中 文 摘 要

面向构件的软件开发方法已成为软件工程领域的热点。目前70%的软件都是基于构件开发的，它秉诚“购买而不创建，组装即成”的开发哲学，从根本上改变了软件开发方式，提高了软件复用性和开发效率。CBD软件开发的核心是构件组装。软件集成测试是组装工作是否有效的检验者，它关系到CBD软件开发工作的成败。由于CBD软件具有源代码未知，应用环境复杂等特点，传统的软件集成测试技术，对于解决这些问题，将会面临严峻的挑战。相比之下，基于UML模型的CBD软件集成测试技术取得了不错的成果，国内外已有不少关于这方面的成果，但基本上都是基于UML1.x的交互图生成测试用例而进行测试，都没有充分整合UML2.0各种模型的新特性而进行更有效的测试。因此，研究基于UML2.0模型的CBD软件集成测试方法具有重要意义。

本文首先在对UML2.0模型中的状态机进行分析的基础上，结合构件图将状态机扩展为具有事件语义的模型(ECSM)。并将“顺序”定义为强顺序和弱顺序以解决UML2.0顺序图(SD)中的“顺序”所描述语义的不准确性问题。然后将ECSM和SD进行有机集成，给出了一种可描述构件交互的模型（可扩展的构件交互模型ECSD）。由于ECSD包含的信息量多而不便于分析，文中构造了可以简化ECSD模型的消息交互流程图模型（MISG），给出了基于MISG模型生成测试场景和测试覆盖准则的算法。同时，分析了CBD软件的测试数据准备生成策略，设计了一种的基于MISG模型的测试用例生成方法，并以ATM系统为例，使用Rational Rose对ATM系统进行建模和分析，得到MISG模型，在此基础上生成了测试用例，达到了预期的效果。最后，本文提出了一种基于UML2.0模型的CBD软件集成测试框架（CSITF），并进行了框架总体设计，关键模块设计及核心算法设计。该模型对设计CBD软件集成测试工具提供了一种思路。

关键词：构件测试；UML2.0；集成测试框架；MISG；

ABSTRACT

Component-oriented software development method has become a research hot in the field of software engineering. Now 70% of the software is Component-Based Development. It is in good faith "rather than creation but purchase and only assembly", which fundamentally changed the way of software development, and improved efficiency of software reusability. Component assembly is the core of the CBD software development. Whether the components assembly work effectively or not, software integrating testing is related to the success or failure of CBD software development as a checker. Due to the characteristics that CBD software source code are unknown, and complex application environments, the traditional software testing techniques will face severe challenges. In contrast, UML-based Integration Testing for Component-based software has achieved good results, There are achievements in this aspect, but basically all are based on the interactive diagraph UML1.x to generate test cases and then test, no more effective testing can be processed without utilizing new features of fully integrated model UML2.0. Thus, research of UML2.0-based CBD software integration testing has the vital significance.

In this paper, based on the analysis of the state machine of UML 2.0, the state machine for event semantics model (ECSM) with the combination of component-diagraph is expanded. For solving the non-semantic accuracy of "sequence" described in SD, Sequences are divided into strong and weak, and a model describing the interactive components (extensible component interaction model ECSD) is constructed by integrating ECSM and SD. Since the ECSD contains great amount of information and not easy for analysis, a new flow chart of interaction (MISG) model is proposed which can simplify ECSD, and a algorithm for the test scenario and test coverage criteria generated based on MISG model is proposed. At the same time, the test data

is analyzed for preparation to generate strategies of CBD software. And ATM system is taken as an example, and Rational Rose is used for the modeling and analysis, The desired results has been achieved on the basis of test cases which have been generated. Finally, this paper proposes UML2.0-based CBD software integration testing framework (CSITF), and designs the architecture of framework, the core modules and the core algorithms. The framework provides a new way for the design of CBD Software integration testing tools.

Keywords: component testing; UML2.0; integration testing framework; MISG;

声　　明

本人郑重声明：所呈交的学位论文，是本人在指导教师的指导下，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。

作者签名：[illegible]　　日期：2009.4.23

关于学位论文使用权的说明

本人完全了解太原科技大学有关保管、使用学位论文的规定，其中包括：①学校有权保管、并向有关部门送交学位论文的原件、复印件与电子版；②学校可以采用影印、缩印或其它复制手段复制并保存学位论文；③学校可允许学位论文被查阅或借阅；④学校可以学术交流为目的，复制赠送和交换学位论文；⑤学校可以公布学位论文的全部或部分内容（保密学位论文在解密后遵守此规定）。

作者签名：[illegible]　　日期：2009.4.23

导师签名：[illegible]　　日期：2009.4.23

第一章 绪 论

1.1 课题研究背景和意义

软件测试是对软件规格说明、软件设计和编码的最后审核，其目的是在产品交付前尽可能发现软件中潜伏的错误[1]。大量统计表明，软件测试工作量往往占软件开发总工作量的40%以上[2]，对于一些要求高可靠性、高安全性的软件，其测试费用可能相当于整个软件开发总费用的3至5倍[3]。因此，研究可靠的软件测试方法和实用有效的软件测试工具是提高软件开发效率，降低软件成本的重要的环节。

随着软件设计方法学的发展，面向构件的软件开发方法正在逐步取代传统的面向对象的软件开发方法，并将成为新的趋势[5]。截至目前为止，大约有80%的软件系统都是基于构件技术的[6]。由于构件具有独立功能，独立发布，遵循特定构件规范，支持可复用和可集成的二进制的功能块[4]等特点，使得基于构件的软件工程更具优势。但同时构件的跨平台，跨语言和源代码不可知的特点，导致了构件化软件系统的测试难度非常大。

其实，对于构件化软件而言，软件测试的问题实质上集中在构件集成测试的问题上。通过学者观察和研究，70%以上的测试问题都来源于构件的交互。所以构件交互测试是CBD软件的集成测试的重点。而OMG组织的UML2.0模型能对CBD软件进行建模，建模过程中能得到各种UML2.0图形，这些图形很好的表达了构件化软件的动态交互过程[8]。因此，研究基于UML2.0模型的可测试模型，并研究构件集成测试方法具有重要意义。

1.2 国内外研究现状

近几年来，构件化软件开发方法在国内外得到了充分的发展，北京大学，中科院软件研究所等发表了不少论文，普元公司，北大青鸟公司，上海构件库等在工程方面取得较好的成果。而关于CBD软件的集成测试的研究主要在国外展开。目前主要成果有以下几个：

（1）基于UML1.x的构件化软件的集成测试技术。目前有两种基于这种测试技术的方法：基于状态的测试技术[12][13][14][17][19][41]和基于事件流的测试技术[15][16]。前一种主要是根据UML1.x的状态图得到构件在交互过程中的状态改变情况，而进行分析，构造基于状态的测试模型。这种方法的缺点是：状态信息并不能充分表达系统

真实运行时的情况，例如：构件间可能发生了交互，但交互没有导致构件状态的变化，这种情况就无法在状态图中反映出来。后一种测试方法主要是根据 UML1.x 模型中的顺序图/协作图来分析交互过程，得到事件流，从而得到相关的可用计算机处理的模型。在这方面比较有代表性的有：Jean 等[9] ，Hoijin[8] ，Fraikin[20] 和 Ye[21] 它们都是都是通过分析序列图来构造测试模型的。这种方法的缺点是：它完全基于事件序列图来构造测试模型，而没有充分的反映构件每次事件作用后的状态信息。因此，考虑怎样把两种方法有机的结合起来，是基于这种方法研究的核心。本文的研究就是基于这个想法来展开的。

(2) 基于序列的技术[33] 这种技术主要是基于数据流图的思想，从两个方面来对构件软件进行测试的：对于单个构件的测试和对于整个构件组合的测试。第一个方面主要是根据构件相关的规格说明文档及第三方提供的说明书，进行信息提取，从而画出数据流图，生成测试序列。后一个方面主要是分析构件间的事情交互情况，得到事件的传播过程，从而推导出事件的交互序列。然后综合这两个方面来进行测试。这种方法的缺点是：如果构件化系统中的构件数据非常大的时候，获取方法序列的难度非常巨大。

(3) 基于构件交互图(CIG)的构件系统测试技术[27] 这种方法的思想是：通过分析得知构件系统中最容易出现问题的地方就是构件间的交互问题，所以它就从这个方面来分析构件间的交互所引起的问题。 主要是采用静态分析交互情况和动态跟踪构件实际运行过程的方法，从而得到各种相关的流图，然后对这些流进行建模和分析，得到测试用例。这种方法的优点是不需要知道构件的源代码就可以进行。缺点是：如果被测试的构件化软件规模非常大，分析交互就非常复杂。

(4) 基于构件软件流图(CBSFG)的测试技术[28] ，这种测试技术的主要思想是借助于数据流图和控制流程图的思想，将 CBD 软件图形化。这个过程主要是通过规格说明文档来推导的一些可视化信息，并能通过图形来表示的。这种方法的优点：它不是用单独的白盒测试或者黑盒测试，而是综合两者来提取一切可视化的信息来构造 CBSFG 图，缺点是这种图形化表示的信息不是非常准确，有时也不好把握，随着构件化软件规模的扩大，难度也随着增加。

(5) 基于内建式的测试技术（Build In Test (BIT)） [22][23][24] 这种测试方法的思想是：构件可测试性的构件，它将一些可用来帮助测试的方法编写好后，植入到构件的内容，当程序启动时，就可以根据这些方法来跟踪和检验测试过程，并将信息

记录下来，从而进行分析。这种方法的优点是：可以非常具有可行性和可操作性。缺点是：植入那些方法，这些方法是否参测试造成影响等都无法预测。

(6)基于 TDS(Technique Development frame words, and Quality Assurance Schemes)[29] 的测试技术。这种技术的思想是：根据构件的接口以及接口中的事件，根据这些信息利用相应的算法进行归纳整理，从而产生测试用例。这种方法的优点是：提出了构件测试充分性的判定条件。缺点是通过这种方法得到的测试信息并足够，真正实现充分性的测试的难度比较大。

1.3 研究内容及论文结构

本文在对 CBD 软件的集成测试和基于 UML2.0 测试的相关理论的认真研究的基础上，首先使用 UML2.0 对 CBD 软件建模，得到 CBD 软件的构件图，状态机和交互图，将构件图和状态图所描述的信息进行整合，构造一种具有事件语义的可描述构件交互的模型（ECSM），然后着重探讨了顺序图中所描述的构件交互语义具有比较大的局限性的问题，并采用将描述的交互场景细化为强弱顺序关系来解决这个问题。由于 ECSM 和 CD 间具有事件语义上的联系，将两者有机的集成起来，构造成一个可充分描述构件间的交互能力的模型（扩展的构件顺序图模型 ECSD），但是这种模型是完全图形化的，并且信息量大不便于处理。因此，在此基础上又提出了一种可用于处理的消息交互数据流模型(MISG)，并分析出了基于 MISG 模型的测试覆盖准则以及基于 MISG 模型生成测试场景的方法。根据文中的模型以及构件的规格说明，产生了测试准备数据，然后提出了一种基于 MISG 模型生成测试用例的方法，并通过 ATM 系统来验证了该方法具有可行性。最后本文提出了一种基于 UML2.0 模型的 CBD 软件的集成测试框架，并进行了框架总体设计，关键模块设计，核心算法设计及核心数据结构。该测试框架为解决 CBD 软件集成测试的核心问题提供了一种思路，同时也为开发 CBD 软件测试工具提供了参考。

本文共分六章，具体章节的内容按如下的顺序编排：

第二章　CBD 软件集成测试的有关问题。首先分析了构件的相关理论，包括构件的定义，构件标准。然后分析了 CBD 软件的特点，以及 CBD 软件集成测试的相关问题。

第三章　UML2.0 及其测试。首先分析了 UML2.0 相关的新特性，并将 UML2.0 与 UML1.x 进行了比较。然后重点分析了 UML2.0 对构件的支持。最后分析了基于 UML2.0 的构件化软件的建模方法。

第四章 一种基于 UML2.0 图的生成测试用例的场景模型。首先分析了构件图对于测试的优点。接着介绍了构件状态机的特点以及用于集成测试中的不足，并结合构件对状态机进行了扩展。然后介绍了构件顺序图，并重点分析了“顺序”所描述的语义具有的局限性。最后，通过分析两者的联系，并将两者集成起来，并生成可处理的消息流程图模型，以及生成测试用例的方法。

第五章 基于 UML2.0 的 CBD 软件集成测试的框架。介绍了 CSITF 的功能描述，原型设计，核心功能的实现，以及相关的数据结构。

第六章 总结与展望。总结语，对系统开发过程中取得的成果和存在的问题进行系统的总结和分析，并对下一步的研究方向和策略进行展望。

第二章 CBD 软件的集成测试中若干问题

构件是按照一定规范编写的具有特定功能的可复用的程序模块。构件的广泛应用，使得 CBD 软件的测试和维护问题越来越受到人们的重视。CBD 软件主要是复用已有构件或者 CTOS 构件构造新的软件系统，从而提高开发效率和质量，因此构件化构件系统的集成测试就显得非常重要了，只有做好集成测试，才能保证新系统的正确性和可靠性。本章将主要讨论当前的主流构件标准，以及构件化软件集成测试中存在的各种问题。

2.1 构件的定义

构件正处于发展阶段，所以目前还没有一个明确的定义。Szyperski[25]把构件定义为"软件构件是一个仅带特定契约接口和显式语境依赖的结构单元"，同时他还写道："软件构件可以独立部署，易于第三方整合。 著名的 Brown[27]和 Wallnau[28]描述构件为"一个非平凡的、几乎独立的、可替换的系统组成部分，它在定义完善的体系结构环境中实现某一清晰的功能" 。Clemens[29]认为：构件是一个独立部署的的单元，是一个第三方合成的单元，构件没有一致性的状态。Gartner Group[30]认为软件是运行时软件，它是一个可动态绑定的、含一个或多个程序的软件包，它作为一个独立单位，通过运行时可辨别的文档化接口加以管理和存取。下面本文将给出构件的定义：

