分类号_____ 学号 M200775118

学校代码 10487 密级_____

華中科技大學

硕士学位论文

基于 RCP 的自动化测试平台的研究与实现

学位申请人：张运英

学 科 专 业：软件工程

指 导 教 师：苏曙光

答 辩 日 期：2009.5.22

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree for the Master of Engineering

Design and Implementation of the Automatic Test System Based on RCP

Candidate : Zhang Yunying

Major : Software Engineering

Supervisor : Su Shuguang

Huazhong University of Science and Technology

Wuhan 430074, P. R. China

May, 2009

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并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。本人授权华中科技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、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于　保密□，　在_____年解密后适用本授权书。
　　不保密□。

（请在以上方框内打“√”）

学位论文作者签名：　　指导教师签名：

日期：　　年　月　日　　日期：　　年　月　日

摘　要

随着通讯行业在整个社会系统中的地位越来越重要，使得通讯产品开发的规模和复杂度也随着其需求和重要度的增加而急速上升，如何保证通讯产品的开发质量就成为了通讯行业所面临的日益严峻的问题。从通迅行业的角度看，自动化测试系统为行业的业务拓展和大规模通讯网络管理提供了一种更为有效的产品质量保证方式；而从测试技术的角度看，自动化测试系统则代表着软件业测试的发展方向。

结合通迅行业特点，本文首先分析了自动化测试系统的整体需求、系统实现的可行性和实现目标，探讨了系统分层式实现方案，并且讨论了关键字驱动技术之下的系统总体结构和系统运作逻辑。接着讨论了系统的层次化体系结构，系统的执行流程和工作状态转换等情况。此后，讨论了系统的两个关键层次富客户端和调度服务器模块的设计实现情况，讨论了这两个层次模块的实现目标和实现方案等。结合系统的统一结构和执行流程，讨论了富客户端的设计实现方式以及实现结果，对客户端的整体插件布局以及核心模块数据驱动模块、测试用例编辑器、子系统返回值和测试报告查看器模块的实现情况进行了重点介绍。最后，讨论了对自动化测试系统的测试情况，讨论了系统交叉测试的任务制定、人员时间分配进行讨论，分析测试执行开展情况和测试所发现故障的解决情况；此后结合系统交叉测试结果，得出系统故障分布的特点和后期制定测试规划等。

自动化测试系统采用多层次 CS 架构模式，本文从系统应用层、调度层、执行层和被测系统的层次角度对自动化测试解决方案进行诠释。自动化测试系统适应通讯行业日益复杂的测试业务流程和愈发繁多的测试任务，其代表现代测试的发展方向。

关键词：自动化测试　　富客户端　　数据驱动　　关键字驱动

Abstract

With the importance of communications in the system of society, the scale and complexity of the output is facing rapidly development besides with the increase of requirement, and it is a serious problem of how to ensure the quality of such software product. From the point of view of communications, the Automatic test system provides the assurance of quality, besides, from the point of view of testing, the system is on behalf of benefits the development in software testing.

Combining with the characteristic of communication, in the first, the author analysised the whole requirement , feasiblity study and the achiefing goals, and discussed multi-layers solution of the system, and analysised the system structure and the system logic of execution by the technology of keyword-driven. And following, he discussed the architecture and the state transition of the system, and then, analysised tow important parts of this system, the Rich Client Platform and the Dispatch Server, discussed the aims and designing patterns of this two parts. Besides, combining with the structure of this system, the discussion revolves around the layout of the RCP GUI, the module of data-driven, the editor of test designing, the module of the return value of sub-system and the module of test report. And finally, the author discussed the test condition of this system, and anysised the cross-test in the development team, setting down the test plan, the distribution of team-member, etc. And by the test result of the system, the author anysised the distribution of the breakdowns, and the organization planning of later period.

The system structure is adopted a multi-layers mode to design, and annotate the solution in the applying layer, the dispatch layer, the executing layer and the being tested system. The Auto-Test system could adopt the complex test business of the field of communication and such heavy-duty test workload, and the technology of Auto-test benefits the heading of test.

Key words： Auto-Test RCP Data-Driven Keyword-Driven

目　录

1 绪论

1.1 项目研究背景

软件测试是软件工程的一个范畴，它作为软件工程的一部分，随着软件生产的产业化运作应运而生。从 20 世纪 70 年代开始，软件产业在社会系统中的地位也越来越重要，而计算机软件开发的规模和复杂度也随其需求和重要度的增加而上升。

在20年前，人们对软件测试的最初认识是“证明程序的正确性”，随后一系列软件测试理论和方法的逐渐被提出。1983年IEEE提出了目前比较认可的软件测试的定义，即：“为了检验某个应用系统的过程是否满足规定的需求，并了解预期结果和实际结果之间的差异，而使用手工和/或自动化手段来运行并验证这一过程”。按照该定义，软件测试的目的是监测和排除缺陷，以确保软件产品在可用性，功能性，可操作性等多方面满足软件需求[6]。

一直以来，大多数国内软件企业对于软件测试的重要性没有足够的认识，从而反映到公司组织形式上是，很多的软件公司没有成立测试部门或测试小组等。对软件测试的理解上也存在误区，有的公司只是在编码阶段后期才会进行测试，有的公司则采用谁开发谁测试的办法，或者甚至测试环节都没有。为什么这些国内的软件公司常忽略软件测试的作用呢？主要原因是企业对软件测试的重要性理解不深，有很多软件开发或管理人员认为程序能试运行基本上就已经成功，没必要成立专门的测试部门或设立测试岗位。其次，很多软件开发企业在为软件开发支付费用后，不希望再为软件的测试支付更多的成本，即使有些项目的开发方有意对软件进行第三方测试，也会考虑到在测试过程中往往需要开发商提供源代码，从而担心其知识产权遭到侵犯，因而不能对软件进行很好的测试。

随着软件危机的出现，软件测试的地位得到提高，人们逐渐认识到，软件开发中，测试的时间越早，测试执行的频率越高，整个软件开发成本降低得越多。另外随着软件开发规模的扩大及其复杂程度的增加，软件测试工作也越来越重要，并且工作难度越来越大，但同时也存在不少认识上的误区。软件测试作为保证软件质量

和可靠性的关键技术，也随着软件工程规模的日益扩大，而面临越来越多挑战。面对繁重的测试工作量以及越来越高的软件质量要求，如何提高测试的质量和效率，就成了许多人深感困惑的问题。

从上世纪 90 年开始，产业界意识到被动的以监测和发现错误为目的的软件测试无法避免在软件开发过程中由于软件需求和设计等方面的缺陷所带来的风险，所以在整个产业界开始从软件质量控制（SQC, Software Quality Control）开始转移到软件质量保证（SQA, Software Quality Assurance），从而使软件测试从单纯的缺陷检测和发现覆盖到整个软件开发过程，同时软件测试的流程和技术也成为独立的研究方向。

因此，在软件测试领域自然而然的引入了的一系列的测试工具来帮助测试人员完成这日益艰巨的测试任务。随着软件技术的不断发展，测试工具也在不断发展，并且测试工具已从单纯手工测试的辅助工具向自动化测试的方向发展。需求引导项目和产品，同样，测试人员在使用测试工具的过程中，对测试工具的要求也越来越高，要求测试工具所覆盖的测试范围也越来越广，并最终促成了各种自动化测试工具的问世。自动化测试工具的引入也为测试人员更加快速、有效地对软件进行测试，提高软件产品的质量等需求的实现提供了可能。

在通讯行业，目前 TD-SCDMA 已经正式开始商用，随着 3G 时代的到来，为国内通讯设备制造厂商提升国际市场份额，带来前所未有的发展机遇。因而，通讯设备厂商将更注重其通讯网络质量及业务、技术水平、产品质量稳定等，从而有机会在未来的竞争中取胜。行业的机遇与挑战也使得对产口出厂前测试的要求越来越高。一方面因为产品业务的扩增，使得通讯网络管理日益复杂；另一方面，所需要支持的电信业务也越来越多。这些特征都使得支持通讯网络和产品的业务实现更加复杂，从而使得测试人员在进行测试时必须设计逻辑更加复杂的测试用例测试新功能，并且需要对原有模块进行更多的回归测试。由此可见，先进测试方法和测试工具的开发就显得非常必要,使得既能提高测试的效率、节约手工测试的时间和扩大测试的覆盖面等。针对行业的发展特征,公司组织开发小组研究制定了本自动化测试平台项目的开发计划，以便开发出适应新需求的自动化测试工具，提升企业的产品竞争力。

1.2 项目研究意义

软件危机是软件界甚至整个计算机界的最热门话题之一，为了解决这场危机，很多软件从业人员、专家和学者做出了大量的努力。现在人们已经逐步认识到所谓的软件危机实际上正是软件中存在的一些错误导致了软件开发在成本、进度和质量上的失控。我们该如何去避免错误的产生和消除已经产生的错误，从而使程序中的错误密度达到尽可能低的程度。与国外相比，在大型软件的测试领域，国内软件产品的质量控制体系和标准还是显得比较模糊。不过可喜的是，国内的很多软件开发机构已经注意到软件测试发展中的这些瓶颈，并且开始重视软件测试技术和测试工具的开发等。比软件测试流程规范化更为困难的是测试人员经验的积累和技能的提高，随着软件测试需求的增加，年轻的测试人员不断地加入测试队伍，如何快速提高他们的能力，使其适应测试工作的需要是亟待解决的问题。

随着测试技术的发展，实现测试自动化的趋势已经不可逆转。目前，测试自动化主要集中在软件测试流程的管理自动化和动态测试方面。自动化测试就是使用软件工具来代替手工测试所进行的一系列动作，其通常是使用脚本或者其他代码驱动应用程序等技术。有两类适用自动化测试的情况：构建版本验证测试（Build verification test, BVT）和回归测试（Regression testing）。其中构建版本验证测试是指针对软件最新版本而做的一系列自动化测试，以确保构建版本足够稳定可以做进一步的测试或做他用；回归测试是指测试以前曾经正常工作的常景[1, 5]。

那么，对于企业用户来讲，什么样的软件测试自动化工具将是他们所需要的呢？回答这个问题之前，我们首先必须注意到，作为企业用户，在企业内部通常存在许多不同种类的应用平台，应用开发技术也不尽相同，甚至在一个应用中可能就跨越了多种平台,同时由于开发时期的不同，也可能导致对同一应用的不同版本之间也存在差异。所以软件测试本身对于这些企业的 QA 和测试部门来说无疑是一个巨大的挑战，他们考虑选择软件测试自动化的出发点就是为了降低这些挑战可能带来的对产品质量和上线周期等方面的风险。企业的自动化测试需求更多的集中在自动化软件测试管理流程和测试用例的标准化复用等方面。他们希望通过自动化软件测

试达到对软件质量的可量化过程管理并提高测试执行效率。

相对于手工测试而言，自动化测试的优势十分明显。自动化测试是确保今天的软件和昨天的软件一样优秀的非常好的方法。换句话说，自动化测试可以发现“回归缺陷”并且可以循环不同的输入。回归测试是软件测试中的重要组成部分,占有很大的比重。大约30%左右的错误(Bug)是通过回归测试发现的[2, 7]。因此，自动化测试对管理人员来说更是有必要进行，它可以节省大量时间，减少测试人员的数量，减少机器的数量等。同时，它还能够完成大量手工无法完成的测试工作，如并发用户测试、大数据量测试、长时间运行可靠性测试等[3]。实施自动化测试意味着我们可以不需要花费太多的人力而更快速成更廉价的开发出高质量的软件，其优点可以概括如下：

（1）提高测试质量:软件开发的过程就是一个持续集成和改进的过程，而每一次修改都有可能产生错误，因而需要对软件进行回归测试。而这种回归测试尤其适合于用自动化测试工具执行，其能以便利的方式验证是否有新的错误引入软件产品。

（2）提高测试效率，缩短测试工作时间：使用自动化测试工具，可以减轻测试工程师的手工测试工作量，同时，测试工具还可以控制管理整个测试过程，能够更好量化测试的进度。

（3）提高测试覆盖率：自动化测试工具将测试人员从繁重的手工测试中解放出来，使他们可以专注于设计良好的测试用例。

（4）执行手工测试不能完成的测试任务：自动化测试工具可以执行一些非功能性方面的测试，如压力测试、负载测试、大数据量测试、崩溃性测试等测试类型。

（5）更好地重现软件缺陷的能力：自动化工具具有一致性和可重复性，由于每次自动化测试运行的脚本是相同的，所以每次执行的测试具有一致性，而由手工测试是很难做到的。

（6）更好地利用资源：理想的自动化测试能够按计划完全自动运行，因而可以减少测试人员的调配，从而一定程度避免开发和测试的冲突。

当然也有许多采用自动化测试的项目不能按预期成功交付，主要原因是自动化

测试的执行者没有认识到自动化测试只是测试过程的一部分，需要设计出合理的自动化测试用例进行测试驱动。在一个软件企业中，无论自动化测试占总测试多少比重，其无疑在测试项目中占据一席之地，且代表着未来的测试方向。它可能不会提高产品的执行性能或提供任保的质量保证，也许自动化测试使用失控还会是一种非常耗时的行为。一个设计非常差的自动化测试将会浪费很多时间和人力，但是如果成功的话，好处将是非常显著。自动化测试所具有的这些特征，正是公司组建项目组进行自动化测试平台开发的意义所在。

1.3 国内外研究现状和发展趋势

自动化测试技术的含义非常广泛，任何帮助流程的自动流转、替换手工性测试的动作、解决重复性问题等，能够帮助测试人员执行测试的相关技术或工具的使用都叫自动化测试技术。例如，一些测试管理工具能帮助测试人员自动统计测试结果并产生测试报告，编写 SQL 语句插入大量数据填充到某个表中，编写脚本程序让版本编译自动进行，采用多线程技术模拟并发请求，利用工具自动记录和监视程序的行为，使用工具自动执行界面上的鼠标单击和键盘输入操作等。自动化测试是测试发展的必然趋势，而自动化测试工具技术也从早期的录制回放技术、脚本技术发展到如今的数据驱动、关键字驱动以及业务驱动技术等。

早期的采用录制回放技术的自动化测试工具的测试框架所采用的原理大多是通过录制应用程序产生的线性脚本进行回放从而达到自动化测试的目的。采用这种方式的自动化测试工具其优点是简单,通过录制就可以得到所需脚本,但是其也存在很大的缺点,首先它不具有逻辑判断能力,并且可维护性差，运行效率比较低下。

目前国内外的自动化测试工具所采用的技术多为数据驱动技术和关键字驱动技术。数据驱动技术是指将自动化测试用例的脚本和脚本数据相分离，其自动化测试工具框架的原理主要是指采用了数据驱动脚本进行测试。将测试数据输入存储在独立的数据文件中，从而形成数据驱动脚本，而测试脚本本身只存放控制信息和逻辑信息等。在进行测试时，将需要的输入数据直接从数据文件中读取出来，与脚本一起构成测试用例进行测试。采用数据驱动的方式，使得同一脚本可以运行于不同

的测试用例中,从而实现了测试脚本与测试数据的分离。采用这种测试技术的优点是可以快速增加相似测试，在测试者增加新的测试时，不必掌握测试工具语言,对此后的类似测试也无额外维护开销；当然缺点则是初始建立测试脚本的开销较大,进行数据扩展的脚本需要针对相同的测试内容并具有相同的测试逻辑，从而一定程度上使得测试逻辑的灵活性受到限制。

关键字驱动脚本技术是以关键词句作为测试事件的驱动，以减少测试脚本的自动化开销，简化测试脚本编写为目标，基于测试能够被分解成为若干独立步骤的观点提出的。关键字脚本技术是将测试步骤细化分解，从而建立一个能被大量引用的功能词句库或者称作关键字库。这样使得要编写大量测试代码才能完成的测试脚本，测试脚本就变成了几个关键字词句的逻辑组合，从而使得开发脚本变的简单和易于阅读维护等。关键字驱动也可以看作是数据驱动的逻辑扩展，采用关键字驱动技术相当于在被测应用程序和录制生成的测试脚本之间增加一个抽象层，从而使得界面元素名和测试内部对象名相分离。关键字驱动技术将界面上的所有元素映射成相对应的一个个逻辑对象，测试则针对这些逻辑对象进行，界面元素的改变只会影响界面元素到逻辑元素的映射表，而不会影响测试。此外，采用关键字驱动可以实现测试描述与具体实现细节的分离。测试描述只说明软件测试要做什么以及期待什么样的结果，而不管怎样执行测试或怎样证实结果。因为测试的实现细节通常与特定的平台以及特定的测试执行工具有着密切的联系，将描述与实现相分离使得测试描述对于应用实现细节不那么敏感，从而有利于测试在工具和平台之间的移植。最后，关键字驱动技术还实现了测试脚本与测试数据的分离。和数据驱动技术类似，关键字驱动技术也可以把测试执行过程中所需的测试数据从脚本中提取出来，在运行时测试脚本再从数据存放处读取预先定制好的数据。采用这种机制，从而实现了测试脚本和测试数据的独立维护。

录制回放技术、数据驱动技术和关键字驱动技术这三种自动化测试技术各司其职、互相独立，并且最大程度地减少相互之间的影响。从关键字驱动的思想可以看出，该种测试框架不仅实现了将数据和脚本相分离，而且实现了测试逻辑和数据的分离，能够大大提高了脚本的复用度和维护性，从而更大限度地体现了测试工具自

动化的思想，因此关键字技术结合数据驱动技术是目前自动化测试技术中应用较为广泛的技术。

目前可供选择的第三方的自动测试工具很多,WinRunner, SilkTest,等。这些测试工具通常可以对软件的图形用户界面(GUI)进行冒烟测试(SmokeTest)和回归测试。WinRunner 是 Mercury Interactive 公司的一种企业级功能测试工具，能够检测应用程序是否达到预期的功能。通过自动录制、检测和回放用户的应用操作，WinRunner 能够有效地帮助测试人员对复杂的企业级应用的不同发布版进行测试。SilkTest 是美国 Segue 公司开发的一种软件 GUI 的自动测试工具。使用 SilkTest 软件可以将被测试软件(Application Un2der Test, AUT)的 GUI 对象(例如,窗口、按钮、菜单、文本等)和在其上进行的各种鼠标或键盘操作录制下来,转换成 Segue 公司开发的一种面向对象的第四代编程语言(4Test)的脚本[5, 8, 9]。

从测试工具出现起，随着软件开发技术的不断进步，测试工具和测试方法也在不断发展，目前主要应用于自动化回归测试(Regression testing)。当然,并不是所有的测试都适合自动化，需要对涉及到的技术要求和测试步骤进行研究分析，确定测试过程中能够进行自动化测试的内容，再选择成熟的自动化测试的工具或者开发相应的自动化测试工具。

在大多数情况下，软件测试自动化可以减少开支，增加有限时间内可执行的测试量。对测试工具能够发挥的作用，大家都已经了解并认可，但是目前很多引入自动化测试工具的软件公司并没有能够让测试自动化发挥应有的作用。主要存在以下问题：

（1）过高的期望:多数情况下，人们对软件测试自动化存在过于乐观的态度、过高的期望，人们都期望通过这种测试自动化的方案能解决目前遇到的所有问题[4]。

（2）缺乏专业的测试人才:软件测试自动化并不是简单地使用测试工具，还需要有良好的测试流程、全面的测试用例等来配合脚本的编写，这就要求测试人员不仅熟悉产品的特性和应用领域、熟悉测试流程，而且很好地掌握测试技术和编程技术。

（3）测试工具本身的问题:一般不会对自动测试脚本做大规模的测试，所以自动测试脚本的质量往往依赖于测试自动化工程师的经验和工作态度,如果自动测试工具不能提供一种机制来保证脚本的质量，那将影响到测试结果的正确性。

尽管目前自动化测试工具的开发和使用都存在一定的问题和风险，但是其发展前景却被普遍看好，它将为自动化测试的广泛应用和推广起支撑作用。

1.4 主要研究内容

本课题的研究对像是自动化测试平台。它采用一种分布式布局的解决方案来实现系统中各层次的灵活调用。图 1-1 是自动化测试平台结构图。平台包括 Client、DispenseServer 和 TestSubsystem 三个功能层次的模块结构。

在系统中，Client 客户端采用 RCP（Rich Client Platform,胖客户端）设计，用户通过操作 Client 来执行平台的一系列操作；DispenseServer 是分发服务器，是平台的一个调度中心，用于处理消息的注册和分发功能；TestSubsystem 是执行测试调度的各个子系统模块，平台正是通过 TestSubsystem 来调度执行底层各种复杂的业务和 GUI 测试等。

