出迁移，而并发 join 结点有一个出迁移和多个入迁移。每一个并发模块都由一个并发 fork 结点开始，以一个并发 join 结点结束。这样，每个并发模块可以看作一个黑盒子，仅有一个入口和一个出口。因此，利用宽度优先搜索算法的思想，很容易找到并发 fork 结点对应的并发 Join 结点。具体算法可参看第五章 5.2.1 节。

4.5.3 生成测试场景

本节将介绍测试场景生成的基本方法和并发模块的实例化方法。

基本方法

为满足前面给出的测试覆盖准则，本文给出了测试场景生成算法，基本方法如下：

（1） 结构化活动图模型，识别出活动图中的对象流、消息流、并发模块以及循环。

（2） 从活动图模型的初始结点开始，对结构化的活动图模型进行深度优先遍历（控制点覆盖、转移边覆盖）。

（3） 处理活动图模型中的并发模块、循环以及分支。对并发模块生成实例化并发

场景（下一节详细介绍）；对循环结构，采用前面的处理技术生成确定次数的循环；对于分支结构，则进行分拆并复制已有的场景。（逻辑路径覆盖）

（4） 根据信号流覆盖准则，处理活动图中的信号流。

（5） 对于已经生成的场景集合，某些场景中可能有用户输入动作。在这些场景中，我们将为这些输入变量生成各自的代表值（包括合法、非法和边界值）并采用“改进的轮转算法“进行组合。（代表值覆盖）

并发模块实例化方法

并发模块中存在运行时的不确定性，因此在测试时需要进行实例化，以保证测试的覆盖率和测试效率。针对并发模块实例化中可能存在的组合爆炸问题，在[27]一文中进行了初步探索，并基于用户定制的策略给出了一种实例化方法－“填空法”。该方法具有一定的理论意义，但难于实现且缺乏实用性。本文对并发模块实例化进行了深入分析和探索，给出了一个更加完善和实用的解决方案，称之为“随机生成过滤法”。

为了避免并发实例化过程中的组合爆炸问题，我们将根据并发活动之间的信息交互，以及用户定义的并发活动约束描述，来完成并发活动的实例化。并发活动之间的约束可以分为下面几种：

（1） 位置约束关系：A>B>C，表示 A、B、C 的先后关系；AB 表示 A 之后紧接着就是 B 活动；~(AB)，表示 AB 不能紧密相邻。

（2） 数据依赖关系：活动之间存在数据的定义使用关系。

（3） 信息交互依赖关系：表示活动间存在消息发送关系。

（4） 主要测试框架：用户认为测试只需要覆盖主要的并发活动框架，其它活动随机插入主要测试框架中即可。

定义好并发活动间的约束类型之后，我们来分析并发模块实例化的特点。我们认为并发模块实例化具有如下难点和特点：

（1） 由于组合爆炸问题，对于复杂的大规模并发模块，要生成所有可能的并发实例化场景是不可行的。

（2） 在对并发模块进行实例化时，用户可能会定义复杂的约束条件。

（3） 并发模块本身的结构也可能相当复杂。

（4） 对于测试者而言，只要生成满足用户给出的约束条件和实例化场景数目即可，并不需要生成所有可能的并发实例。

为此，我们认为采用图论算法中传统的深度优先搜索或者宽度优先搜索算法生成测试用例是不可行的。在本文中，我们提出一种随机生成过滤的方法，来对并发模块进行实例化。下面结合例子来介绍并发模块实例化方法 —— “随机生成过滤法”。

对于并发模块中的每一个结点，我们都定义了两个值，分别为 TW, W。W 是该结点随机生成的权值，而 TW 则是该结点的计算权值。

随机生成过滤法如下：

（1） 对每个结点，采用随机数生成算法赋予 W 一个随机值。

（2） 计算并发模块中每个结点的 TW 值，$TW(A) = W(A) + \max\{ TW(A_{out}) \}$。其中，A 表示并发模块中的结点，$\max\{ TW(A_{out})\}$ 则表示 A 结点的所有出迁移目标结点中 TW 的最大值。

（3） 处理分支。利用类似于宽度优先的轮转法，优先选择当前访问次数最少的一个分支。

（4） 将参与场景的所有结点按 TW 值从大到小进行排列，即可得到一个并发模块的实例。由于 TW 的计算方法，必然保证了每个并发模块实例化场景的正确性。

（5） 如果用户有条件约束，则根据条件检查场景实例是否符合要求，并进行过滤。

（6） 对上述过程循环多次，总可以得到满足约束条件的指定数目的实例化并发场景。

下面给出一个例子（见下图 4-5），以说明并发模块的一个实例化场景生成过程。

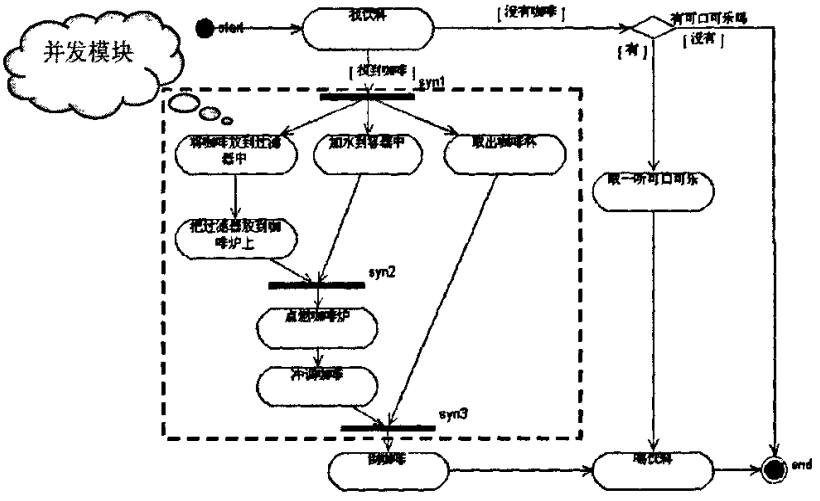


图 4-5 活动图模型—Drink Coffee

（1） 为并发模块的各结点自动生成 W 值，并计算 TW 值，如下表 4-1 所示。

表 4-1 并发模块中各结点的 W 与 TW 值

结点名称	W	TW	结点名称	W	TW
sync1	95	1129	sync2	21	624
将咖啡放到过滤器中	108	786	点燃咖啡炉	302	603
加水到容器	410	1034	冲调咖啡	100	301
取出咖啡杯	504	705	sync3	201	201
把过滤器放到咖啡炉上	54	678			

（2） 由于这里的并发模块没有分支，因此并发场景包括上述九个结点。按照 TW 值排序可得到一条随机生成的实例化场景：sync1—加水到容器—将咖啡放到过滤器中—取出咖啡杯—把过滤器放到咖啡炉上—sync2—点燃咖啡炉—冲调咖啡—sync3。

（3） 如果用户有约束条件，则根据约束条件来判断该场景是否符合约束条件。如果不符合，则被过滤掉。

“随机生成过滤法“是一个不断循环的过程，具有简单实用的优点。

4.5.4 测试数据生成与组合方法

基于 UML 活动图的测试用例由测试场景和测试数据两个部分组成，因此，测试数

据的生成同样具有重要的意义。如何对 UML 活动图模型中的输入输出变量进行描述，以及如何根据这些信息生成相应的输入值，是本节要研究的重点。

输入输出数据描述

由于被测程序的输入输出数据的类型十分庞杂，而测试剖面定义中的其他信息描述则相对简单，因此，这里仅给出输入输出的详细描述方法。

当 UML 活动图模型用于描述系统或子系统级的工作流程时，常常会涉及用户与系统之间的交互行为。因此，我们将输入分为两种：数值型输入和用户操作；输出则仅包括数值类型。

Input = {Data, Operation}

Output = {Data}

一般而言，输入输出数据的描述可以分为三个部分：数据结构描述、数据值域约束描述和数据值域相关描述[26]，它们分别描述一个输入或输出数据量的数据结构及其每个数据项的值域约束关系，以及各个输入输出数据量之间的值域约束关系。但对应于被测软件的交互式执行过程，输入输出数据量之间还存在着某种“时序关系”，即活动之间的执行顺序。因此，针对活动图模型的特点，其输入输出数据的描述应当包含上述四个方面。表 4-2 列出了各部分的具体内容以及用途。

表 4-2 测试剖面描述中关于输入输出数据描述的基本框架

描述特征	详细内容	用途
数据结构描述	描述输入量的数据结构	测试数据生成
	描述输出量的数据结构	结果验证
数据值域约束描述	描述输入量的值域约束关系	测试数据生成。
	描述输出量的值域约束关系	结果验证
活动内部的数据值域相关性描述	描述输入量之间的约束关系	测试数据生成
	描述输出量之间的约束信息关系	结果验证
	描述输入输出量之间的约束信息关系	结果验证
活动间的数据值域相关性描述	描述输入量之间的“时序”约束关系	测试数据生成
	描述输出量之间的“时序”约束关系	结果验证
	描述输入输出量之间的“时序”约束关系	结果验证

数据结构描述可以用两种方法表示，即 UML 类图和结构规范语言。数据值域约束描述和数据值域相关性描述则分别采用值域约束规范语言和值域相关规范语言来表示。

数据结构描述

从数据结构描述的角度来看，输入输出数据又可以分为复合类型数据和简单类型数据。复合类型数据可通过 UML 类图和结构规范语言来定义。对于 UML 类图定义的复合型数据，我们可以进行分析并转化成结构规范语言定义的数据，因此这里仅介绍如何用结构规范语言来定义复合类型的数据。

利用结构规范语言，各种复合数据类型可以通过下面四种基本结构进行构造：

（1） 组合结构：一个结构由若干个子结构组成，表示为 S：$\{S_1, S_2,...S_n\}$。组合结构类似于编程语言中的 Struct。

（2） 选择结构：一个结构可以是列举的几个结构中的一种，表示为 S: $S_1|S_2|...|S_n$。选择结构类似于编程语言中的 Union。

（3） 重复结构：一个结构可以是基于某一类结构的有限重复，表示为 S: $S_i[m..n]$。重复结构类似于编程语言中的 Array。

（4） 预定义的简单类型数据：即下面定义的 SimpleData。

下面采用扩展的 BNF 范式来描述数值型输入输出的数据结构：

<Data> ::= <Struct>|<Union> |<Array> | <SimpleData>

<SimpleData> ::= <IntData> | <LongData> | <RealData> | <CharData> | <StringData> | <BooleanData> | <EnumData> | <TimeData>

<Struct>::= "{" {<Data>","}$_1^n$ <Data> "}"

<Union> ::= {<Data>"|"}$_1^n$ <Data>

<Array> ::= <Data>" ["<LowerIndex>".."<UpperIndex>"]"

<LowerIndex> ::= <IntValue>

<UpperIndex> ::= <IntValue>

<IntData> ::= <intType>:<Identifier>

<LongData> ::= <longType>:<Identifier>

....