构件定义 构件是一个具有独立功能的可复用的功能模块，它通过接口向外提供服务。构件可以定义为一个六元组：{spe，imp，dep，sta，pae,test}，其中 spe 为构件说明，imp 为构件实现体，dep 表示可被组装，sta 为构件模型，Pac 表示可包装，test 表示可被测试。

（1）构件说明。它是构件定义中最重要的部分。构件说明主要包括：构件的适用环境及使用方法说明，接口定义，构件所具有的功能说明，构件的组装规格说明。

（2）构件实现体。构件一般是由一方定义，一方开发，另一方使用的流程。由于功能需求不同，所以需要产生多个构件实现体。使其具有更强的复用性。

（3）可被组装。构件的设计目的就是为了能用构件快速组装成新的系统。可组装主要包括：构件组装的基础设施说明，组装时顺序要求，组装时的异常处理说明，组装有效的验证工作。

（4）构件模型 由于构件最终的目的是为了大量复用，因此在开发时需要选择尽量通用的构件模型来做开发，构件模型定义了关于发重用软件构件和构件之间相互通信的一组标准的描述。应具有自描述性，可集成性和可定制性。

（5）可包装。这点主要是从功能组合来考虑的。通过对不同的构件进行包装或者改造，能具有新的功能。

（6）可测试性。由于一般是商业构件，因此能够将构件设计成可测试的，对将来复有时有着很多的支持。

2.2 构件标准

目前软件行业内的构件标准有四个：OMG 提出的 CORBA(Common Object Request Breaker Architecture)，微软提出的 COM(Component Object Model)，还有 sun 公司提出的 EJB(Enterprise Java Bean)和 web 服务。

CORBA 的优点在于其与开发语言无关的语言独立性，与开发者无关的厂商独立性以及与操作系统无关的平台独立性，即提供跨平台的能力，其缺点在于其技术规范十分复杂，而且没有规定实现细节。因此不同版本的 CORBA 产品互不相同，会带来互操作性与移植的大量问题。另外，CORBA 开发工具相对而言比较缺乏，开发难度较大。其体系结构图如图 2-1 所示

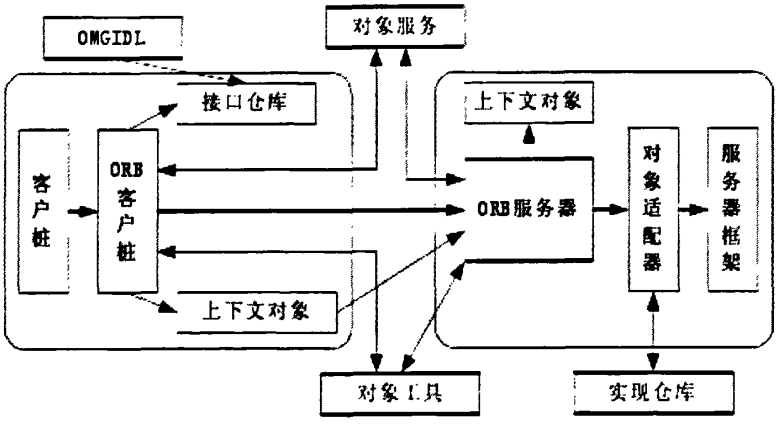

图 2-1 CORBA 体系结构

Fig2-1 Architecture Of CORBA

COM 体系结构如图 2-2 所示，其优点在于开发容易、商品化构件多和易于使用，其缺点在于其技术为微软专有。技术上的垄断带来的结果就是 COM 技术只能在 Windows 平台上实现，代码的可重用性和跨平台性能相对较差。COM+把 COM，DCOM 和 MTS(Microsoft Transaction Server)组合在了一起，使 COM+成为一个完整

的构件体系结构。三者有机地统一起来，同时新增了一些服务如负载平衡、内存数据库、事件模型、队列服务等，形成一个概念新、功能强的构件体系结构。

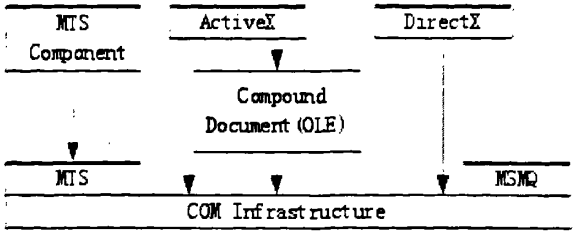

图 2-2 COM/COM+/DCOM 体系结构

Fig2-2 Architecture Of COM/COM+/DCOM

EJB 体系结构如图 2-3 所示。它是由这两种核心分布式技术演化来的，提供一个标准的分布的、基于 OO 的组件架构，屏蔽复杂的系统级功能需求，Write once, run anywhere，与非 Java 应用程序之间的互操作能力。

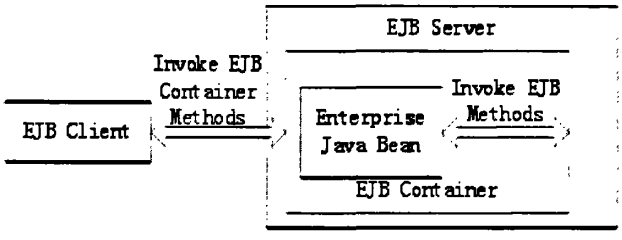

图 2-3 RMI/EJB 体系结构

Fig 2-3 Architecture Of RMI/EJB

WEB 服务是一套构件开发标准，如果 2-4 所示，它主要包括 SOAP，WSDL（WEB 服务描述语言），UDDI（通用描述发现，集成）和 WSIL（WEB 服务检查语言）有三个独立的角色：服务提供者，服务请求者和服务代理者。三个角色通过服务的发布，查找，绑定进行交互。服务提供者通过用代理者的发布接口让客户能访问服务来通知代理者服务存在。发布信息描述了服务并说明服务定位在哪儿。服务请求者询问代理者服务的定位。根据从代理者获得的服务信息，请求者可以绑定或者调用这个服务。

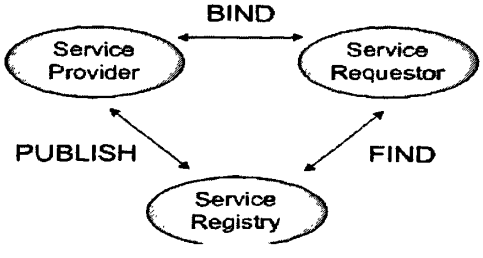

图 2-4 web 服务体系结构

Fig 2-4 Architecture Of Web Service

2.3 CBD 软件的集成测试的特点

由于 CBD 软件具有与传统软件不同的新特性[2][3]：(1)由源代码不可见的构件组装的高松耦合的系统。(2)构件更换容易，系统演化速度快。(3)系统的可靠性高度依赖于构件及其构件间的交互。(4)系统具有很好的扩展性和复用性[4]。从而导致了构件系统测试难度大。而构件通常是通过接口向外提供服务的，客户端只能通过接口来调用相关方法得到服务。一般用来构造系统的构件都是经过完整的单元测试，但是当它们构造系统时可能会产生各种交互错误，还有可能是接口提供的功能大于当前新系统的功能，而且这些错误不易被观察到。本文主要是从构件使用的角度来分析 CBD 软件的集成测试。我们从以下几个方面来探讨基于构件的开发对软件测试的影响，以及基于构件的软件测试的特点。

(1) 构件是 CBD 软件的最小组成单位　CBD 软件是复用已有的或自行开发的构件而组成的。构件是封装了数据和方法的功能休。我们可以认为构件是构成 CBD 软件的基本单位，构件质量的好坏直接影响到 CBD 软件的质量。

(2) 需要两种不同目的的测试工作　CBD软件的开发过程就是将现有的构件进行集成的过程。其中，构件可能是自行开发或者是第三方提供的。如果是自行开发，为了保证构件的质量，需要测试人员对构件进行单元测试。如果是第三方提供的也需要进行功能验证[33][40]。当把构件进行组装时则需要进行集成测试，集成测试是保证新软件系统质量的关键。两种测试工作虽然出发点不一样，但都得做好，这样才能确保最终的系统质量。

(3) 测试方法的选择　由于出于公司效率等的考虑，在开发CBD软件时，一般会考虑从构件库中检索合适的商业构件或者开源构件，这样的构件可能源代码不可得到或者文档不全等，如果要在组装前对构件进行白盒测试，难度会非常大。但可以考虑进行功能验证测试。即黑盒测试比较可行，也可以进行基于模型的测试[23]。

2.4 构件化软件系统的集成测试中的问题

CBD方法相对于传统的开发方法体现出了许多新优势，但它的新特点也给软件测试带来了新的挑战。我们从两个方面分析构件及构件软件的测试问题。

(1) 从构件生产者来看，主要是包括以下的问题：

A. 测试是否充分性的判定。由于构件规模和构件类型等不同，很难有个确定的判定依据说明测试的程度。

B. 测试准备数据的产生是否充分。由于对于单个构件的测试类似于白盒测试，所以很难设计出充分的测试数据。

C. 搭建构件的测试环境。对单个构件进行测试，需要模拟出构件运行的环境。构造测试驱动器和打桩对于构件的多线程，死锁，跨平台，跨语言等特殊情况很难模拟。

D. 构件的可复用性测试。由于构件设计的目的就是要重复使用可能需要在多种环境中应用。所以也要测试。

(2) 从构件使用者角度来看，对构件化软件测试面临如下的问题：

A. 测试是否充分的问题依然存在。

B. 构件的测试顺序问题。如果采用自底向上的集成测试方法，则每次的测试都是建立在相对正确的基础上的，但如果采用其他的集成测试方式则需要驱动模块和打桩。

C. 冗余测试问题。由于在组装过程中，每新增加一个构件，理论上应该重测以前的工作，但是工作量大，而且很多是重复工作，所以怎样取舍而进行回归测试也很复杂。

D. 源代码是否可用。因为很多构件都是第三方提供的，只可能提供一些文档，因此测试难度也比较大。

E. 编程语言、操作系统平台、硬件结构的混杂性。由于构件设计时就是尽可能被复用，而复用的环境可能会是不同的操作系统，不同的语言环境和不同的硬件平台，因此对于这些环境中的测试难度比较大。

F. 测试分布式软件时的监控问题。由于分布式软件一般部署在不同的硬件平台上，所以在测试过程中，跟踪就比较复杂。

G. 事件重构。系统中可能存在异步消息和同步消息，仅仅对系统做常规的测试是不充分的，需要对系统做一些重构而进行测试。

H. 构件的竞争和死锁。这类问题非常难处理。

I. 多线程问题。可能存在有些功能在多线程下可以运行但在单线程下无法运行。而且这种问题不好判断。

J. 容错测试问题。许多系统要求有高度的容错能力，如核电站、宇宙飞船等，因为在测试时有些错误很难出现，因此可以使用故障注入技术引入缺陷，观察系统的错误处理代码能否正常工作。但引入的错误不一定会激发，有时要

控制或改变系统的流程来激发这些错误。而且故障发生后不一定立刻表现为失效，当运行到后续模块，甚至是几天之后才表现为软件失效，为查找出错误的根源带来困难。

第三章 UML2.0 理论及其测试

UML2.0（Unified Modeling Language Version 2.0）是一种定义良好、易于表达、功能强大且普遍适用的建模语言[35]。它溶入了软件工程领域的新思想、新方法和新技术。它为高质量的软件开发与测试提供了有力的保障。本章将介绍 UML2.0 的新特性，设计目标，主要组织特征，基于 UML 的传统软件测试方法，以及基于 UML2.0 面向构件((Object-Oriented Component OOC)开发与测试方法。

3.1 UML 介绍

UML 是由著名的面向对象技术专家 Grady Booch、Ivar Jacobson 和 James Rumbaugh 发起，在 Booch 表示法、OOSE(Object-Oriented Software Engineering)方法和 OMT(Object Modeling Technology)方法的基础上，广泛征求意见，集众家之长，反复修改而完成的。在美国，UML 已经获得工业界和学术界的广泛支持。使用 UML 模型，是当前软件界成功地进行软件开发所不可缺少的关键环节。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 提供了一种对伙伴系统进行可视化，详述构造和文档化的工业标准机制。但这种静态定义却无法描述 UML 惊人的发展速度。UML 利用一套动态的，支持不断发展的特征组织其思想。这种不断发展的思想已经使 UML 经历了 Version 1.0，1.1，1.2，1.3，1.4，发展到了今天的 Version 2.0，并且提供的功能发生了很大的变化。UML2.0 还使 UML 的发展延伸到一些新领域。

3.1.1 UML2.0 设计目标

UML2.0 继承了 UML1.x 的思想，就是要让设计者设计出高复用性，高准确性和高扩展性的软件系统。所以 UML2.0 在开始设计前就提出了设计目标。

A. 使得该语言对软件实体进行建模变得更加可行。

B. 对流程和动作建模提供更健壮的机制，

C. 为不同工具之间的通信创建一个标准。

D. 在一个标准的建模框架内协调 UML。

E. 支持基于组件的开发，包括插件可替代性的完整定义。

F. 允许软件进行上下文的较完整的建模，尤其是随着中间件和 Web 应用技术的不断增长，非常需要进行此项功能。

G. 提供用于重要语言的标准用户配置文件或扩展。

H. 支持通过交互模式的自动实现。

I. 增强 UML 对运行时体系结构的支持，包括对内部结构的建模支持，由包括对内部结构的动态行为的描述。

J. 利用更清晰的规范来描述行为链接和特化规则，从而进一步改进状态机。

K. 利用更好的控制流和数据特征改进活动图。

L. 支持交互机制的组合，定义交互，可选交互和并行执行的顺序。

3.1.2 UML2.0 与 UML1.X 的区别

UML2.0 作为 OMG 组织的倾力之作，它更正和扩展了以前的版本。尤其是描述构件的能力大大增强。通过笔者比较和分析 UML1.X 和 UML2.0，归纳了了两者的主要区别，主要表现在以下几个方面：