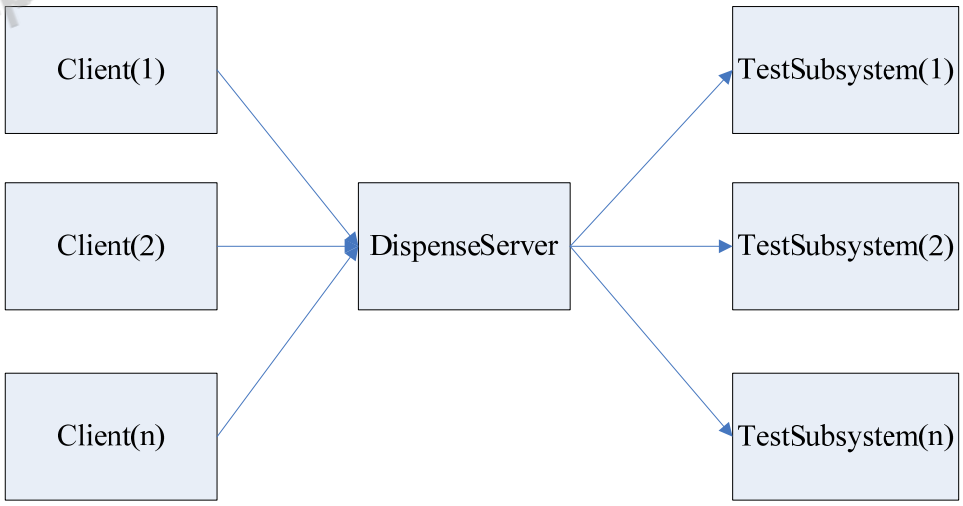

图 1-1　自动化测试平台结构图

在平台中，Client 是用户打交道的胖客户端，用户通过 Client 客户端来与整个系统打交道。包括编辑自动化测试任务脚本，消息控制执行测试任务，以及查看自

动化测试报告等功能。DispenseServer 则是平台的调度中心，其一方面为 Client 客户端屏蔽复杂的测试调度，另一方面也将对应的测试任务分发到对应的 TestSubsystem 测试调度子系统，避免 TestSubsystem 解析复杂的测试任务类别。TestSubsystem 测试调度子系统则是平台中真正的测试调度功能单位，其将调度实际的测试执行单元。

2　主要技术分析

2.1　自动化测试技术

软件自动化测试，是一种让计算机代替测试人员进行测试的技术，是指编写软件去测试其他软件。也可以被理解为：使用一个商业通用测试自动化工具编写一个软件来测试其他软件；或者被理解为：编写驱动被测试应用程序的测试脚步本以执行键盘、鼠标动作和后台进程，并验证应用程序响应和行为[10]。

软件自动化测试的实现基础是可以通过设计特殊程序来模拟测试人员对计算机的操作过程和操作行为等，或者该程序能够像编译系统那样对计算机程序进行检查。软件测试自动化实现的原理和方法主要包括：直接对代码进行静态和动态分析、测试过程的捕获和回放、测试脚本技术和虚拟用户技术等。

（1）代码分析:类似高编编译系统的分析工具，其通过定义一些类、对象、函数、变量等的定义规则和一些语法规则，可对不同高级语言进行代码分析；在分析代码时，分析工具进行语法扫描，找出代码中不符合编码规范的地方，并可以根据某种质量模型评价代码质量，生成系统的调用关系图等。

（2）录制和回放:录制和回放是一种黑盒测试方法，其可以捕获记录将用户的每一步操作。通过捕获程序用户界面的像素坐标或程序显示对象的位置，以及一些用户操作、状态属性的变化等，然后将捕获的记录转换为一种脚本语言所描述的过程，以模拟用户的操作。在回放时，将脚本语言所描述的过程转换为屏幕上的操作，并将被测系统的输出记录与预先给定的标准结果进行比较。

（3）脚本技术:脚本是计算机程序的一种形式，可以看作是一些测试工具执行的指令集合。脚本的产生可以直接用脚本语言编写，也可以为减少脚本编程的工作量而通过录制测试操作并对其做修改而产生。目前常用的脚本技术很多，分为：线性脚本、结构化脚本、共享脚本、数据驱动脚本和关键字驱动脚本等。

（4）虚拟用户技术:虚拟用户技术通过模拟真实用户的行为来对被测程序施加负载，以获取其测试性能指标值，如事务的响应时间、服务器的吞吐量等。

开发自动化测试的关键在于使重复性的测试和传统发现最多错误的任务实现自动化，同时能够使测试的可维护性更好，便于在其他项目中重用等。目前测试自动化工具的应用主要集中在回归测试、冲击测试和错误管理等方面。

2.2 分布式处理与分层处理

分布式处理是指由多个自主相互连接的信息处理系统，在一个高级操作系统协调之下共同完成同一任务的一种处理方式。 在网络出现之前，信息处理一般采用集中式处理或分散式处理方式。集中式处理是指把各项信息包括远程信息都传输到系统的统一信息处理系统中进行信息处理。分散式处理的最大优点是能够简单地提取数据、变换格式或者进行其他数据加工，最后输出结果。而要把这些适用于分散式处理的工作场景集中于统一的程序中是非常困难和代价昂贵的。后来随着要求在数据产生场所进行局部处理的数据比重不断增加，使得系统应答时间和计算机处理能力不能满足需求，因而人们常常把分散设置的各计算机组合成计算机网络，从而形成分布式处理系统。这种处理方式既结合了集中式和分散式处理的优缺点。分布式处理系统模型图，如图 2-1 所示。分布式处理将装备从集中的处理系统中分散开来，能够直接从信息源取得信息并进行相互协调处理等操作。

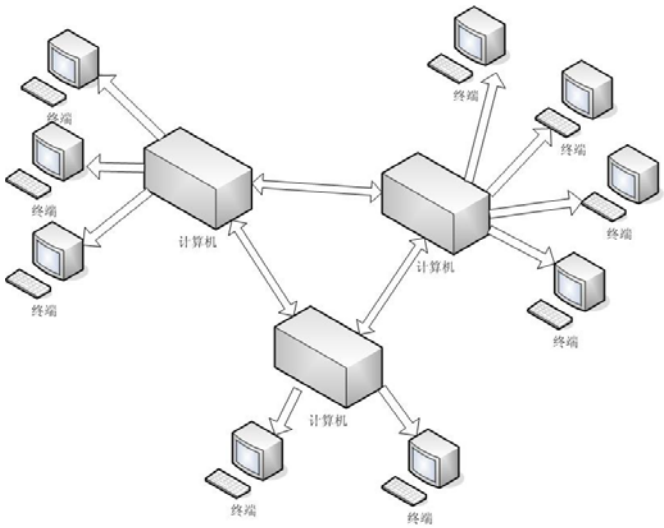

图 2-1　分布式处理系统模型图

分布式处理系统能够在较短时间内动态组合成面向不同服务对象的处理系统，然而这些组合于于用户来说却是透明的，用户只需指定系统干什么而不必指出哪个部件提供这些特定的服务。系统各组成部分是自主的，但不是无政府状态，而是遵循某个计划由高级操作系统进行协调工作。在一个计算机网中有多台主机，但这还不一定都是分布式处理。如果这多台主机的系统不具备动态组合及任务再指派的能力，那么它们仍然是集中式处理，因此可以说，高级操作系统是分布式处理的关键。在分布式系统中不再使用完整的信息，各个组成部分提供自己的状态信息，高级操作系统根据这些状态信息进行任务协调和资源再分配，各组成部分之间没有层次关系而是一种自主关系。

层次系统组织成的层次结构中,每一层需为上层服务，并作为其下相邻层次客户。在一些层次系统中，除了一些精心挑选的输出函数外，内部的层次只对相邻的层可见。这样的系统中构件在一些层次中实现了虚拟机[12]。连接件通过决定层间如何交互的协议来定义，拓扑约束包括对相邻层间交互的约束。图 2-2 是层次系统体系结构图，其特点可概括如图 2-2 所示。

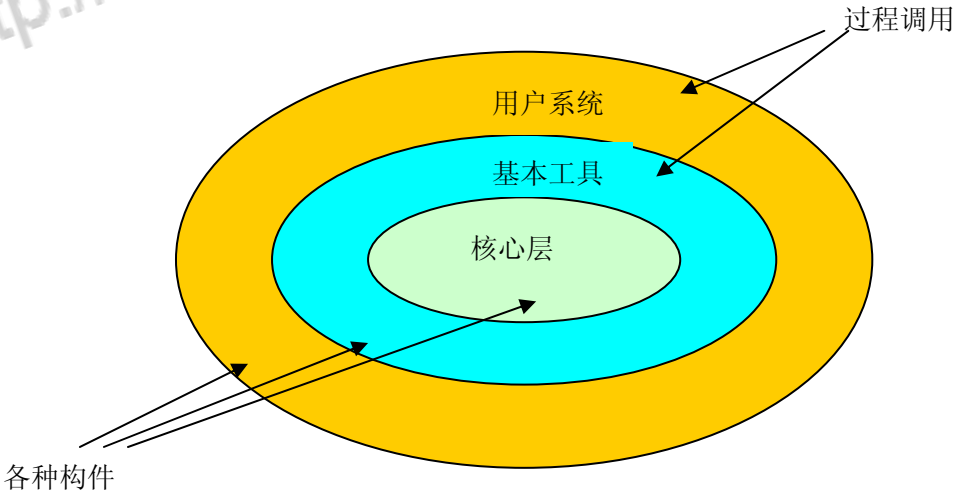

图 2-2　层次系统体系结构图

（1）支持基于抽象程度递增的系统设计，使设计者可以把一个复杂系统按递增的步骤进行分解，然后对分解的各层实行层次封装。

（2）支持功能增强，因为每一层至多和相邻的上下层交互，因此功能的改变最多影响相邻的上下层，因而使得各个层次实现的功能点相对独立。

（3）支持重用，只要定义好服务接口，对于同一层的不同实现可以交换使用这些接口，从而可以允许各种不同的实现方法。

2.3 Eclipse RCP 富客户端技术

RCP 的全称是 Rich Client Platform，是富客户端的意思。富客户端这个词最早出现于 90 年代初，人们还在利用 VB，delphi 开发桌面应用程序的时候就已经出现了。随着桌面应用程序数量的不断增长，各种各样的应用程序孕育而生，小到 Windows 自带的扫雷游戏，大到企业级桌面 ERP 系统，桌面应用充斥着我们的生活。

Eclipse 是一个开放源代码、基于 Java 的可扩展开发平台。就其本身而言，它只是一个框架和框架下的一些服务，它可以看作是一些插件组件加载在其内核之上而构建起来开发环境。但真正的 Eclipse 是一个提供了完善插件机制的 RCP 平台，它以 SWT/JFACE 作为界面元素组件，提供给用户一个名为 Workbench 的 UI 平台，加上它本身优秀的插件机制，能够构造出扩展能力强、性能优秀、并提供给用户良好 UI 体验的 RCP。Eclipse 的插件结构图，如图 2-3 所示。

除了小型的运行时内核外，Eclipse 中的所有东西都是插件。从这个角度来讲，所有功能部件都是以同等的方式创建的。但由图我们也可以看到，Workbench 和 Workspace 是 Eclipse 平台的两个必备的插件---它们提供了大多数插件使用的扩展点。

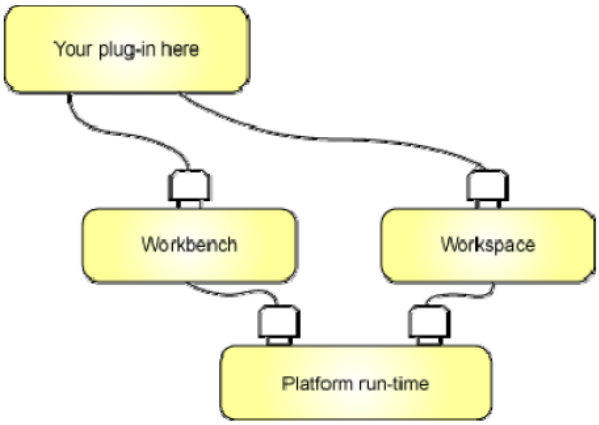

图 2-3　插件结构图

使用 RCP 来开发 Java 桌面应用可以把开发的焦点转移到系统的逻辑功能上，而不是界面上。我们自己的程序可以继承 Eclipse 的风格与功能，而不用自己去编写诸如菜单，工具条，子窗口等等的界面元素。甚至我们可以把 Eclipse 本身的功能插件，比如 Console 拿来放在自己的程序里，这样就避免了很多重复开发。

2.4 XML 可扩展标识语言技术

XML 可扩展标记语言(extensible markup language)与 HTML 超文本标记语言(hypertext markup language)类似，都是从所有标记语言的原语 SGML(standard generalized markup language,准备通用标记语言)那里派生出来的。

XML 的广泛使用和巨大潜力使其在现在和将来成为不争的标准，随着越来越多的规范对 XML 的支持，使得 XML 的功能日趋强大，不仅是在 Web 世界，在整个软件系统架构过程中都发挥出巨大作用。目前 XML 的应用很广泛，包括设计置标语言、数据交换和 WEB 服务等。XML 主要是一种数据描述方法，其魅力要在与其相关的技术结合中才能显示出来。与 XML 相关的技术很多，以下介绍几组与 XML 紧密相关的方面：

（1）DTD 与 Schema:DTD(document type definition,文档类型定义)和 Schema 是用来对文档格式进行定义的语言,相当于数据库中需要定义数据模式一样，DTD 和 Schema 决定了文档的内容的类型。其中 DTD 是从 SGML 继承而来，而 Schema 是专门为 XML 文档格式而设计，它们都规定 XML 文件的逻辑结构，定义了 XML 文件中的元素，元素的属性和元素之间的关系等。

（2）Xpath,Xpointer 与 Xlink: Xlink 是 XML 标准的一部分，用于定义对 XML 的链接。Xpath 是一门语言，用于把 XML 文档作为带有各种结点的树来查看，使用其可以定位 XML 文档树的任意结点。Xpointer 是对 Xpath 的扩展，它可以确定结点的位置和范围，通过字符串匹配查找信息。

（3）DOM、SAX 和 XML Parser:都是用来解析 XML 文档的接口 API 函数。不同的是，采用 DOM 处理 XML 文档，在处理前要对整个文档进行分析，把整个 XML 文档转换成树状结构放到内存中；而采用 SAX 是事件驱动，每当它发现一个

新的 XML 标记，就用一个 SAX 解析器注册句柄，激活回调函数；而 XML Parser 则是一个处理 XML 文档的软件包，它能减轻应用程序处理 XML 数据的负担。

2.5 本章小结

在本章中我们讨论了本课题开发中应用到的关键技术，主要包括自动化测试技术、跨平台的分布式技术和分层技术，Eclipse RCP 富客户端技术，XML 通讯接口技术，基于 Java 的 GUI 组件 SWT/JFace 技术，Socket 消息通讯技术，Ftp 文件传输技术，以及面向对象设计模式技术等。

3　自动化测试系统需求分析和结构设计

本章从讨论自动化测试系统的总体需求和体系结构设计准则，引导出对系统体系结构和执行流程的研究分析,并讨论了系统的富客户端和调度服务器模块的结构设计等研究情况。

3.1　自动化测试系统总体需求

3.1.1　自动化测试系统的整体需求

通讯业的测试一般流程是先搭建好通讯测试环境，然后按照测试规程手工逐项规程中规定的各项测试步骤。然而，对通讯产品和软件版本的测试，有很大一部分测试工作是进行回归测试，即对通迅产品和软件引入新功能后，对原有功能进行验证性测试。而随着的通讯业务范围的扩大，回归测试的工作量已变得越来越繁重，例如对整个软件版本的语音、数据、短消息、切换、补充业务等方面回归测试下来，测试人员的工作量非常大；并且随着原有通迅技术的成熟，很多通迅产品故障缺陷却越来越少，大部分工作只是一种正确性的验证。由这种行业趋势和技术导向，因而产生了对自动化测试系统的需求。

这种工具不但要对各种版本透明通用，而且在不需要人干预的情况下，能够实现自动控制、自动测试、自动分析数据、得出结论，并高效率的完成测试任务。

（1）节约手工测试的时间,实现测试用例的重用反复执行。能支持成熟的测试规程，编写测试用例对其有效测试，并能够方便移植，从而达到测试用例的重用。

（2）宽测试面，实现基本的通讯业务测试，具易于业务扩展。支持目前通迅的基本业务，如语音、数据、短消息、切换、补充业务等；支持目前基本的网络前后台管理软件测试，且当软件发生变化扩展时，测试系统容易升级扩展。

（3）自动化测试实现的基本功能目标: 实现自动化输入测试参数，自动处理，自动结果比较，过程记录，自动结果分析报告。

3.1.2 自动化测试系统设计目标

自动化测试系统架构理想模型，如图 3-1 所示。系统通过对功能模块粒度进行多层次封装，支持实现自动化测试应用。

（1）数据驱动测试的模式:实现测试数据与测试脚本相分离，这样适合于各种测试业务配置，且能够分别重用测试数据和测试逻辑。

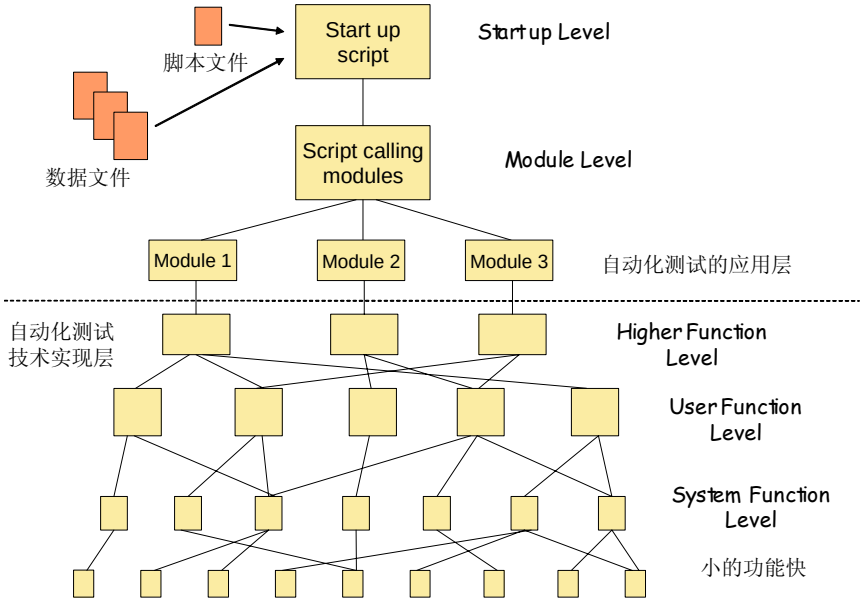

图 3-1　自动化测试系统架构理想模型

（2）测试执行人员可以根据需要定制脚本:只要自动化测试技术实现层能够支持的测试业务，测试执行人员可以增加测试用例，不影响测试人员主观能动性的发挥。

（3）自动化测试的技术实现层与自动化测试的应用层分离，都采取分层的模式:如果自动化测试平台或者测试产品改变，自动化测试的应用层部分，即测试用例部分可以进行平滑的移植。只要改动自动化测试的技术实现层在新的平台或者产品上实现，即可做到测试用例的共享。

（4）系统严格区分前后台的功能分工:后台负责提供各测试小的功能块，前台提供测试用例的编辑功能，在前台抽取出这些小的功能点以便测试设计人员通过组合装配功能点来编辑测试用例。图 3-2 自动化测试系统的层次图，可见平台的前后

台分工划分情况。

由图 3-2 可知，整个系统分为自动化测试的应用层和技术实现层。测试设计人员和测试人员工作于自动化测试应用层，依据测试规程和测试功能点，编写测试用例和进行自动化测试。测试开发人员只需要在自动化测试的技术实现层开发维护好各个执行小的子功能模块，就可以在自动化测试系统中实现上层测试人员所要执行的测试用例。

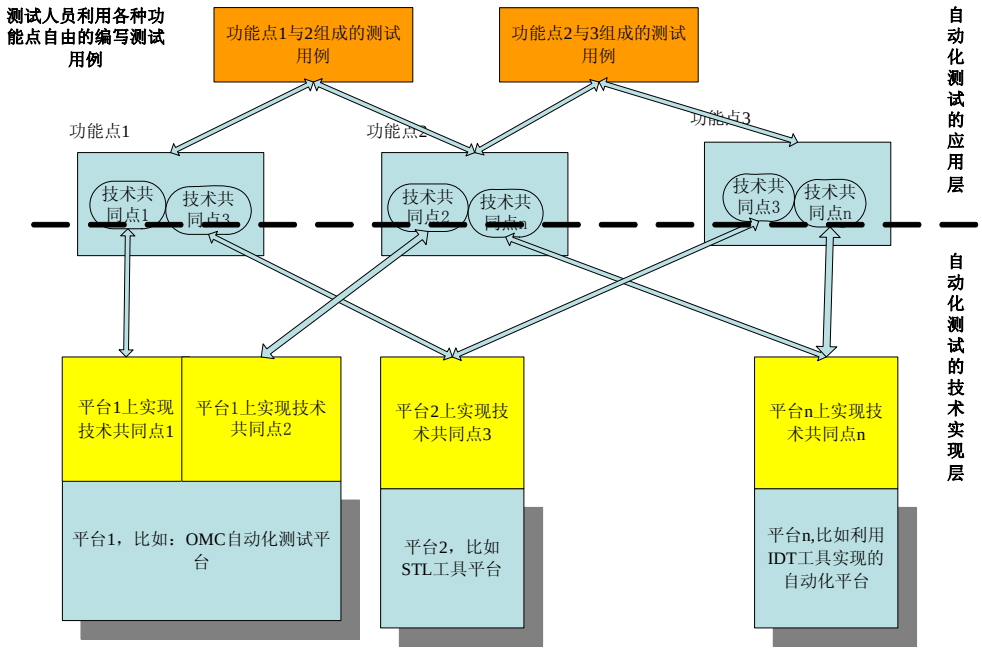

图 3-2　自动化测试系统的层次图

3.2　层次化系统体系结构

3.2.1　系统体系结构设计准则

依据系统实现方案和总体设计，系统体系结构应该既能满足实现系统运行逻辑，又能方便系统的设计开发。从任务开始订制执行到最终到底层实现会经过很多功能模块的交互、调度等，所以我们在架构分层设计的基础上，要求高一级管理低一级的调度协调；低一级的如果有逻辑关系统一提交上级来处理。任务层应有统一的多任务调度，任务间的关系都通过上层处理；单个任务设计为任务对象，由任务管理处理单个任务。