<intType> ::= int

<longType> ::= long

....

上面略去了一些简单数据类型的定义。此外，Identifier 和 IntValue 分别表示标识符和整数。

数据值域约束描述

由于复杂数据结构都由简单数据组合而成，因此，这里仅给出关于简单类型数据的值域约束描述：

（1） int，long, real 类型：int，long，real 类型的值域约束采用域界的形式，当只有一个连续值域时，表示为：[下界值...上界值]；当有多个连续值域时，表示为：[下界值 1...上界值 1，...，下界值 n...上界值 n]。此外，real 类型的值域约束描述中还应包括小数点后的精确位。

（2） char，string 类型：char 类型从本质上讲是 0x00~oxFF 之间的一个数，因此可以采用与 int 类型相同的值域描述方法。String 类型的值域约束包括三个方面的内容：字符串长度、字符串掩码格式、字符串可取值集合。字符串长度用上下界表示为：[下界长度...上界长度]（上界长度可以等于下界长度）。字符串掩码可以采用*,?来表示。例如“*.mdl”表示以“.mdl”结尾的所有字符串。字符串可取值集合列出了可取的字符串值，可用{String1, String2, ..., Stringn}来表示。

此外，还有一类特殊的输入量，即用户操作（UserOperations）。单独提出用户操作类型是为了更好地支持交互式软件的测试。用户操作（UserOperations）的描述由操作名称、操作目标和操作描述三个基本部分组成：

UserOperations = { OperationName, OperationTarget, OperationDescription}

其中 OperationTarget = {Menu, Button, Window, Toolbar...}，表示操作目标的类型及其名称。不同类型的操作目标包含着一组特定的操作集合。例如，Menu 的操作集合为{MoveOver, Click, ...}， Window 的操作集合为{Open, Close, Move, Resize, ...}。OperationDescription 为一个字符串，用于给出操作的具体描述。

数据值域相关描述

[26]提出了三种基本的值域相关规范，包括：集合相关规范；函数相关规范；条件相关规范。它们适合于描述单一活动中不同数据量之间的约束关系。集合相关规范支持集合的并、交、补运算，以及子集、真子集关系的判断。函数相关规范支持二元函数关系。条件相关规范则支持基本的逻辑关系运算符，包括>,<, =, >=, <=, !=关系。

对于活动之间的数据值域相关性，则需增加“测试场景相关规范”，用以描述这些数据值域与测试场景之间的相关性关系。测试场景相关规范用以描述和限定测试场景或测试场景子集合与特定数据量之间的关联性，即描述在哪些特定的场景下，一个数据量是否可以取值于某个特定的值域。作为缺省，在任何测试场景中，数据量可以取值于其任何值域。

测试数据生成与组合策略

复合类型变量的存在以及测试场景与测试数据的不可分割性，决定了测试数据生成与组合方法分为三个层次：第一个层次是简单类型测试数据的生成方法；第二个层次是复合类型测试数据的生成，它建立在第一个层次的基础上；最后一个层次是测试数据与测试场景的结合方法，它建立在前两个层次基础之上。

基本域测试数据生成策略

基本域是指一个输入量的基本值域单位，即在任何测试场景中，基本域中的所有值均被认为是等效的。输入量的基本域包括合法基本域、非法基本域和边界基本域，它们分别是由彼此等效的合法值、非法值和边界值组成的基本域。

为简单类型的变量生成测试数据时，我们结合传统黑盒测试中所采用的等价类划分和边界值方法，首先为各输入数据生成合法等价类值域、非法等价类值域和边界等价类值域。然后，每当生成一个新的测试用例时，再从相应的值域中（随机地或依据一定的策略）选择一个代表值。

单个复合类型变量的测试数据生成算法是一个综合递归的过程。通过遍历变量数据结构描述中给出的各数据项，并参照变量的值域约束描述、活动内部的数据值域相关性描述以及活动间的数据值域相关性描述，动态生成测试数据。

测试数据组合方案

单个变量的测试数据生成后，需要针对特定场景，为其中所涉及到的每个输入变量生成适当的测试数据，从而与测试场景相结合产生一组测试用例。

结合边界值分析和等价类划分的思想，我们提出一种改进的轮转法用于测试数据的组合。改进轮转法的组合规则如下：

（1） 场景中所有变量的基本域都要覆盖到。

（2） 组合时，首先利用轮转法对所有变量的合法基本域进行组合，保证所有合法基本域都覆盖到。

（3） 测试变量的非法基本域时，结合等价类划分时候的思想，一次只选择一个变量的非法基本域，其它变量都选取合法基本域。

（4） 边界基本域的测试与非法基本域一样进行，一次只选择一个变量的边界基本域，其它变量都选取合法基本域。

（5） 选择变量的合法基本域时，如果某个变量的所有合法基本域都已经覆盖，则按顺序（或随机）选择合法基本域中没有使用的值。

表 4-3 中用一个例子对这种方法进行了说明。假设有三个变量，其类型和测试数据基本域如下表所示。

表 4-3 输入变量值域划分表

变量名	类型	基 本 域		
		合法基本域	非法基本域	边界基本域
nCount	Int	[2,9] [61,99]	[-65535] [42] [65535]	[1] [10] [60] [100]
fTemp	Real	[-99.99, 99.99]	[-100.01] [100.01]	[-100.00] [100.00]
bFlag	Boolean	[true] [false]		

利用改进的轮转法进行组合后，测试用例集共有 13 个，如下：

TCSets(nCount, fTemp, bFlag) =

{ (2, -99.99, true), (61, 99.99, false), (-65535, -99.99+Δ, true), (42, 99.99-Δ, false), (65535, -99.99+2*Δ, true), (1, 99.99-2*Δ, true), (10, -99.99+3*Δ, false), (60, 99.99-3*Δ, true), (100, -99.99+4*Δ, false), (9, -100.01, true), (99, 100.01, false), (9-Δ, -100.00, false), (61+Δ, 100.00, true) }

其中Δ称作微调量，是一个适当的微小增量，保证上述取值均在规定的基本域内。

改进的轮转法具有如下优点：

（1） 覆盖率高。该方法覆盖了合法域、非法域和边界域，综合了等价类划分和边

界值分析法的优点。同时，由于微调量Δ的引入，使得在测试数据生成过程中，一个基本域的出现频率越高，所选中的不同的值的数量也就越多，从而增大对此类数据的实际测试覆盖率。此外，采用改进的轮转法还保证了对非法域和边界域的充分覆盖。

（2） 测试用例的数量与变量数及其基本域的数量呈线性关系。假设一个特定的测试场景 CS_k 中包含有 n 个输入变量，记为 $V_1, V_2,...,V_n$，每个变量的合法基本域数目、非法基本域数目和边界基本域数目表示如下：

$NV_i = N(VF_i)$ 变量 V_i 的合法基本域数目，VF_i 是合法基本域集合

$NI_i = N(IF_i)$ 变量 V_i 的非法基本域数目，IF_i 是非法基本域集合

$NE_i = N(EF_i)$ 变量 V_i 的边界基本域数目，EF_i 是边界基本域集合

则采用改进的轮转算法所生成的测试用例集的数目为：

$$N_c(CS_k) = Max(NV_i) + \sum_{i=1}^{n} NIi + \sum_{i=1}^{n} NEi$$

由于活动中的这些变量可能被多个测试场景所使用，因此，对于所有测试场景，其测试用例集的数目上限为：

$$N_c <= \sum_{k=1}^{m} N_c(CS_k)$$

其中 m 为测试场景的数目。

4.6 本章小结

本章首先分析了活动图模型适合的范围，并给出了活动图模型的形式化描述。接着，详细论述了基于活动图模型的测试用例生成方法，包括测试剖面的定义，活动图模型的结构化，测试场景的生成，测试数据的描述、生成与组合方法等。

第 5 章 测试用例生成系统的设计与实现

上一章我们详细论述了基于 UML 活动图模型设计和生成测试用例的方法。本章将介绍基于此方法的测试用例生成系统的设计实现。下面将介绍系统的整体结构及关键算法，并给出相应的实例。

5.1 系统结构与组织

测试用例生成系统选择 Rational Rose 作为分析和设计面向对象软件系统的可视化建模工具，并通过插件形式与 Rational Rose 集成，在后端则将“软件测试过程控制平台”作为测试大纲与测试用例库的存储和管理平台，从而最大程度上方便软件测试者使用和管理测试大纲和测试用例。