A. 顺序图现在还有多个片断，允许在一个顺序图上显示更复杂的选择。顺序图片断支持顺序图元素的重用。

B. 新增两种交互图；时序图和交互概观图，为用户查看系统的动态行为提供新工具。

C. 图现在表达一流的思想，不再与状态机合在一起了新的活动图包括对大量附加流特征的支持，并且不再把重点放在各种状态转换上，而是放在沿着各个活动沿流动的令牌流上。

D. 扩展方面，UML2.0 着重强调用户配置文件的使用，以及各个模型级别，以便各个模型把重心放在各自的主要目的上，并使人们利用通用方法来解决各种问题，这些扩展更易于人们利用某些级别的模型生成代码。

E. 方面的扩展：复合结构的引入使得用户可以显示高层类别之间的连接，以及该类别所需要的运行时元素。

F. 对组件的环境需求提供明确规则，协议状态机和端口提供了在量支持组件开发的特征。

G. 用于定义某个类别的内部结构的规则更加明确，使得人们更容易于分解一个类别。

H. UML1.X 的协作图现在已经不使用了，变成了通信图。

3.1.3 UML 的定义

UML 是一种标准的图形化建模语言，它是面向实例分析与设计的一种标准表示。它不是一种可视化的程序设计语言，而是一种可视化的建模语言；不是工具或知识库的规格说明，而是一种建模语言规格说明，是一种表示的标准。UML 的定义

包括 UML 语义和 UML 表示法两个部分：UML 语义描述基于 UML 的精确元模型定义。元模型为 UML 的所有元素在语法和语义上提供了简单、一致、通用的定义性说明，使开发者能在语义上取得一致，消除了因人而异的表达方法所造成的影响。对于 UML 可以从以下几个方面来理解。

A. 不是过程，也不是方法，但允许任何一种过程和方法使用它；

B. 使用、表达能力强，进行可视化建模；

C. 与具体的实现无关，可应用于任何语言平台和工具平台；

D. 与具体的过程无关，可应用于任何软件开发的过程；

E. 简单并且可扩展，具有扩展和专有化机制，便于扩展，无需对核心概念进行修改；

F. 面向实例的设计与开发中涌现出的高级概念(例如协作、框架、模式和组件)提供支持，强调在软件开发过程中，对架构、框架、模式和组件的重用；

G. 可升级，具有广阔的适用性和可用性；

H. 有利于面向实例工具的市场成长；

3.1.4 UML2.0 主要组织特征

视图(view)：它显示了被建系统的各个方面，它是由多个图组成的一个抽象模型。在建立系统时需要建立多个视图，来显示多个方面的特征，从而描绘一个完整的系统。主要包括：用例视图，逻辑视图，实现视图，进程视图和部署视图。

图(Diagram)：它包括描述 UML 视图内的各种图形元素。通过图的相互组合提供被建模系统的所有视图。主要包括：用例图，类图，对象图，状态机(行为状态机，协议状态机)，活动图，顺序图，通信图，组件图，部署图和复合结构图。

模型元素(Model Element)：它代表普通面向对象的概念以及它们间的关系：关联、依赖和泛化关系。同一个模型可以在不同的 UML 图中使用，都保持其原有的特性。

通用机制(General Mechanism)：它可为元素提供一些额外的注释，信息或语义，还可提供扩展机制对 UML 扩展。主要包括：修饰，注释，规格说明。扩展包括构造型，标记和约束。

模型驱动架构(Model Driver Architechure,MDA)：将 UML 封装在一个 MDA 中，它力求 UML 模型变得更加可行，并且能与开发支持工具更好发集成。

3.2 基于 UML 的软件测试

基于 UML 的软件测试属于基于规格说明的软件测试。UML 模型在指导测试方面

有如下优点[39]：

(1) 通用性：UML 作为软件工程领域内的标准建模语言，目前已被广大软件设计者广泛使用，并取得了好评，同时也有很多商业软件工具来支持它。比如 Rational rose，OpenUML 等。

(2) 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，并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。综合上述原因，我们认为 UML 不仅是软件开发的重要建模工具，同时也是指导测试的重要模型。可将测试融合到软件开发的过程中。

(3) 强大的描述能力：UML 设计者吸取大量的优秀的软件思想，具有强大的描述软件开发各个阶段的能力。UML 包括了各种视图和模型，他们从不同的视觉和层次描述了软件系统的结构、行为及软件的使用。

(4) 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。

(5) 可复用性：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) 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。

3.2.1 测试阶段

一般软件测试过程包括三个阶段：单元测试，集成测试和系统测试。结合 UML 在软件设计阶段所刻画的侧重点不同，可将 UML 用于不同的测试阶段：对于单元测试，可以用状态图；对于集成和系统测试而言，仅从单个图所需的数据是不能完成

测试，因此需要从多个模型图中得到信息。本文的测试就是以构件图，顺序图和状态图相结来做测试。

表 3.1 UML 图与软件测试的关系

Table3-1 The Relationship between UML and Software Testing

软件测试各阶段	用到的 UML2.0 模型
单元测试	状态图
集成测试	类图，用例图，顺序图，通信图，活动图，状态机，交互概览图，组件图，时序图
系统测试	用例图，类图，对象图，状态机(行为状态机，协议状态机)，交互概览图，活动图，顺序图，通信图，组件图，部署图，复合结构图，时序图

3.2.2 基于 UML 的软件测试过程

UML 建模贯穿了整个软件过程，基于 UML 的软件测试过程如图 3-1 所示。

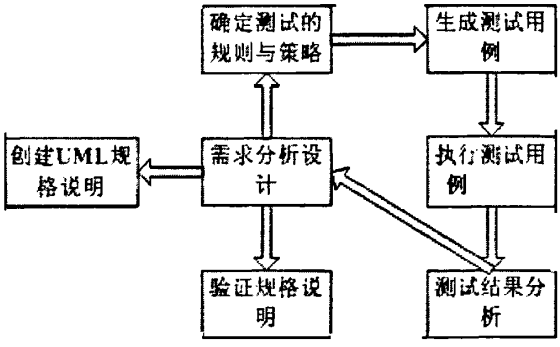

图 3-1 基于 UML 的软件测试过程

Fig3-1 A Testing Process of Software Based on UML

基于 UML 的软件测试过程，从图中可知测试过程可以分为以下主要步骤：

（1）创建 UML 相关规格说明文档。可从用户需求说明文档，软件概要和详细设计文档以及源代码等中产生规格说明文档。

（2）验证 UML 规格说明文档。由于 UML 规格只能从相关文档中整理得到，所以需对 UML 规格说明文档进行验证，主要包括 UML 模型是否是可跟踪性、是否是最优性、是否具有一致性、是否是结构良好性和有效等。可以通过手工审核，也可以利用工具来验证。整个过程中应根据实际的情况进行修正。

（3）生成测试覆盖准则和确定测试方法。主要是根据测试方法来确定测试的充分性，以及根据测试方法来准备测试数据。

（4）确定生成测试用例策略。根据规格说明文档，再结合测试方法，就可以得用测试用例生成算法，并产生测试用例。

（5）执行测试用例和分析。把生成的测试用例通过手工或工具执行，并将测试结果保存。

（6）测试结果分析。通过分析测试结果和预期的结果来验证测试用例是否有效，是否高效，并将相关信息记录到文档中。

3.3 UML2.0 中的构件

3.3.1 UML2.0 中对构件的定义

在 UML2-0 中，构件是系统的模块化部分，它封装了自己的内容，且它的声明在其环境中是可以替换的，构件利用接口提供服务。有以下几方面的含义：

（1）一个构件表示系统的一个模块部分，而且是一个自包含的单元，它封装了其内部成分的状态和行为。

（2）按照提供和请求接口，构件定义其行为。

（3）构件是可替换的单元，在设计时和运行时基于接口的兼容性，若一个构件能提供功能与另一个构件相同的功能，则前者就能替换后者。

（4）构件也可被看做是一种数据结构，是可执行的，是可实例化的。

（5）构件具有属性、操作和可见性，并能参与关联和泛化。

（6）可以用构件来装配大粒度的构件，主要是通过构件组合来实现。

UML2.0 把构件分为基本构件和包装构件。基本构件注重于把构件定义为在系统中可执行的元素。包装构件扩展了基本构件的概念，它注重于把构件定义为一组相关的元素，这组元素为开发过程的一部分。相当于定义了构件的命名空间。在构件的命名空间中，可以包括类、接口、构件、包、用例、依赖（如映射）和制品。

3.3.2 构件的接口

接口是表示对一组相关的操作进行声明的一种建模元素，它指定了一种契约，这些契约必须由实现这个接口的构件的任何实例完成。可以按各种约束（如前置和后置条件）的形式把一个接口与一个职责相关联，可以对通过这个接口的交互规定次序。接口分为提供接口和请求接口。把构件实现的接口称为提供接口，这意味着构件的提供接口是给其它构件提供服务的。构件可以直接实现提供接口，构件的子构件也可以实现提供接口。实现接口的构件支持由该接口所拥有的特征，此外，构件必须与接口拥有的约束相容。构件使用的接口被称为请求接口，即构件向其它构

件请求服务时要遵循的接口。一个构件可以实现多个接口，一个构件可以请求多个接口，一个接口可以由多个不同的构件实现。只要一个构件服从由另一个构件的提供和请求接口表达的约束，它就能和这样的环境进行交互。通过添加新增功能的新构件类型，能够扩展系统。

3.3.3 构件的端口

构件的端口声明一个构件的总的行为来说是有用的，构件的实现仅需保证要实现在全部的提供接口中的操作。使用端口是为了能进一步控制这种实现。

端口是一个封装构件的显式的对外窗口，是有标识的。在一个封装的构件中，所有进出构件的消息（调用或信号）都要通过端口。可用一个显式的对象实现端口，它也可能只是一个虚拟的概念。端口允许把构件的接口分成离散的组，并独立使用。端口是构件的一部分，端口的实例随着它们所属的构件的实例一起被创建和撤消。端口也可以具有多重性，用于指定在构件实例中特定端口的可能数目。构件实例中的每一个端口都有一个端口实例的排列。虽然排列中的端口都满足同样的接口并接受同种类的请求服务，但它们可能有不同的状态和数据值。例如，具有较高优先级的端口实例优先被满足。

3.3.4 连接件

如果一个端口提供一个特定的接口而另一个端口需要这个接口，且接口是兼容的，那么这两个端口便是可连接的。连接端口意味着请求端口要调用提供端口中的操作，以得到服务。设立端口和接口的优点在于在设计时，两个构件彼此不需要了解对方的内部，只要它们的接口是相互兼容的即可。如果需要使用新的构件，可以把这些端口重新连接到其它提供相同接口的构件上。把两个端口之间的衔接称作为连接件。为了连接构件或把端口与构件内的部件相连，UML 定义了两种连接件：委托连接件和装配连接件。

3.4 基于 UML 的 CBD 软件工程过程

基于构件的软件开发过程[35]可以分为需求分析与领域分析、构件的分析与设计、构件的测试、构件的检索、构件的组装以及集成测试与系统测试等阶段，如图 3-2 所示，在以上的阶段都可以结合 UML2.0 技术，使得每个阶段的任务清晰、有的放矢。

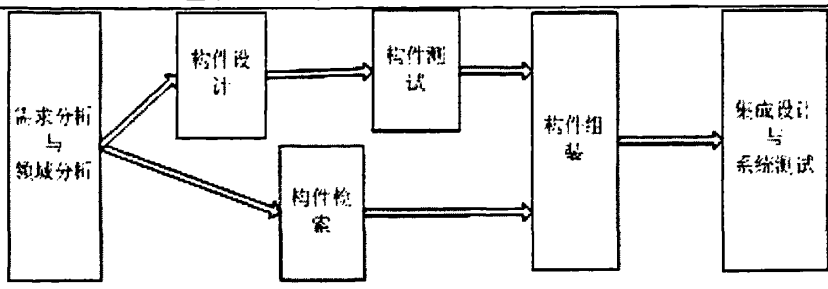

图 3-2 CBD 软件开发过程

Fig3-2 A process of CBD Software Development

3.4.1 基于 UML 的需求分析与领域分析

在用UML做需求分析与领域分析时，使用用例(case)来理解和捕捉系统的功能，它在整个开发过程中起着非常重要的作用，目前流行的用例驱动开发就是这个的写照,如图3-3所示。首先，它描述了待开发系统的功能需求；其次它将系统功能看成一个黑盒，从外部行为者的角度来理解系统；再次，它驱动了需求分析阶段以后的各个阶段的开发工作，不仅可以保证在开发过程中系统所有功能得以实现，而且对以后的测试和检测所开发的系统极其有利，应该说用例模型影响到了整个系统开发的各个阶段和软件测试的各个阶段。

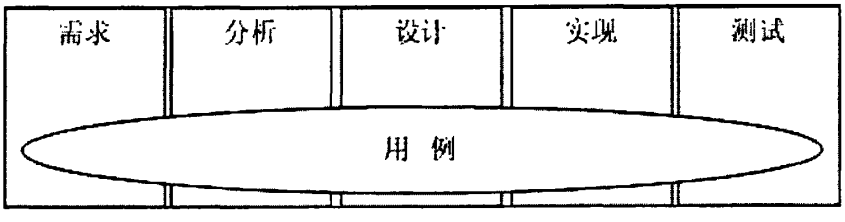

图 3-3 用例驱动开发模型

Fig 3-3 Use Case Driven Development Model

3.4.2 基于 UML 的构件的分析与设计

利用用例图来驱动构件的分析与设计，此阶段可以分为以下几个步骤：（1）用 UML 中的顺序图来标识用例中的对象以及对象间的交互，（2）用 UML 中的静态图(类图)构件中类之间的关系，（3）用 UML 中的组件图来表示组件与外界的交互。借助 UML2.0 中的各种动态图可以方便的分构件间的通信，也能方便我们分析出构件的对外接口。