（1）应用层应是富客户端，具有较强的用户交互能力。提供用例编辑器，方便用户组装原子化关键字为测试项模块，并能够给出人性化功能，便于制定复杂的逻辑用例；提供报告查看器，指导用户分析测试报告；同时需要与调度服务器实现多对一的分布式对应关系。

（2）测试用例应交由统一的测试用例调度模块处理，测试用例之间的关系都通过高一层的测试用例调度处理；单个测试用例设计为对象，由测试用例调度模块按照用例逻辑调度分发执行。

（3）执行层的每个测试工具之间的调度通过统一调度服务端处理，彼此之间不直接交互，且它们之间有专用的接口与下层被测模块交互。

3.2.2 层次化系统体系结构

系统按层次可划分为应用层、调度层、执行层和被测系统。自动化测试系统体系结构图，如图 3-3 所示。

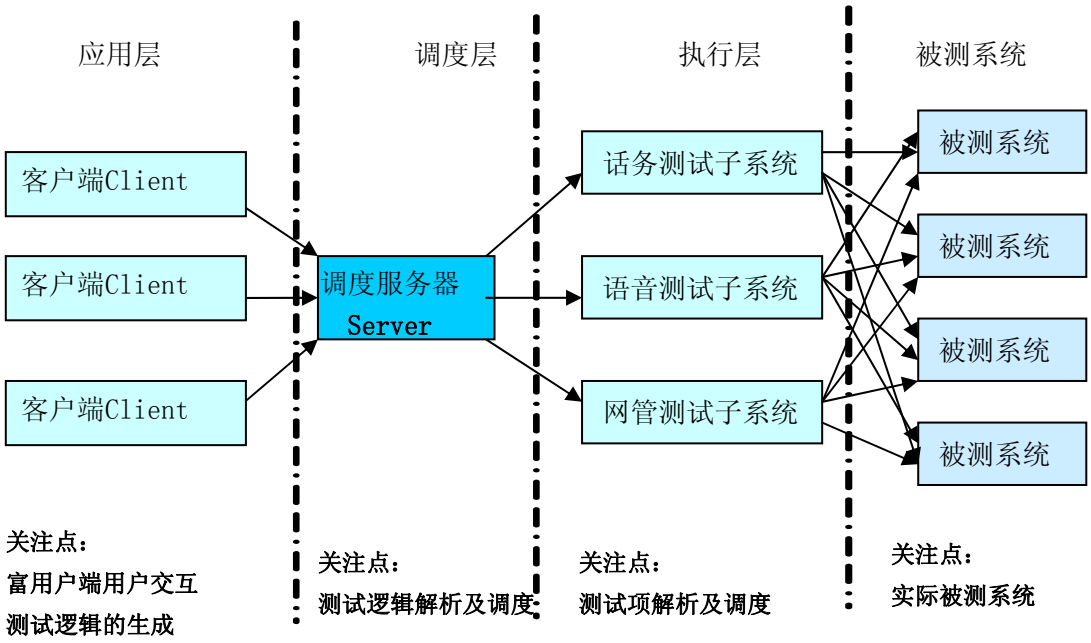

图 3-3　自动化测试系统体系结构图

（1）其中应用层为系统的富客户端,客户端既是用户与系统交互的接口，也是测试任务编辑、测试逻辑生成、测试控制以及测试报告分析展示的前台软件；

（2）系统的调度层为一个调度服务器，其与富客户端和后台执行层均为分布

式对应关系，其为消息处理和终端注册中心，同时其负责按照客户端用户制定的测试任务逻辑来拆分测试用例，并分发到下层执行系统中去；

（3）而系统的执行层则属于和通迅业务联系紧密的功能模块，在系统中其称为测试子系统，用于对后台的被测系统执行各自专一的测试业务；

（4）被测系统是指后台被测试子系统执行测试的通讯软件或硬件设备。

3.3 自动化测试系统执行流程

3.3.1 系统运作逻辑概述

自动化测试系统的用户定位为系统测试人员。在他们执行系统测试时，方便其实现业务规程的自动化运行。由于系统测试用例的复杂性，自动化测试系统必须体现分工分层的概念，要实现系统测试的自动化要有不同人员承担不同角色，完成各自分工，并相互协作。自动化测试系统运作逻辑模型，如图 3-4 所示。

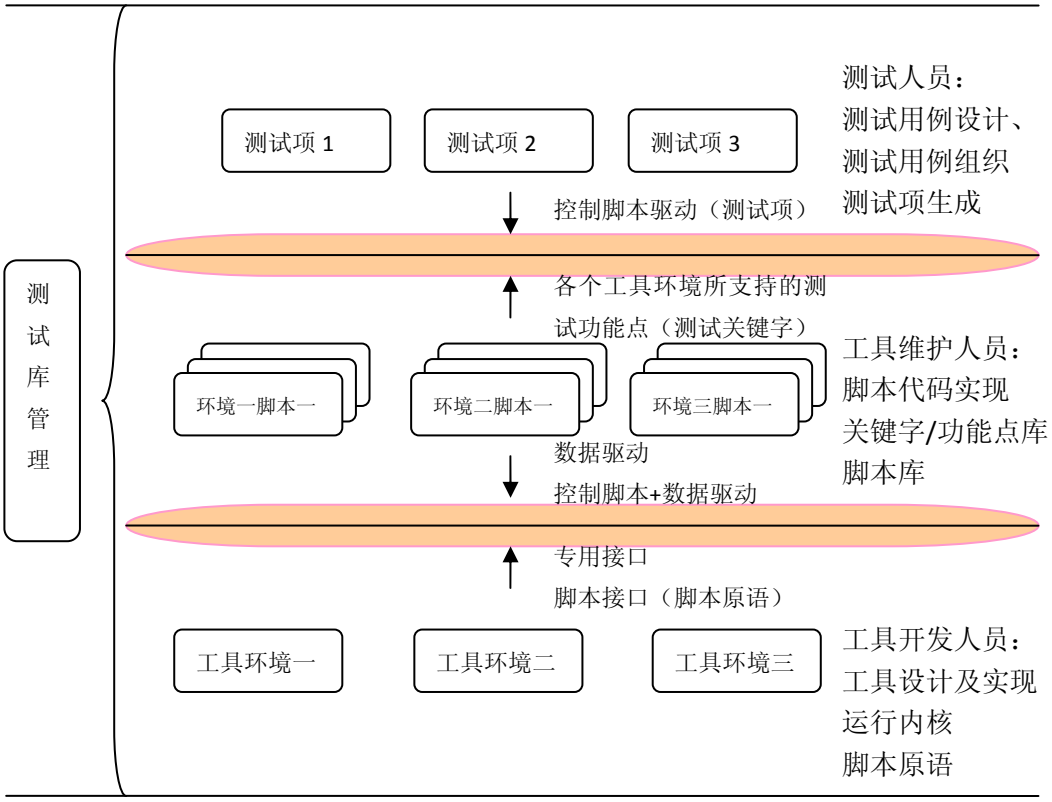

图 3-4　自动化测试系统运作逻辑模型

系统中引入的一个核心概念是关键字，即针对应用层业务功能点抽象出来的描述体，包括关键字名及参数，其对应用层可视为一个可以配置参数的原子化操作。关键字对上层表示设置此描述体即可在运行时实现此功能点，对下层则表示它是对某项执行体的封装描述，根据此描述执行体要实现具体的运行代码。

系统中人员分三类角色，测试人员、工具维护人员、工具开发人员，各自专注于不同层面工作内容。测试人员关注于测试用例的设计与组织，根据测试规程定义自动化测试用例步骤及逻辑顺序，然后通过组合关键字来实现用例的编辑，即不用测试人员关注底层的运行代码就可将整个用例的运作过程描述好。工具维护人员的职责就是将关键字的描述用工具的脚本代码来实现，即相对于同样的业务功能关键字是其描述体，而脚本代码则是其执行体。工具开发人员更多关注的是工具运行平台的开发，做为支持层向上提供最基本的脚本原语，所谓的脚本代码、关键字、测试用例等都可看成是应用层内容，最终都是由脚本原语来实现的。

3.3.2 数据执行流程

由于自动化测试系统结构的多层次性和实时性，使得系统各模块之间的通讯显得非常重要，在系统执行自动化测试时，数据流需要从系统的最上层逐层向下传递数据，最后再依据测试执行结果，逐层向上整合和传输数据。自动化测试系统的数据执行流图，如图 3-5 所示。

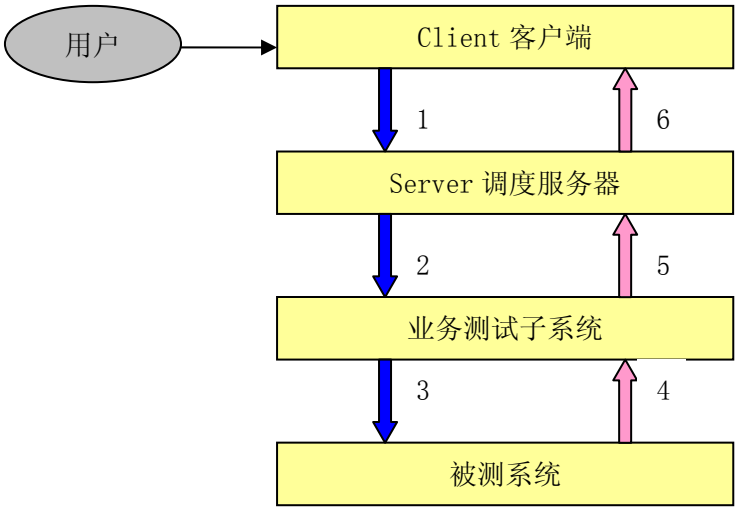

图 3-5　自动化测试系统数据执行流图

在系统中，用户通过与富客户端的交互制定测试任务、下达测试命令和查看分析测试报告。

在用户在客户端编辑好测试任务，用户下达测试任务命令后，系统自动化测试系统数据执行流图来控制执行数据走向，如图 3-6 所示。

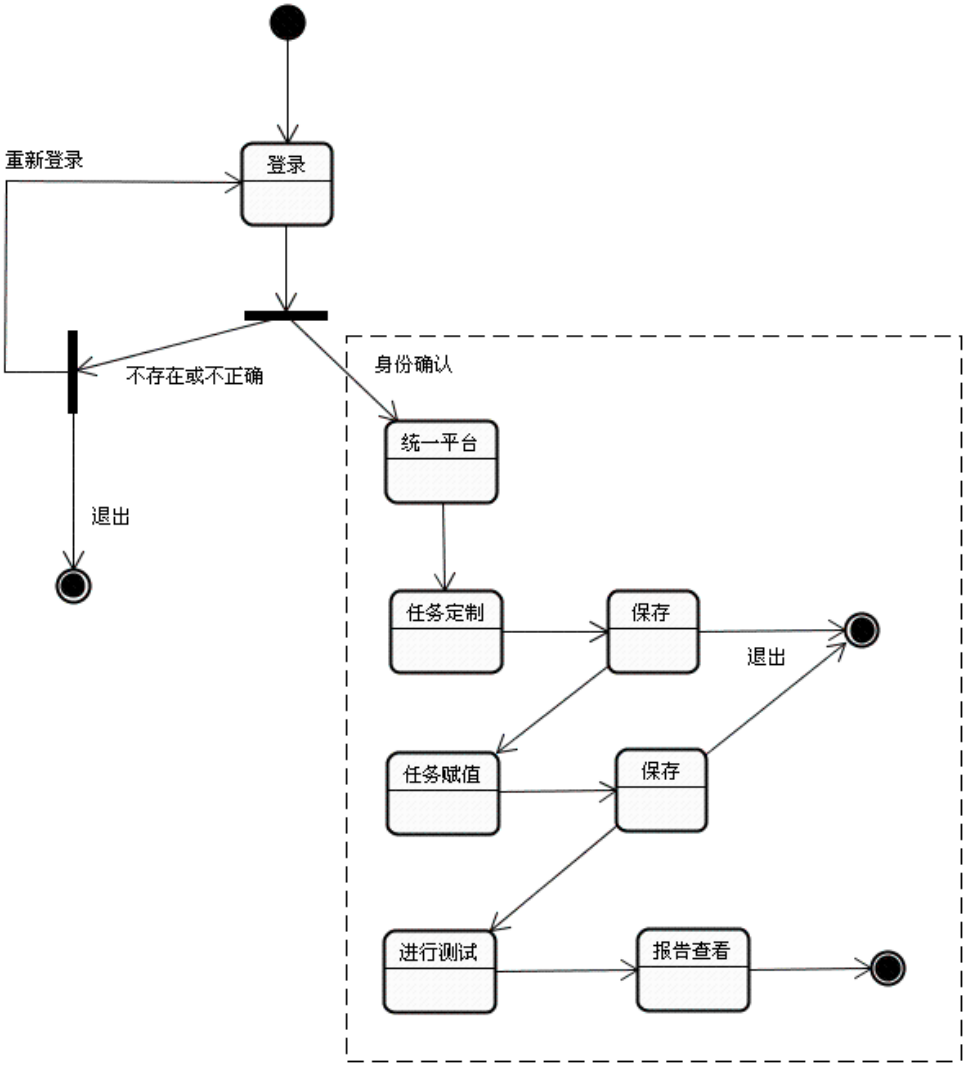

图 3-6　系统状态转换图

（1）客户端向调度服务器注册测试任务，并将测试任务通过 FTP 传至调度服务器执行。

（2）调度服务器拆分每一个测试任务中的测试用例，并且按照每个测试项的类型实时下发到对应的业务测试子系统。

（3）业务测试子系统在接收到调度服务器下发的测试项后，对测试项中的所

有测试关键字（原子化操作）进行拆分，并对相应的被测系统执行这些测试关键字操作。

（4）业务测试子系统收集到其对应的被测系统的测试关键字操作的执行结果或者执行异常，并组装成测试项级的测试项执行报告。

（5）业务测试子系统将测试项级的测试报告和日志异常文件送至调度服务器端。

（6）调度服务器收集到所有业务子系统测试项级的执行报告，按照测试用例和测试任务的逻辑重新组装测试报告，并将组装好的测试报告送至客户端。测试人员通过客户端的测试报告查看器对测试报告进行查看分析。

3.3.3 系统的工作状态转换

自动化测试平台为用户展现的就是一个测试平台，从用户制定测试任务到执行测试，再到最后分析测试报告，系统在用户的操作之下，亦呈现出一系列的工作状态转变。

对于用户而言，与其打交道的就是客户端，因此其所有操作和可见视图主要为客户端为用户展现出来的工作状态。

（1）首先，用户需要登陆系统（平台），并对其进行一些基本的属性设置，在用户身份验证和属性设置完毕后，如果用户身份验证通过，系统将处于其初始状态，否则系统退出。

（2）这时，用户可以进行测试任务定制，可以跟据测试规程来编写自己的测试逻辑，并对测试逻辑进行保存，使所编写的测试用例固化在系统文件中，当然此时用户可以选择退出系统。

（3）任务定制之后，对测试用例的测试逻辑编辑就完成了，用户需要对任务进行赋值，这时用户就需要针对相应的测试现场设定一些测试数据，同样需要对已赋值的任务进行保存固化。

（4）测试任务逻辑和数据均保存完毕后，用户便可进行测试了，通过操作客户端，来启动系统后台对测试任务进行测试，此行系统处于进行测试的状态。这时后台调度服务器将按照测试逻辑拆分测试用例，并分发调度后台测试子系统进行测

试，同时将与客户端实时通信，使用户可以看到运行任务的实时信息。

（5）最后，当测试任务运行完毕，用户通过客户端的报告管理器和报告查看器，对相应的测试报告进行查看分析。

3.4 富客户端分析与设计

3.4.1 客户端总体目标和性能需求

自动化测试系统的客户端为测试人员提供一个统一的操作入口，使用户在一致化的界面上根据测试规程方便地将测试自动化。客户端的总目标是免去用户配置文件和环境的烦琐，用直观和流畅的GUI与用户进行工作交互，平台的后端根据定制的任务完成整个测试过程，最后统一反馈测试结果。

由于客户端在功能上是一个集测试用例编辑、测试任务控制和测试报告展示等功能综合于一体的富客户端平台。故其需要满足实现系统功能模块，对能流畅地与调度服务器实现分布式对应关系；同时又由于客户端是整个系统的操作入口，其直接与最终用户打交道，故其须提供符合用户操作习惯的使用流程，需要满足一些特定的性能需求。

（1）时间性能:对于界面的响应应该控制在合适的时间（1～2 秒），对于长时间作业的操作，应该给出进度条提示。

（2）资源重用性能:对于生成的测试任务结构，要求尽可能地提高重用度，允许用户拷贝到其它地方再进行使用。

（3）可扩展性:对于测试规程的变化所带来的测试用例逻辑复杂化，客户端用例编辑器应该容易扩展支持更多的测试逻辑。同时，对于底层测试业务的增加，客户端编辑器应该能易扩展模拟其业务，以便用户设计出相应的测试用例。

（4）数据驱动性能:将测试任务的数据与逻辑相分离，且分别可以重用。便于测试任务设计人员设计或重用测试逻辑，测试任务执行人员设计或重用测试数据。

（5）易用性:因为自动化测试系统的用户有很大一部分是系统测试人员，而这种用户的编程经验不足，故客户端的操作需要符合通用软件的操作，且客户端需要给出操作说明书和操作提示等。

（6）高容量性:用户在执行测试时，可能一次执行一个测试任务，或者同时执行多个测试任务，这使得客户端和服务器之间的通信显得尤其重要。需要控制上行下行数据，还有测试任务控制数据。因而，客户端的通信需要满足高容量性的要求，应对高容量和网络不好的情况。

3.4.2 客户端总体用例说明

客户端是用户操作平台进行自动化测试的入口，采用富客户端架构，用户通过客户端 GUI 登录平台，进行自动化测试的工作。客户端总体用例图给出系统的总体功能需求，并对各个用例进行编号，方便后面进行细化的描述，如图 3-7 所示。

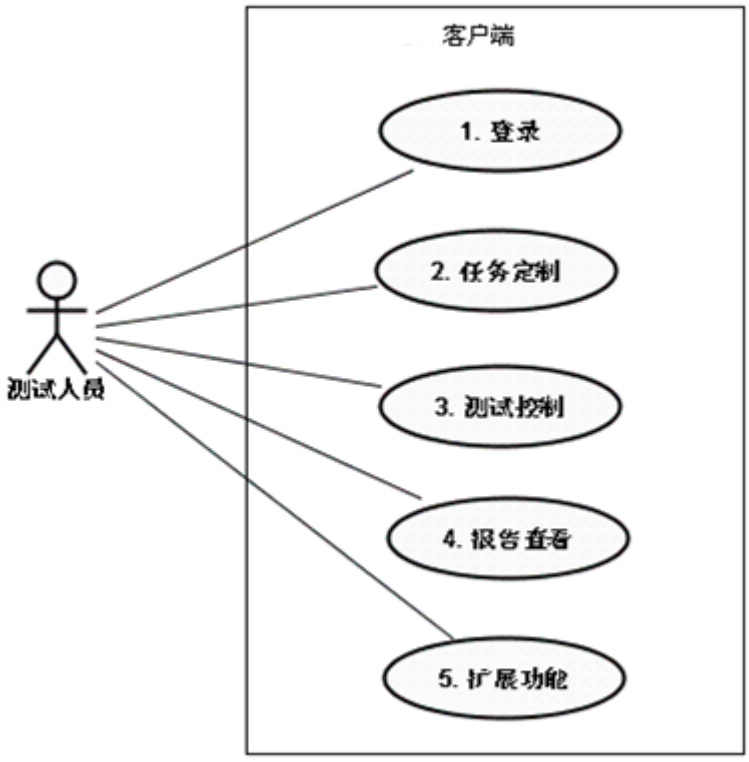

图 3-7　客户端总体用例图

（1）登录：包括用户登录身份确认、注销的常规功能，以及用户属性的一些基本设置。

（2）任务定制：测试人员根据测试规程，把测试的任务划分成平台规定的结构，最后生成各种逻辑上已经确定的“模板”，每次运行从“模板”生成一个实例，并

对该实例进行赋值，成为可以运行的任务。该用例提供实现这种结构的功能。

（3）测试控制：实时查看现在的任务队列，反映任务的运行状态，并提供启动、暂停和停止等的控制功能。

（4）报告查看：实时信息反馈和最终测试结果的数据收集，并能够引导用户分析测试报告结果。

（5）扩展功能：平台要完成核心的测试功能，还必须提供一些其它的辅助工具的功能，如任务运行定时设置、多任务串行化运行、网络性能监视等，为用户提供人性化操作接口。

3.4.3 客户端系统结构

自动化测试系统的客户端采用的是 Eclipse RCP 的富客户端体系结构。因此其界面的 GUI 就相当于是一个 Eclipse 内核和一个自动化测试客户端插件的集合。在前文已介绍了 Eclipse RCP 技术，由此可知，Eclipse RCP 实际上就是一种插件结构，在 Eclipse Runtime 的内核上，扩展自己的插件，即可得到自己的富客户端应用程序。自动化测试系统客户端结构图，如图 3-8 所示。采用插件结构开发客户端的一个优点是，使得软件的可扩展性好，同时开发 GUI 的开发成本降低的同时，能够将开发的重心用在开发客户端所要实现的丰富功能之上。

