UML 建模技术在虚拟实验仿真系统开发中的应用研究

摘 要

针对教育资源相对不足的现状，我国正在积极发展网络教育，运用先进的信息技术和教育技术整合各类教育资源。实验教学一直是网络教育的一个薄弱环节，迫切需要建立多种实用、高效的网上虚拟实验室。虚拟实验仿真系统研究课题就是在这种情况下提出的。

虚拟实验仿真系统分为仿真平台和管理平台两个部分。仿真平台仿真了电类实验（电路分析、电路与信号和数字电路）中大部分实验器材，学生可以在仿真平台进行实验。仿真平台采用 ActiveX 技术，使用 Visual C++语言在 Windows 操作系统下开发。管理平台增加了学籍管理、实验管理、学生实验等模块，与仿真平台一起构成了一个完整的系统，可以较好的为高校远程实验教学服务。管理平台基于 B/S 结构，在 JAVA 平台上进行开发。

本论文介绍了 UML 建模技术在虚拟实验仿真系统开发中的应用，结合软件工程思想讨论了面向对象的分析、面向对象的设计、面向对象的测试的理论，以及这些理论在虚拟实验仿真系统中开发各个阶段的应用。

论文的第二章介绍了 UML 技术的相关内容，以及面向对象技术和面向对象的软件工程。第三章详细介绍了 UML 技术在系统开发需求分析、系统设计、对象设计阶段的应用，也讨论了系统实现阶段和测试阶段的工作。

关键词： UML 面向对象设计 软件工程 虚拟实验

STUDY ON UML MODELING TECHNOLOGY IN THE DEVELOPMNET PROCESS OF VITUAL EXPERIMENT SYSTEM

ABSTRACT

According to the situation of the relative lack of the education resources, our country is developing network education and use advanced information technology and education technology to optimize all kinds of education resources. Experiment education need establish many applied and effective network virtual laboratory because experiment education is a weak part of network education. The study of the virtual experiment system has been brought forward because of the above mentioned.

Virtual experiment system has two parts: emulation platform and management platform. The emulation platform has emulated the most apparatus of electric experiments, including circuit analysis experiment, circuit and signal experiment and digital circuit experiment. Students can finish their experiment on the emulation platform. The development of emulation system used ActiveX technology and Visual C++ programming language under Windows operation system. The management platform added "user management", "experiment management", "student experiment" modules and so on. The emulation platform and the management platform make up of a full system. The full system can serve distant experiment education of universities. The development of the management platform is based on B/S architecture and use java technology.

The paper introduced the study on UML in the development process of virtual experiment system, discussed the theories of Object-Oriented Analysis, Object-Oriented Design and Object-Oriented Test and the application of these theories.

The second part of the paper introduced UML technology, Object-Oriented technology and Object-Oriented software engineering. The third part of the paper introduced in detail the application of UML in requirement analysis, system design and object design and discussed the work of system implementation and testing.

KEY WORDS: UML, Object-Oriented Design, Software Engineering, Virtual Experiment

独创性（或创新性）声明

本人声明所呈交的论文是本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，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：＿＿＿＿＿＿＿＿　　日期：＿＿＿＿＿＿＿＿

关于论文使用授权的说明

学位论文作者完全了解北京邮电大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属北京邮电大学。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磁盘，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。（保密的学位论文在解密后遵守此规定）

保密论文注释：本学位论文属于保密在__年解密后适用本授权书。非保密论文注释：本学位论文不属于保密范围，适用本授权书。

本人签名：＿＿＿＿＿＿＿＿　　日期：＿＿＿＿＿＿＿＿

导师签名：＿＿＿＿＿＿＿＿　　日期：＿＿＿＿＿＿＿＿

第 1 章　引言

1.1 论文背景

1.1.1 远程教育概况

远程教育是学生与教师、学生与教育组织之间主要采取多种媒体方式进行系统教学和通信联系的教育形式，是将课程传送给校园外的一处或多处学生的教育。现代远程教育则是指通过音频、视频（直播或录像）以及包括实时和非实时在内的计算机技术把课程传送到校园外的教育。

世界远程教育的历史可以追溯到本世纪 30 年代。随着先进的信息技术，特别是互联网的出现，远程教育的特征发生了深刻的变化。远程教育技术大致经历了以下 4 个阶段的发展：

第一代远程教育技术的特征是单向传输。这一时期远程教育技术主要用于从老师到学生的信息传递，这种传递模式没能起到学生之间沟通的作用，仅实现了师生之间有限的交流。当时的传输技术还受到时间的限制（例如学生们收听收音机和收看电视节目的时间是预先安排好的）。