3.4.3 基于 UML 的构件测试

在组装前应该保证单个构件是完全正确的。因此就得进行构件的单元测试。结合 UML 类图分析构件内部类间的组织关系，以及各种对外接口的情况。再对照构件规

格说明文档来验证被测试的构件是否和软件设计阶段的功能说明相符合。通过编写测试用例，测试构件是否有功能缺陷和接口缺陷。总体来说，这个测试过程和面向对象的软件的单元测试是一样的。

3.4.4 构件的检索与组装

由于构件软件的开发核心就是构件组装，被组装的构件有可能是自行开发的，也可能是 CTOS（第三方提供的）。为了日后复用，这些构件都将存入构件库中。所以我们在开发新的系统的时候就需要从构件库中检索出构件，并进行组装。这个过程比较复杂。构件图是组装的依据。文献[39]提供了一种组装方法。

3.4.5 系统和集成测试

在软件构件组装好了以后，紧接着进行的就是软件集成测试和系统测试。集成测试的目的是尽可能多地发现构件协作关系中的软件故障，这些软件故障一般都发生在软件各部分的行为交互上，特别是对基于构件的软件系统问题更加突出。系统测试是指测试整个系统，以验证系统是否完成了需求定义的功能，因此属于功能测试，相比之下，集成测试显得相当重要，这也是本课题研究的内容。本文将在 UML2.0 的基础上，提出测试模型，完成集成测试的相关研究。

第四章 一种基于 UML2.0 图生成测试用例的模型

在前一章中，分析了 UML2.0 图的相关特性，以及在 CBD 软件中的应用，本章将着重分析 UML2.0 模型中的构件图，顺序图和状态图，并在此基础上提出了一种 MISG 模型，最后将给出一种基于 MISG 模型生成测试场景的算法以及生成测试用例的算法。

4.1 UML2.0 构件图模型(Component Diagram CD)

构件图是 UML2.0 模型中的一种重要的描述软件系统结构的图形。它从软件架构的角度来描述一个系统的主要功能，如系统分成几个子系统，以及它们之间的关系以及它们分配到哪些节点上等。在构件图中，系统中的每个构件都使用构件符号来表示，通常构件图看起来像是构件图标的集合，这些图标代表系统中的物理结构，一个简单的构件图如图 4.1 所示。

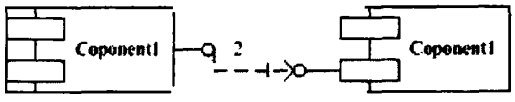

图 4-1 一个简单的构件图

Fig4-1 A Simple CD

构件图在 CBD 软件的集成测试过程中发挥着重要的作用，主要表现在以下几个方面：

（1）CD 确定了由哪些具体的构件来构造系统。因而在集成测试时可以根据这些构件或构件的组合分析产生测试准备数据。

（2）CD 确定了构件间的依赖关系，因而可以根据构件图对软件系统进行 UML 建模，从而提取具体场景的交互图和状态图，为构件间的交互分析提供了基础。

（3）CD 描述了构件通过接口对外提供服务的过程，为分析顺序图中的缺陷提供了依据，同时也为生成的测试场景的有效性提供了依据。

4.2 基于 UML2.0 状态机的构件状态行为的描述

在构件化软件系统中，当具体执行一个项目的业务过程时，参与到这个过程中的构件会响应各种操作，构件的状态可能会发生变化，这些变化是构件交互分析的重要部分。而状态机正好可以非常正确的描述这些信息。

4.2.1 UML2.0 状态机(State Machine　SM)

SM 描述事件如何引起对象状态的迁移。对应于消息的事件被发送到对象，要求对象做某件事情，这个事情被称作动作，动作导致了对象状态的变化。状态图通常包括两个部分：状态和转换。一个状态是指在对象的生命期中的一个条件或状况，在此期间对象将满足某些条件，执行某些活动或引起某些事件。转换是两个状态之间的一种关系，表示对象将在第一个状态中执行一定的动作，并在某个特定事件发生和某个特定条件满足时进入第二个状态。在图形上，一个转换用一条从原状态到目标状态的有向实线来表示，实线的上方表明相应的事件、监护条件和动作。一个简单的状态机如图 4-2 所示。.在 UML2.0 中定义了两种类型的状态机。

a. 行为状态机(Behavioral state machine BSM)描述类的生命周期的所有细节

b. 协议状态机(Protocol state machine PSM) 仅关注状态的转换和用来管理各个操作

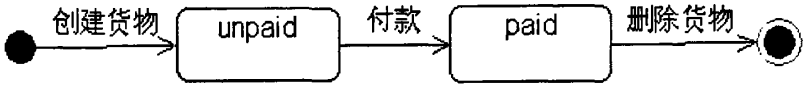

图:4-2 一个简单的状态机

Fig: 4-2 A Simple SM

在状态图中，状态的转换是由事件触发的(除了完成转换)，事件包括调用事件、变更事件、信号事件、时间事件等等，但最常见的就是对象接收到某个消息，转换发生时会执行某些动作，这些动作通常是向别的对象发送消息。另一方面在交互图中，对象之间的消息和状态图中的事件及动作是一致的。消息的发送通常产生一个动作，这个动作导致接收者对象的事件的接收。状态图解释了发送一个消息所导致的不同结果，因而也解释了一个对象的不同行为，这些行为依赖于对象的当前状态。

4.2.2 扩展 SM 描述构件的状态行为(CSM)

通过前面的分析可知：构件是类的集合体，在它的生命周期中也应具有多种状态。在构件系统运行过程中，构件的状态会因事件的作用而发生变化。因此在构件的整个周期中，对它的状态进行建模，则可得到构件的状态机。CSM 从全局上描述了构件交互过程中的构件状态转换情况。

可以看出标准的 UML SM 并不能充分描述构件的动态行为，如事件对应的接口等信息都没有反映在 SM 中。因此本文将对 UML 状态图的语法定义进行扩展，使其具有事件语义，并能够描述构件的动态行为，特别是构件间的交互。本文将 UML SM 作为产生如同黑盒子测试那样的测试基础。根据构件图提供的信息，对状态图进行

建模，将触发状态改变的事件语义添加到相关的状态上去。本文引入 CSP(Communicating Sequential Processes)[41]机制，定义了在 CSP 中通信中需要测试者统一命名的转换标签，用于标示构件间交互调用关系。名字的前缀是表示触发（引入）或发出（导出）事件。在状态图中的转换标签如下所示：

$$comoponent_a.in_m?sendEvent \wedge component_b.in_n!receiveEven$$

上述转换标签可以被解释为：收到来自构件 $component_a$ 接口的 $interface_m$ 的一个触发事件 sendEvent，触发了发送给接口 $component_b$ 的导出事件 receiverEvent。其中触发的（接受的）事件可用"_"符号标识，同时导出的事件可用"^"符号标识。具体实例如图 4-4 所示。本文把通过这种方式的扩展得到的模型称为 ECSD 模型。ECSD 不仅可以描述了状态的迁移，还描述了状态发生改变的因果关系，同时也为与构件顺序图集成提供了基础。

4.3 基于 UML2.0 顺序图/通信图的构件交互行为的描述

UML2.0 顺序图(如图 4-3 所示)是 UML 四种交互图中最重要的交互图。它描述了在时间顺序上多个对象在自己的生命周期内与其它对象间通过消息传递的交互行为.。UML 顺序图有两个维度：纵向是时间；横向是不同的对象. 这两个维度是可以互换的，横向也可以表示时间，纵向可以表示不同的对象。通常情况下，顺序图中只注重时间的先后顺序，在为实时系统建模时，这个时间轴可以有实际的度量。横向上对象之间的顺序是无意义的.

在 UML2.0 中可将顺序图划分成多个片段(包括分支,条件,循环和递归).。组合片段封装了顺序图中的部分内容，这些片段由一个框架包围，每个构架的上面都有一个记录处理方式的算子。有条件限制的用 alt 关键字标识，有循环处理的用 loop 标识，有交互引用的用 ref 标识。

4.3.1 基于 UML 的构件顺序图形式化定义

定义 4-1 (构件顺序图 CSD)可以表示为一个八元组

$$CSD =< Cpt, Msg, Event, f_m \to e, f_o \to e, f_e \to l >$$

其中：$Cpt=\{cpt_1,cpt_2...,cpt_n\}$，是实例的集合。$Cpt_1$，$Cpt_2$，…，$Cpt_m$ 都是顺序图中的构件。

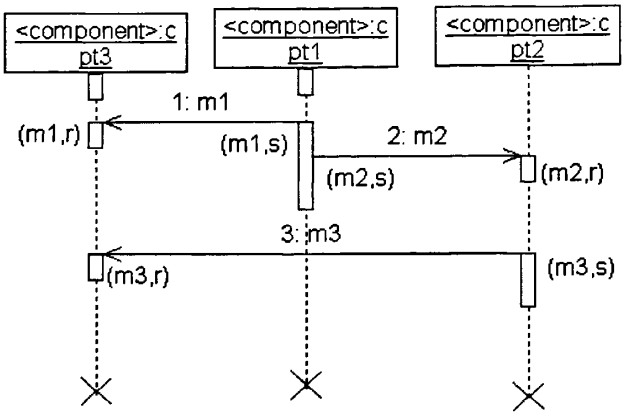

图:4-3 一个简单的顺序图模型

Fig 4-3 A Simple Model of SD

$$\text{Msg} \in \text{guard} \times \text{message} \times name \times parameter\_list$$

是消息的集合，顺序图中的每一个消息都形如:"[卫士条件]消息名(参数)".

$$Event = Msg \times (send, receive)$$

是事件集合。事件是指消息的发送和接收。对于消息 msg，发送事件可用 $<msg, send>$ 表示，接收事件用 $<msg, receive>$ 表示。顺序图中所有发送消息事件的集合记为 S，所有接收消息事件的集合记为 R：$R \cap S = \varnothing, S \cup R = Event$。

$f_{m->e}$ 是顺序图中位点的集合。位点是实例的生命线上发送或接收消息的点，每个位点都形如：<cpt,l>，其中 cpt∈Cpt.l 为构件实例 cpt 的位点.

$F_{m->e}$ 是 Message 到 Event 的映射关系，$f_{m \to e}(event) = msg$，表示事件 e 所对应的消息，因为顺序图中的每个事件都会产生特定的消息。

$f_{o->e}$ 是 Event 到 Cpt 的映射关系，$f_{o \to e}(event) = Cpt$ 表示实例的事件，因为顺序图中的事件只有外部事件和内部事件(由实例产生的事件)，这里指的是内部事件。对于实例 Cpt_i 具有的全部事件 $Evnet_i$，$Event_i = \{e \mid e \in foe(event) = Cpt_i\}$。

$f_{e->l}$ 是消息的事件到位点的映射关系，$f_{e->l}(e)$=loc 表示消息的事件可将消息传播到相应的位点，反映了消息的传播过程。

根据定义 1，图 4.3 的模型可以表示为：

Cpt＝{cpt1, cpt2, cpt3}

Msg＝{1,2,3}(用编号代替消息)

Event＝{(1,s),(1,r),(2,s),(3,r),(4,s)……}

4.3.2 CSD 语义分析

CSD 给出了消息发生的顺序，但没有给出消息接收和发送的事件的顺序，而消息的发生对于系统运行而言实际上是构件实例执行发送和接收消息的事件。由于顺序图中出现异步消息时，可以会引起消息并发，这时顺序图中给出的消息顺序就不是绝对的。虽然顺序图中消息的位置是自顶向下错落安排的，但对于并发的消息，实际发生的先后顺序并不是它们在顺序中的位置关系所显示的结果，而是可以同时或按任意顺序发生，这就是顺序图所固有的缺陷。因此本文将消息的顺序理解为弱顺序和强顺序。

定义 4-2 弱顺序:<=={(m', m) such as: $Dv = |Pm1 - Pm|, Dv' = |Pm - Pm'|, Dv < Dv'$, m_1 指顺序图中第一个发生的消息，P 为消息发生点的垂直方向坐标位置，Dv 和 Dv' 分别表示消息 m 和 m' 与 m_1 垂直方向上的距离}.

弱顺序是通过顺序图中的时间轴表示出来的，可直接观察到。它只是从表面层说明了消息的前后顺序关系，它说明了两个方面的内容：

任何消息的发送事件和接受事件满足因果关系，即发送事件一定先于接受事件，对于一个特定的构件实例 Cpt，传播的消息总是与一个接受事件和一个发送事件相联系的，那么消息的顺序可映射到对应的事件的顺序上去。所以在 Cpt 的生命线上的所有事件 Events 总有一个局部的全序关系"<"，在时间轴上就存在事件的顺序关系，即上方的事件先于下方的事件发生，定义 4-2 就充分说明了这点。

由于弱顺序并没有完全描述系统实际运行时真实的消息传播信息，因为顺序图无法反映出系统底层的架构信息和运行时情况，比如说存在异步消息就使得在顺序图中直接得到的顺序就不正确，这正是顺序图本身的局限性。因此下面我们定义了强顺序来弥补这个问题.将强顺序用 " << " 表示。它可以反映出系统本身固有的特性和底层结构所限定的事件间的真实顺序。这些系统特性主要有：是同步消息还是异步消息，消息的队列存储机制是 FIFO 还是随机存取的，消息传递过程中是否有异常发生等等。对于特定的两个事件 $event_1$,$event_2$,强顺序<<定义如下：

定义 4-3 强顺序：表示为"<<"，它表示系统中真实的事件的顺序，并不能从顺序图反映出来的。它应该具备以下原则：

对于同一个构件实例的消息，满足：

原则一 发送先于接收原则

对于一个实例 Cpt 的消息 msg∈Msg，总有<msg,s><<<msg,r>,即对于一个消息必经过发送事件发送后才能接收，这是最基本的条件。

原则二　发送先于原则

对于一个构件 Cpt 的消息 msg∈Msg，在该构件生命线位于<msg,s>上方的任何事件 evnet,都有 e<<<msg,s>。也即对于一个构件的任一发送事件，位于该构件生命线上方的所有的发送事件部是先于这个发送事件发生.在时轴上，同一构件的发送事件的顺序是确定的。

对于不同构件实例的消息，应满足：

先发送先接收原则：不同构件间发送消息，应有先发送的消息先接收，也即消息的接收只依赖于消息的发送事件.