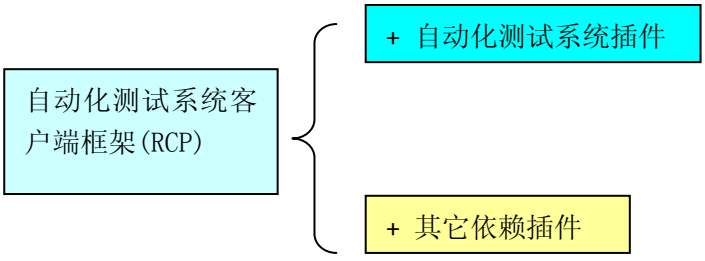

图 3-8　自动化测试系统客户端结构图

由此可见，自动化测试系统客户端实际是一个 Eclipse RCP 的应用，其在 Eclipse 运行时的内核之上，添加了自己功能模块的插件，以及支持该功能模块插件所需要的其它插件。客户端核心功能是提供测试用例编辑、测试运行控制和测试报告查看的功能。自动化测试系统客户端模块图，如图 3-9 所示。

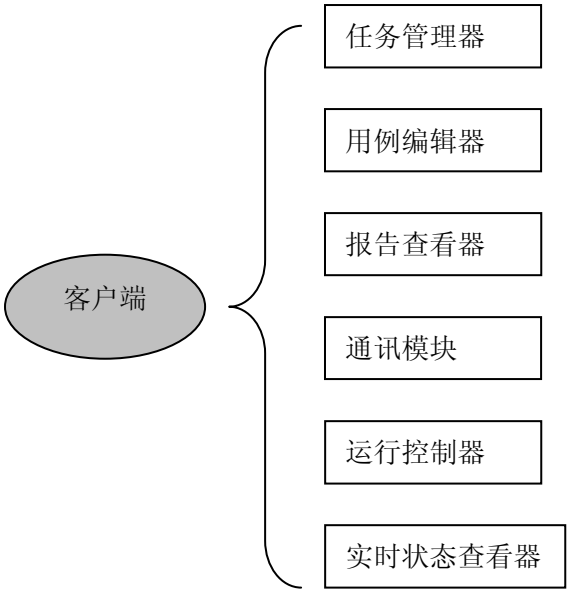

图 3-9　自动化测试系统客户端模块图

（1）任务管理器：提供测试任务组织和管理的基本功能。

（2）用例编辑器：提供对测试用例的复杂逻辑编辑功能和数据驱动的功能。

（3）报告查看器：用于查看测试的最终报告内容，并为用户分析报告导航。

（4）通讯模块：与调度服务器连接、断开、消息通讯和 FTP 文件上下传模块。

（5）运行控制器：控制已经处于运行状态测试用例的模块。

（6）实时状态查看器：查看系统当前正在运行或已经运行完毕的测试任务状态。

3.4.4　客户端的布局说明

按照 Eclipse RCP 插件体系结构和客户端功能模块划分，自动化测试系统客户端的功能模块均封装在自动化测试的插件中，而界面为用户展现出来的，将是按照 RCP 的布局规则，将相近的功能点视图和编辑器封装成一个个的透视图。

从而实现了，用户在进行某项功能时，系统即切换到这种透视图模式之下，为用户展现其操作所需的视图、编辑器和一些局部操作接口。

自动化测试系统客户端界面基本结构，如图 3-10 所示。

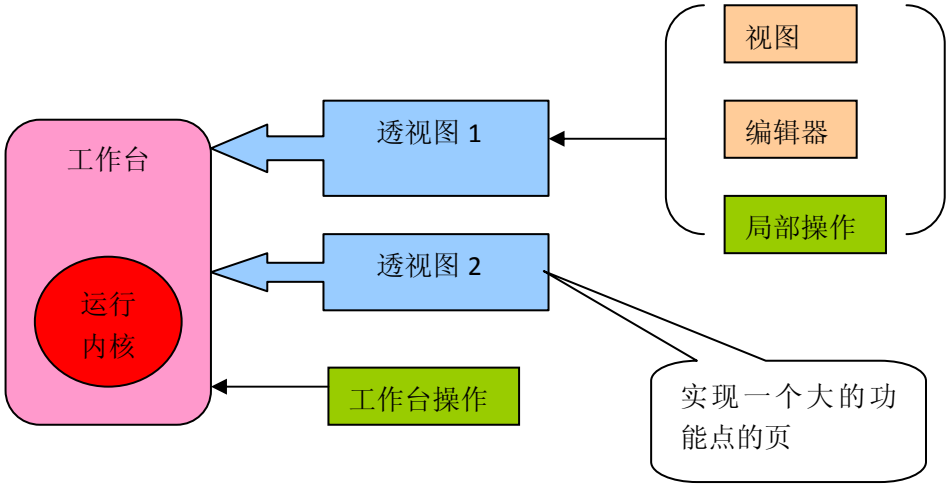

图 3-10　自动化测试系统客户端界面基本结构

其中工作台是由一个或是多个透视图构成，每个透视图可以看成是“一页”，类似于书籍和页的关系。每个透视图都是由各种视图和编辑器按照一定的形式组织起来的。一个透视图用于实现一个较大的功能集合，这些功能的实现由组成它的视图和编辑器共同合作完成。整个平台运行在 Eclipse 提供的内核上，内核本身不关心外部实现。

自动化测试系统客户端按照各个子功能模块之间相似模块的划分，依据 Eclipse RCP 的透视图和视图编辑器的组织原理，将客户端的视图和编辑器做了如图 3-11 自动化测试系统客户端的总体布局规划图所示的划分。

客户端由一个透视图构成，这个透视图包含的所有视图的描述如下：

（1）（V1）任务管理器:提供测试任务组织和管理的基本功能。支持创建、修改、删除、属性查看与编辑测试任务的功能等。

（2）（V2）报告管理器:提供测试报告的组织与管理功能。支持测试报告的删除，查找和重新载入功能等。

（3）（V3）大纲视图:用例编辑器的实体文件(包含测试项和测试关键字)管理。支持测试项的打开、新建、移动、重命名、导入、导出，支持测试关键字的参数赋值和属性设置等。

（4）（V4）属性视图:提供各种节点的属性查看和编辑功能。“任务管理器”中

各种任务节点的属性快速浏览，测试用例节点的部分属性编辑。

（5）（V5）控制台视图:用于打印一些反馈信息。类似普通的 IDE 控制台。

（6）（V6）子系统视图:用于查询连接到调度服务器的测试子系统的状态。

（7）（V7）文件夹视图:Windows 文件夹结构在平台中的映射。用于查找和加载测试项文件。

（8）（V8）模板库视图:提供以模板的形式组织测试项文件。比文件夹视图形式提供更多的描述信息和更加合理的组织结构。

（9）（V9）关键字同步结果视图:展示测试关键字同步结果信息。根据同步信息进行手工修复等操作。

（10）（V10）业务测试项同步结果视图:展示业务测试项同步结果信息。根据同步信息进行手工修复等操作。

（11）（V11）变量重名结果视图:展示业务变量重命名结果,用于追踪重命名点。

（12）（E1）用例逻辑编辑器：提供对测试用例复杂逻辑的编辑功能以及数据驱动功能，支持增删改查，以及拖放，功能编辑和数据编辑等功能。

（13）（E2）报告查看器：用于查看测试的最终报告内容和为用户分析报告结果提供导航支持。

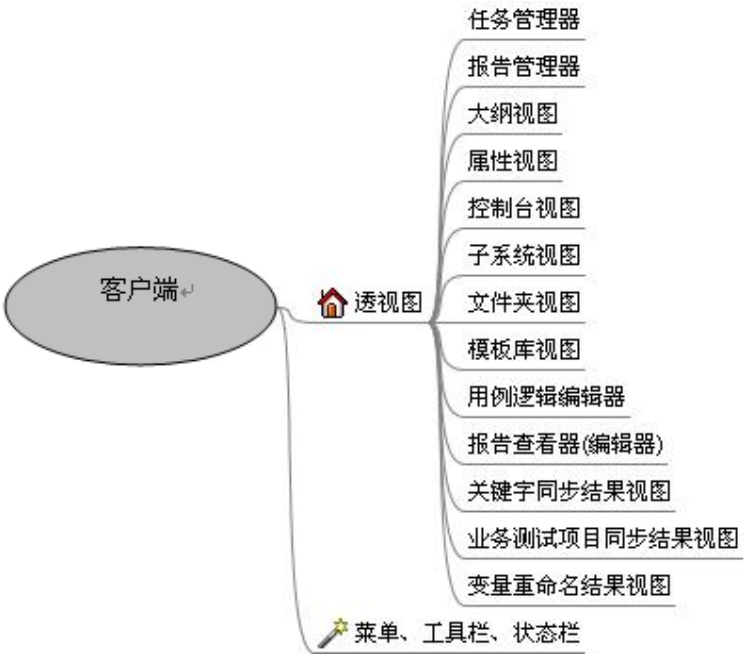

图 3-11　自动化测试系统客户端的总体布局规划图

3.5 分布式调度服务器分析

3.5.1 调度服务器目标分析

由前文介绍自动化测试系统架构可知，调度服务器在整个自动化测试系统层次中起到一个调度支点中心的作用。

（1）一方面调度服务器需要与多个客户端进行分布式对应，这使得调度服务器需要能够实现对多响应终端的处理能力，能够同时响应多方客户的任务响应要求，并且每个客户端可能同时并发多个测试任务；（2）另一方面，调度服务器是对测试用例逻辑拆分，并对下层测试子系统的测试项任务指派中心，因此需要调度服务器能在及时拆分用例逻辑的同时，还能对下层返回报告按照测试任务用例逻辑进行报告的拼装；（3）同时，调度服务器需要能处理测试用例过程数据，不但能够处理测试用例中已经赋值的静态数据，而且能够对测试用例的中间动态数据进行处理。从而体现动态数据驱动的功能。

3.5.2 调度服务器实现方案

调度服务器测试处理模型图，如图 3-12 所示。

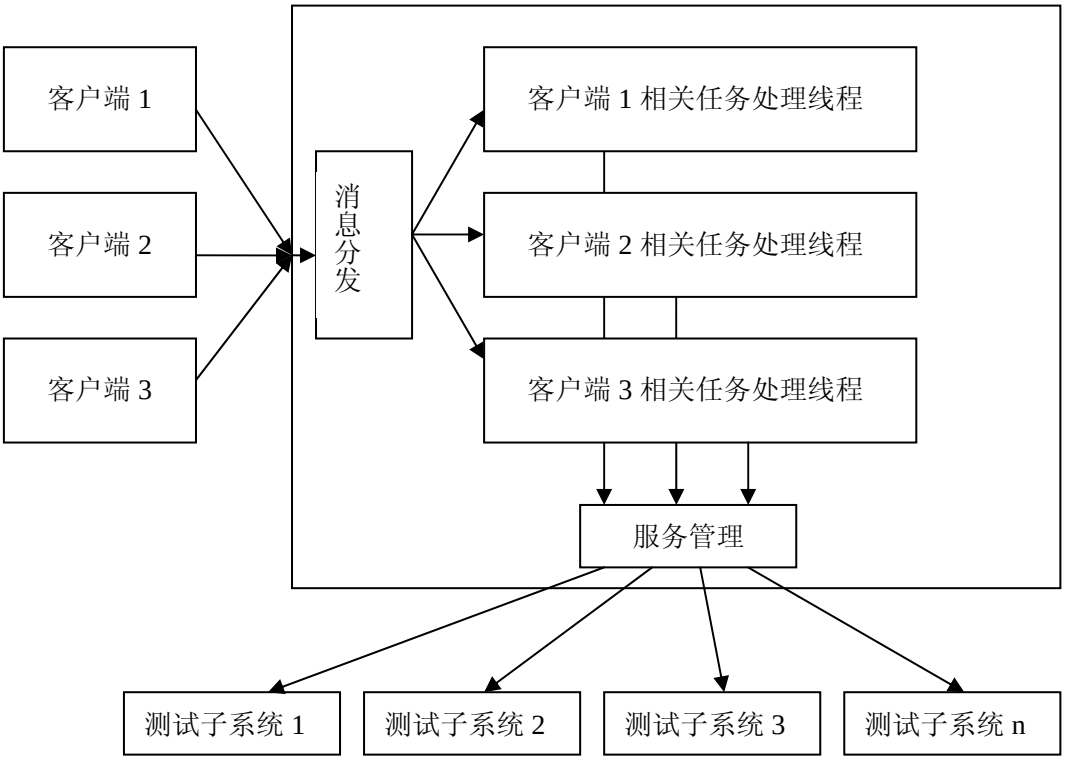

图 3-12　调度服务器测试处理模型图

在自动化测试系统中，调度服务器处于整个系统的调度中心地位，其负责连接客户端和对应的测试子系统，且与客户端和测试子系统都是一对多的关系。因此在系统中，客户端与调度服务器，以及测试子系统与调度服务器都是 C/S 的结构。

对于多客户端请求的情况处理流程如下：

（1）服务器通过消息监听模块监听客户端消息，并将其转给消息分发器。

（2）消息分发器根据请求信息，访问客户端管理模块，获取该客户端的处理线程，并由客户端线程对相应任务的测试实体进行拆分。

（3）通过服务管理模块对客户端任务中相应测试实体进行调度，派发相应的测试子系统来执行测试。

调度服务器的功能模块图，如图 3-13 所示。各个模块相互协作完成服务器分布式调度的功能。

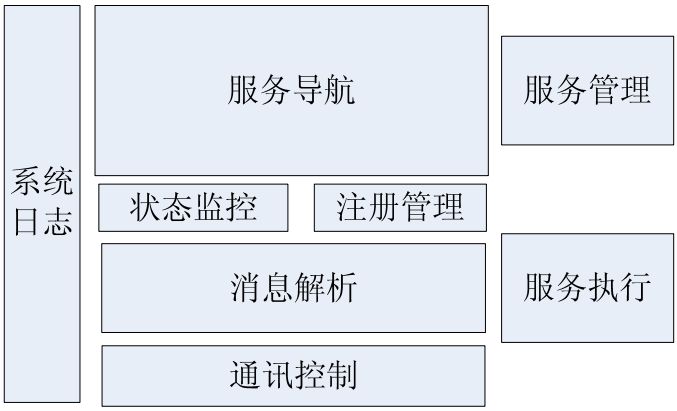

图 3-13　调度服务器的功能模块图

调度服务器框架主要由以上组件组成：

（1）服务导航:根据解析后的请求服务消息的内容调度已在服务器上注册的服务，即查找相应的测试子系统执行相应的测试实体。

（2）服务管理:主要负责管理已经注册的服务组件和服务列表。

（3）服务执行:根据由服务导航调度的服务组件来执行相应的操作。

（4）状态监控:监控链接到调度服务器的各个客户端/测试子系统的相关信息。

（5）注册管理:对链接到调度服务器的客户端或测试子系统进行链接注册。

（6）消息解析:负责把通讯层传来的消息进行解析，并且对调度服务器发出的

消息也进行对应格式的封装。

（7）通讯控制:调度服务器与外界交流接口，主要是采用 socket 消息通迅。

3.6 本章小结

本章首先讨论了自动化测试系统的体系结构设计准则，从而引导出对系统体系结构的研究分析；然后对系统的执行流程分析，研究其数据执行流程和系统的工作状态转换。此后，本章对系统的两个关键层次富客户端和调度服务器模块进行分析设计，研究讨论了自动化测试系统的富客户端总体目标、性能需求、总体用例、客户端系统结构和布局说明等；讨论了调度服务器的实现目标和实现方案等。

4　自动化测试系统客户端实现

根据自动化测试系统的结构设计和执行流程，本章主要分析讨论了富客户端界面结构、用例编辑器、数据驱动和报告查看器等模块的设计实现情况。

4.1　客户端界面结构实现

4.1.1　客户端界面总体布局规划

客户端采用 Eclipse RCP 插件体系结构，按照 RCP 的布局规则，其界面亦与 Eclipse 相象。客户端的视图模式即为透视图模式，在透视图上为用户展现其操作所需的视图、编辑器和一些局部操作接口。

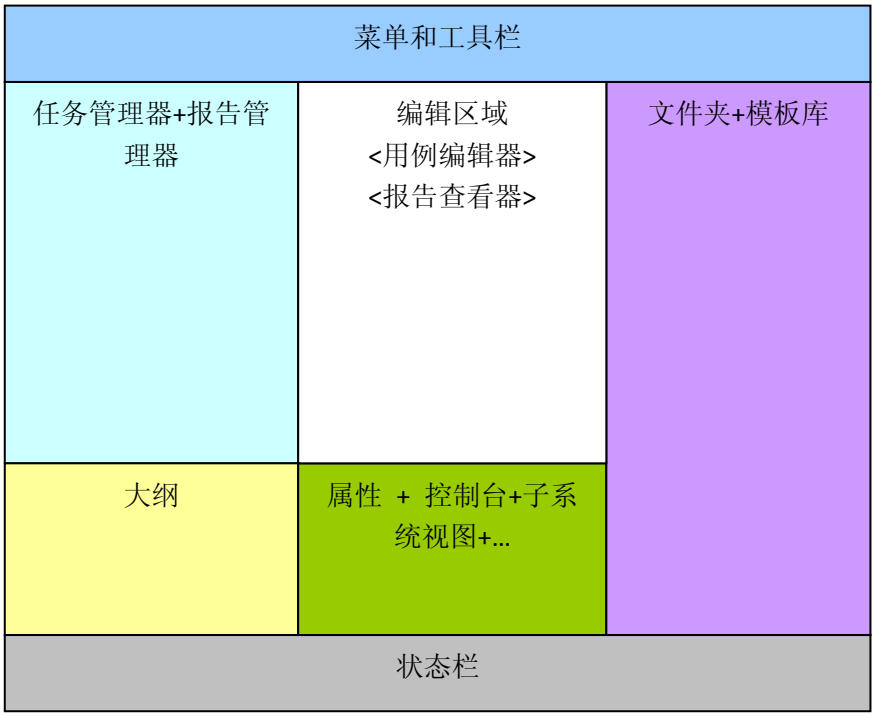

图 4-1　自动化测试系统客户端界面总体布局

由前文的客户端的布局说明，客户端采用上图的界面布局模式。如图 4-1 自动化测试系统客户端界面总体布局所示，客户端采用一个上中下的布局结构。

（1）上端是菜单和工具栏，是一些对应视图模式下用户操作的功能菜单和按

钮。用户可以通过操作菜单和工具栏实现对系统相应的操作，完成一样编辑和运行等操作。

（2）中部是系统的主要显示区，其主要包括一些视图和编辑器。按照默认布局，右边是任务管理器视图、报告管理器视图和大纲视图；中部上方是编辑区域，包括用例编辑器和报告查看器，该区域是用户的主要活动操作空间；中下部是属性、控制台、子系统等一系列视图，用于辅助系统编辑和运行之用；中部的右边部分是文件夹和模板库视图，主要用于方便用户添加寻找测试项。