系统的逻辑结构

本系统的逻辑结构图如图 5-1 所示，该图给出了系统与使用者及外界系统的关系。

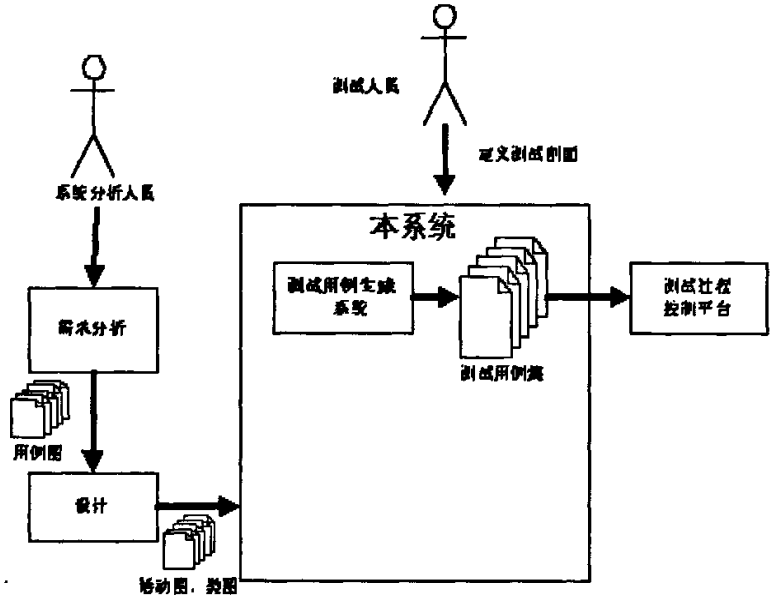


图 5-1 系统逻辑结构图

活动图模型是本系统的输入。系统分析和设计人员在建立起活动图模型后，测试人员首先对活动图模型进行完整性与一致性检查，然后利用本系统定义测试剖面以描述被

测系统的用户输入和操作等测试相关的约束信息，进一步生成测试用例的集合。测试大纲与测试用例集存放在“软件测试过程控制平台”（Notes 库）中，以方便测试人员的管理和使用。测试用例生成系统所生成的测试大纲与测试用例集可以用于指导测试，对软件模型本身进行检查，还可以用于对程序进行走查。

系统的实现机制

本系统以插件的形式与 Rational Rose 工具集成，后端以“软件测试过程控制平台”作为测试用例库的管理平台，使得软件开发、测试用例设计生成与测试实施管理等工作紧密集成[27]。图 5-2 是本系统的实现机制。

图 5-2 系统实现机制

由于 Rational Rose 是目前被普遍使用的 UML 可视化建模工具，而且 Rational Rose 的可扩展接口 REI 不仅可以用于提取模型信息，同时也提供了与其它工具集成的机制。为此，我们选择 Rational Rose 作为 UML 的建模工具。

Rational Rose 提供对插件（Add－In）的支持，其企业版能在一个 Rose 对话中具有多个独立、激活的插件。为了增强可扩展性，Rose 提供了一套完整的接口，可供用户

通过使用 Rose 自动化工具访问当前模型。Rose 的扩展接口(REI)共提供了 105 个接口类[28]，这些类既包括各种模型图的信息，也包括 Rose 平台的一些设置信息。

作为 Rose 的一个插件（Rose Add-in），本系统通过 Rational Rose 的可扩展接口 REI 抽取 UML 活动图模型数据信息，并根据测试人员定义的测试剖面，设计并自动生成测试大纲、测试用例集合。测试大纲以及测试用例集合将通过 Lotus Notes 的应用程序接口 API（Lotus C++ API）输出到测试过程控制平台 Notes 库中，供测试人员和质量保证人员使用。此外，系统还调用了 Matalab Statistical Toolbox 的随机数生成函数来生成测试数据。

系统包图

从实现的角度来看，本系统可以分为三个层次，其系统包图如图 5-3 所示：

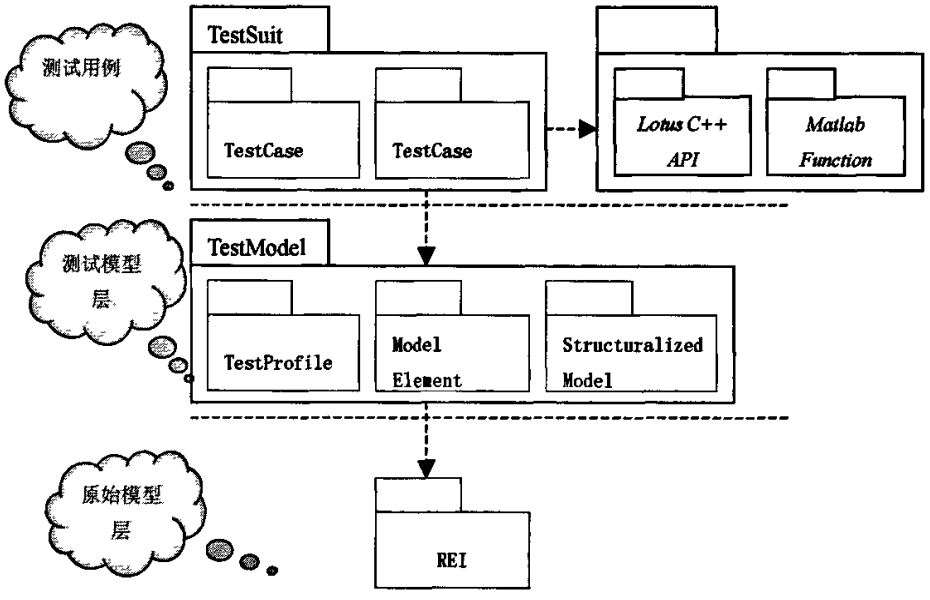


图 5-3 系统包图

这三层从下往上分别是：

（1） 原始模型层：在这一层中，活动图的原始模型信息存放在 mdl 文件中，我们可以通过 Rational Rose 的可扩展接口获得。可以认为这是最底层的原始模型数据信息。

（2） 测试模型层：为了根据 UML 活动图模型生成测试用例，必须对原始的活动图模型进行处理，包括活动图模型结构化与测试剖面定义两个部分。

TestProfile 即为测试剖面，其中包括了与活动图模型测试剖面相关的类。StructuralModel 是经过结构化处理之后的模型，包括对象流矩阵、并发模块、循环信息等。ModelElement 则记录了模型中各种元素的具体信息，包括结点（活动、对象、状态、决策、并发等）、迁移等信息。StucturalModel 包依赖于该包。

（3） 测试用例层：包括 TestCase 包。这个包用于根据测试模型（包括测试剖面定义和结构化的活动图模型）来生成测试用例。测试用例的生成，需要调用 Lotus C++ API 和 Matlab Function 的相关类和函数。

下面我们将要介绍的测试用例生成系统主要集中在上面两层，即测试模型层与测试用例层，具体内容包括结构化活动图模型、定义测试剖面、生成测试用例等。

5.2 测试用例生成系统的实现

在测试用例生成系统中，我们将着重介绍结构化活动图模型、定义测试剖面、生成测试数据等部分的关键算法。最后将给出具体实例以展示本系统的功能。

5.2.1 结构化活动图模型的实现

循环识别

为了识别循环，我们借用了有向图深度优先搜索算法中的几个概念[29]。

在有向图的深度优先搜索算法中，一条未检查的弧(U,W)可以按照下列四种情况，分为如下几类：

（1） W 是未访问过的结点，（U,W）归为树枝弧（见图 5-4a）

（2） W 是 U 的已形成的 dfs 森林中直系后代结点，则(U,W)称为前向弧（见图 5-4b）。无论（U，W）是树枝弧还是前向弧，dfn(W)>dfn(U)。细分一下，树枝边总使得搜索导向一个新的未访问的结点，且 dfn(U)+1 = dfn(W)。

（3） W 是 U 的已形成的 dfs 森林中直系祖先结点，则（U，W）称为后向弧（见图 5-4c）。

（4） U 和 W 在已形成的 dfs 森林中没有直系上下关系，并且有 dfn（W）<dfn（U），则称（U，W）是横叉弧。无论（U,W）是后向弧还是横叉弧，dfn(W)>dfn(U)（见图 5-4d）。

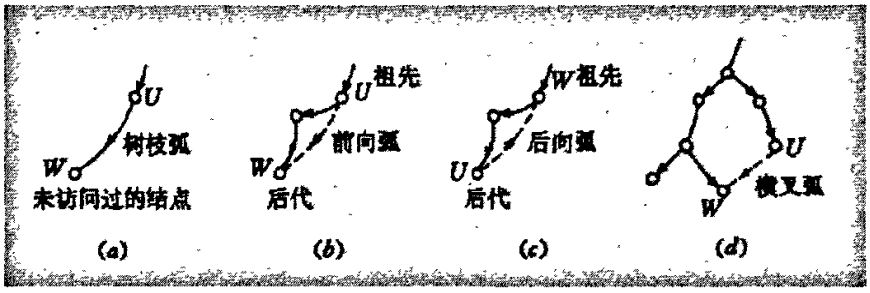


图 5-4 有向图的弧分类

从图论的角度来看，活动图模型是一种更加严格的有向图。经过分析，可以看出，构成循环的迁移实际上是有向图中的后向弧。因此，循环的识别可以通过深度优先搜索算法识别出后向弧即可。

目前，我们仅仅处理结构化的循环，包括简单循环、串接循环和嵌套循环。处理方法参看本文第四章。

对象流识别与处理

在 UML 活动图模型中，对象流不同于控制流，它表现了对象与动作之间的关系。对象可能是动作的输入，也可能是动作的输出。对象流中同时也隐含了一种控制流关系。

根据第四章中所给出的对象流处理规则，我们建立起对象和活动之间的关系矩阵。（这里，我们将同类名同对象名但不同状态名的对象记为不同的对象。）

设 AD 是一个 UML 活动图模型，其中 $A=\{a_1, a_2, \ldots, a_n\}$为结点集合，$O=\{o_1, o_2, \ldots, o_k\}$为活动图中对象的集合。我们称矩阵 $M_{n,k}=(m_{i,j})$为对象流矩阵，其中：

$$m_{ij}=\begin{cases}0, & \text{当 } o_j \text{ 不是 } a_i \text{ 的输入或输出对象}\\ 1, & \text{当 } o_j \text{ 是 } a_i \text{ 的输出对象}\\ 2, & \text{当 } o_j \text{ 是 } a_i \text{ 的输入对象}\end{cases}$$