第二代技术出现在 1960 年，大大改进了第一代技术对时间的依赖性。录像机和有线电视的出现，使远程教育课程传播部分不受时间限制，将录制好的课程内容的录像带发给学生，使他们可以随时观看。然而，在别的方面，这一代远程教育技术同上一代相比并没有太大的不同：学生之间、师生之间的交流还是很少。

第三代远程教育技术同以前相比，教师可以传送大量更加复杂的信息给学生，使学生之间、师生之间可以通过电子邮件、聊天室和电子公告牌进行交流。计算机辅助教学、计算机模拟以及其他通过计算机磁盘、光盘和互联网等途径的电子资源进一步表现出这一代远程教育的特征。

第四代远程教育技术更加先进。学生之间、师生之间的交流得到了加强。进

行交换的信息的数量和种类显著增加，信息更新更快捷。从而减少了远程教育对时间和空间的依赖性，使实现真正意义上的虚拟大学成为可能。

现代远程教育融合了第二、三、四代技术，是随着现代信息技术的发展而产生的一种新型教育方式。计算机技术、多媒体技术、通信技术的发展，特别是因特网(Internet)的迅猛发展，使远程教育的手段有了质的飞跃，成为高新技术条件下的远程教育。

根据远程教育的定义，可归纳出它有如下一些特点：

- 学生与教师分离
- 采用特定的传输系统和传播媒体进行教学
- 信息的传输方式多种多样
- 学习的场所和形式灵活多变

与传统教育相比，远程教育的优势在于它可以突破时空的限制；提供更多的学习机会；扩大教学规模；提高教学质量；降低教学的成本。基于远程教育的特点和优势，许多有识之士已经认识到发展远程教育的重要意义和广阔前景。

1.1.2 课题的提出

针对教育规模大、教育资源相对不足的现状，我国正在积极发展网络教育，运用先进的信息技术和教育技术整合各类教育资源。实验教学一直是网络教育的一个薄弱环节。迫切需要建立多种实用、高效的网上虚拟实验室。利用网上虚拟实验室进行教学，可以解决传统实验教学方式在时间和空间上的限制，大大节约实验成本和经费。并可更好地培养学生的自学及创新能力。随着远程教育的不断普及，网上虚拟实验室也必将得到越来越广泛的应用。

该课题是北京邮电大学网络教育学院的项目，它包括三个子课题：电路分析虚拟实验、电路信号虚拟实验、数字电路虚拟实验。课题的提出主要为北京邮电大学的网络教育服务，课题同时包括了一个虚拟实验的管理平台的开发。本文主要介绍了 UML 技术在电路分析虚拟实验系统的应用。

1.1.3 课题的开发目标

依据电路分析、电路信号、数字电路实验课程的教学内容，用虚拟的方法模拟实验中实际用到的器材设备，提供给用户与真实环境相同的情景，用户按照要

求设计完成这三门课程的实验。针对成人本科教育，开发配套的可在网上开展实验的虚拟实验系统，解决相关的实验教学问题，同时为其他课程的虚拟实验系统的开发提供可重用的技术和代码。

通过使用本系统，使学生完成本科教学大纲要求的各种重要的电路实验。系统具有提示及帮助功能，以便指导学生完成实验。系统对实验结果进行记录，并可以重现实验结果。系统具有良好的可重用性与自由性，系统对实验的过程没有限定，可以随意添加或者减少使用的电路设备；在自主实验中，用户运用系统提供的设备和功能，可以建立一个属于自己的电路。如果用户为教师，则还可以在定制实验之后，对指定的学生发布实验。学生完成实验提交后，教师可对其批改评分。系统具有良好的界面效果。所模拟的电路器材操作界面与实际器材一致，其他设备的操作界面与实际使用的器材接近，这将极大的方便用户的学习与使用；同时系统的实验场景尽量仿照真实的实验环境，增强用户使用本系统的兴趣与沉浸感。

学习了电路的相关知识后，用户将可以通过本仿真系统来进行电路实验。本系统提供的仿真界面与真实的器材操作界面基本相同，用户即使没有操作电路的经验也可以在本系统上做实验，并有助于适应真实的器材操作；同时有操作真实器材经验的用户可以很快熟悉本仿真系统。