（3）下端是状态栏，用于显示系统与调度服务器的联机状态，以及当用户执行一些耗时操作时，方便显示一些操作的进度信息。

4.1.2 客户端界面的扩展接口实现

依据客户端的布局规划，在 Eclipse Runtime 扩展出界面视图和编辑器扩展点。图 4-2 为 Plugin.xml 中的扩展示意图。Plugin.xml 为客户端插件接口扩展点的一个注册文件。

```
<extension
      point="org.eclipse.ui.views">
   <category
         id="com.zte.autotest.stl.category.stl"
         name="测试"/>
   <category
         id="com.zte.autotest.stl.category.resource"
         name="资源"/>
   <view
         category="com.zte.autotest.stl.category.stl"
         class="com.zte.autotest.stl.ui.views.TaskView"
         icon="icons/obj16/taskbase.gif"
         id="com.zte.autotest.stl.ui.views.TaskView"
         name="任务管理器"/>
   <view
         category="com.zte.autotest.stl.category.resource"
         class="com.zte.autotest.stl.ui.views.TemplateView"
         icon="icons/obj16/template_view.gif"
         id="com.zte.autotest.stl.ui.views.TemplateView"
         name="模板库"/>
   <view
         category="com.zte.autotest.stl.category.stl"
         class="com.zte.autotest.stl.ui.views.ReportView"
         icon="icons/obj16/report_view.gif"
         id="com.zte.autotest.stl.ui.views.ReportView"
         name="报告管理器"/>
   <view
         category="com.zte.autotest.stl.category.resource"
         class="com.zte.autotest.stl.ui.views.FileBrowseView"
         icon="icons/obj16/folder.gif"
         id="com.zte.autotest.stl.ui.views.FileBrowseView"
         name="文件夹"/>
   <view
         category="com.zte.autotest.stl.category.resource"
         class="com.zte.autotest.stl.ui.views.SubSystemResView"
         icon="icons/obj16/subsystem.gif"
         id="com.zte.autotest.stl.ui.views.SubSystemResView"
         name="子系统"/>
```

图 4-2　Plugin.xml 中的扩展示意图

依据每个视图和编辑器的功能要求，为其定义添加显示所需的控件，并对这些控件的属性和动作操作事件定义特定规则，便可得到大致的客户端界面和界面操作功能。

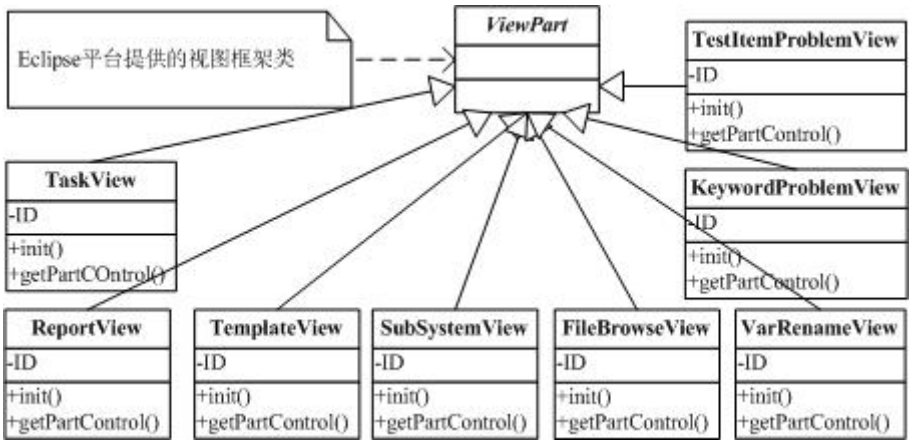

图 4-3　自动化测试系统客户端视图实现类图

图 4-3 为自动化测试系统客户端视图实现类图，依据客户端功能需要，分别由 TaskView(任务管理器视图), ReportView（报告管理器视图），TemplateView（模板库视图），SubSystemView（子系统视图），FileBrowserView(文件夹视图)，VarRenameView（变量重命名视图），KeywordProblemView（关键字问题视图），TestItemProblemView（业务测试项问题视图）这 8 个视图对 Eclipse 平台的视图框架抽象类继承，并对每个视图单独添加其实现控件和控件事件，便得到所有视图。

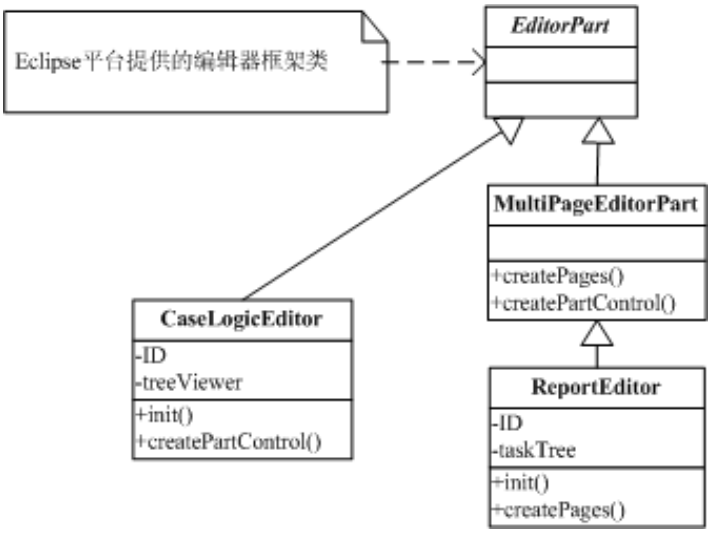

图 4-4　自动化测试系统客户端编辑器实现类图

自动化测试系统客户端编辑器实现类图是客户端 CaseLogicEditor(用例编辑器)和 ReportEditor(报告编辑器)继承图，如图 4-4 所示。其中用例编辑器是直接继承自 Eclipse 平台所提供的编辑器框架类 EditorPart，而报告编辑器由于需要实现多页的显示，故其继承了 MultiPageEditorPart 编辑器抽象类。

用例编辑器中包含有 treeViewer 表示一个用例的树结构，而方法体 createPartContriol()则表示设置用例编辑器的显示规则。报告编辑器中包含有 taskTree 属性，表示相应报告编辑器中所属的测试用例树结构，而方法体 createPages()则表示生成报告编辑器的多个 Tab 显示页，每个 Tab 分别代表报告的一种显示模式。

由于自动化测试系统客户端采用 Eclipse RCP 的插件结构设计，所以其界面 GUI 也具有和 Eclipse 类似的视图和编辑器结构。自动化测试系统客户端基本界面图，如图 4-5 所示。

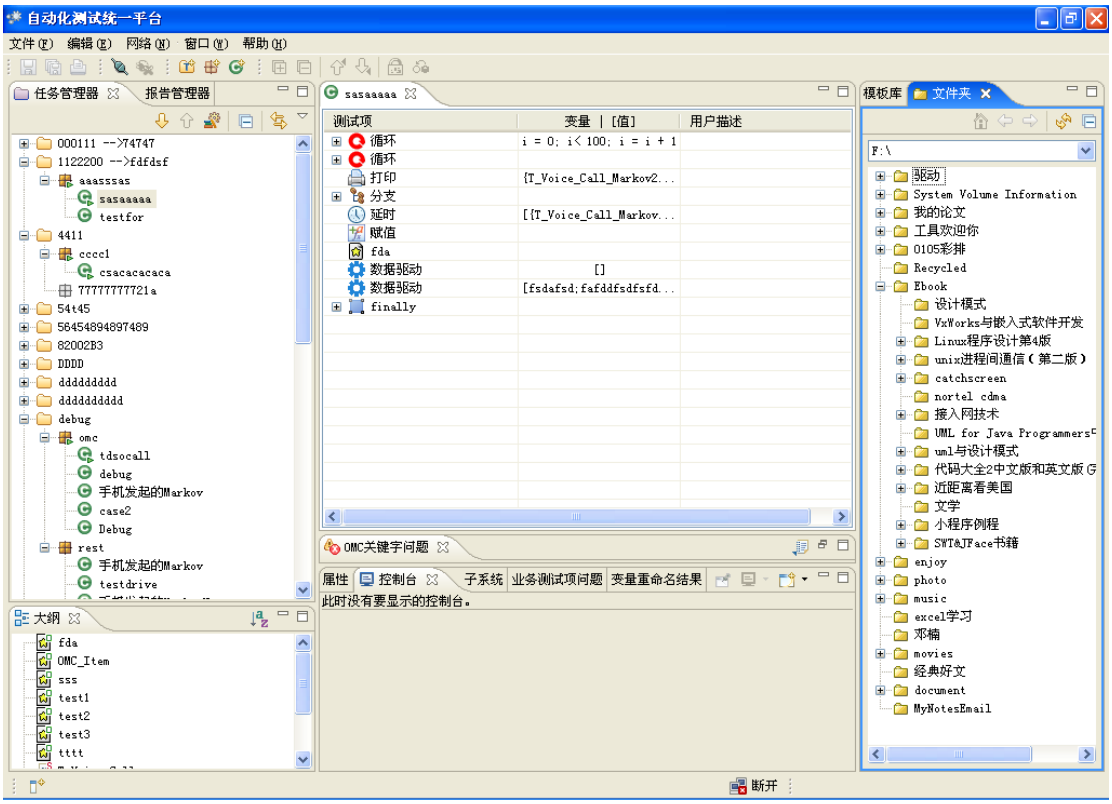

图 4-5　自动化测试系统客户端基本界面图

客户端界面左边有两个视图，分别为任务管理器视图和报告管理器视图，分别用来对测试任务和测试报告进行管理。由上图可见，任务管理器中采用的是一种树

状结构对测试任务进行管理，每个文件夹图标代表一个测试任务，其所包含的子节点代表测试套，测试套的子节点是一个一个的测试用例，也就是测试逻辑的封装体。

4.2 用例编辑器实现

4.2.1 测试用例任务定制

客户端的一个重要的功能模块是用例编辑模块，其为用户提供图形化的测试用例逻辑定制规则，并按照系统测试任务的文件结构生成最终的 XML 测试任务文件以供调度服务器处理。用例编辑器的核心功能即为系统的测试任务定制。任务定制的用例细化图，如图 4-6 所示。

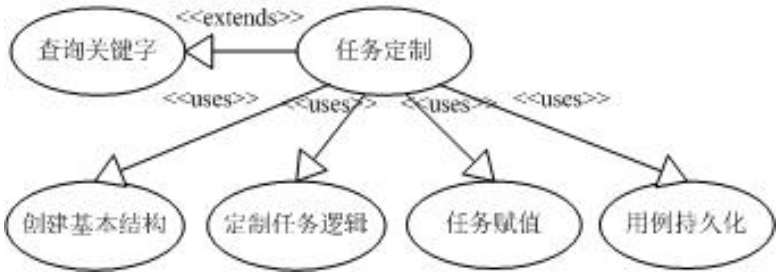

图 4-6　任务定制的用例细化图

用户在客户端进行用例编辑的功能模块可以分为创建测试任务基本结构、定制任务逻辑、任务赋值和用例持久化等。

创建测试任务基本结构为用户按照测试规程建立测试任务、测试套和测试用例的操作过程。定制任务逻辑为在新建的测试用例中设置用例运行逻辑节点和测试项等。任务赋值为给测试用例各逻辑节点赋值，设置测试数据的过程等。用例持久化为用户选择保存测试任务，完成对测试用例逻辑和测试数据的保存，生成测试任务文件的过程。查询关键字是对系统中测试的原子化操作单元进行查找的功能模块。

由于客户端采用 Eclipse RCP 结构设计，所以用例编辑器在客户端相当于是在自动化测试客户端插件上新增一个编辑器扩展点。故在开发该功能模块时，首先需要进行此扩展。用例编辑器插件扩展示意图，如图 4-7 所示。

图 4-7　用例编辑器插件扩展示意图

4.2.2　定制任务逻辑

在实现用例编辑器模块中，实现的难点在于定制任务逻辑模块。测试任务中具体的复杂逻辑关系体现在测试用例中的各个逻辑节点和测试项。用例编辑器支持的测试用例逻辑实质上就是一种对测试子系统原子操作的可视化编辑。系统也正是通过对原子操作的封装和各种测试逻辑节点的可视化包装，从而降低对测试人员编程技术的要求，而将工作重心放在测试规程和分析测试所发现的故障之上。

用例编辑器支持具体的逻辑关系包括：顺序、分支、循环、并行、时钟、数据驱动等，并且各种逻辑关系要求可以嵌套。支持的逻辑关系用例逻辑关系图，如图 4-8 所示。

（1）顺序：测试项之间的先后执行次序

（2）分支：根据参数的值选择进行不同的分支点，类似于程序语言中的 if else 和 switch 的功能。分支点的多少由用户创建，每个分支都会带一个默认分支点。

（3）循环：重复执行操作，类似程序语言中的 for 的功能；单体循环和组合循环的功能差别在于组合循环的一个过程有可能包含多个测试项或是其它逻辑，单体只是对单个测试项进行循环。

（4）跳出和继续：类似编程语言中 break 和 continue，用户可以控制用例的逻辑直接跳出或是绕过当前循环，但是注意跳出不是跳出循环，而是跳出整个用例，直接进入到 finally 节点。

（5）打印：打印功能，用户可以设置打印常量、变量或表达式信息。

（6）并行：几种操作同时进行；为了表示出并行的效果，在并行节点下面包含有若干通道，这些通道的数量由用户创建决定，每一个通道表示要执行并行的一个总体，客户端会对这个通道进行并行操作，而不管里面包含的具体内容。

（7）时钟:延时的作用，使用执行停顿用户指定的时间，这个时间可以手动设置。

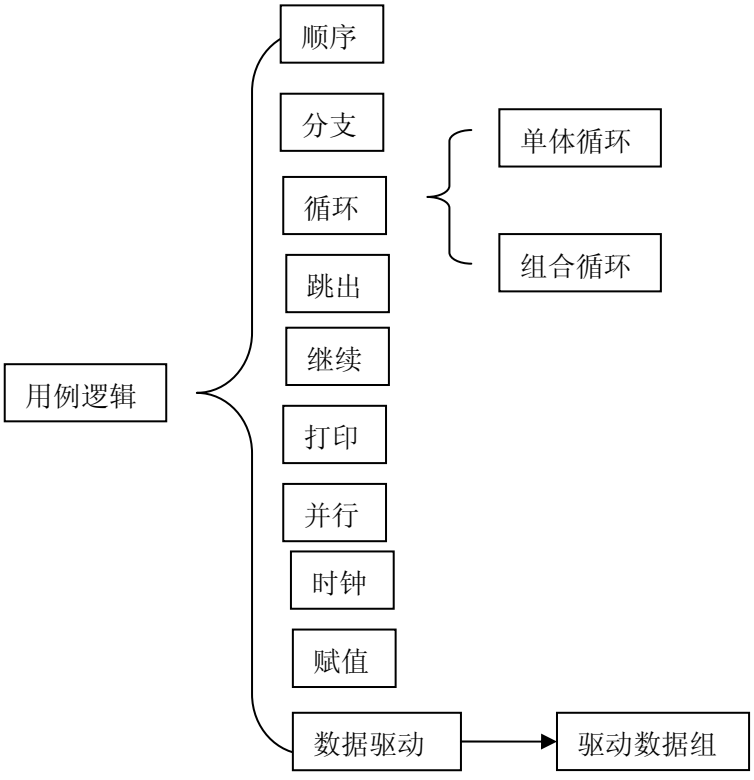

图 4-8　用例逻辑关系图

4.2.3　用例编辑器实现

用例编辑器的实现主要分为界面编辑器事件实现部分、用例编辑处理器部分和底层节点模型封装部分等。用例编辑器实现类图，如图 4-9 所示。

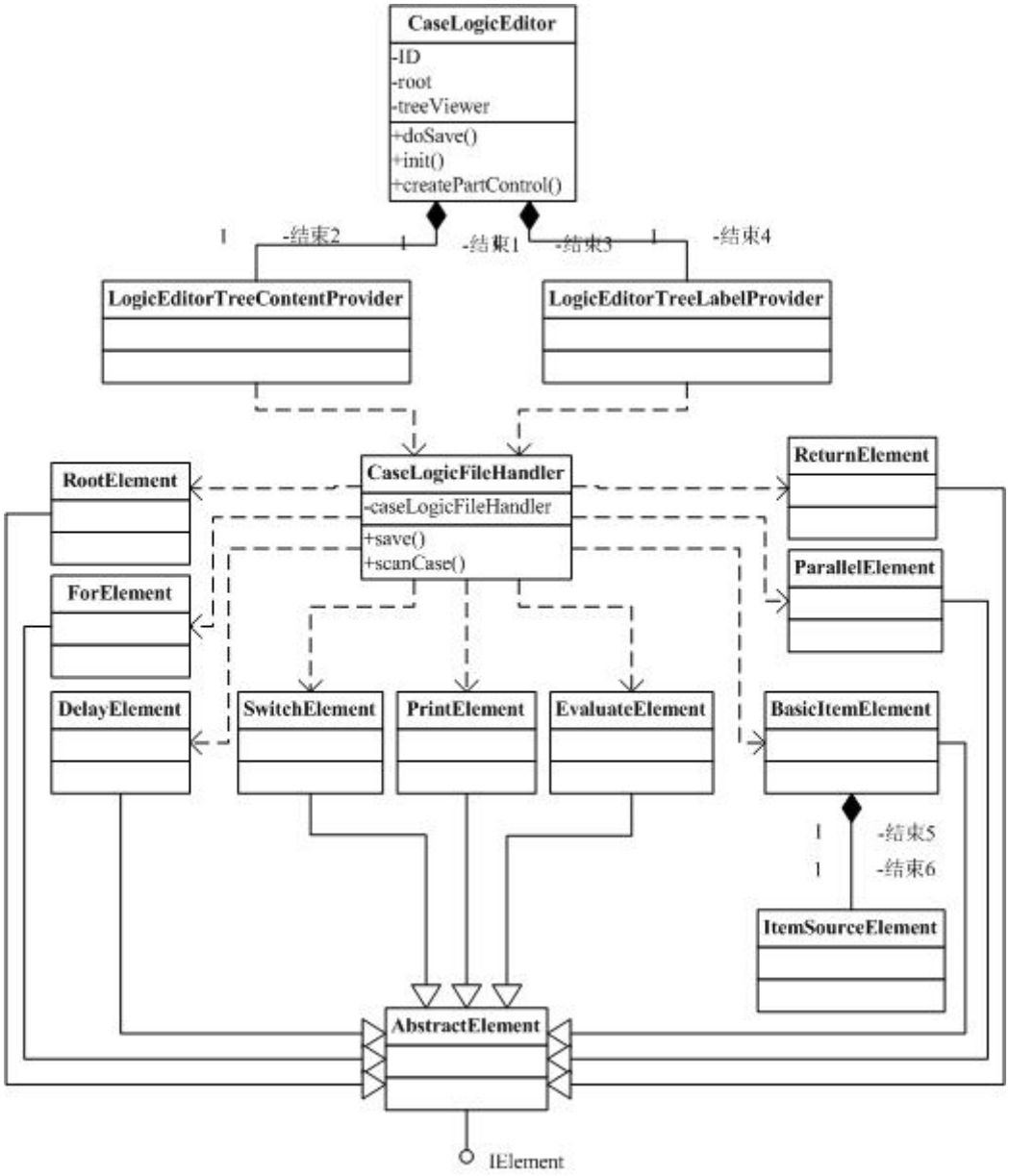

图 4-9　用例编辑器实现类图

（1）CaseLogicEditor 是用例编辑器类，由其对插件该扩展点的实现。其为用户展现的界面模型是一个树状表格，用树结构来表示用例的逻辑关系，而用表格来方便用户编辑和前台显示。类的 ID 属性表示用例编辑器 ID,为客户端扩展点 ID 号，能够方便其他模块查找调用用例编辑器； treeViewer 属性是用例的逻辑树，其树结构的每一个树节点项均表示一个逻辑节点，这些逻辑节点可以是普通的逻辑节点，也可以是测试项节点；root 属性是用例逻辑的根节点，用于方便获取用例逻辑。

（2）LogicEditorTreeContentProvider 和 LogicEditorTreeLabelProvider 为 Case LogicEditor 的 treeViewer 属性逻辑树的内容和标签填充器，其屏幕了用例编辑器复杂的内容节点获取和标签显示。

（3）CaseLogicFileHandler 是用例编辑器处理器，其主要用于测试用例的保存生成符合调度的用例文件，以及将用例文件解析成用例逻辑树结构。

（4）RootElement，ForElement，DelayElement，SwitchElement，PrintElement，EvaluateElement，ReturnElement，ParallelElement 和 BasicElement 均是对 AbstactElement 的扩展节点，均为实现用例逻辑的树节点模型。AbstactElement 为用例逻辑任务树抽象节点，其定义了逻辑节点的一些共性特征；RootElement 为用例逻辑根节点，其包含了一些用例特征的属性和方法；BasicElement 为测试项节点，其为 ItemSourceElement 测试项实体文件类在测试用例中的映象，一个测试项实体文件类在同一个测试用例中允许有多个映象。

在客户端按照任务定制的流程便可实现用例编辑。从用户的角度看，用例编辑器的功能就是由一个树型表格编辑器和附庸其上一些事件功能组合而成；而从自动化测试系统来看，用例编辑器就是一个生成系统接口格式用例文件的编辑单元。用例编辑器的实现模型图，如图 4-10 所示。