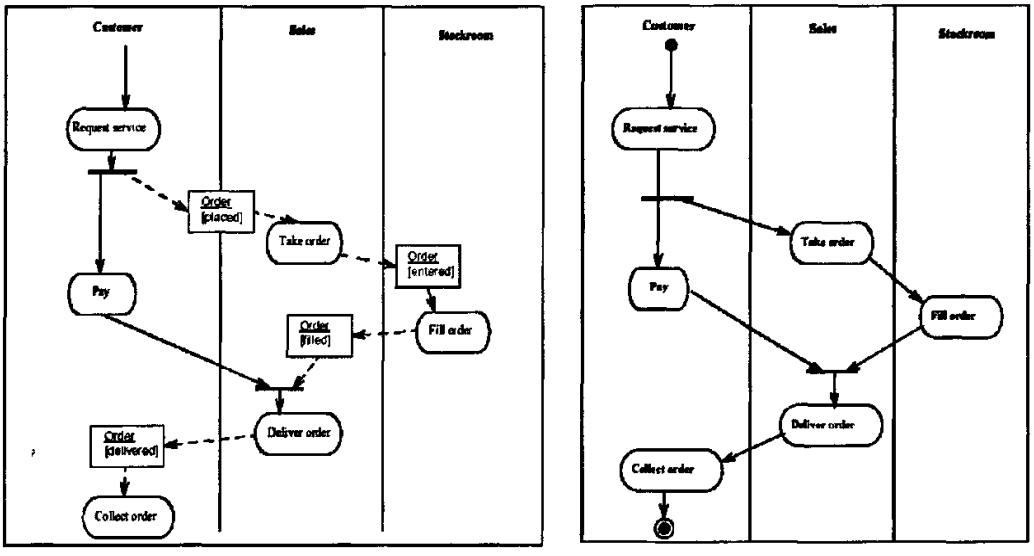
对象流的识别与处理过程如下：

（1） 识别出活动图中所有的对象流，建立起对象流矩阵 M。

（2） 将对象流转化成控制流。对每个结点 a_i, 对于所有使得 $m_{ij}=2$ 的 j，如果存在 $m_{sj}=1$, 则增加一条从 a_s 到 a_i 之间的控制流迁移；相反，对于每个结点 a_i, 对于所有使得 $m_{ij}=1$ 的 j，如果存在 $m_{sj}=2$，则增加一条从 a_i 到 a_s 的控制流迁移。

（3） 保留对象信息作为活动的输入输出。对于结点 a_i，如果 $m_{ij}=1$，则 o_j 是结点 a_i 的输出对象，在活动结束后可对 o_j 对象进行检查；如果 $m_{ij}=2$，则 o_j 是结点 a_i 的输入对象，在活动开始前对 o_j 对象进行检查。

图 5-5 的例子分别给出了对象流处理前后的活动图模型。



（a）包含对象流的活动图　　（b）处理之后的活动图

图 5-5 对象流的处理实例

并发模块识别

并发模块的识别采用了有向图的宽度优先搜索算法。正如第四章所言，并发模块有自己的特点，因此需要对有向图的宽度优先搜索算法进行修改。

每一个并发模块都由一个并发 fork 结点开始，以一个并发 join 结点结束。并发 fork 结点有一个入迁移和多个出迁移，并发 jion 结点有多个入迁移和一个出迁移。因此，每个并发模块可以看作仅有一个入口和出口的黑盒子。

首先定义算法中涉及到的关键数据结构：

CStringList　listSVID：　并发模块的结点列表

CStringArray　asTransID：　并发模块中的迁移列表

int　nPathCount：目前活动图中的出迁移数和

识别算法如下：

（1） 从活动图模型的开始结点起，利用宽度优先搜索算法遍历活动图模型的结点。若结点为并发模块的开始结点，则标识着一个并发模块的开始。将该节点加入到 listSVID 的尾部。

（2） 初始化处理：将并发模块开始结点的所有出迁移加入到 asTransID 中，设 nPathCount = 结点出迁移数目。

（3） 循环处理。

while(listSVID 非空) && (nPathCount != 1)

{

//1 获得当前要处理结点（listSVID 的头结点）

// 2 判断结点的所有入迁移是否访问过，若还有未被访问，置 flag = false

// 3 若 flag == false, 则表明结点还有入迁移没有访问到。

将结点移到 listSVID 的尾部，推迟处理该节点。

// 4 如果 flag == true, 表明该结点的所有入迁移都已经访问过了。

将结点的所有出迁移放入 asTransID 中。

将出迁移的 target 放入到 listSVID 尾部（保证不重复）

修改 nPathCount = nPathCount – 入迁移数＋出迁移数

}

（4） 当循环结束时，必然有 nPathCount =1, 并发模块的识别完成。当前的结点就是并发模块的并发 Join 结点。

利用该算法，可以识别出活动图中的所有并发模块。

5.2.2 定义测试剖面的实现

在第四章中，我们说明了引入 UML 活动图测试剖面的原因，并给出了 UML 活动图模型的测试剖面定义及逻辑结构。本节我们将给出定义测试剖面的具体实现。

测试剖面的描述

在表示和存储测试剖面时，我们选择了 Rose 模型文件的 petal 格式。

为什么采用 petal 文件的格式，而没有采用 XML 或者 OCL 语言？有如下几点原因：

（1） 灵活性和完整性：首先必须要能够完整地定义测试剖面中的所有信息；其次，还应该具备一定的灵活性，以方便测试剖面定义中元素的增删改查，以及将来的可扩充性。由于 OCL 只是一种约束语言，可以对模型中的元素提供一种更加严格的约束，而且主要用于 Invariant 和 pre/post 约束信息，因此并不能用于表示所有这些测试相关的信息。

（2） 可集成性：petal 格式可以很好的表示和保存这些信息，此外 Petal 格式本身是 Rose 模型的文件格式，因此，如果用 Petal 格式定义测试剖面信息，可以很好地与 Rose 模型文件集成，这一点 petal 格式要优于 XML 格式。而且，很容易将 petal 格式转换成 XML 格式。

Petal 语法分析

Rational Rose 的模型文件（.mdl）是一种 ASCII 码文件，其格式为 Petal，但它的格式并没有正式文档记载。根据[30]的分析，Petal 格式大体类似于 lisp 数据结构，通过圆括号（）嵌套多个层次，从而形成一颗结点树。该结点树的主要数据结构为 Object。例如：

```
(object Petal
    version     44
    _written    "Rose 7.1.9642.27"
    charSet     134)
```

每个 object 都有一个 name，此处是“Petal”，以及其它一些属性。Object 的两个常用属性是 quid 和 quidu，分别用于表示 object 的唯一标识符以及该 object 所引用（或相关联）的其它对象的唯一标识符。

下面给出 Petal 文件的部分语法：

（1） Petal Node：Petal 文件定义了如下几种类型的结点。

```
<PetalNode> → <Object>|<Literal>|<List>
<List>→ (list <name> <Object>*)
<Literal> → <Value>|<Tuple>|<Tag>|<Location>|<stringliteral>|<int>|<boolean>|<float>
<Value> → (value <name><stringliteral>)
 <Tuple>→ (<String><int>)
<Tag> → @<int>
<Location> → (<int>, <int>)
```

除了 Object 和 List 外，其它结点都是叶结点。

（2） Object

Object 具有一系列的属性，如 name, tags，quid 等，也可以包含其它的结点对象。

```
<Object> → (Obejct <name> <string>* [<Tag>]
            (<name><PetalNode>)*)
```

（3） Literals：字面常量

petal 语法中定义了一些标准字面常量，包括 string， number, hexnumber, digit, int, float, boolean, char 等，这里不再赘述。

下面给出 Rose 模型文件的语法（Petal 格式）。

```
<petalfile> → <petal><design>
<petal> → (Object petal
          version <number>
          _written <ident>
          charSet <number>)
<design> → (object Design "Logic View"
            is_unit TRUE
            is_loaded TRUE
            quid <ident>
            defaults <defaults>
            root_usecase_package <UseCasePackage>
            root_category <LogicalCategory>
            root_subsystem <SubSystem>
            process_structure <Process>
            properties <properties>
            )
```

定义测试剖面的实现

根据前面给出的测试剖面逻辑定义、Petal 语法以及 UML 模型文件，我们首先给出定义测试剖面的实现层类图，如图 5-6 所示。

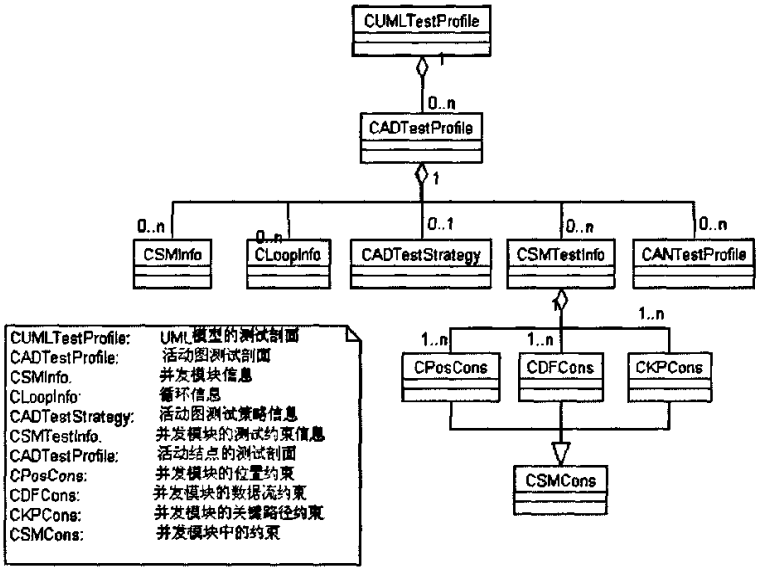


图 5-6 测试剖面的实现层类图

对测试剖面的实现层类图，我们给出如下几个关键类的说明：

CUMLTestProfile：用于描述整个 UML 模型的测试剖面。由于 UML 模型中存在各种不同的模型，如活动图、状态图、用例图等，这些模型可能会有自己的测试剖面定义。因此，CUMLTestProfile 中包括多个图的测试剖面。目前我们仅定义了 UML 活动图的测试剖面，以后还可以针对不同的图进行扩充。