实际上"<<"关系是"<"关系的一个特例，强顺序是系统内在的根本逻辑关系，只有通过分析系统需求才可能得到。所以通过顺序图是不能表现出来的，而顺序图中直观反映出来的顺序往往并不是系统真实的情况，或者有时候说是错误的。因而可能造成了用户的理解偏差以及产生不完全正确的时间约束顺序。如果直接使用弱顺序来分析顺序图，将不能准确反映系统的行为特征，出现许多不可预知的错误。比如图 4-1 中，如果认为消息 4 一定先于消息消息 5 发生，这是有问题的。因为在消息 2 发生后，消息 4,5 有可能是同时发生的。由于存在不正确的共享访问，低效率的资源使用，死锁，饥饿和优先级倒置等问题，这会导致最终生成的系统的不准确性和不完备性，从而出现错误。所以在使用 UML 顺序图分析构件的交互行为时，应结合系统实际情况，并需对得到的"顺序"进行验证，从而得到可信的构件交互序列。

在 CSD 中,可能存在各种组合片断。在这些片断内又可能存在多种结构(循环，分支等)，也遵守上述的消息顺序规则。对于不含片断的简单的 CSD，实质就代表了系统运行时的一个场景。它的运动轨迹就是相应的事件序列 ($event_1$,$event_2$,$event_3$...,$envent_n$)，其中事件序列所表示的含义是排在前面的事件在它后面的事件前发生，但是事件之间存在着强顺序，即系统实质运行情况。因此并不是所有的事件序列都是顺序图中表示出来的。由于强顺序并不是一个全序关系，所以一个顺序图的场景可能允许多个事件序列。对于复杂的顺序图，在分析满足事件关系的同时，还要分析各个片断内的情况，后面将做详细说明。

4.3.3 扩展构件顺序图（ECSD）

UML 顺序图描述了系统一个场景的运行过程。它强调了事件导致消息的传递。在这个过程中每个构件实例都会存在不同程度上的交互，每次交互后构件实例的状态很可能会发生改变。而交互后的实例的状态在顺序图中的表示却是不变的，因为在协作图中实例的状态是静态的，在交互的过程中是无法感知到的。所以，就需要一些信息来反映出顺序图中构件实例的状态的变化情况，从而需要对顺序图进行扩展。前文中扩展的 CSM 模型中，引入了事件，使得静态的状态具有了交互的能力。结合对应的顺序图可以发现两种模型具有这样的联系：触发状态改变的事件正好是顺序图中触发消息的事件。通过特定构件的事件可以将两者融合到一个模型中，笔者选择本文了把状态图集成到顺序图中，使得扩展后的顺序图能够最大限度的展现构件间的交互情况。ECSD 生成算法步骤如下：

第一步：利用 UML2.0 对将要测试的构件系统建模，构造出构件图。提取出构件所有的可能状态和构件接口的相关信息。构造出其对应的顺序图和扩展的状态图。

第二步：针对选定的测试场景，找出参与的构件，画出所有构件的 CSM 图和 CSD 图。并结合构件图构造出扩展的 CSM 图。

第三步：选定一个构件实例，依次选定状态图中的每个状态，分析导致其状态发生改变的事件，并根据这个事件在顺序图中找出通过该事件发送的消息，在该实例该消息处的激活期上附加上状态。重复此操作直到所有构件实例的状态附加完毕。

4.4 消息交互流图模型(MISG)

前文中将 CSD 和 CSM 进行了集成，构造了具有很强的动态描述能力的 ECSD。本节将在 ECSD 的基础上，提出一种消息交互流图(MISG)模型，并实现了由 ECSD 转换为 MISG 模型的算法。

4.4.1 消息交互流图(MISG)概念

MISG 的基本思想是将 ECSD 中的消息以及它们所关联的构件实例抽象为 MISG 中的执行节点，而将 ECSD 中的分支条件抽象为 MISG 中的判定节点，连接各节点的弧(因为 MISG 是有向图)则对应着顺序图中的消息流关系。形式上，可以将 MISG 表示为一个二元组 MISG=(NODE，EDGE)。其中：NODE 表示顶点，EDGE 表示边。对于顺序图中异步消息的处理，先通过 CSD 中的弱顺序进行构造，然后用强顺序来进行检验和修正。

4.2.2 MISG 的存储结构

生成 MISG 后，需要对其进行搜索遍历以得到满足特定要求的测试场景。为了实现自动处理，首先要对 MISG 设计适当的存储结构。针对 MISG 的特点，我们采用了扩展的十字链表作为它的存储结构。对于 MISG 中的每个节点，存在一个相应的顶点结构，它由八个域组成：

表 4-1 MISG 顶点数据结构

Table 4-1 Data Structure of Node of MISG

$S(n_i)$	$R(n_i)$	Mark	Info	Firsthead	Firsttail	$S(n_i)$state	$R(n_i)$state

其中，$S(n_i)$表示节点 n_i 所对应消息的发送构件实例，$R(n_i)$表示节点 ni 所对应消息的接收构件实例，Info 域存储顶点的相关数据信息，这里用对应的消息 n_i 表示，firsthead 和 firsttail 是两个链域，分别指向以当前顶点为弧头或弧尾的第一个弧节点，Mark 为标志域，当该节点被访问后，Mark=1，否则 Mark=0。S(ni)state 表示节点 ni 所对应消息的发送构件实例的状态，$R(n_i)$state 表示节点 n_i 所对应的接受构件实例的状态。这两个域刻画了实例在交互过程的状态变化。

对于 MISG 中连接各个消息节点的边，设置一个响应的弧节点，它由四个域组成：

表 4-2 MISG 顶点数据结构

Table 4-2 Data Structure of Node of MISG

Tailvex	Headvex	Hnextlink	tnexxtlink	conditions

其中，tailvex 和 headvex 表示当前弧的弧尾和孤头所代表的顶点在 MISG 中的位置，该域中值的表示方法与顶点节点 Info 域中的表示法相对应。Hnextlink 和 tnextlink 为两个链域，分别指向与当前弧具有相同弧头和相同弧尾的下一条弧。Conditions 表示消息发生时的各种约束条件。

4.4.3 具体算法

根据 ECSD 规模的不同，本文分两种情况讨论生成 MISG 模型的算法。

（1）对于简单的 ECSD（即不含有片断）。

根据 CSD 弱顺序原则将图中的消息及关系的构件实例抽象成 MISG 模型中的结点。将各结点的连线抽象成弧。然后根据 CSD 强顺序原则进行验证和修正。

（2）对于复杂的 ECSD。

由于一个 CSD 可能包含很多片断，片断中又嵌套片断。但是所有这些结构中最基本的只有顺序结构，alt 片断(分支结构)，loop 片断(循环结构)和 ref 片断(交互结构)，所以可以将顺序图理解为若干个小的片断结构逐步复合而成，是一个由小变大的过程。因此本文采用以下的步骤来生成 MISG 模型。

第一步：对于 $CSD=<Cpt,Msg,Event,f_m\to e,f_o\to e,f_e\to l>$，观察该图，如果含有片断，调用第五步的方法。将其封装为 Snippet，并将其保存到数组 Array 中。用同样的方法完成所有的最外层的片断。

第二步：将第一步中的 Array 做为顶级事件元素，利用弱关系，构造一个顶级的事件序列 eventListTop，再对照强关系，对 eventListTop 进行调整。

第三步：对第一步中的 Array 做进一步的分析。如果含有次子片断，则利用第一步，第二步，第五步的方法操作，并将结果进行保存。

第四步：重复利用第三步的方法，直到操作到最低层的片断。

第五步：由于片断中可能含有各种结构及其组合。这步主要是对这种情况的处理。如果存在 alt，则用 if/(if else)/(switch case)表示；如果存在 loop，则用 while/for 表达式表示；如果存在 ref，则将 ref 片断写进调用他的事件序列中。

4.4.4 基于 MISG 模型的测试覆盖准则

Goodenoug [43]等人于 1975 年提出了软件测试充分性的概念，它是判定测试数据集对于被测程序是否充分的准则。一个系统有大量的测试需求，不可能完全测试，因此需要根据实际情况提供一套科学的测试覆盖准则来检验测试工作是否充分，是否能尽可能的保证系统的正确性。

构件化软件系统的集成测试的主要内容是构件之间的交互测试。MISG 模型是由 ECSD 演变而来，将构件间的交互情况用图的形态等价的展示出来。因此我们可以从 MISG 中提取覆盖准则，并辅助于其它信息，包括消息传递前后构件实例的状态信息的变化，顺序图中嵌套的各个片断所要考虑的路径及强弱关系原则约束的顺序等。本文将结合这些因素，归纳出以下几个测试覆盖准则：

（1）构件实例覆盖准则

对于 $\forall component_i \in components, \exists method_i \in messagePath$，其中，component 和 method 都属于同一个构件，确定满足该条件约束的消息序列的最小集（关键路径）。该准则最大限度的要求参与交互的构件至少被覆盖一次。

（2）消息覆盖准则

对于 $\forall message_i \in MessageSet, \exists message_j \in MessagePath, message_i = message_j$，确定满足该条件的消息序列的最小集（关键路径）。该规则最大限度的要求参与交互的消息都应至少被覆盖一次。

（3）消息对覆盖准则

对于 $\forall message_i \in MessageSet, message_j \in MessageSet \exists message_{j,i} \in MessagePath$，确定满足该条件的消息序列的最小集（关键路径）。该规则最大限度的要求参与交互的消息组合都应至少被执行一次。

（4）逻辑路径覆盖准则

由于 ECSD 中存在分支，循环和片断等情况，需对各种可能的消息序列进行遍历，确定满足该条件的消息序列的最小集（关键路径）。该规则要求要对可能的逻辑上的路径进行至少执行一次。

（5）Z 路覆盖准则

对含有循环的的结构，需要分别执行循环 0 次，1 次，2 次。

（6）边界覆盖准则

对于边界值，约束条件等要进行至少覆盖一次。

4.4.5 基于 MISG 生成测试场景

前文中已将 ECSD 转化成了 MISG，下面将给出由 MISG 模型生成测试场景的方法。

定义 4-4 测试场景(Test scenario TS)是生成的 MISG 是一条由顶点和有向弧构成的一个有顺序的顶点序列。

测试场景表示了顺序图中的一个场景的线程执行的完整踪迹，也是参与顺序图的实例之间的交互的踪迹，消息的传递激活构件实例方法的执行是构件实例之间控制流的转移，而场景中通过消息激活的方法调用的参数定义和使用、实例的创建、使用和撤销明显表示了一条在控制流上执行的数据流．而在 MIDG 模型中可等价的表现为一条连通的路径。

根据我们提出的测试覆盖准则和不同的测试需求，可以根据不同的测试覆盖算法生成测试场景。下面将给出每种情况生成场景的方法。

（1）基于构件实例覆盖产生测试场景的算法（CreateTSByComponet）

主要思想：对于 ECSD 和 MISG，在两者转化的过程中，所有的构件实例都需要遍历到。具体步骤为：取 MISG 的初始节点 n_i,如果 n_i.firsttail.headvex 不为空，

则取 n_i 的接受/发送构件实例对放到集合 list 中，如果 list 包含了所有构件实例，则可以得到一个 TS,遍历结束。否则查找 n_i 的后 继结点进行操作

（2）基于消息对覆盖产生测试场景的算法(CreateTSByMessagePair)

主要思想：MISG 中的顶点个数(sumVex)与消息的数目对应的。所以在 sumVex 范围内，是否有消息的 $S(n_i)=R(n_i)$。从面可以确认是否覆盖完全。

（3）基于消息覆盖产生测试场景的算法（CreateTSByMessage）

主要思想：访问结点 n_i,如果 n_i.mark=1;则将后继结点入队列。如果当前指针指向了队链表队列的头元素，则遍历完成：如果 n_i.firsttail.hnextlink 不为空时，就将结点指向分支结点。如果为空的话则重复前面过程继续操作。

（4）基于逻辑路径覆盖产生测试场景的算法（CreateTSByLogicMessage）

主要思想：逻辑路径覆盖与消息覆盖的测试场景生成方法大体一致，主要不同之处有两点：一是对于分支节点的处理，不是任选邻接点，而是对所有的邻接点都需要加以考虑；二是对循环的处理，一般采用 z 路径覆盖准则，而两次循环是检测数据初值 / 使用故障的最小要求，进一步的覆盖是循环的边界条件，因为它是一个最主要的控制错误来源。在本文中。为了更加全面地检测错误。考虑跳过循环体，和执行循环体 1 次、2 次、最大次数和中间值次数 5 种情形，限于篇幅，这里不再详述。由上述算法产生的测试场景都是合法的测试消息序列，为了实现软件测试的根本目的。检测系统的容错能力和在异常处理方面存在的错误，可以在合法测试消息序列的基础上，采用一定的策略，生成若干非法的测试消息，称之为健壮性测试场景，

在实际应用中我们采用了如下几个方法。

（1）重复法对测试消息序列中的任意消息 $m_i(1\leq i\leq n)$，在它和其直接后继之间重复该消息 1 次，以检查测试消息序列对冗余的处理。使用该方法，至少可以生成 n 个冗余的测试消息序列。

（2）替换法对测试消息序列中的任意消息 m_i $(1\leq i\leq n)$，若 $S(m_i)$和 $R(m_i)$之间还存在其它的消息 $m_{i'}$，则用 $m_{i'}$ 替换 m_i，形成新的测试消息序列。