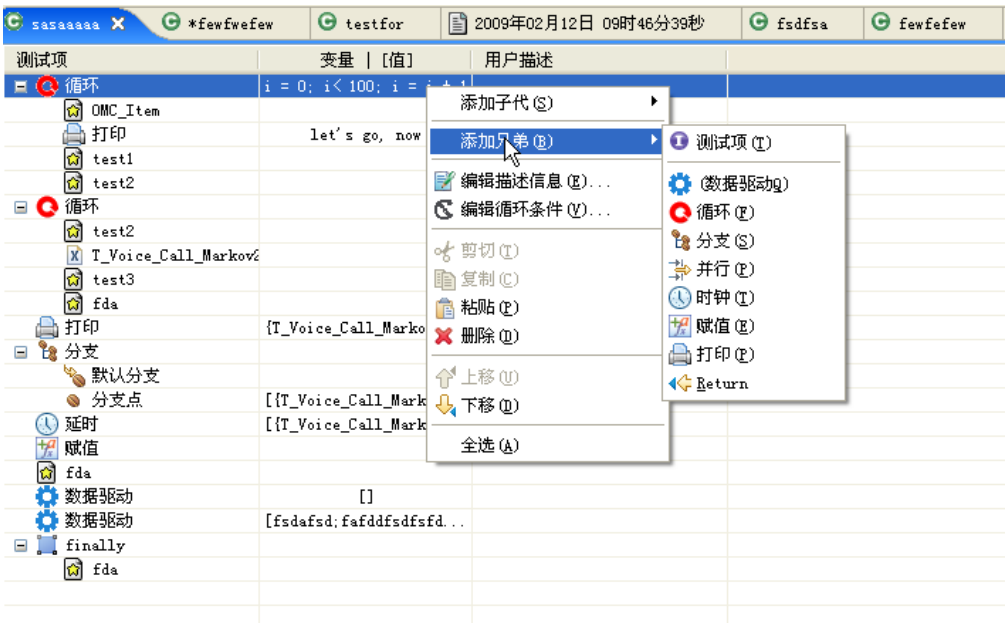

图 4-10　用例编辑器的实现模型图

图 4-10 中的树型表格即为用例树状结构，也是用例编辑器的显示界面，表格的第一列表示节点，第二列表示节点内部赋值情况，第三列表示节点描述信息，用户可以按照测试规程修改用例逻辑；同时用户可以对逻辑节点选择右键功能菜单来对其进行赋值或修改，由于不同的逻辑节点对应不同的逻辑功能，因而其所对应的右键菜单项也都不一样。

4.3 数据驱动模块实现

4.3.1 数据驱动模块内部关系

数据驱动是为了实现用例复杂逻辑与测试数据相分离而开发的一个模块，其功能依然是扩展用例编辑的灵活性。因此，可以说数据驱动模块实质上是用例编辑器模块的一个扩展子模块。数据驱动模块结构图，如图 4-11 所示。

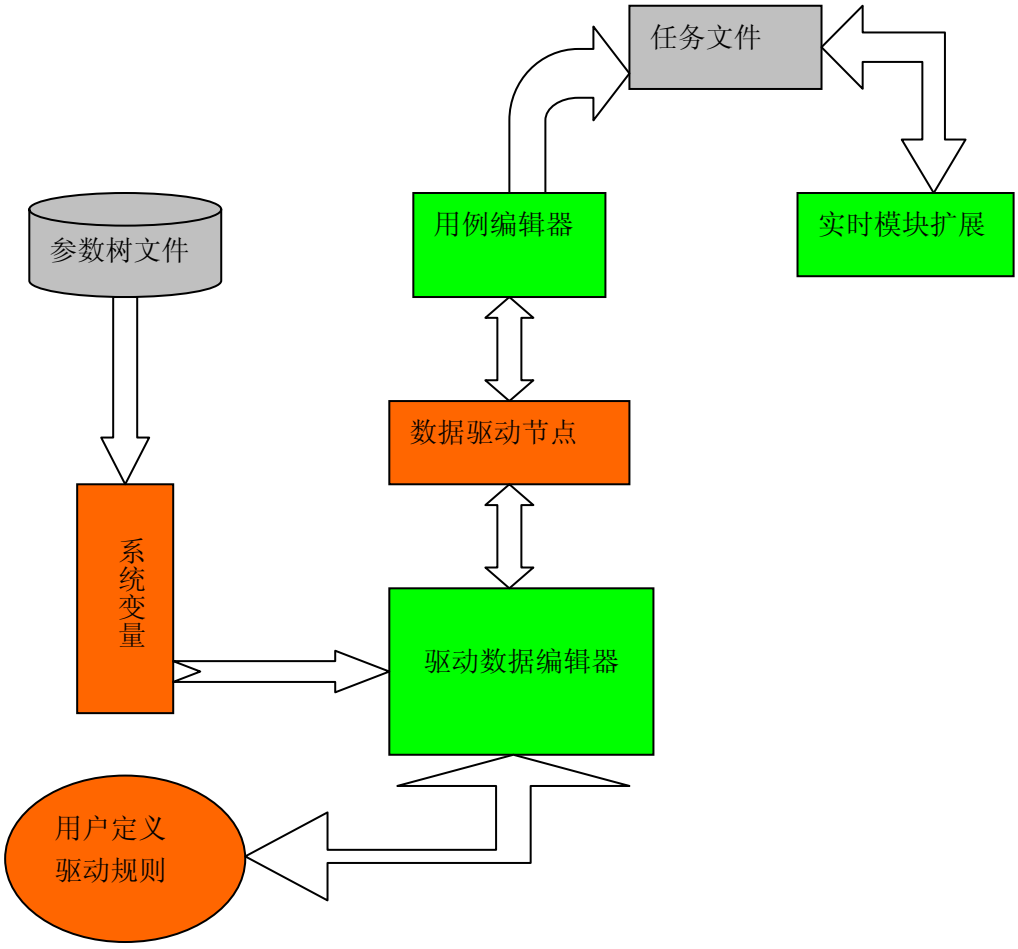

图 4-11　数据驱动模块结构图

数据驱动的核心部分在于驱动数据编辑器，用户正是通过在驱动数据编辑器上定义测试的驱动规则，从而生成数据驱动式的测试用例，以便后台调度服务器运行调度。

数据驱动编辑器与用例编辑器打交道的接口是数据驱动节点，用户正是依据测试规程在数据驱动编辑器中生成好数据驱动节点，再将数据驱动节点作为一个整体赋值到用例编辑器中，从而实现带数据驱动的测试用例任务。

4.3.2 数据驱动接口示意

在用例编辑器中，数据驱动节点可看作是一个类似于循环节点的容器类逻辑节点。用户通过与数据驱动编辑器交互，定义好数据驱动规则后，便可借用用例编辑器接口，将数据驱动节点数据生成固化的 xml 测试用例文件。数据驱动节点接口示意图，如图 4-12 所示。

在数据驱动节点接口示意图中，drivedata 表示数据驱动节点，其中包含有 variables 表示数据驱动变量，data 表示驱动数据。而 data 节点中又饮食有静态数据 group 组和动态数据 groupstext。在数据驱动编辑器的主要功能是依据数据驱动接口生成用户规则的驱动数据，并且实习驱动数据的增、删、修改和规则定义等功能。

```
<drivedata>
  <variables>
    <var name="carrierId" desc="载频号" type="int" />
    <var name="cellName" desc="小区名称" type="String" />
  </variables>
  <!--
  新增加属性 isdynamic 标识是否是动态数据，
  true表示动态数据；false表示静态数据，在客户端显示及
  服务器调度时，只有一种形式生效
  -->
  <data ignoreblank="true" isdynamic="true/false">
    <group carrierId="1" cellName="11" execute="false" />
    <group carrierId="2" cellName="12" execute="true" />
    <group carrierId="" cellName="13" execute="true" />
    <group carrierId="4" cellName="14" execute="true" />
    <group carrierId="" cellName="15" execute="true" />
    <!--  客户端使用测试项返回值变量，在调度时换成实际值，
      格式如'2#12||4#14' -->
    <groupstext>{测试项名:关键字ID:返回值名}</groupstext>
  </data>
</drivedata>
```

图 4-12　数据驱动节点接口示意图

4.3.3 支持数据驱动的系统运行流程

带数据驱动的自动化测试运行流程图，如图 4-13 所示。

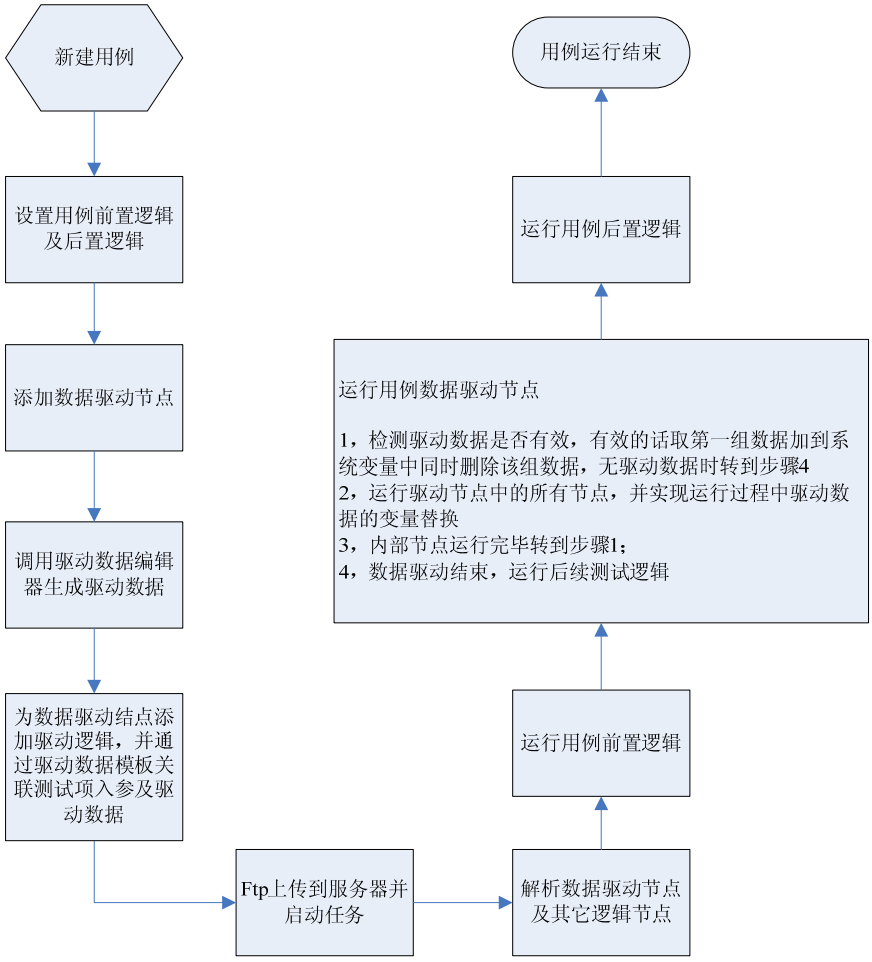

图 4-13　带数据驱动的自动化测试运行流程图

描述了在系统中，从用户新建了测试用例并定义数据驱动，到最后测试用例的运行解析完成等。

（1）新建测试用例，并在测试用例相应的测试逻辑处增加数据驱动节点。数据驱动节点相当于一个包含有驱动数据的容器节点，用来对其容器内的子节点运行逻辑进行驱动。

（2）调用数据驱动编辑器，对数据驱动节点进行驱动数据定义和编辑。可以设置动态数据和静态数据，并选择运行所需要的数据格式。

（3）对数据驱动节点进行子节点和逻辑赋值，生成数据驱动节点驱动逻辑。

（4）启动任务，运行解析测试用例。由调度服务器开如执行调度后台测试子系统执行设置好的测试任务。

（5）调度解析每组驱动数据。当解析到数据驱动节点时，系统将按照用户设置好的驱动数据组或者动态数据依次循环执行数据驱动节点中的测试逻辑。

（6）用例运行结束。

4.3.4 驱动数据编辑器的实现

驱动数据编辑器的实现类图，如图 4-14 所示。

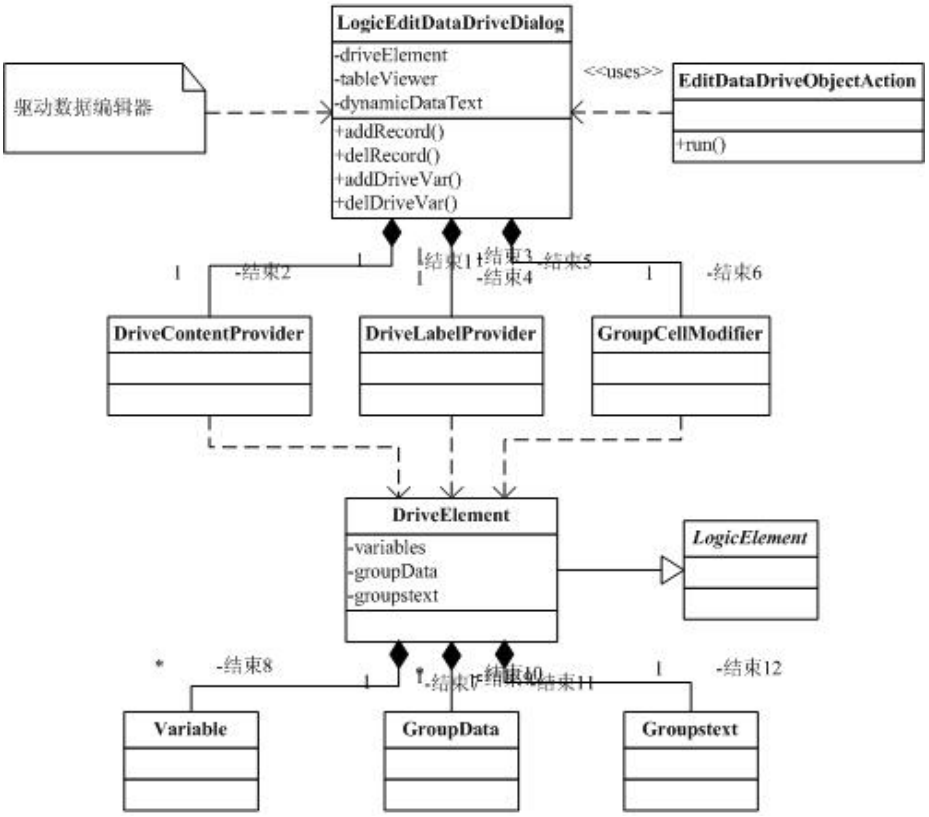

图 4-14　驱动数据编辑器的实现类图

该模块的设计，严格按照 MVC（模型-视图-控制器）的设计层次进行设计。

（1）LogicEditDataDriveDialog 是驱动数据编辑器的界面对话框类。用户可通过操作界面 EditDataDriveObjectAction 触发件件来打开 LogicEditDataDriveDialog 驱动数据编辑器。同时在 LogicEditDataDriveDialog 类中，包含有数据驱动节点等属性，以及增加记录、删除记录、增加驱动变量和删除驱动变量等方法。

（2）驱动数据编辑器类图的第二层次包括 DriveContentProvider, DriveLabel Provider 和 GroupCellModifier 三个控制类。其中，DriveContentProvider 是内容提供器，为 LogicEditDataDriveDialog 提供内容显示；DriveLabelProvider 是标签提供器，为 LogicEditDataDriveDialog 提供文字和标签的显示信息；GroupCellModifier 为内容修改器，其为 LogicEditDataDriveDialog 的静态驱动数据表格提供修改功能。

（3）DriveElement 为数据驱动逻辑节点，用于在内存映象中存储逻辑数据，并在用户选择保存时，将其驱动数据映射为测试用例的 xml 文件内容。DriveElement 是继承自抽象的 LogicElement 节点，从而使数据驱动节点很容易与用例编辑器结合。

（4）Variable 为驱动变量类，用于定义数据驱动的驱动变量的属性和方法模型；GroupData 为静态驱动数据类，用于定义静态驱动数据的格式和调用规则等；Groupstext 为动态驱动数据类，用于定义动态驱动数据的来源和调用规则等。

以下是数据驱动的实现示意图，按照数据驱动静态和动态数据的区别，数据驱动静态模式示意图，如图 4-15 所示。

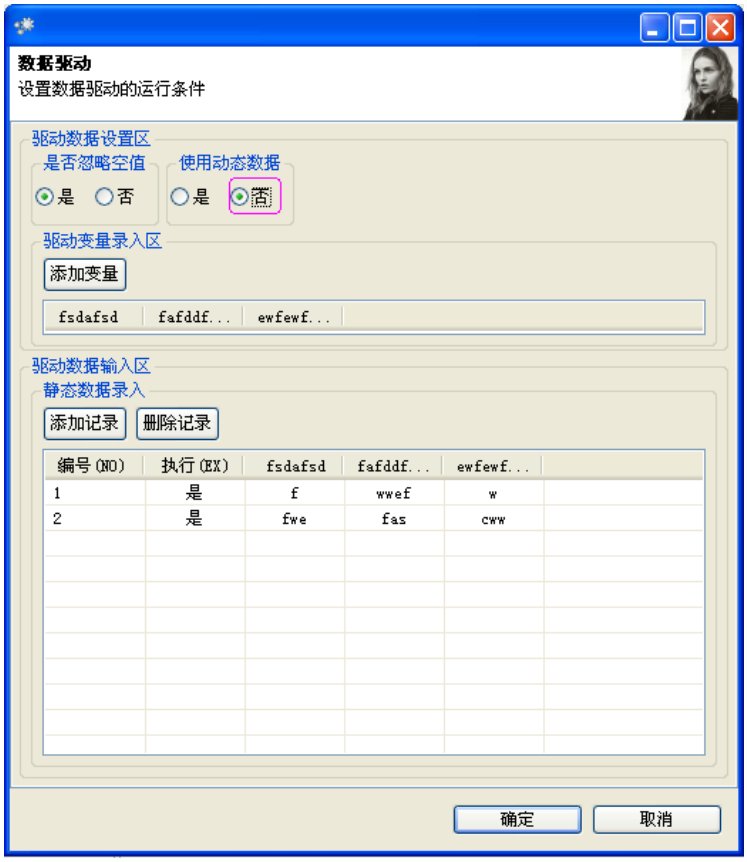

图 4-15　数据驱动静态模式示意图

数据驱动动态模式示意图，如图 4-16 所示。为了保持驱动数据编辑器入口的统一性，客户端采用动态布局方式，依据用户选择的驱动数据模型来动态调整界面布局。测试人员可以根据测试用例场景选择设置相应的驱动数据格式。

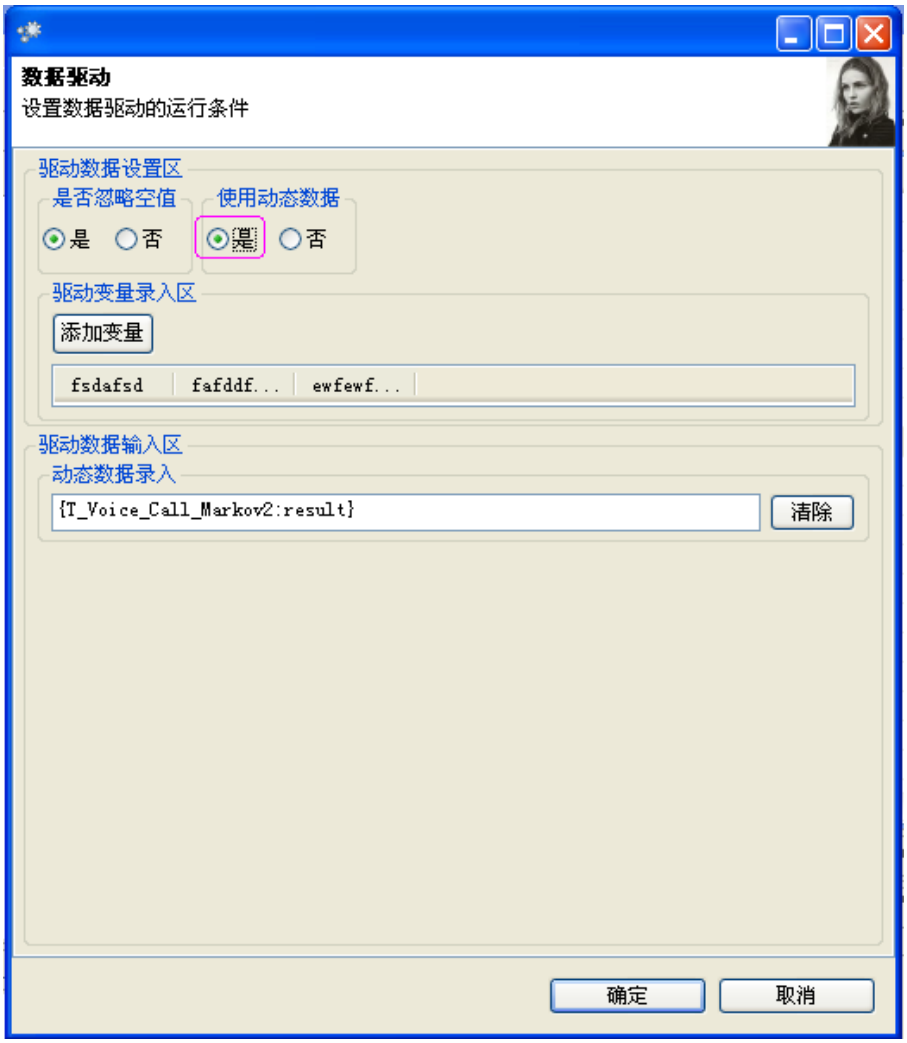

图 4-16　数据驱动动态模式示意图

数据驱动编辑器界面主要分为两部分，上部分是驱动数据设置区，下部分是驱动数据输入区。其中驱动数据设置区又分为数据模式的设置和驱动数据变量的设置；而驱动数据的输入区，则依据驱动数据模式可以动态显示静态数据的表格或者动态数据录入方式。测试人员按照用例编辑原则设置好驱动数据和数据驱动逻辑之后，当运行测试任务的数据驱动逻辑时，调度服务器会先解析出驱动数据和驱动逻辑，然后将每一组驱动数据记录对该子逻辑进行一次运行驱动。