CADTestProfile：用于描述 UML 活动图模型的测试剖面。UML 活动图模型的测试剖面中包括如下信息：CSMInfo，活动图模型中的并发模块信息；CLoopInfo，活动图模型中的循环信息；CADTestStrategy，基于活动图模型生成测试用例的策略信息，包括测试覆盖准则、测试数据组合方法等；CSMTestInfo，并发模块的测试约束信息，包括并发模块中用户指定的结点之间的约束条件等；CANTestProfile，活动结点的测试剖面定义，包括该活动的权值、活动的输入输出信息、活动的前置条件后置条件不变式等信息。

由于空间的原因，我们将基于 Petal 格式的测试剖面具体定义以及相关实例放在了

附录中。

用户界面设计

为了便于用户使用，我们采用了“测试剖面的用户操作界面 ↔ 测试剖面对象树 ↔ 测试剖面文件”的三级转换策略。用户可以通过测试剖面定义界面指定测试剖面元素的相关信息，程序将这些信息构造成一棵测试剖面对象树，并保存到测试剖面文件中（Petal 格式）。以后进行测试时，可以从测试剖面文件中提取这些信息。

图 5-7 是定义测试剖面的用户界面，包括活动图测试信息、并发模块测试信息、结点测试信息三个部分。

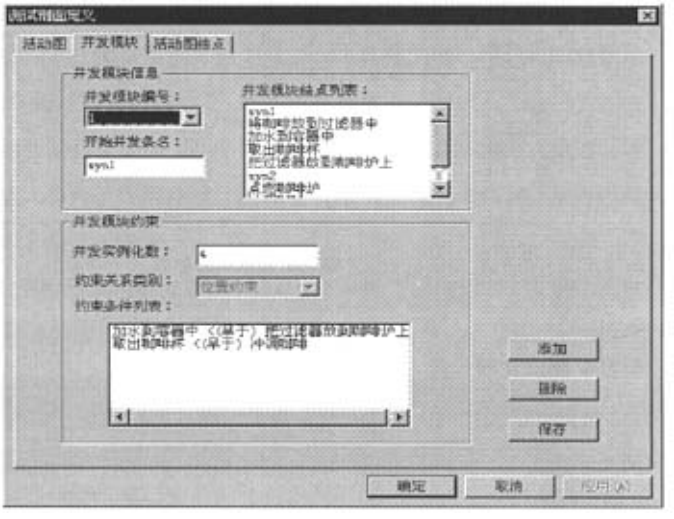


图 5-7 测试剖面定义界面

5.2.3 生成测试数据的实现

我们采用 Matlab Statistic ToolBox 来生成各种分布的测试数据。

Matlab 是美国 MathWorks 公司的产品，是一种以矩阵为基本编程单位的高效数值计算语言。它已经被证明是在应用数学、物理、工程学和其它设计复杂数值计算等应用领域中解决问题的优秀工具。C、C++等高级程序设计语言在数值处理分析和算法工具等方面，效率远远低于 Matlab 语言；并且本课题所需要的随机数生成函数已经在 Matlab 的 Statistic Toolbox 中有了完备的实现，所以我们直接可以通过 Matlab 和 C、C++的接口调用其函数，从而节省时间和精力去完成课题其他更重要的部分。

Matlab 中的随机数发生器主要有 beta 分布随机数、指数分布随机数、泊松分布随机数、正态分布、离散型均匀分布、连续型均匀分布随机数[31]等等。本课题利用 Matlab

中的随机数生成器，为简单类型的输入变量生成随机的合法、非法和边界值。

图 5-8 是所生成的测试数据结果片断。

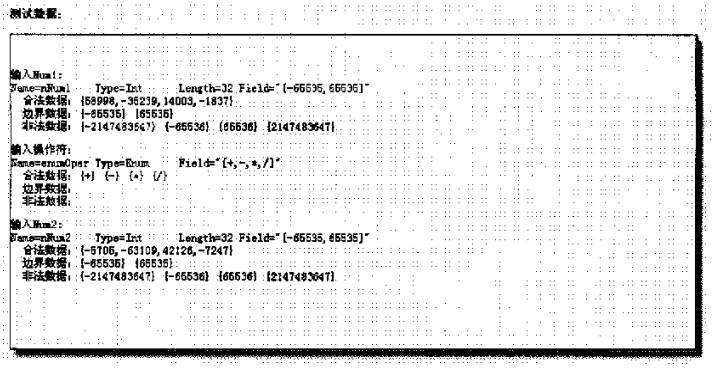


图 5-8 测试数据生成

数据的组合则根据第四章中给出的改进轮转法进行。

5.2.4 测试文档设计

最后介绍测试用例生成系统中测试大纲与测试用例文档的设计。图 5-9 给出了相应的类示意图。

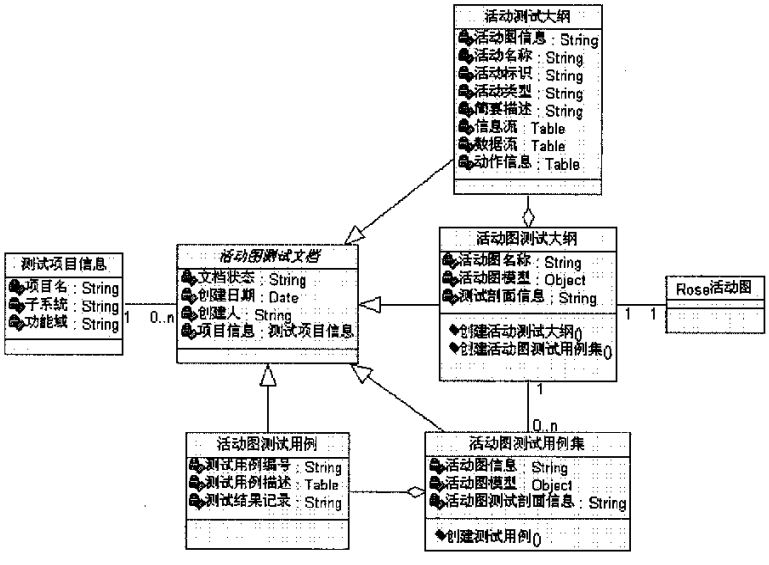


图 5-9 测试文档类图

在测试过程控制平台中，每类文档的创建都使用系统提供的专用模版。测试大纲与

测试用例模版是通过对 UML 活动图和图中活动的特性分析，以及大量已有实例总结而得到的。

测试大纲包括活动图测试大纲和活动测试大纲。测试用例文档则包括活动图测试用例集文档和活动图测试用例文档。根据特定的测试覆盖准则，每一个活动图模型都可以生成一个测试用例集文档，其中包含了多个测试用例文档。测试用例文档主要用于指导软件测试执行人员直接进行测试。测试用例文档中给出一个具体的测试用例，即测试场景与测试数据的合成。测试场景指导测试执行人员怎样执行每一步测试，并给出了相应输入数据，以及其它测试相关的信息，如某一个操作的前置条件后置条件等，这些，可以使得测试更加完善。测试人员可以使用测试大纲和测试用例文档指导测试，也可以用于代码走查，以及需求分析和设计的检查。