电路实验仿真系统提供一个虚拟实验平台，其核心是对电路作数值计算的程序，也就是"电路仿真程序"。利用平台，教学的模式实现了"实验"和理论紧密配合，做到先在平台上"搭"电路，看见电路"能干什么"，"干的怎么样"，再从理论上理解"为什么会是这样"，推论出有一般意义的结论。学生可以随时"设计"电路，到平台上仿真，检验设计的正确性，考查自己对知识理解和积累的程度。该系统可以用在电路技术基础课程的辅助教学上，诸如：电路理论，电路分析等。在广度和深度上可以满足高校本科、高职乃至中等职业学校电类或非电类不同层次的相类似课程的教学需要。操作界面的总体布局如下：

界面左右有器材栏，属性栏，中间为实验区。下方有信息提示区。属性栏、器材栏为竖直窗体，位置可自由移动（单击边框，鼠标拖动）。

属性栏：提供用户在实验区中所选择的器材的属性和和对复杂器材的操作。属性栏将随着器材的大小而改变大小。

器材栏：提供当前实验所要使用的器材。使用器材的图标和相应描述文字进行显示和说明。

实验区：在此区域中，搭建实验电路，进行实验操作，仪表读数等。

提示信息出现在实验区的左上方，提示信息有不同的种类，用户可以选择显示何种信息。具体实现操作界面如图 1-1 所示。

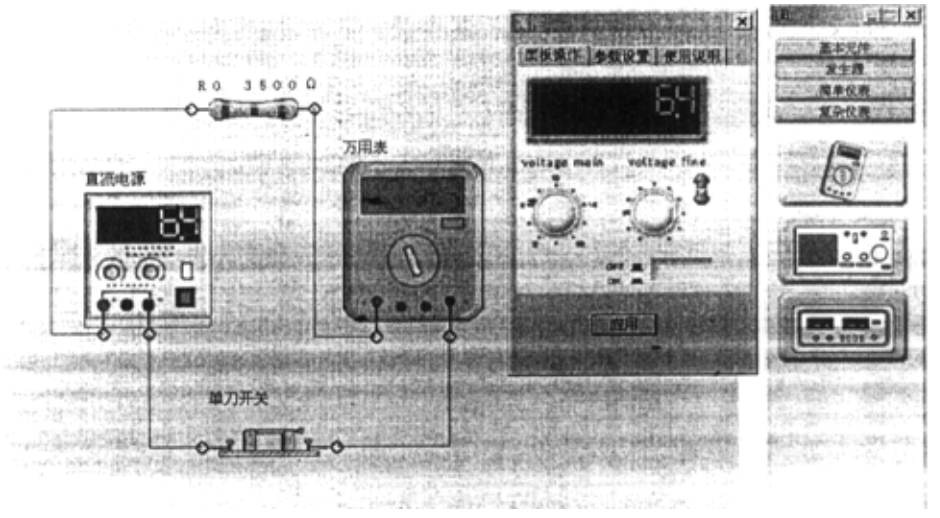

图 1-1 实验系统操作界面

1.1.4 课题的技术理论背景

系统总体设计主要采用 MVC 机制，即模型、视图、控制器。三者之间相互联系并保证足够松的耦合。其中模型主要用来记录数据和实现一些与操作没有过多关系的功能函数，如负责提供具体的数据操作接口、算法的功能实现以及处理控制器不能处理的信息；视图负责绘制，提供给用户绘制的接口，并用来表示器材以及各种操作在外观产生的影响；控制器主要负责响应各种鼠标键盘事件，同时调用模型和视图提供的接口函数来修改具体的数据和绘制信息。

系统基本上由三层组成：引擎层、仿真框架层、具体仿真实现层。引擎层负责底层的绘制，通用界面的绘制，底层信息的采集，信息处理泵的运转，信息处理函数的调用，以及整体的初始化与销毁。仿真框架层提供了一个针对各种仿真系统的较通用的 Framework，它是在引擎层的基础上实现的，负责仿真流程的运转，为上层的具体实现层提供针对引擎层的功能调用接口。具体仿真实现层实现

了电路仿真这一具体仿真系统的各种功能，包括电路的模拟仿真算法，以及各种器材的具体布局的模拟。

课题充分利用了已有的框架、函数库和各种设计模式，从而提高构建系统的效率。本课题主要使用了 MFC 类库和 STD 标准函数库。

在整个开发中，我们在 Windows 环境下，使用 Direct3D 作为绘制的底层接口，充分利用硬件性能，达到了在消耗很少资源的情况下快速绘制，从而很好的模拟了示波器等电路设备。本软件作为虚拟实验平台，通过使用组件技术将用户信息管理、实验管理、成绩管理等与仿真模拟比较无关的部分放在了组件之外，使用网页语言来连接数据库并进行仿真无关的逻辑运算，而将仿真组件的重点放在了仿真平台的构建上。