4.4 测试子系统返回值模块实现

4.4.1 测试子系统返回值执行流程

由于自动化测试系统的层次结构架构，早期的系统版本中，调度服务器与测试子系统之间的数据流向为单向数据流。调度服务器对测试任务进行拆分和分发至测试子系统，测试子系统在执行完的测试子任务后，将测试结果回送至调度服务器，再由调度服务器合成传送至客户端。这种模式有个缺点就是只能做到数据的静态传输，而不能实现动态传输处理。为解决这个问题，对系统的功能扩展，增加了返回值功能，实现对测试任务子系统中间数据的动态使用。带返回值的系统数据执行流图，如图 4-17 所示。

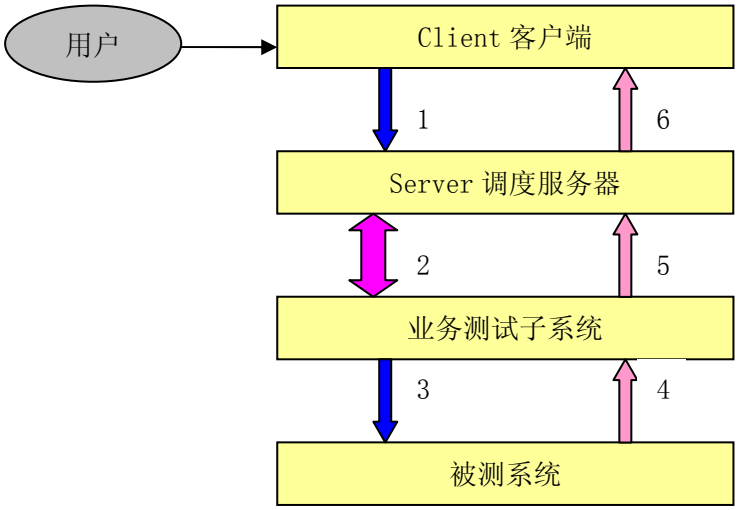

图 4-17　带返回值的系统数据执行流图

数据执行流的变化在于调度服务器与业务测试子系统之间的数据执行。由原来的调度服务器直接下发测试子任务给业务测试子系统，业务子系统在运行完测试子任务后上传测试结果给调度服务器，增加了业务测试子系统在每执行完测试关键字之后，需要实时返回测试关键字返回值到调度服务器的数据传递模式。这种模式使得调度服务器可以在测试逻辑中动态的调用这些返回值。而在系统的客户端，则需要对用例编辑器进行扩展，添加支持返回值的一系列编辑接口。

4.4.2 测试项返回值实现

测试项返回值模块算法实现的难点在于当对用例编辑器中的节点引用测试项

返回值时，需要依据当前引用节点在用例逻辑中的位置动态加载符合逻辑的测试项队列。由于每个测试用例都是采用逻辑树的数据结构组织而成，故在测试用例的特定节点处对用例逻辑前面的测试项的返回值进行引用时，需要生成一个符合运行逻辑的测试项对列表，并对对列表中的每个测试项的所有返回值进行抽象构造。取测试项返回值算法描述图，如图 4-18 所示。

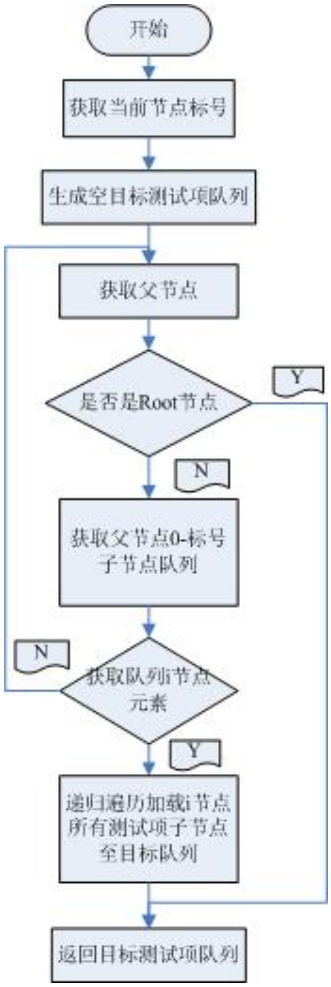

图 4-18　获取测试项返回值算法描述图

由于测试用例的逻辑是一个顺序执行结构，所以在引用测试项返回值时，引用节点只能引用在用例逻辑中先它运行的测试项的返回值。故在该算法实现中，外循环需要从引用节点开始循环向上获取其父节点，从而获取其标号靠前的兄弟队列；而内循环则是对标号靠前的每个兄弟节点依次递归扫描加载其所有测试项子节点；最终便得到当前所选节点的合法加载测试项队列。

测试项返回值界面图，如图 4-19 所示。

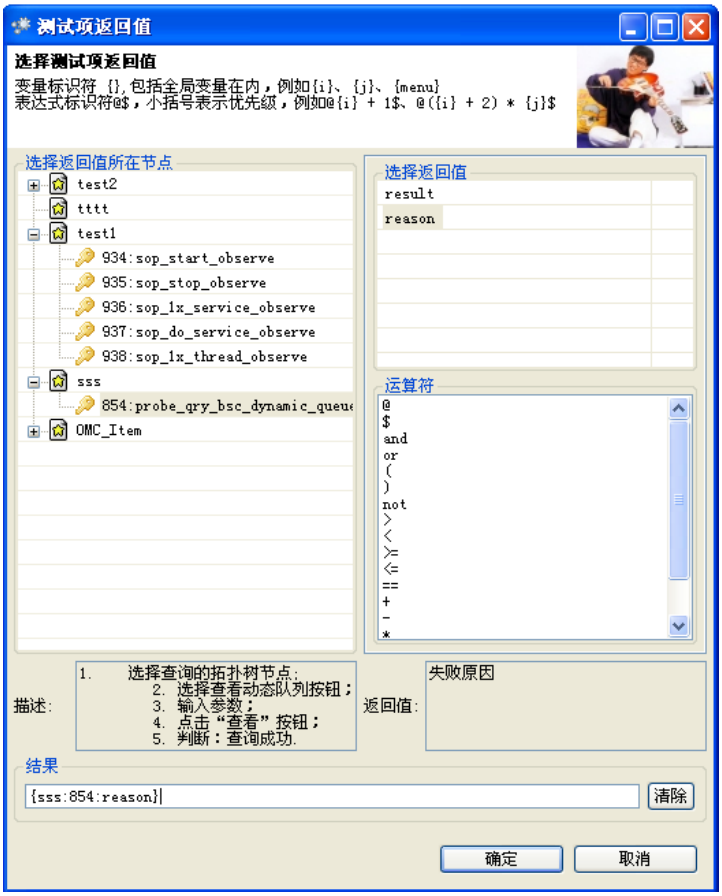

图 4-19　测试项返回值界面图

界面右边为测试项和关键字二级树结构，一个测试项可能对应多个关键字，测试人员通过选择所需要的关键字来打开该关键字所包含的返回值列表；左边为用户所选择对应的关键字返回值列表和一个表达式选择列表。用户选择所需的返回值后，可以选择编辑所需要的表达式，并做为最终选择结果返回。结果输入框内用来显示返回值或者返回值表达式的格式，在用户点击确定时，将用正则表达式匹配结果输入框里的选值。

4.5　测试报告查看器模块实现

4.5.1　报告查看器模块流程

测试报告是系统对测试任务测试执行结果的信息返回集合，用户正是通过分析

测试报告来查找和分析被测软件和硬件装置所存在的问题。报告查看器则是负责将最终测试报告以一种人性化的方式展现给用户的接口。报告查看器模块关系示意图，如图 4-20 所示。从图 4-20 中，可以看出 Client(客户端)、Server(服务器)、用户、Task(任务)和 Report(报告)几者的关系。

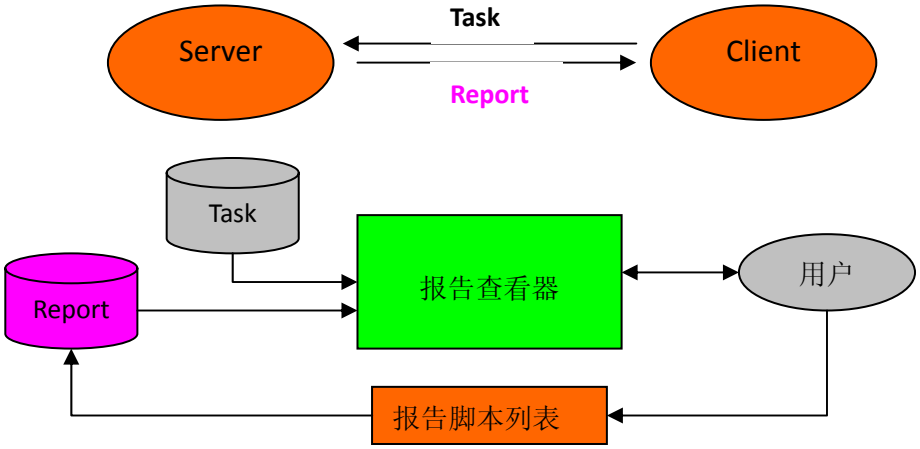

图 4-20　报告查看器模块关系示意图

用户通过对测试任务执行测试，查看测试自动化系统返回的测试报告，来达到其对分析测试结果的目的。在系统中，客户端 n 发送测试任务到调度服务器，供其调度并下发到相应测试子系统执行测试；各测试子系统执行完调度服务器分发的测试子任务后，将返回报告信息到调度服务器，再由调度服务器将报告收集并发送回客户端。

而对于用户而言，当需要查看测试任务报告信息时，与其交互的是客户端的报告管理器和报告查看器。报告脚本列表中的每条记录对应一个测试任务报告，报告查看器为方便用户查看对应报告的测试信息提供接口，并且为用户分析测试报告提供导航功能。

4.5.2　报告查看器的功能实现

从客户端的界面视图看报告查看器就相当于是一个 RCP 编辑器，故在 Plugin.xml 插件扩展表中扩展报告编辑器功能点，同时在报告管理器中为报告查看器增加报告查看入口。按照测试报告的后台解析和前台显示规则，设计报告查看器实现，其类图实现报告查看器实现类图，如图 4-21 所示。

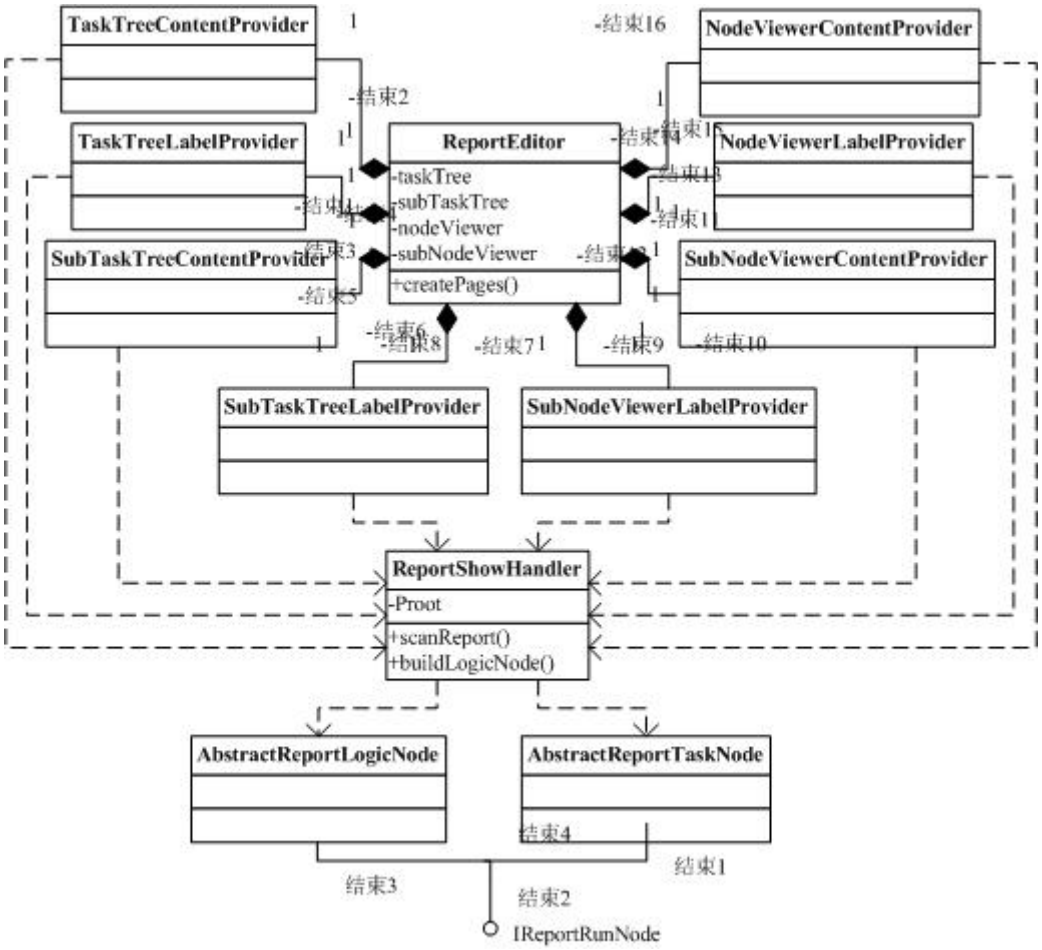

图 4-21　报告查看器实现类图

（1）ReportEditor 是该模块的报告编辑器类，主要用于显示报告查看器的界面内容。该类是实现报告查看器复杂功能所有控件的容器类，也是界面布局管理的核心类。其包含有 taskTree,subTaskTree,nodeViewer,subNodeViewer 等显示控件。

（2）TaskTreeContentProvider 和 TaskTreeLabelProvider 是报告任务树 taskTree 的内容和标签填充器；SubTaskTreeContentProvider 和 SubTaskTreeLabelProvider 是任务子树 subTaskTree 的内容和标签填充器；NodeViewerContentProvider 和 NodeViewerLabelProvider 是选中任务节点的报告内容和标签填充器；SubNodeViewerContentProvider 和 SubNodeViewerLabelProvider 是选中任务节点的子节点的内容和标签填充器。这 8 大填充器共同处理报告编辑器中烦锁的内容和显示规则。

（3）ReportShowHandler 类主要功能是负责对报告的文件解析，将相应的 XML 格式报告文件解析成一棵树状的数据结构，在内存中生成报告结构映象。并且对树结构中的每个节点的报告信息均包含其中，以便显示端可以直接从内存中读取到所有报告映象。

（4）类图的底层是一些节点类的封装和公用接口类。报告系统中报告树中所有节点均须要实现 IReportRunNode 接口，该接口定义了一些结点运行路径和用例使能的方法。支持的报告子节点类主要有两类，一类是任务节点类，一类是逻辑节点类，分别需要继承抽象类 AbstractReportTaskNode 和 AbstractReportLogicNode。由于任务节点类和报告逻辑类的节点类型太多，故在类图中只标明了两个抽象类。

4.5.3 报告导航功能实现

由于系统支持的测试任务规模的扩大和测试用例逻辑的复杂化，使得测试报告中所包含的节点越来越多，同时节点所附带的报告内容也越来越多。这就需要报告查看器中对测试报告的提供导航功能，为用户分析测试报告提供一些辅助功能。导航模块功能图，如图 4-22 所示。

图 4-22　导航模块功能图

导航模块主要起到方便用户快速查看寻找测试报告运行错误以及协助分析报告的功能。对导航功能模块进行功能分解，可以得到测试报告的展开功能、折叠功能、前向定位功能、反向定位功能、刷新功能和切换功能。

展开、折叠和刷新功能的实现比较简单，只须对调用 TreeViewer 树视图的 API

进行相应的展开和折叠功能，然后在界面的导航条处增加相应的操作接口即可实理。

对于切换功能，则报告查看器定义了四种切换模式，分别是：错误树模式、成功树模式、运行树模式和完全树模式。切换功能模块通过分析测试用例级节点的运行成功失败情况，实时生成相应的任务树加载至 taskTree 显示，从而在即不破坏测试用例逻辑的前提下，做到多功能视图的切换。

对于前向定位和反向定位功能，主要是为用户避免手工查找失败测试项的烦锁而开发的功能。该功能主要是用一种循环查找的算法，对错误测试项进行循环查找和显示。错误查找定位功能算法描述图，如图 4-23 所示。

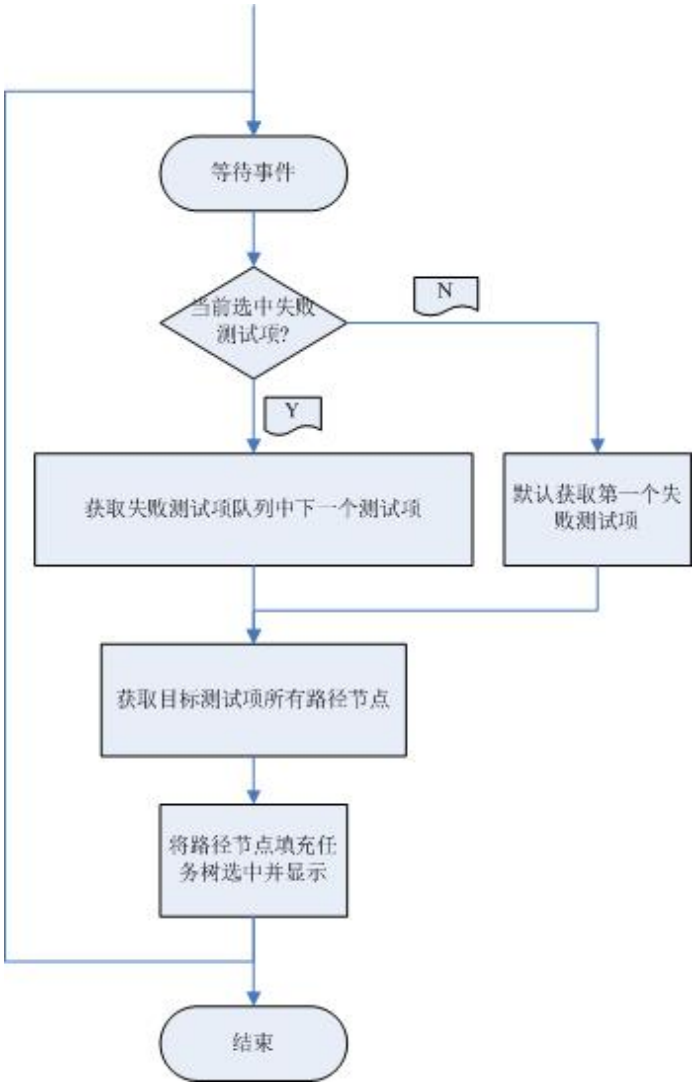

图 4-23　错误查找定位功能算法描述图

系统首先会跟据当前任务树来获取一个失败测试项队列；当报告查看器接收到前向定位或反向定位的事件命令后，客户端首先判断当前是否有错失败测试项处于选中状态；如果有选中的失败测试项，则从失败测试项队列中获取下一个失败测试项，否则就默认选中第一个失败测试项；获取了目标失败测试项之后，系统便获取目标测试项的所有路径节点，此时需要从目标节点依次向上层遍历，直到在路径中添加根节点为止；最后将该路径填充到任务树中，并设置选中目标失败测试项，便可以实现报告查看器信息更新的目的，从而完成一次的失败查找定位事件。

4.5.4 报告查看器实现评析

由于在同行评审中，客户代表对报告查看器的任务树选中 Tab 页拆分报告树为任务树和逻辑树提出易用性质疑，认为该种实习方式会影响用户的使用习惯，所以最终的报告查看器实现时未对报告任务树进行拆分，而是增加了测试用例节点事件，方便用户快速查看所选测试用例下的测试逻辑和报告运行信息。报告查看器实现模型图，如图 4-24 所示。