（3）对调法对测试消息序列中任意一对邻接消息(MIFG 中的邻接点) m_i 和 m_j ，将 m_i 和 m_j，在序列中的位置进行对调，产生新的测试消息序列，以检测消息的时序特性。

4.5 测试数据准备

要对软件进行测试不仅要有测试序列，更关键的是要有有效的测试数据。测试数据是软件测试的关键，测试数据的质量直接影响到软件测试的质量。基本测试数据集是根据被测构件实例的每一个输入操作的定义，通过采用测试技术而得到的数据集合，可针对每一个输入产生相应的数据集。

本文对顺序图进行了扩展，并利用 OCL 将各种约束信息附加到了图中，因此只需要针对一个特定的测试场景 TS，所对应图中的路径，就可以得到该测试用例的测试数据，具体步骤如下：

第一步：提取 TS 中的构件的配置文件，得到构件相关的数据规约。

第二步：根据 OCL 信息，提取相关约束信息

第三步：利用等价类划分的方法，将第一步，第二步中的条件进行整理和归约，形成测试数据列表。并根据测试结果来检查和完善测试数据。

4.6 生成测试用例

测试用例（Test Case）是为某个特殊目标而编制的一组测试输入、执行条件以及预期结果，以便测试某个程序路径或核实是否满足某个特定需求。测试用例是 CBD 软件质量的重要保障。测试用例应包括的四个部分:环境条件、输入、方法调用序列、预期输出。

本文在前面讨论的基础上提出了基于扩展顺序图生成测试用例的方法。具体步骤如下：

第一步：分析用例图，产生与用例图相关的顺序图和状态较，产生扩展的构件状态机。

第二步：利用 ECSD 生成算法，产生 ECSD，并生成 MISG 模型。

第三步：利用 TS 生成算法，产生该用例图的所有测试场景，遍历每一个场景，记录各个方法调用时的约束(包括状态信息和支条件)以及场景的输入和最终输出，用 and 运算符将各个约束相连。获得约束表达式后，要对它进行化简。对于状态约束，需要依据状态的 OCL 约束信息将它替换成实际的构件实例属性值的约束以便于化简。如果在约束条件中存在。运算符，需要将一个约束条件化简成多个不含 or 的约束。最终获取各个场景的约束条件、输入和预期输出。

第四步：使用等价划分方法确定场景的环境条件。首先，我们从顺序图中确定所有的测试单元(testunit)，在顺序图中，每一个交互的对象就是一个测试单元;对于每一

个测试单元，我们从带 OCL 约束的顺序图中导出相应的环境设置，环境设置的每一项被称为一个范畴(category);然后根据顺序图和状态图中的一些约束信息将每个范畴划分成多个选择(choice)，一个选择是一组相似值的集合，在所有的选择确定以后，与每个场景相关的选择将被组合在一起构成该场景的环境条件。对于简单系统，测试用例数目不大，采用常用的用例管理方法即可。但对于复杂系统，用例数目大，需要将测试用例复合成用例集，并采用分层的方法进行管理。

4.7 实例分析

本文以 ATM 系统做实例分析。该系统有三个构件：CardReader(读卡构件)，ATMGUI(ATM 用户界面构件，提供 User 操作 Account 的接口)和 Account(User 帐户构件)。本文取在 ATM 机取钱为用例场景。大致过程是：User 向 cardReader 插入卡，如果读卡失败则退卡，成功则初始化 ATMGUI，并打开 Account，提示 User 输入密码。如果失败则提示重新输入，最多可输三次，如果成功，User 可通过 ATMGUI 向 Account 发送取钱的命令。如果 Account 钱不够，则向 User 返回信息。否则取钱成功。并关闭 Account,退卡。

第一步:用例图的一个顺序图和对应的状态图

取钱过程的 CSD 如 4-4 图所示。有三个构件参与了交互。Account 是交互中的重点，因此本文只考虑 Account 的状态图，如 4-4 图所示。它有三个状态：close 状态（帐户没有被打开），open 状态（帐户信息打开）和 overdraft（帐户透支），它在交互过程会因各种消息而导致状态的改变。因此采用前文的方法对其进行扩展。对于 Account，当 cardReader?open(info)^account!时，Account 的状态会变为 Open；当 ATMGUI?exit(into)^account!do(close),Account 的状态转换为 Close：当 ATMGUI?get(info)^account!Check(money)时，Account 的状态转换为 overdraft。

第二步: 生成 ECSD

结合上述 CSD 和 ECSM，通过算法将 ECSM 中的状态附加到 CSD 中，生成 ATM 取钱的 ECSD，如图所示。它非常直观的反映了在取钱的过程 Account 的状态改变情况。使得比单纯 CSD 表达信息要丰富很多。

第三步:产生 ATM_MISG 模型，并产生测试场景

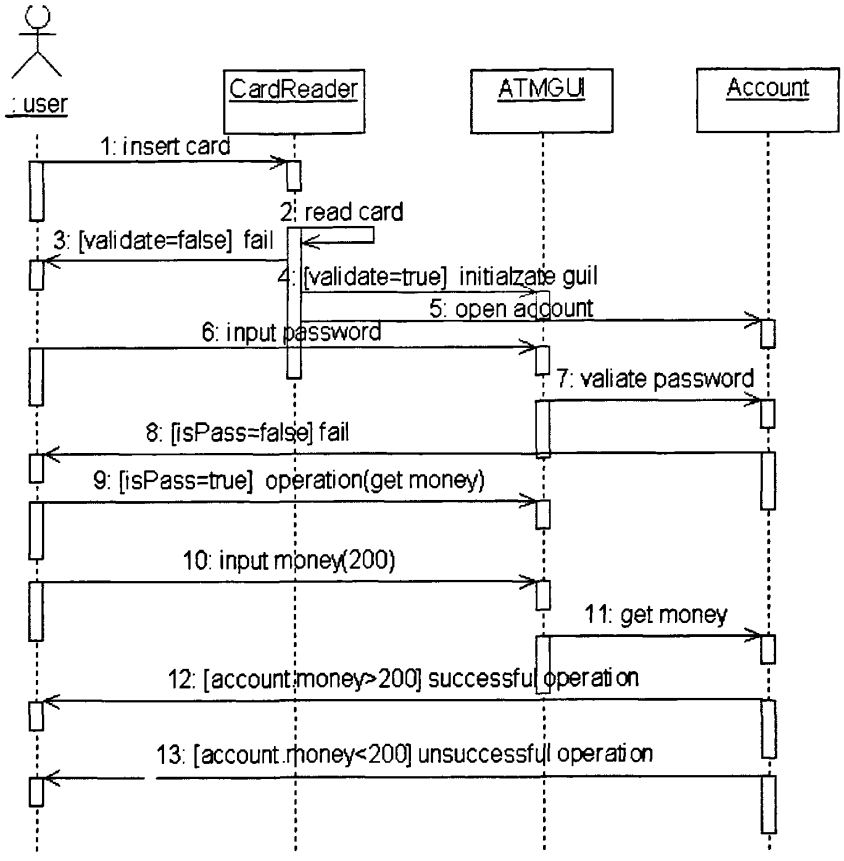

图:4-4 ATM 取款顺序图

Fig4-4 A CSD of ATM

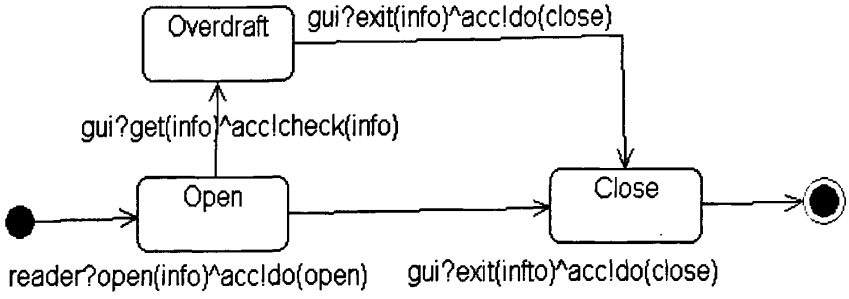

图:4-5 帐户状态机

Fig 4-5 CSM of Account

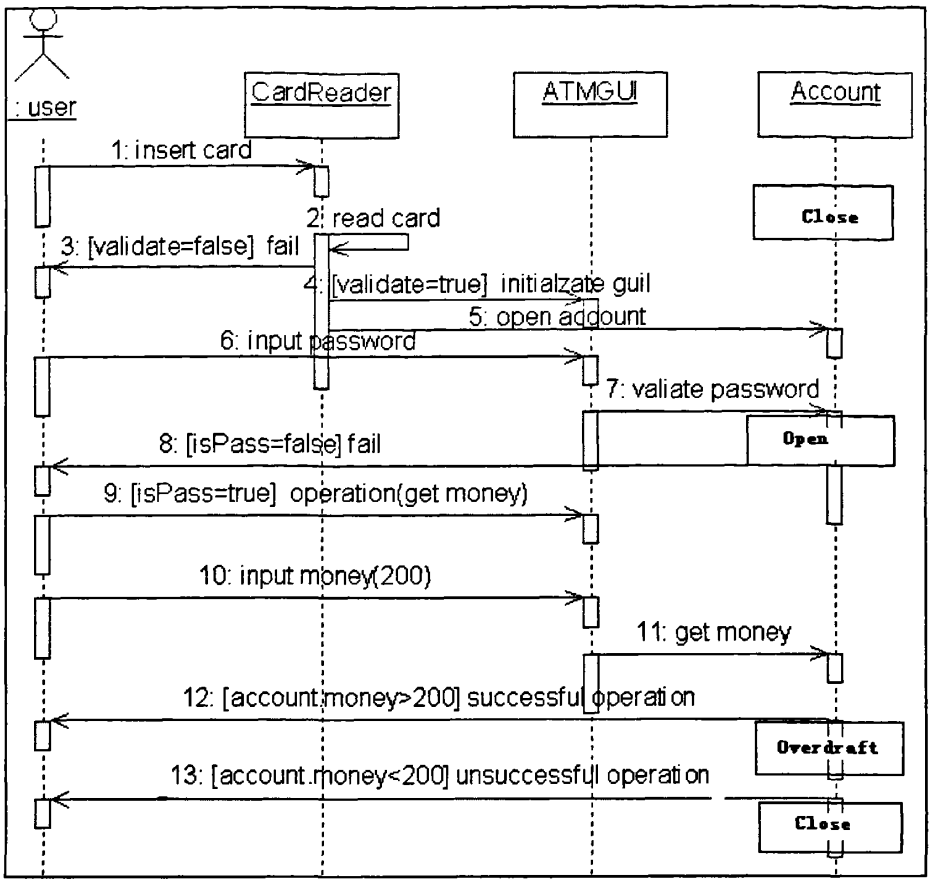

图:4-6 帐户扩展状态机

Fig 4-6 ECSM of Acount

为产生测试场景，先需要将 ECSD 转化成 MISG 模型，从而可以数学模型来分析和非形式化的 ECSD 了、应用算法产生 ATM_MISG 模型，如图 4-7 所示

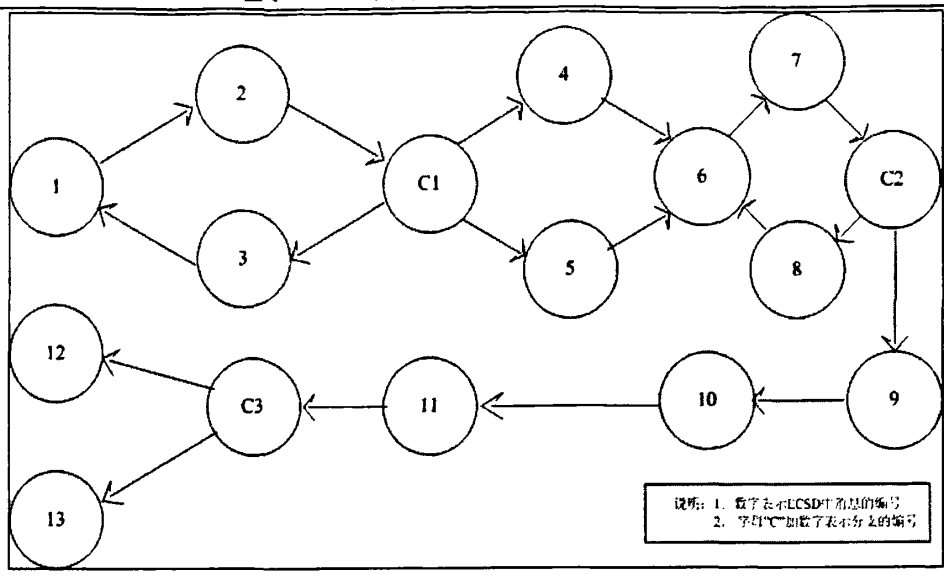

图:4-7 ATM MISG 模型

Fig 4-7 A Model of ATM MISG

图中的结点是由消息及相关的构件实例的结合体，根据 MISG 的存储结构定义，为图中每个节点和弧构造出相关数据信息。对于结点 11，节点信息可表示为：

表 4-3 MISG 顶点数据结构

Table 4-3 Data Structure of Node of MISG

atmgui	account	0/1	11	10	12/13	open	overdraft

弧结点的信息可以表示为：

表 4-4 MISG 顶点数据结构

Table 4-4 Data Structure of Node of MISG

12/13	10	null	null	1.金额数目是否正确 2.帐户金额信息

利用 TS 生成算法生成该 MISG 模型中的可能的测试场景。

(1,2,4,6,7,9,10,11,12),(1,2,4,6,7,9,10,11,13),(1,2,3),(1,2,5,6,7,8),

(1,2,5,6,7,9,10,11,12),(1,2,5,6,7,9,10,11,13)