本系统所生成的测试大纲和测试用例文档均以 Lotus Notes 文档的形式，提供给测试设计和执行人员使用。其界面如图 5-10 所示：

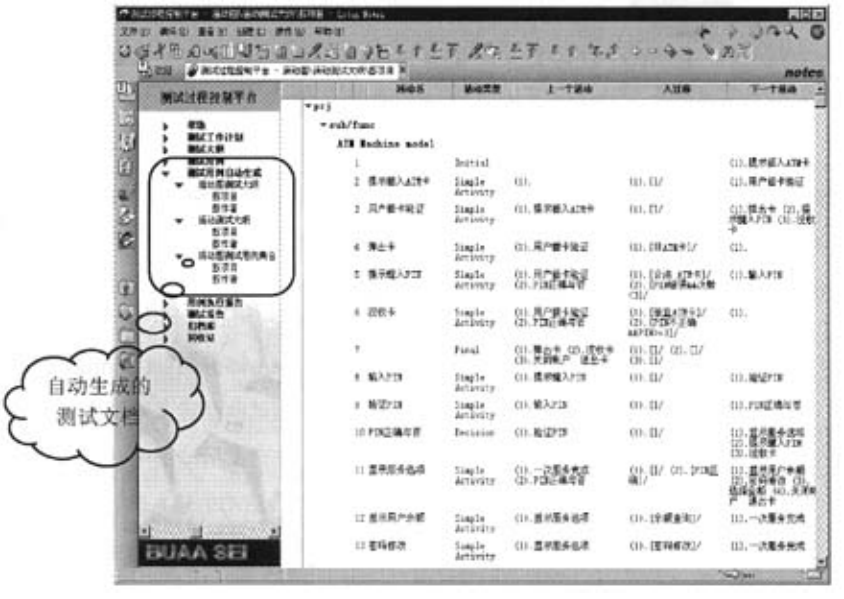


图 5-10 与测试过程控制平台集成的测试文档

5.3 简单实例介绍

本节将通过三个例子来介绍测试用例生成系统。

（1） Drink Coffee 例子

首先结合 Drink Coffee 的例子（如图 5-11 所示）来介绍该系统的使用流程：

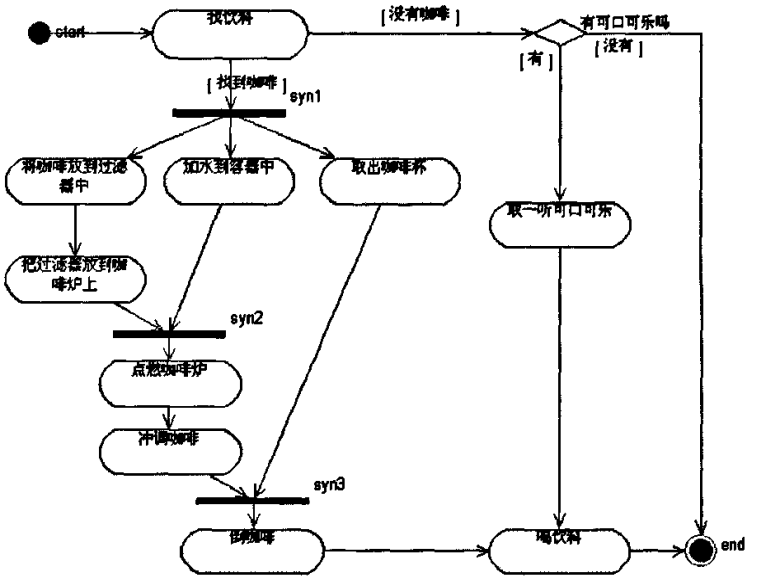


图 5-11 喝咖啡—测试用例生成系统示例

（1） 启动 Rational Rose，打开要测试的活动图模型文件 Drink Coffee。

（2） 定义活动图模型的测试剖面：选择 Tools→TestCaseAddIn→Test Config，弹出“测试剖面定义”对话框（如图 5-7 所示）。在测试剖面定义对话框中，用户选择活动图测试策略、测试覆盖准则等信息，并定义活动图中活动结点的测试剖面信息，包括活动结点的权值、活动种类、活动的输入输出等信息。如果活动图模型中存在并发模块，还可以定义并发模块的测试约束信息。完成后按下“保存”按钮即可。如果测试剖面定义已经存在，则在对话框中会看到这些信息，用户可以对这些信息进行修改。

（3） 生成测试大纲和测试用例集：选择 Tools → TestCaseAddIn → TestCase

Generantion，弹出“测试用例生成”对话框。填写相关信息（被测模型所对应的系统、子系统、功能域等），选择测试过程控制平台所在的 Lotus Notes 库。按下 OK 按钮，系统将生成测试用例大纲与测试用例，并发布到测试过程控制平台 Notes 库中。

在本例中，我们着重考察活动图中并发模块的处理能力，因此对并发模块定义了如下约束信息：

并发模块中活动之间的位置约束关系，包括：①加水到容器中 <(早于) 把过滤器放到咖啡炉上。②取出咖啡杯 <(早于) 冲调咖啡。此外，我们还要求对并发模块生成 4 个实例化场景。

利用测试用例生成系统，我们为该活动图模型生成了 6 个测试用例，考虑到空间问题，我们仅仅给出它的简要形式。实际生成的测试用例文档包括了每一步要执行的活动、活动之间的迁移条件、活动的前置条件后置条件等，如果活动中存在输入，测试用例中还会给出相应的数值。

测试用例 1： start → 找饮料 → syn1 → 将咖啡放到过滤器中 →加水到容器中 → 把过滤器放到咖啡炉上 → syn2 → 点燃咖啡炉→取出咖啡杯 → 冲调咖啡→ syn3 → 倒咖啡 → 喝饮料 → end。

测试用例 2：start →找饮料 →syn1 →将咖啡放到过滤器中→加水到容器中→把过滤器放到咖啡炉上→syn2 →取出咖啡杯→点燃咖啡炉→冲调咖啡→syn3 →倒咖啡→喝饮料→ end。

测试用例 3： start→找饮料→syn1→加水到容器中→将咖啡放到过滤器中→把过滤器放到咖啡炉上→syn2→点燃咖啡炉→取出咖啡杯→冲调咖啡→syn3→倒咖啡→喝饮料→end。

测试用例 4： start→找饮料→syn1→将咖啡放到过滤器中→加水到容器中→把过滤器放到咖啡炉上→取出咖啡杯→syn2→点燃咖啡炉→冲调咖啡→syn3→倒咖啡→喝饮料→end。

测试用例 5： start→找饮料→有可口可乐吗→取一听可口可乐→喝饮料→end。

测试用例 6：start→找饮料→有可口可乐吗→end。

（2） ATM 机的例子

图 5-12 是一个 ATM 机活动图的例子。

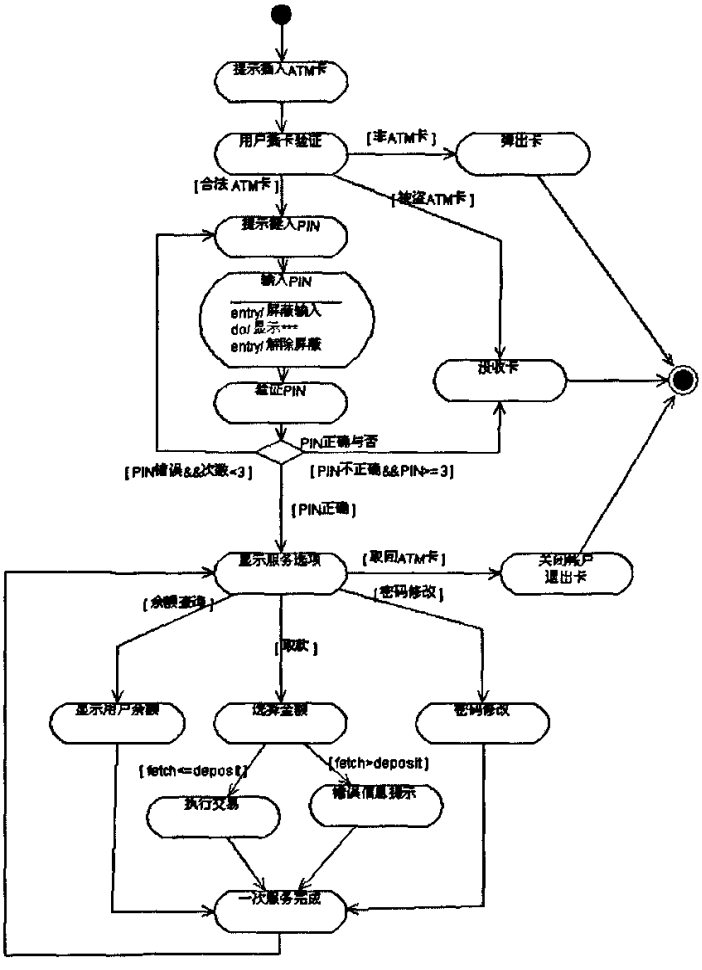


图 5-12　ATM 机活动图一测试用例生成系统示例

根据 ATM 机活动图模型所生成的测试用例如下：

测试用例 1：Initial→提示插入 ATM 卡→用户插卡验证→弹出卡→Final

测试用例 2:Initial→提示插入 ATM 卡→用户插卡验证→提示键入 PIN→输入 PIN→验证 PIN →PIN 正确与否→显示服务选项→关闭帐户→Final

测试用例 3：Initial→提示插入 ATM 卡→用户插卡验证→提示键入 PIN→输入 PIN→验证 PIN→PIN 正确与否→显示服务选项→显示用户余额→一次服务完成→关闭帐户→Final

测试用例 4:Initial→提示插入 ATM 卡→用户插卡验证→提示键入 PIN→输入 PIN→验证 PIN →PIN 正确与否→显示服务选项→密码修改→一次服务完成→关闭帐户→Final

测试用例 5:Initial→提示插入 ATM 卡→用户插卡验证→提示键入 PIN→输入 PIN→验证 PIN →PIN 正确与否→显示服务选项→选择金额→执行交易→一次服务完成→关闭帐户→Final

测试用例 6:Initial→提示插入 ATM 卡→用户插卡验证→提示键入 PIN→输入 PIN→验证 PIN →PIN 正确与否→显示服务选项→选择金额→错误信息提示→一次服务完成→关闭帐户→Final

测试用例 7:Initial→提示插入 ATM 卡→用户插卡验证→提示键入 PIN→输入 PIN→验证 PIN →PIN 正确与否→提示键入 PIN→输入 PIN→验证 PIN→PIN 正确与否→显示服务选项→关闭帐户→Final

测试用例 8:Initial→提示插入 ATM 卡→用户插卡验证→提示键入 PIN→输入 PIN→验证 PIN →PIN 正确与否→没收卡→Final

测试用例 9：Initial→提示插入 ATM 卡→用户插卡验证→没收卡→Final

（3） 计算器的例子

最后是一个简化的计算器活动图示例（如图 5-13 所示）。在该活动图模型中，我们仅支持＋、－、×，/运算和＝运算。该活动图包含了三个嵌套的循环。定义好模型的测试剖面之后，生成的测试用例如下：