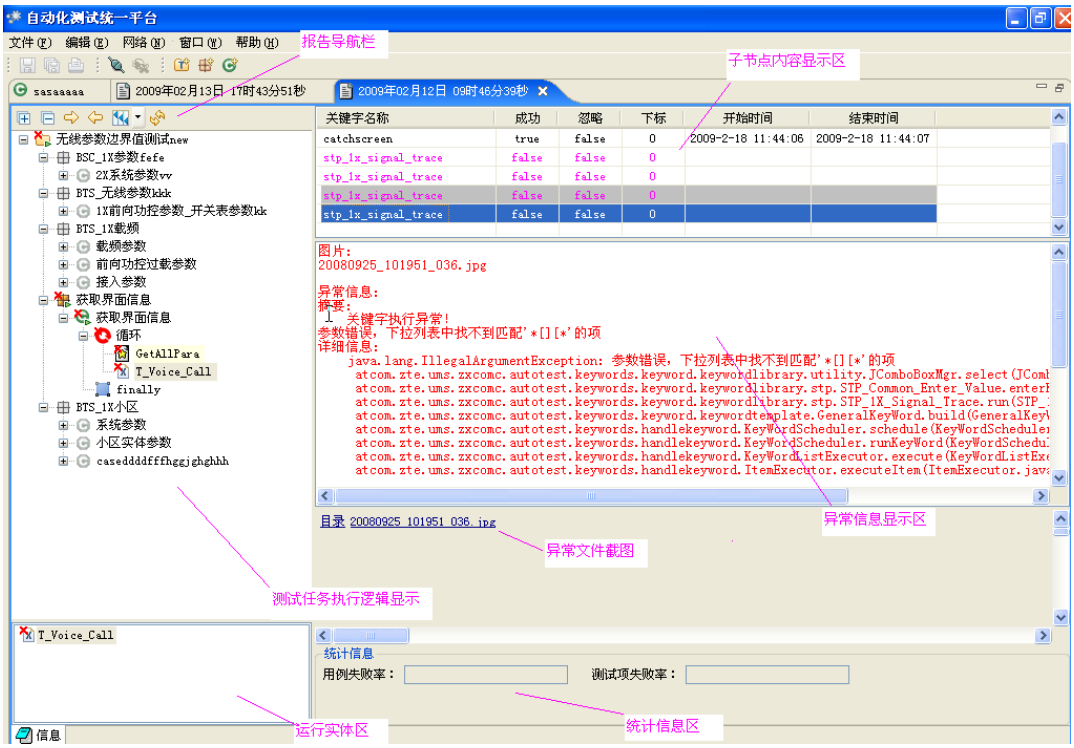

图 4-24　报告查看器实现模型

如图 4-24 可见，报告查看器是左右结构显示，左边是报告任务节点，右边是报告内容显示模块。左部中间是报告任务树结构，其包含有任务节点和测试逻辑结构；左上方报告任务树上面是导航条模块，其为一组功功按钮；左下方是报告子节点显示模块，其为一个列表式的树状结构。右边的报告内容显示模块分为四部分，最上部是报告结点内容信息显示区，当前显示的是报告关键字表格列表；中间显示的是当前选中的子结点的报告内容异常显示区；中间偏下部是一个文件和图片链接区，显示所选报告子节点内所包含的被测系统捕获的异常图片和日志文件链接等；右下方是一个统计信息模块，支持对测试用例失败率和测试项失败率的实时统计等。

4.6 本章小结

本章根据自动化测试系统的统一结构设计和执行流程，分析讨论了富客户端一些核心模块的设计实现方式以及实现结果等。

首先介绍了客户端的界面结构设计、客户端主要视图和编辑器的总体布局，以及在实现客户端界面时基于插件的接口扩展和最终界面的实现结果等。

其后讨论了，客户端用例编辑核心模块测试用例编辑器的实现情况，包括测试用例任务定制、测试任务逻辑定制和最终的用例编辑器实现结果等。此后讨论了测试自动化系统工具用例编辑功能拓展的两个功能模块，数据驱动模块和测试子系统返回值模块。讨论分析了数据驱动模块的内部关系、与调度层的接口示意、数据驱动运作流程和最终数据驱动实现结果；分析了测试项子系统返回值执行流程以及关键字返回值和测试项返回值的实现方式等。

最后讨论了测试报告查看器模块的实现方式，分别从报告查看器的模块流程、界面规划、模块功能设计实现、测试报告导航功能和最终报告查看器实现评析等几个方面，对报告查看器模块的实现方式进行分析。

5 自动化测试系统的测试分析

本章主要讨论了对自动化测试系统所进行的一些交叉测试情况，并讨论对测试结果的分析和后期测试规划等研究情况。

5.1 测试概念说明

在软件工程领域，测试是软件生存周期中一个独立的，关键的阶段，也是保证软件质量的重要手段。测试代表了规约、设计和编码的最终检查，是软件质量保证的最后一个环节。

测试的目标是设计出最可能发现最多数量错误，并耗费最少时间和最小代价的测试用例。测试的种类也比较多，一般传统的测试方法包括有黑盒测试和白盒测试，这两种测试方式可以用于所有的环境、体系结构和应用程序。此外还包括一些其他的测试技术，如面向对象测试技术、（嵌入式）实时操作系统测试技术、C/S 体系结构测试技术、GUI 测试技术、帮助（文档）测试技术等。

黑盒测试在于发现功能需求错误，而不考虑程序的内部工作。一般采用黑盒测试来检验系统功能不对或遗漏、界面错误、数据结构或外部数据库访问错误、性能错误、初始化和终止错误等方面。而白盒测试注重于程序控制结构，其一般包括基本路径测试和控制结构测试等，其目的在于测试程序条件的错误和程序的其他错误等。

5.2 系统功能性交叉测试

5.2.1 测试目的和测试环境

从自动化测试系统的功能上看，其为一个自动化测试工具，用来对手机通迅业务和网络管理软件进行测试；而从软件工程的角度看，自动化测试系统本身亦属于软件的范畴，其开发过程中同样包括系统需求、系统设计和开发编码等软件开发周期过程。因此，在软件开发的过程中同样需要对系统进行充分的测试，只有首先保

证好软件的质量，才能让自动化测试工具本身有资格进行自动化测试其他软件和业务。

由于自动化测试系统架构层次和系统实现模块较多，需要对系统架构中的富客户端、调度服务器和测试子系统三个层次的模块进行测试。因此为了提升测试效率，项目开发小组采取了内部交叉测试的方式，分工对系统各模块进行黑盒测试。

由于目前，自动化测试系统试用于手机基本业务测试环境和基站网络管理软件测试两个方面，故执行交叉测试时，分别选择了一套基本业务测试环境和一套基站网络管理软件测试环境进行测试。其中系统自测环境选用的是基站收发台 BTS I1 和 CBTS I2；同时对软件的配置管理采用服务器 IBM System X3500，用于对历史数据库进行维护管理等。在人员和时间配备方面，项目小组分配人员 6 人，测试时间 2 周共计 10 个工作日，对系统各模块进行内部交叉测试。

5.2.2 测试过程记录

此次系统自测主要是针对富客户端平台和分布式调度服务器所开发的一些核心功能而进行的，主要包括数据驱动模块、报告查看器、用例编辑器、任务解析调度和报告生成等模块。

自动化测试系统自测记录表，如表 5-1 所示。

表 5-1　自动化测试系统自测记录表

模块	功能点	问题记录	故障确认解决	测试人员
数据驱动功能	1 数据驱动节点嵌套	1 驱动变量类型定义未加以限制	解决	Z
	2 驱动变量名称限制规则			
	3 静态驱动数据编辑	2 保存驱动数据时，未能保存数据使能项	解决	
	4 动态驱动数据编辑			
	5 驱动逻辑内部调度规则	3 用例编辑器中驱动节点界面显示需加强	解决	
	6 驱动数据为空时调度执行			

续表 5-1　自动化测试系统自测记录表

报告查看功能	1 能正确浏览和删除报告	1 有环境不显示导航条	解决	L
	2 报告各节点解析规则	2 未解析测试项返回值	解决	
	3 报告导航功能			
用例编辑功能	1 用例内部逻辑节点编辑	1 查找关键字功能偶尔不能有效查找	解决	Z
	2 逻辑节点属性编辑			
	3 容器类逻辑节点运行逻辑			
	4 测试项编辑			
任务运行控制	1 任务控制功能键逻辑	1 任务运行使能功能出现过不正确显示	未重现故障	L
	2 定时任务和串行任务功能			
关键字同步	1 能正确检测测试项中内容改变	1 关键字列表文件刷新加载更新有延迟	解决	D
变量支持功能	1 各层次级变量赋值有效性	1 变量就近层次选择出现错误	解决	L
	2 参数树设置			
	3 调度服务器设置			
	4 运行任务设置			
实时状态查看	1 实时显示运行任务各逻辑节点执行状态	1 当任务运行完毕后，测试完成时间显示不正确	解决	L
子系统注册	1 各子系统能正确注册连接	未发现故障		Y
	2 处理子系统断链			
任务解析调度	1 解析任意用例逻辑	1 测试用例中关键字为空时异常抛出	解决	D
	2 分发调度任务	2 停止测试任务再运行时产生过分发错误	解决	
报告生成	1 生成报告逻辑	1 测试项节点内部运行成功/失败信息统计有误	解决	Z
	2 生成各报告子节点信息			
	3 报告正常上报			

其依据测试计划，对系统各个功能模块执行交叉测试的结果统计表格，按照测试计划对各个模块和功能点进行测试划分和人员划分。在系统自测期内，完成了各个功能模块的测试并且对各个功能点所发现的故障进行质量故障管理，从而跟踪每个故障的修复情况。

5.3 系统自测现象和结果分析

虽然从开发目的和功能上看，自动化测试系统本身是一个测试工具，用于测试其他通讯软件和硬件，但从软件工程的角度看，在软件开发过程中不可避免会产生一些软件故障。于是在版本大规模运用之前，在开发小组采取交叉自测能够提高发现并解决这些故障，从而降低软件的开发风险和后期维护成本。

通过交叉自测，从自动化测试系统自测结果表可以看出，在数据驱动模块和用例编辑模块发现的故障较多，而在任务运行控制、实时状态查看、子系统注册和任务解析调度等模块发现的故障较少。数据驱动模块和用例编辑模块是富客户端的两个重要模块，也是与测试用例编辑和测试数据编辑联系最紧密的两个模块。由富客户端是用户直接打交道的模块，其为用户提供界面操作接口，同时也实现系统一些重要模块功能。客户端的特征，使得其故障较容易发现，且修正效果也容易确认。同时需要控制客户端的故障，客户端的体验直接决定用户对系统的使用感受。

而调度层的任务解析调度和报告生成等功能则需要通过执行多种测试用例才能测试完成。故在后期的测试计划中，将考虑对客户端依旧执行黑盒测试，而对调度服务器则执行白盒测试，以便提供更稳定的系统。由于系统处于试用阶段，故后期的测试计划将结合测试人员在实际测试环境的使用而进行制定和调整，从而对系统功能和性能进行扩展优化，以便适用更多的应用场景，为用户提供更加稳定的功能。

5.4 本章小结

本章主要讨论了对自动化测试系统所进行的一些测试情况。首先讨论了目前软件测试中使用的黑盒、白盒测试等测试技术，然后分析了自动化测试系统内部交叉测试的实施情况。

对系统交叉测试的任务制定、人员时间分配进行讨论，分析测试执行开展情况和测试所发现故障的解决情况；此后结合系统交叉测试结果，得出系统故障分布的特点和后期制定测试规划等。

6 总结与展望

6.1 全文总结

自动化测试系统是为了适应通讯行业日益复杂的测试业务流程和愈发繁多的测试任务而研发的测试处理系统。自动化测试系统的研究工作对于替代效率低下的系统手工测试，进行大规模的回归测试和冒烟测试，规范化测试管理流程，提高被测试产品的质量都有着重要的现实意义。本论文对自动化测试系统的需求和系统要求进行分析设计，并讨论分析了自动化测试系统的系统体系结构和执行流程，最后从用例编辑模块、数据驱动模块、子系统返回值模块和报告查看模块等，讨论分析了自动化测试系统客户端实现等。以完成的工作有：

（1）分析系统各层次模块中，应用层、调度层、执行子系统和被测系统在自动化测试系统中的功能实现和系统数据执行流程。

（2）讨论了富客户端可扩展性界面框架，分析了其支持多种视图和编辑器，且可针对不同的应用场景给出不同视图显示的应用实现。

（3）分析了用例编辑器模块的功能实现，讨论了其支持顺序、循环、嵌套等多种形式的测试逻辑和测试节点数据赋值特点。

（4）分析探讨了系统实现的数据驱动功能，研究了在测试用例中测试逻辑和测试数据的分离和重复性使用的数据驱动特征，讨论了其所支持静态和动态两种驱动数据形式。

（5）对系统的测试项返回值功能进行分析，讨论了返回值功能在应用层客户端的测试项和关键字返回值编辑实现以及调度层对执行数据的动态引用等实现机制。

（6）测试报告管理和查看模块实现模块进行分析，讨论了测试报告的实时显示加载、对报告内容的错误定位导航等功能实现。

6.2 展望

自动化测试系统的大部分功能和性能需求已经实现，目前试用版本已经发布，

然而由于很多需求需要用户在使用过程中可能需要扩展，同时针对通迅技术的不断发展，自动化测试所需支持的业务也会增多，许多功能还需要进一步的完善，对本系统的研究仍有许多后继工作需要探讨：

（1）系统界面扩展接口，提供更加人性化接口，降低自动化测试系统操作门槛，方便普通测试人员快速学习使用。

（2）测试用例管理功能扩展，支持复用测试用例中的部分测试逻辑和测试数据，实现更加灵活的测试用例管理。

（3）业务应用拓展，能快速扩展不同通讯业务，要求针对不同的应用场景，系统容易拓展增加对新业务的支持等。

由于时间的仓促和笔者水平的有限，论文中存在错误在所难免，请各位老师和同学指正。

致　谢

时光如逝，转眼间研究生的研究和学习生活即将结束，在此之际，我要向所有曾经给予过我关怀与帮助的人表示衷心的谢意。

首先，我要感谢我的导师苏曙光教授的悉心指导和关怀。苏曙光老师严谨治学的态度和对学生一丝不苟的要求激励着我学习和研究上的每一点进步。她渊博的知识、孜孜不倦的工作热情和无私的敬业精神无时不在的教育着我、感染着我，她对我的影响将使我终生受益。

研究生学习期间，我有幸来到中兴通讯股份有限公司实习，同项目组的领导和同事们陪伴我度过了紧张而又快乐的时光，这里我向他们一并表示感谢。要感谢我的师傅张宏强工程师，项目经理左军工程师，科长周胜宝工程师和部长张欣工程师，在课题进行中给予我很多学术上和资料上的帮助和不可多得的思维启发，在我的科研和学习工作中，都从他那里学到了很多书本上难以学到、不可多得的知识。同时要感谢同项目组的罗漩、邓楠、宋花伦和武娟，是大家的团结协作才使项目得以顺利进行，和他们的交流也使我在技术上和认识上有了新的收获，和他们的朝夕相处使我感到了集体的温暖和大家庭的快乐。感谢部门领导和同事，你们的关心和帮助，你们的友谊让我度过了许多快乐的时光。

最后深深感谢永远支持我的亲人们，你们的鼓励与支持永远是我人生路上探索前行的巨大动力和信心源泉。

参考文献

[1] Lydia Ash. Web 测试指南. 北京: 机械工业出版社, 2004: 2-5

[2] Cem Kaner. Improving the Maintainability of Automated Test Suites. www. kaner. com/lawst1. htm, 1997

[3] 王倩. 软件自动化测试工具的分类与选择. 知识讲座, 2008, 2(8): 51-53

[4] 孟晓鸿. 谈软件测试自动化. 中国新技术新产品, 2008, 3(10): 11-13

[5] 郭伟斌, 郭锡坤. 自动化测试的研究和探讨. 电脑开发与应用, 2008, 4(12): 10-14

[6] 高雯雯, 兰雨晴, 高静. 自动化测试执行管理工具的研究与设计. 计算机研究应用, 2008, 25(1): 126-127

[7] 乔元. 软件测试工具比较. 软件测试网. http:\\www. 51testing. com, 2005. 10

[8] 李刚毅, 金蓓弘. 自动化回归测试的技术和实现. 计算机应用研究, 2006, 1(2): 186-189

[9] 路晓丽, 葛玮, 龚晓庆等. 软件测试技术. 北京: 机械工业出版社, 2007: 8-11

[10] Mark Fewster, Dorothy Graham. 软件测试自动化技术与实例详解. 舒智勇, 包晓露, 焦跃译. 北京: 电子工业出版社, 2000: 18-21

[11] 张友生. 软件体系结构. 北京: 清华大学出版社, 2006: 11-14

[12] Wendy Boggs, Michael Boggs. UML with Rational Rose 从入门到精通. 邱仲潘译. 北京: 电子工业出版社, 2000: 3-6

[13] 向润. 黑盒测试方法探讨. 软件导刊, 2009, 8(1): 33-35

[14] 古乐, 史九林. 软件测试概论. 北京: 清华大学出版社, 2004: 47-49

[15] 马雪英, 姚砺, 叶澄清. 回归测试自动化工具研究. 计算机科学, 2005, 32(3): 162-165

[16] Csondes T, Kotnyek B. Grdddy Algorithm for the Test Selection Problem in Protocol Conformance Testing. Journal of Circuit, System and Computers, World

Scientific Publishing Company, 2002, 11(3): 273-281

[17] Wegener, Joachim, Baresel, et al. Evolutionary Test Environment for Automatic Structural Testing. Information and Software Technology, 2001, 43(14): 841-854

[18] 刘慕涛, 张磊, 王艳等. 基于 XML 的 API 自动化测试工具设计与实现. 计算机工程, 2007, 33(13): 96-98

[19] 朱菊, 王志坚, 杨雪. 基于数据驱动的软件自动化测试框架. 计算机技术与发展, 2006, 16(5): 68-70

[20] Bash J. Test Automation Snake Oil. http: //www. satisfice. com/ articles/ test-automation-snake-oil. pdf, 1999

[21] 刘晓丹, 武君胜, 刘博. 基于数据驱动的自动化测试平台设计. 科学技术与工程, 2008, 8(3): 779-782

[22] Bertolino A, Mirandola R, Peciola E A case study in Branch testing automation Systems Software, 1997, 1(4): 47-59

[23] 周秦源. 浅谈我国 IT 行业中的软件测试. 中国科技信息, 2005, 14(5): 69

[24] 许静, 陈宏刚, 王庆人. 软件测试方法简述与展望. 计算机工程与应用, 2003, 13(1): 35-37

[25] 陈骏. 软件测试技术研究. 信息科学, 2009, 1(5): 32

[26] 王倩. 软件自动化测试工具的分类与选择. 知识讲座, 2008, 8(2): 51-53

[27] Hoffman D. A course on software test automation design Software Quality Methods, LLC(SQM), 2002

[28] 韩振斌, 苗克坚. 一种分布式软件自动化测试工具的设计与实现, 2007, 7(8): 174-177

[29] Pettichord B. Deconstructing GUI Test Automation. STQE. http:\\www. stqemagazine. com, 2003

[30] 蒋云, 赵佳宝. 自动化测试脚本自动生成技术的研究. 计算机技术与发展, 2007, 17(7): 4-7

[31] Kaner C, Falk J, Nguyen HQ. Testing Computer Software. Canada: Wiley

Computer Publishing, 1999

[32] 接卉, 兰雨晴, 骆沛. 一种关键字驱动的自动化测试框架. 计算机应用研究, 2009, 26(3): 927-929

[33] DUSTION E. The automated testing life-cycle methodology. Proc of International Conference on Practical Software Assurance Techniques and Practical Software Test Techniques, 2000: 1-16

[34] John W Satzinger, Robert B Jackson, Stephen D Burd. Systems analysis and design in a changing world. New York: Course Technology Division of Thomson Learning, 2000: 126-257

[35] 尤永康, 刘乃琦. 自动化回归测试在 JAVA 项目中的实现. 计算机应用, 2005, 25(1): 88-90

[36] Liu C. Platform-Independent And Tool-Neutral Test Descriptions For Automated Software Testing. Information and Computer Science, University of California, 2002

[37] 冯玉才, 唐艳, 周淳. 关键字驱动自动化测试的原理和实现. 计算机应用, 2004, 24(8): 140-142

[38] Li Kanglin, Wu Mengqi. 高效软件测试自动化. 曹文静, 谈利群译. 北京: 电子工业出版社, 2004: 47-49

[39] 孟微, 罗省贤. 功能测试可回归性研究. 计算机工程与设计, 2009, 30(1): 125-128

[40] 刘正升, 蒋志忠, 杨日杰. 测试领域新技术的发展及应用. 中国科技核心期刊, 2009, 28(1): 16-19

[41] 美国国家仪器公司. 构建以软件为中心的下一代自动化测试系统. 电子技术应用, 2008, 8: 10-12

[42] 金大海, 宫云战. 数据驱动自动化测试方法研究. 装甲兵工程学院学报, 2004, 18(2): 95-97