可能数量很多，可采用求解关键路径的算法来减少数量。通过测试场景和 ATM 系统的说明以及 ECSD 中的约束信息，则可以生成 ATM 取钱过程的测试用例。

第五章 基于 UM 图的 CBD 软件集成测试框架(CSITF)

通过前面一章的介绍，我们对通过 UML2.0 模型产生测试用例的整个过程的分析有了深刻的认识。本章将从原型系统设计的角度对系统的需求、系统框架、涉及的相关技术和主要的功能模块作详尽的阐述。

5.1 原型系统功能需求

原型总体框架设计如图 5-1 所示，主要包括：（1）基于 UML2.0 对 CBD 软件建模部分：包括生成构件图，构件交互图和构件状态图。这些信息通过 UML 建模得到。本文采用 Rational Rose 产生的.mdl 文件，需用 UML 语法分析器提取有效信息[47]。并将这些信息存入到我们指定的数据结构中。（2）测试场景生成工具。主要包括 MISG 模型的定义及由 ECSD 生成 MISG 模型的方法。这步实现了由非形式化为数据模型的转化。从而可以利用图的理论来分析问题。（3）测试用例的生成和管理。主要包括测试覆盖准则，测试准备数据和测试用例修正。这步主要是提供测试数据来源和进行测试充分性的判定。

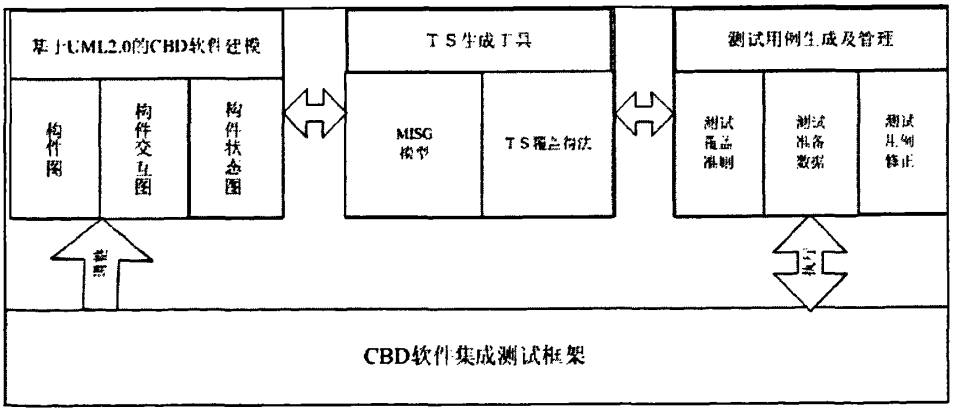

图 5.1 CBD 软件集成测试原型框架

Fig 5.1 A Integrated Testing Framework Of CBD Software

5.2 原型设计中的关键技术的实现

5.2.1 ECSD 构造策略

ECSD 主要是由 USD,CSM 和 CSD 中的信息集成整合而成。生成过程如图 5.2 所示。主要用到 MDL 到 XML 的文件转换算法(MXPraser)，ECSM 算法(Ecsm)和 CSM 和 CSD 集成算法(Ecsd)。

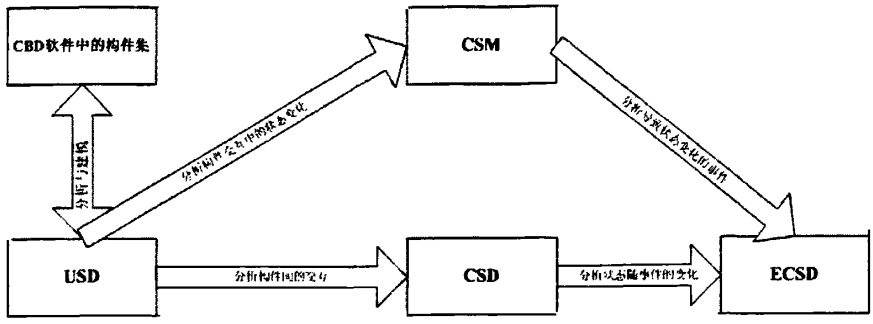

图 5.2 ECSD 构造过程

Fig 5.2 A Process Of ECSD

算法 5-1 MXPraser 算法。目的是将 UML 图转换成可操作的形式化文件。本文选用了 RationalRose 建模生成 MDL 文件，并使用文献[47]中的方法将 MDL 文件转换成 XML 文件。并通过定义 XMI Schema 将 MDL 文件减化我们需要的信息。

算法 5-2 Ecsm 算法。目的是将普通的 CSM 扩展成有状态事件驱动的图。利用 UML2.0l 图形标准扩展[27]中的标记值，约束和构造型将前章中的信息加到 UML 图中。然后使用 Ecsm 生成 XML 文件，然后对文件进行操作。实现代码如下：

```
public void XMLParser (String xmlpath,Object entity) {
  DocumentBuilderFactory domfac=DocumentBuilderFactory.newInstance () ;
  try {
    DocumentBuilder dombuilder=domfac.newDocumentBuilder () ;
    InputStream is=new FileInputStream ("bin library.xml") ;
    Document doc=dombuilder.parse (is) ;
    Element root=doc.getDocumentElement () ;
    class clazz=enttiy.getClass();
    NodeList List=root.getChildNodes () ;
    if(List.=null){
      for (int i=0; i<List.size();i--) {
        Node entity=List.item (i) ;
        if (entity.getNodeType () ==Node.ELEMENT_NODE) {
          //entity.getAttributes () 。getNamedItem ("对象属性名") 。getNodeValue () ;
          //利用上面的方法可以得到对象的各个字段的值
          for (Node node=book.getFirstChild () ; node! =null; node=node.getNextSibling
            if (node.getNodeType () ==Node.ELEMENT_NODE) {
              if (node.getNodeName () 。equals ("对象属性名") ) {
                //使用node.getNodeValue () 得到结点的值;
                //使用node.getFirstChild () 。getNodeValue () 得到孩子结点的值;
                }}} }}}
  } catch(ParserConfigurationException e){
    e.printStackTrace () ;
  } catch (FileNotFoundException e) {
    e.printStackTrace ()
  } catch (SAXException e) {
    e.printStackTrace () ;
  } catch (IOException e) {
```

算法 5-3 Ecsd 算法：实现方法是将 CSD 使用 XMLPraser 生成 CSD XML 文档，并通过类似于 Ecsm 算法得到基于 ECSD 数据结构的内容。然后通过两图的事件关系将基于 ECSM 的状态和约束信息添加到 CSD 中形成 ECSD 数据结构。核心实现代码如下：

```
//生成ECSD算法
//该算法需要两个参数:csd和ecsm
public void createECSD(CSD csd,ECSM ecsm){
  int msgCount;//CSD中消息序列的总数
  int csmCount;//特定的构件的状态的总个数
  Comonent Component;//构件实例
  Msg msg;//消息对象
  for(int i=0;i<msgCount;i++){
    componet=msg[i].componet;//接收消息的构件
    //如果消息的接收事件和状态图中的构件的接收事件相同，则要改变构件的状;
    if(msg[i].Component.REventId==csd.component.stateinfo.REventID){
      //从csd中查找状态并设置到构件的状态列表中
      for(int m=0;m<csmCount;i++){
        Component.stateinfo.add(stateinfo[i].stateinfo);
      } }
  }
}
```

5.2.2 测试场景(TS)生成策略

该生成策略是本文中最核心的算法之一，它有两个过程：将 ECSD 等价转换为 MISG 模型，在 MISG 模型的基础上得到测试场景。实现过程如图 5.3 所示。实现算法有：ECSD 到 MISG 的算法(ETM)和 MISG 到 TS 的算法(MTT)。

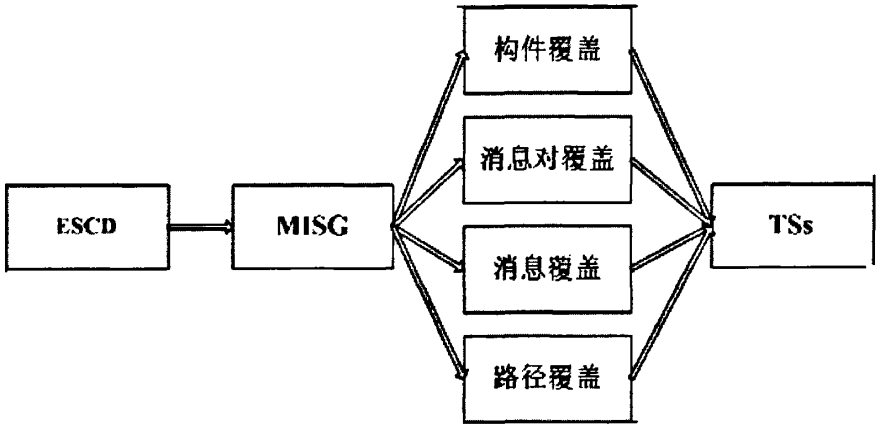

图 5.3 ESCD 生成 TS 过程

Fig5-3 A process of Creating ESCD

ETM 算法：主要分析 ECSD 中的消息，构件和连线进行等价转换为 MISG 模型中定义的数据结构。由于篇幅。具体算法省略。

MTT 算法。主要是根据 MISG 模型，将通过构件，消息，消息对和逻辑路径覆盖策略得到 TS。主要核心代码如下：

算法 5-4 构件覆盖产生 TS 算法

```
public List CreateTSByComponent{
    int sumMsg;//消息的总数目
    List Component;//构件列表
    List result,path;//path为测试场景
    int count=0;
    VexNode node;//为十字链表的初始结点
    while(node.firstTail.headVex!=null){
      path.add("("+S(node)+r(node)+")");
      result.put(S(node));
      result.put(r(node));
    }
    for(int i=0;i<Component.size();i--)
      for(int j=0;j<result.size();j++){
        if(Component.get(i).equals(result.get(j))){
          count++;
          continue;
        }
      }
    if(count==Component.size())//则得到一个测试场景
      break;
    slse {
        node=node.firstTail.headVex;//将node后继放入队列
        if(node.mark==1)//选取路径队列中没有访问的节点进行循环
           continue;
    }
    return path;
}
```

算法 5-5 消息覆盖产生 TS 算法

```
public List createTsByMsgPair(//通过消息来得到T S
  int msgCount;//十字链表中消息总数
  List tsPath;//路径
  //如果消息中的任意两两组合都能在有相应的发送事件与接收事件的匹配
  for(int i=1;i<msgCount;i++){
    for(int t=1;i<t;i++){
      if(node[i].S(node[i])==n[j].R(n[j]))
        tsPath.add(node[i]);
        tsPath.add(node[j])
    }
  }
  return tePath;
}
```

算法 5-6 消息对覆盖产生 TS 算法

```
private void  docase(){
  //将当前结点指向node.next;
  if(node.next.firstail.hnextlink~=null)
    return 1;
  else
    return 0;
}
private void dealcase(){
    //将node做为分支结点
  //并将node.next存入队列
  //在分支中进行遍历
  //如果当前结点的node.firstail.hnextlink==null,则结束
  //否则继续向后查找
}
private void output(){
  //操作队列将序列打印出来
}

public List CreateTSByMsg(){
  Node node;//设为链表的初始结点
  Queue,queue;//存放结点
  node.mark=1;
  node.inQueue(node);//将node放入到队列
  if(node.firstTail.headVex==null)
    output();
  docase();
  dealcase();
  if(docase()==1){
    dealcase();
  }
  else if(docase()==1){
    dealcase(j,1);
  }
  else{
    node.next=node.firstTail.headvex;;
    no=node.next;
    node.mark=1;
    queue.inQueue(node);
  }
}
```

```
public List CreateTSByMsg(){
  Node node;//设为链表的初始结点
  Queue,queue;//存放结点
  node.mark=1;
  node.inQueue(node);//将node放入到队列
  if(node.firstTail.headVex==null)
    output();
  docase();
  dealcase();
  if(docase()==1){
    dealcase();
  }
  else if(docase()==1){
    dealcase(j,1);
  }
  else{
    node.next=node.firstTail.headvex;;
    no=node.next;
    node.mark=1;
    queue.inQueue(node);
  }
}
```

5.3 测试用例生成及集成测试策略

本文生成测试用例的过程如图 5.4 所示。前面已经给出了生成测试场景的算法，已为生成测试用例打下了基础。再结合各种图的约束条件以及构件的说明文档，可以得到测试用例的相关数据。从而就可以得到系统的测试用例集合了。

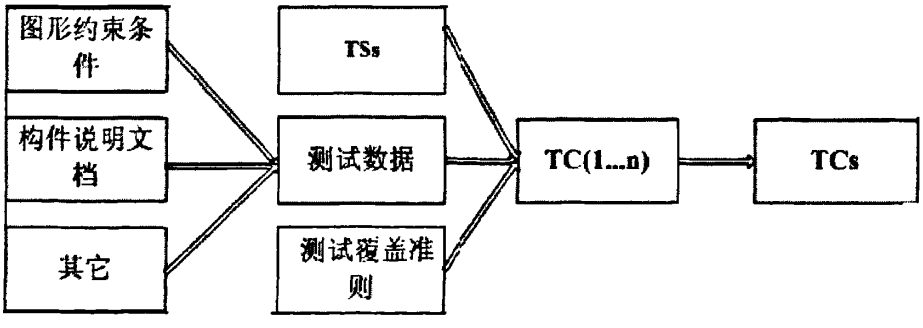

图 5.4 生成测试用例过程

Fig5.4 A Process Of Create TS

算法 5-7 生成测试用例核心算法：

```
public void createTS(){
  //从TS集合中取出一个
  //根据对应的事件序列，分析得到相关的各种约束条件，产生输入数据
  //根据序列和说明文档，分析得到预期结果
  //用TC数据结构将数据保存到数据库中
  //根据测试情况，再进行修改
  //最后检查是否符合覆盖准则
}
```

5.4 主要数据结构

该框架主要由构件表,测试用例表,测试场景表,构件状态表组成.分别介绍如下:

状态图中状态的数据结构

```
class StateInfo{
  private String id;//状态编号
  private String name;//状态名称
  private String preEventID;//发送消息的事件
  private String preComponentID;//发送事件的在的构件
  private String nextEventID;//接收的事件
  private String nextComponentID;//接收事件的构件
  private String stateInfo;//状态的相关信息
  private Map conditions;//约束信息列表
}
```

(2)扩展构件状态机的数据结构

```
class ECSM{
  private String id;//按照编码规则的编号
  private String componentID;//描述的构件编号
  private List stateInfo;//状态列表信息
}
```

(3)顺序图中的消息的数据结构

```
class Msg{
  private int id;//消息编号
  private String SEventID://发送消息的事件的编号
  private String REventID;//接受消息的事件的编号
  private String RComponentID;//接收事件所属的构件
  private String SComponentID;//发送事件所属的构件
  private Map conditions;//消息的相关约束条件
  private String info;//相关数据信息
}
```

(4)构件的数据结构

```
class Component{//描述构件的相关信息
int cID, //构件的唯一标识符
int cType, //构件的类型,包括简单构件,复合构件
String cName,//构件的名称
int Member[NUM],//内部构件集合,当为复合构件时,否则为空
int sendM[NUM],//发送消息集合,
int recM[NUM],//接受消息集合,
};
```

(5)构件环境参数的数据结构

```
class ComParam{ //构件参数信息
int PID,//参数编号
int cID, //构件编号
String cPName,//参数名称
int cPType,//参数类型
String sService,//参数所对应的服务
int value, //参数的值,如果有
String scope, //参数的范围,如果有
String note, //备注信息
};
```

(6)测试用例数据结构

```
class TestCase{//测试用例信息
String tID,//编号
String tName,//名称
Date   tDate, //日期
String tPerson,//录入人
int    tType,//类型
String Path, //测试用例的执行序列
String startState,//开始状态
String endState,//结束状态
char   mark, //测试标记
};
```

(7) ECSD 数据结构

```
clas ArcNode{//表示弧结点信息的类
  Object sender ;//发送消息的构件实例
  Object receiver ;//接受消息的构件实例
  int mark;//标志该节点是否被访问过,0:未访问1:访问过
  String info;//该顶点的相关的数据信息,即消息编号
  ArcNode firstHead;//指向以当前顶点为弧头的第一弧节点
  ArcNode firsttail;//指向以当前顶点为弧尾的第一弧节点
  String senderStatus;//表示当前发送消息的构件实例的状态
  String receiverStatus;//表示当前接受消息的构件实例的状态
}
```

(8)MISG 中的弧的数据结构

```
class ECSD{
  private String id;//编号
  private List<Msg> msg;//消息列表
  private String info;  //相关信息
};
```

(9)MISG 中的结点的数据结构

```
class VexNode{//MISG顶点存储结构
  Map constri constraintCondtion;//消息所具有的各种约束条件
  int tailVex;//表示当前弧的弧头所代表的顶点在MISG中的位置
  int headVex;//表示当前弧的弧尾所代表的顶点在MISG中的位置
  ArcNode  sameHeadNextLink;//指向与当前弧具有相同弧头的下一条弧
  ArcNode  sameTailNextLink;//指向与当前弧具有相同弧尾的下一条弧
}
```

(10)MISG 中的十字链表的数据结构

```
class OLGraph{//十字链表类
  VexNode List[MAX_VERTER_NUM];//表头元素
  int vexnum;//有向图的当前顶点数目
  int arcnum;//有向图的当前弧的数目
```

本章主要对 CBD 软件集成测试的原型框架及其相关技术进行了阐述，使得我们对基于模型的测试方法从工程实践上有了深刻的理解。同时也为设计 CBD 软件集成测试工具提供了一种思路。

第六章 总结与展望

6.1 总结

本文从基于 UML2.0 的软件测试技术的角度出发，结合 CBD 软件集成测试的特点，着重探讨了基于构件交互生成测试用例测试的方法和 CBD 软件集成测试框架的搭建。并给出了一种基于 UML2.0 模型生成测试用例的方法以及测试框架（CSITF）。主要完成的工作如下：

（1）扩展构件状态机，生成 ECSM 模型。分析了状态机的在测试中的不足，然后结合构件图将状态机扩展为具有事件语义的可描述构件交互的模型（ECSM）。

（2）集成 ECSM 和 CSD，生成 ECSD 模型。分析了构件顺序图中所描述的构件交互具有一定局限性的问题，并将交互关系细化为强弱顺序来解决这个问题。同时，通过 CSD 和 ECSM 之间的事件语义关系，将两者有机集成为可充分描述构件交互的模型（ECSD）

（3）构造 MISG 模型，并生成测试用例。由于 ECSD 模型信息量大，不便于处理，进而提出了消息交互流图（MISG），给出了由 ECSD 转化为 MISG 模型的算法以及基于 MISG 模型的测试覆盖准则，并分析出了 CBD 软件测试准备数据生成策略，最后给出了一种基于 MISG 生成测试用例的方法。

（4）提出了 CBD 软件集成测试框架(CSITF)。分析了 CSITF 的功能需求，设计出了系统结构图和关键模块。

6.2 展望

由于 CBD 软件固有的特点使得集成测试难度非常大，而且软件测试的充分性一直是学术界的一个难题。由于作者的水平和研究时间有限，目前只是给出了一种测试方法和原型系统。这为后续的研究奠定了基础，后续的主要工作表现在以下几个方面：

（1）需进一步完善基于 UML2.0 图的新特性在软件测试中的应用。本文只是从构件图，顺序图和状态图角度来考虑了问题，所以需从更多的角度来分析和研究。

（2）需进一步完善 MISG 模型及其生成测试场景的算法。本文中的 MISG 模型只支持 UML2.0 中的比较简单的图形的情况。因此需加强模型适应复杂的 UML2.0 图的能力。同时生成算法需优化以提高准确性和高效性。

（3）本文的原型系统也需改进，同时作者只实现了系统中部分核心功能，因此需开发出完整的系统用于实际的应用中，以检验和改进系统。

参考文献

[1] 郑人杰，软件测试、质量与可靠性，北京：国防工业出版社，1994:53-54

[2] 朱鸿，金凌紫，软件质量保障与测试，北京，科学出版社，1997: 258-259

[3] S. Keene,Modeling software R&M characteristics,Reliability Review ,part I:VOL 17, 1997 Jun,pp 5-28;part II:vol 17,2003 Sep,pp 13-22.

[4] Weynker EJ; Testing component-based software a cautionary tale[J] IEEE Trans on Software Engineeriing.2002,11(7) 54-59

[5] Poul G Component-based software development approach new opportunities and challeges[C] IN Proc of Technology of Object-Oriented Language.2004:375-383

[6] 单锦辉，姜瑛，孙萍.软件测试研究进展[J].北京大学学报(自然科学版)，200541(l):134-145

[7] 朱三元，钱乐秋.软件工程技术概论「M].北京:科学出版社，2002.168-193

[8] Wanglin zhang,Jiesong Yuan,xiaofeng Yu etc. Generating test cases form UML Activity Diagrams based on Gray-box method. Proc 11th Asla-Pacific Software Enginerring Conference(APSEC04),Busan,Korea,2006,284-291)

[9] J.Hartman, C. Imoberdorf and M. Meisinger, UML_Based Integration testing,2000

[10] Ye Wu,Mei-Hwa Chen, and Jeff Offutt. UML_Based Integration Testing for Component_Based Software Systems,Ottawa,Canada,Feb.10-12,2003

[11] 齐治昌,谭庆平,宁洪.软件工程[M].高等教育出版社.,2004,4

[12] Bertolino A,Basnieri etal.A practical approach to UML_Based derivation of integration tests,in Proceeding of QWE2000,Brussels,Belgium,Paper3T(CD2ROM),2000

[13] SmithMD,RobaonDJ.A framework of testing object-oriented programs[J],Journal of Objected Progrmgming.2003,3

[14] Marko Jantti,Tanja Trol>UML_Based Testing:A Case Study,Proceedings of NUML'2004 2nd Nordic Workshop on the Unified Modeling Language,August,2004

[15] 张毅坤，施风呼，刘军等。基于 UML 状态图的类测试用例自动生成方法 计算机工程 2003.12

[16] Edwanls S H.A Framework for Pratical,Automated Black-Box Testing of Component-Based Software software Testing[D],Verification and Reliability,2006,11(2) 97-111.

[17] Yoon H,Choi B. An Effective Testing Technique for Component Composition is EJBs[C],In:Proceeding of the 8th Asia Pacific Softeare Engineeing Conference Macau SAR,2005,229-236.

[18] Martins E,Toyota C M,Yangwaw R L.Constructiong Self-Testable Software Components[c]: In: Proceedings of the Internatianl Conference on Dependable System and Networds,Goteborg,2006,151-160

[19] Lee S,C,Offutt J.Generating Test Cases for XML_Based Web Component Interactions using Mutation Analysis In: Proceeding of the 12th International Sysmpoaium on Software realiability Enginerring,Hong Kong,2004:200-209

[20] Component+Partners.Built-In Testing for Component-Based Development EC IST Frameword Project IST-1999-20162 Component +Friday,9,November 2001

[21] Orao A, Harrokl MJ,Rosenblum D,et al>Using Component Metacontent to Support the Regression Testing of Component-Based Software[C] In:Proceedings of the International Conference on Software Maintenance Florence,2001,716-725

[22] Ma YS, Oh SUM, Bae DH,et al. Framework for Third Party Testing of Component Softwaret[C]:In: Proceedings of the 8th Asia Pacific Software Engineering Conference Macau SAR,2004,431-434

[23] Jams-Erol Erolsspm agims {emler Broam :upms Davod Fadp UML 2.0 ToolKit 电子工业出版社 2004.10

[24] 杨芙清，梅宏，李克勤.软件复用与软件构件技术[J]，电子学报，1999,2.

[25] C. Szyperski. Component Software-Beyond OO Programming. Addison-wesley,1997，2

[26] 杨芙清，王千祥，梅宏，陈兆良.基于复用的软件生产技术[J]，中国科学(E 辑)，2001,31(4).

[27] Ye Wu and Dai Pan and Mei_Hwa Chen.Techniques for Testing Component-Based Software.Seventh International Conference on Engineering of Complex Computer Systems.

[28] Td Lewis. The next 10,000 years part2[J] IEEE Computer,2006 29(5)

[29] Lan W Brown,Kurt C Wallnau, The Corrent State of CASE[J] IEEE software,1998,15(5)

[30] Adler RM.Emerging Standard for Component Software.IEEE Computer,2000,28(3),68-77

[31] http://www.microfot.com

[32] http://www.sun.com

[33] 冯茂岩 基于构件的软件测试若干问题分析 计算机与数字工程 2006 34(6)

[34] 黄柳青 构件中国 清华大学出版社 2006.5

[35] 聂长海，徐宝文.一种最小测试用例集生成方法软件学报计算机学报第 Vol 12:(12) 2003.12

[36] 张毅坤，施凤鸣，姚全珠，刘军，付长龙.基于以 L 状态图的类测试用例自动生成方法计算机工程 2003.12

[37] 曹严元，张为群种基于 CB。的软件测试方法.计算机科学，2005，2

[38] Clemens Szypershi Dominik GruntzStephanMurer 著，王千祥译.构件化软件-超越面向构件实例的编程[M].电子工业出版社，2004，37-45，415-421.

[39] 张毅坤，叶涛，邢传玉.基于合约检查的构件化软件集成测试框架[J].计算机工程.2006.

[40] 单锦辉，姜瑛，孙萍.软件测试研究进展[J].北京大学学报(自然科学版)，2005，41(l):134-145

[41] 15.江泽凡、王林章，李宣东，郑国梁.基于 UN 几顺序图的测试方法闭.计算机科学，2004，7，131-135.

[42] 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[]J.计算机科学，2004，8，179-184.

[43] Xie K,Xu B W,Nie C,H Component System Regrssion Testing Method Based ON CTAM[J] Journal of SouthEast University 2005,21(3) 282-286

[44] Mats S,Runcson P.A case study on regression test suite maintenance in system evolution[C] In Proc of the 20th IEEE International Confrerece on Software Maintenance,2004:302-310

[45] 谢棠棠，张为群一种基于 UN 工模型的系统测试方法[1J.西南师范大学学报(自然科学版)，2005，2.

[46] Hartman J, Imoberdorf C, Meisinger M. UML-based Integration Testing[C]. Proceeding of the International Symposium in Software Testing and Analysis (ISSTA'00), IEEE Computer Society Press, 2000

[47] MDL 到 XML 转换：http://www.51cto.com/art/200601/17063_4.htm

作者在读研期间参与科研及论文发表情况

[1] 宋建华，谭瑛，周小波，基于场景和 UML 图的构件系统集成测试方法研究 太原科技大学学报 2009. 5

致谢

首先要感谢我的导师谭瑛教授和周小波讲师。谭老师严谨并灵活的治学态度、谦虚且和蔼的待人方式、敏捷而严密的思维等等都让我获益匪浅。周老师严谨的治学、渊博的知识、平易近人的态度、朴素的生活作风等等也都是我学习的榜样。两位导师耐心对我进行指导，为我顺利完成课题奠定了坚实的基础，起到了重要的导航作用，他们在我研究生期间带给我的为学、做人、处世各方面的知识会让我受益终生!

其次要感谢太原博华软件公司和太原清华网络有限公司的全体员工，在我在博华实习的那段日子里，他们给我了很多的指导和帮助，进一步丰富了我的知识面，使我对计算机软件的认识到了另一个高度!

还要感谢师兄胡文鹏，张亮，黄海涛和王国强，他们在我平时的学习和写论文期间，给了我热情的鼓舞和帮助，同时也给了我许多建设性的想法，使我从中受益匪浅!

最后我要感谢我的家人，你们一直是我前进的动力!感谢所有帮助、指导、关怀、批评过我的人。