测试用例 1：Initial→启动计算器→输入 Num1→输入操作符→输入 Num2→查看运算结果(=)→关闭计算器→Final。

测试用例 2：Initial→启动计算器→输入 Num1→输入操作符→输入 Num2→输入操作符→输入 Num2→查看运算结果(=)→关闭计算器→Final。

测试用例 3： Initial→启动计算器→输入 Num1→输入操作符→输入 Num2→查看运算结果(=)→输入操作符→输

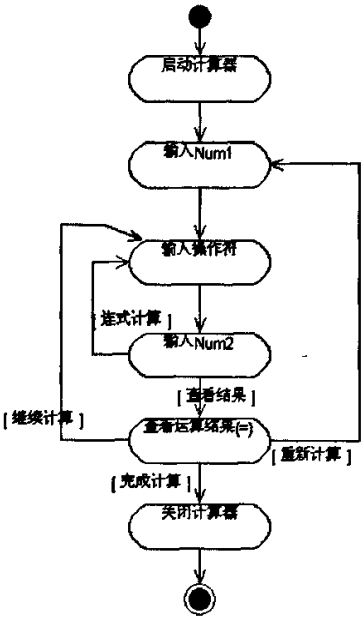


图 5-13 计算器活动图

入 Num2→查看运算结果(=)→关闭计算器→Final。

测试用例 4：Initial→启动计算器→输入 Num1→输入操作符→输入 Num2→查看运算结果(=)→输入 Num1→输入操作符→输入 Num2→查看运算结果(=)→关闭计算器→Final。

下面给出一个实际生成的测试用例，如图 5-14 所示。该测试用例对应于上面以简化形式表示的测试用例 1。实际生成的测试用例中包含了详细的测试步骤，包括到下一步操作的迁移条件、每一步操作的权值、操作的前置条件、后置条件、不变式等约束信息、操作的输入变量描述信息以及系统自动生成的测试数据（包括合法、非法、边界值）、操作完成后的迁移条件等。

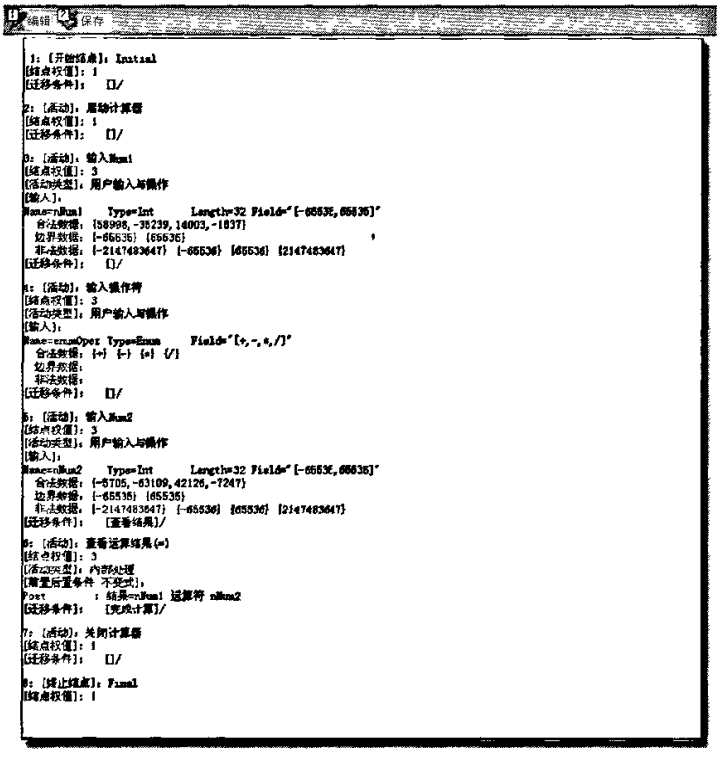


图 5-14 一个具体的测试用例

5.4 本章小结

本章详细介绍了本系统的设计与实现。首先给出了系统的逻辑结构、设计方案、分层包图，然后详细介绍了测试用例生成系统的实现，包括结构化活动图模型、定义测试

剖面、生成测试数据的实现、以及系统中的关键算法。系统最后给出了一个或多个例子来说明系统的使用流程以及产生的结果。

第 6 章 总结与展望

在软件测试的重要性日益增强的今天，研究如何根据需求设计阶段的制品自动生成测试用例具有重大的实际价值。本文以 UML 模型为对象，阐述了基于 UML 模型的测试方法，重点是基于 UML 活动图模型的测试用例生成的研究工作以及取得的成果。下面总结本文的研究成果和存在的不足，并确定今后进一步研究的方向。

6.1 工作总结

本论文主要进行了下面三个方面的研究与开发工作：

（1） 研究了 UML 各模型用于指导测试的可行性及测试策略

◆分析 UML 各种模型用于指导测试的可行性和测试策略，包括静态模型（如类模型）和动态模型（如用例模型、状态模型、活动模型、交互模型），进而指出 UML 活动图着重描述系统所支持的业务过程及其动态行为，因此不仅是进行业务需求分析和系统设计的有力工具，同时也是系统测试的重要依据。

（2） 研究了基于 UML 活动图模型的测试用例设计与生成方法，包括：

◆ 基于活动图模型控制流结构的测试场景生成。在测试场景生成部分，本文针对活动图模型的结构化问题提出了对象流处理方法及并发模块的实例化方法。

◆ 针对活动的输入量的测试数据生成。在测试数据生成中，针对测试数据的描述与生成组合问题，为活动图模型定义了测试剖面，用于描述活动图模型中活动结点的输入输出等测试相关信息，并提出了改进的轮转法以实现测试数据的组合。

（3） 实现了相应的测试支持工具：本课题基于 Rational Rose 工具，设计并开发了基于活动图模型的测试用例设计与生成工具，它提供以下功能：

◆ 基于 UML 活动图模型定义测试剖面。

◆ 基于 UML 活动图模型生成测试大纲与测试用例。

此外，本工具还与测试过程控制平台（Notes 库）进行了集成，从而方便测试人员对测试用例库的使用和管理。

6.2 工作展望

由于时间的紧迫，本系统虽然已经完成了大量研究和开发的工作，但所实现的测试用例生成工具还只是一个原型。就方法研究和系统实现而言，还有如下问题需要进一步研究。

从纵向角度来讲，复杂类型测试数据生成与组合算法的实现。本文给出了测试数据空间的描述方法，简单类型和复杂类型变量的测试数据生成方法，以及测试数据的组合策略，但目前这些方法在工具里还没有完全实现。

从横向的角度来讲，包括：

1 如何将基于 UML 模型的测试与统计测试理论相结合。包括如何将 UML 模型与使用建模相融合，生成测试用例，并根据测试执行结果来评估系统的质量参数，包括可靠性、平均失效时间等。本文所研究的是一致性测试，而统计测试的重点在于可靠性。

2 模型的一致性问题。这一问题并不是基于 UML 模型生成测试用例的重点，但它会影响所生成测试用例的质量。因此，模型的一致性检查也是一个值得考虑的问题。

3 如何基于 UML 模型来生成测试框架，包括测试计划、测试大纲以及测试用例等。不同层次的 UML 模型对应于不同级别的测试。由于基于 UML 模型的软件开发过程都是用例驱动的，因此可以考虑以用例图为中心来组织各模型的用例驱动的测试方法。

最后，系统的实用性还需要不断增强，并在使用过程中进行评估和不断改进。

附　录

1 基于 Petal 格式的测试剖面定义

```
<UMLTestProfile> → { // Rose 模型的测试剖面信息，最顶层结点
                     Object   UMLTestProfile
                     UniqueID    ident
                     Name        string
                     ADTestProfile    <list>    // 活动图测试剖面列表
                     // 下面可以扩充其它模型的测试剖面
                   }
<ADTestProfile> → { // 活动图模型的测试剖面定义
                     Object       ADTestProfile
                     Name         string
                     UniqueID     string
                     Time         datetime      // 该活动图测试剖面定义的时间
                     SyncModules  <List>        // 并发模块信息列表
                     Loops        <List>        // 循环块的信息
                     ADTestStrategy  object     // 活动图测试策略
                     SMsTestInfo     <List>     // 并发模块的测试信息
                     ANTestProfile   <List>     // 活动结点的测试剖面定义
                   }
<SyncModule> → { // 活动图模型中的并发模块信息
                   Object  SyncModule
                   Index   int
                   StartNode   object        // 并发模块开始结点
                   EndNode     object        // 并发模块结束结点
                   ActivityNodes   <List>    // 并发模块中所有结点的列表
                 }

<Node> → { // 对应于活动图中的结点
           Object Node
           NodeName string
           NodeID    ident
         }
<Loop> → { // 活动图模型中的循环后向弧
           Object          Loop
           LoopTransID     Ident
         }
<ADTestStrategy> → { // 活动图的测试策略
```

```
                Object ADTestStrategy

                TestStrategy   string    //  测试策略
                TestCoverage string      //  测试覆盖准则
                DataKind string          //  测试数据生成的类型（合法、非法、边界）
                DataCombine string       //  测试数据组合策略
              }
<SMTestInfo> → { // 一个并发模块的测试信息配置
                Object SMTestInfo
                Index  int
                InstanceCount int        //并发实例化的数目
                ConstraintType string    // 约束类型
                ConstriantType string    // 并发模块的约束类型
                SMPosConfig <List>       // 位置类型的约束信息描述 Position
                SMDFConfig <List>        // 数据流类型的约束信息描述 DataFlow
                SMKPConfig <List>        // 关键路径类型的约束信息描述 KeyPath
              }

<SMPosConfig> → { // 位置类型的约束信息
                  Object    SMPosConfig
                  Node1     object      // 结点 1
                  PosKind   string      // 位置关系
                  Node2     object      // 结点 2
                }
<SMDFConfig> → { // 数据流类型的约束信息 DataFlow
                  Object          SMDFConfig
                  ProduceNode   object        // 生产结点
                  ConsumeNode   object        // 消费结点
                }
<SMKPConfig> → { // 关键路径类型的约束信息  KeyPath
                  Object     SMKPConfig
                  Path       <List>         // 组成关键路径的结点
                }
<ANTestprofile> → { // 活动节点的测试剖面定义
                  Object        ANTestProfile
                  Node          object        // 活动结点
                  Weight        int           // 结点权值
                  NodeType      string        // 活动结点类型
                  InVars        <List>        // 结点的输入信息
                  OutVars       <List>        // 结点的输出信息
                  Relations     <List>        // 输入输出变量之间的关系
                  PrePosts      <List>        // 结点的前置条件、后置条件、不变式等信息
```

```
                }
<IntInVar> →{ // int 类型的输入变量描述
                Object      IntInVar
                Name        string      // 输入变量名
                Type        "int"       // 变量类型 int
                Length      int         // 整形的长度：16，32，64
                Field       string      // 值域约束
                Comment     string      // 输入变量的其它注释信息
              }
<OutVar> →{ // 输出变量描述
                Object      OutVar
                Name        string      // 输出变量名
                Type        string      // 变量类型
                Comment     string      // 输出变量的其它约束信息
              }
<Relation> →{ // 变量间的约束关系
                Object      Relation
                Content     string      // 变量间约束关系描述
              }
<PrePost> →{ // 活动节点的前置条件、后置条件、不变式等约束信息
                Object      PrePost
                Type        string      // 类型：Pre, Post, Invariant
                Content     string      // 信息内容
              }
```

关于输入变量的描述，除了 Int 类型，还包括 String, char, bool, float, enum 等。这里不再一一列举。

2 活动图模型测试剖面的一个实例

```
{object UMLTestProfile
    UniqueID"3E23D9E802BA"
    Name    "F:\program\com\ForRose\TestRose\Debug\RoseModel\模型例子.mdl"
    ADTestProfiles {list ADTestProfiles
    {object ADTestProfile
            UniqueID"3E2FB1110368"
            Name    "CalcuAD"
            Time    "2003-01-27 11:04"
            Loops   {list Loops
                {object Loop
                    LoopTransID"3E2FB1F900E1"}
                {object Loop
```

```
            LoopTransID"3E2FB1AB0355"}
        {object Loop
            LoopTransID"3E2FB21B02D4"}}
    ADTestStrategy    {object ADTestStrategy
        TestStrategy   "测试场景与数据相结合"
        TestCoverage "基本路径覆盖"
        TestDataKind "合法非法边界"
        TestDataCombine  "轮转法"}
    ANTestProfile   {list ANTestProfile
        {object   ANTestProfile
            Node     {object Node
                          UniqueID "3E2FB124009E"
                           Name     "输入 Num1"}
            Weight   "3"
            NodeType     "用户输入与操作"
            InVars   {list InVars
                    {object   IntInVar
                        Name    "nNum1"
                        Type     "Int"
                        Length   "32"
                        Field    "[-65535,65535]"
                        Comment "第一个运算数"}}}
        {object   ANTestProfile
            Node     {object Node
                            UniqueID"3E2FB149028C"
                            Name     "输入操作符"}
            Weight   "3"
            NodeType     "用户输入与操作"
            InVars    {list InVars
                    {object   EnumInVar
                        Name   "enumOper"
                        Type    "Enum"
                        Field    "[+,-,*,/]"
                        Comment ""}}}
        {object   ANTestProfile
            Node     {object Node
                        UniqueID"3E2FB1510021"
                        Name     "输入 Num2"}
            Weight   "3"
            NodeType     "用户输入与操作"
            InVars    {list InVars
                {object   IntInVar
```

```
                    Name      "nNum2"
                    Type      "Int"
                    Length    "32"
                    Field     "[-65535,65535]"
                    Comment ""}}}
            {object   ANTestProfile
                Node      {object Node
                              UniqueID     "3E2FB15B0075"
                                Name "查看运算结果(=)"}
                Weight    "3"
                NodeType      "内部处理"
                PrePosts  {list PrePosts
                    {object   PrePost
                        Type      "Post"
                        Content "结果=nNum1 运算符 nNum2"}}}}}
{object ADTestProfile
        UniqueID"3E23DEC40178"
        Name      "ATM Machine model"
        Time      "2003-01-27 11:05"
        Loops     {list Loops
            {object Loop
                LoopTransID"3E23E42E0384"}
            {object Loop
                LoopTransID"3E23E3660228"}}
        ADTestStrategy      {object ADTestStrategy
            TestStrategy    "测试场景与数据相结合"
            TestCoverage "基本路径覆盖"
            TestDataKind "合法非法边界"
            TestDataCombine   "轮转法"}}}}
```

参考文献

[1] 刘超，程序交互流程图及其测试覆盖准则，软件学报，1998.6，9(6)：458-463

[2] Aynur Abdurazik and Jeff Offutt，“Generating Test Cases from UML Specifications”，UML 99，P416-429，October 1999

[3] 兰毓华，毛法尧，曹化工，基于 Z 规格说明的软件测试用例自动生成，计算机学报，1999，22(9)：963-969

[4] 郑人杰，殷人昆，陶永雷，实用软件工程（第二版），清华大学出版社，1997.12

[5] 刘超，金茂忠，软件测试过程的基本模型 POCERM，北京航空航天大学学报，1997.2，Vol.23：56-60，

[6] Tsai W T，Volovik D，Tkeefe T F，“Automated test case generation for programs specified by relational algebra queries”，IEEE Trans on Software Engineering，1990，16(3)，316-324

[7] Weyuker E，Goradia T，Singh A. “Automatically generating test case data from Boolean specification”，IEEE Trans on Software Engineering，1994，20(4)，353-363

[8] A.J. Offutt and Yiwei Xiong，Shaoying Liu，“Criteria for Generating Specification-based Tests”，Fifth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '99)，October 1999

[9] Chris Rudram，“ Generating Test Cases from UML”，University of Sheffield，June 23，2000

[10] A.S. Evans，“Reasoning with UML Class Diagrams”，Workshop on Industrial Strength Formal Methods (WIFT'98)，IEEE Press，1998

[11] J. Lilius and I. Porres Paltor. “vUML: a Tool for Verifying UML Models”，Proceedings of ASE'99，IEEE Computer，1998

[12] Tsun S. Chow，“Testing design modeled by finite-state machines”，IEEE Transactions on Software Engineering，1978，4(3)：178-187

[13] Susumu Fujiwara, Gregor v. Bochmann, etc, "Test Selection Based on Finite State Models", IEEE Transactions on Software Engineering, Vol, 17, No.6, June 1991

[14] James A. Whittaker and Michael G. Thomason, "A Markov Chain Model for Statistical Software Testing", IEEE Transactions on Software Engineering, Vol. 20, October 1994

[15] Ibrahim K. El-Far and James A. Whittaker, "Model-based Software Testing", Florida Institute of Technology, 2001

[16] 黄陇，陈致明，于洪敏，于秀山，基于UML的软件测试用例自动生成技术研究，计算机应用与软件，2004年 21卷 11期：16-17,113 页

[17] UML Specification 1.3, http://www.omg.com ,1999

[18] Ivar Jacobson, Grady Booch, James Rumbaugh, The Unified Software Development Process, Addison Wesley Longman, Inc, December 1999.

[19] Robert.V.Binder, Testing Object-Oriented System, Addison-Wesley, 2000

[20] 张毅坤，施风鸣，姚全珠，刘军，付长龙，基于UML状态图的类测试用例自动生成方法，计算机工程，2003年29卷21期：91—93

[21] 田志刚，朱小冬，甘茂治，基于顺序图的软件可靠性测试用例生成方法，计算机工程与设计，2005年26卷10期

[22] 沈剑乐，王林章，李宣东，郑国梁，一个基于UML顺序图的场景测试用例生成方法，计算机科学，2004年8期

[23] 刘超，张莉，可视化面向对象建模技术，北京航空航天大学出版社，1999.7

[24] Benoit Baudry, Yves Le Traon, and Gerson Sunye, "Testability Analysis of a UML Class Diagram", Proceeding of the Eight IEEE Symposium on Software Metrics(METRICS' 02), 2002

[25] Roger S. Pressman 著，黄柏素 梅宏译，软件工程一实践者的研究方法(第四版)，机械工业出版社，1999.10

[26] 林振，盛浩林，吴定一，测试数据的规范描述与自动生成，计算机工程，1994，20(2)：52-54

[27] 袁洁松，王林章，李宣东，郑国梁，一个基于灰盒方法从UML活动图生成测试用例的工具，计算机研究与发展，2006年01期

[28] Rational Software Corporation, Rational Rose 在线帮助－Rational Rose Extensibility Interface, 2001

[29] 吴文虎，王建德，图论的算法与程序设计，清华大学出版社，1997

[30] M. Dahm, Grammar and API for Rational Rose Petal files, July 19, 2001

[31] User's Guide: Matlab Statistics Toolbox, version 3, http://www.mathworks.com

致　谢

首先我要衷心地感谢我的导师王晓琳老师。本论文及相应的研究工作都是在王老师的悉心指导下完成的。王老师淡泊宽厚的为人、渊博的知识、严谨求实和追求创新的治学态度都使我受益匪浅。在王老师倾尽了心血的指导和关心下，帮助我顺利完成了学业，在我即将离开校园之际，谨向王老师表示最诚挚的感谢。

感谢各位任课老师在生活上和学业上给予我的关心和帮助，各位老师高尚的为师品德和宽厚热情的待人态度为我树立了学习的榜样。

感谢我的研究生的同学们，在我研究生学习期间他们在学业和生活上都给予了我很多的关心和帮助，并鼓励我不断前行。

最后感谢我的家人和所有的朋友们，你们在生活上支持我，给我鼓励和帮助，在此我也要将我最诚挚的谢意献给你们。

基于UML活动图模型的测试用例生成方法的研究

作者： 粘新育
学位授予单位： 山东大学