1.2 论文的主要内容和本人承担的工作

在该虚拟实验仿真系统的整个开发过程中，我们最大限度的运用统一建模语言 UML 技术，使用 UML 对需求进行建模，并结合用例驱动的开发过程，从需求分析到数据库设计，从详细设计到系统实现及测试，都将以用例为指导。

论文重点描述了系统开发的分析和设计阶段对 UML 技术的使用。在需求分析阶段，采用 UML 技术中的用例图，对系统的需求建模，清晰的描述了系统的需求；在系统设计阶段，采用 UML 技术中的配置图，对系统的 B/S 架构下的分布式计算环境建模，展示了系统运行时各个逻辑组成部分的分布；在对象设计阶段，采用 UML 技术中的类图，对系统中的对象及对象之间的关系进行建模，使系统的静态结构一目了然。

在面向对象的系统开发中，UML 技术是一个强有力的工具，可以使系统开更加高效，更加符合面向对象的原则。虚拟实验仿真系统的开发，大量的采用了 UML 技术，取得了良好的效果。

作者在该项目中完成的主要工作如下：

1. 管理平台的需求分析、系统设计、对象设计以及数据库设计，“学生实验”、“实验管理”等模块的编码和测试工作。
2. 仿真平台的控制器的设计、实现及测试。
3. 为整个系统（包括管理平台和仿真平台）制作安装程序。
4. 使用 ROSE 为系统建模，绘制了系统分析和设计需要的用例图和类图。

第 2 章 UML 技术概述

2.1 UML 技术的发展

UML 是在常用的一些面向对象建模技术 OMT(Object-oriented Modeling Technology)的基础上，由三位面向对象大师——Grady Booch、Ivar Jacobson、Jim Rumbauth 在 Rational 公司的组织下，统一了各种方法和不同的标准，于 1997 年 11 月 17 日正式由对象管理组织（OMG）批准 UML1.1 成为世界标准建模语言。我们称之为统一建模语言——Unified Modeling Language。UML 是一种编制系统蓝图的标准化语言，可以对复杂的系统建立可视化的系统模型。

UML 是一种定义良好，易于表达，功能强大且普遍实用的建模语言。它溶入了软件工程领域的新思想、新方法和新技术，不仅可以支持面向对象的分析与设计，更重要的是能够有力地支持从需求分析开始的软件开发的全过程。UML 是一种图形语言，在软件开发的整个过程中，从需求分析到结构设计、数据库建模，UML 都提供了模型化和可视化的支持。

UML 是在多种面向对象建模方法的基础上发展起来的建模语言，它支持面向对象的技术和方法，能够准确方便的表达面向对象的概念，体现面向对象的风格。在多种面向对象建模方法流派并存和相互竞争的局面中，UML 树起了统一的旗帜，使不同厂商开发的系统模型能够基于共同的概念，使用相同的表示法，建立彼此风格一致的模型。UML 的定义有两个主要组成部分：语义和表示法。UML 的语义用自然语言描述，表示法定义了 UML 的可视化标准表示符号，这决定了 UML 是一种可视化的建模语言。这些图形符号和文字用于建立应用级的模型，在语义上，模型是元模型的实例。此外 UML 的定义还给出了语法结构的精确规约。

2.2 UML 技术的体系结构

UML 是由图和元模型组成的。图是 UML 的语法，而元模型则给出图的意思，是 UML 的语义。也就是说 UML 是用元模型来描述的，元模型是四层元模型体系结构模式中的一层。此模式的其他层次分别是：元-元模型层、模型层和用户对象层。

元模型的体系结构模式已被证明可以用来定义复杂模型所要求的精确语义，这种复杂模型通常需要被可靠地保存、共享、操作以及在工具之间进行交换。它的特点如下：

1). 它在每一层都递归地定义语义结构，从而使语义更精确、更正规。

2). 它可用来定义重量级和轻量级扩展机制，如定义新的元类和构造型。

3). 它在体系结构上将 UML 元模型与其他基于 4 层元模型体系结构的标准(比如 MOF 和用于模型交换的 XMI Facility）统一起来。

在元模型层，UML 元模型又被分解为三个逻辑子包：基础包、行为元素包和模型管理包。其中基础包由核心、扩展机制和数据类型三个子包构成，它是描述模型静态结构的语言底层结构，支持类图、对象图、构件图、部署图等结构图。行为元素包是描述模型动态行为的语言上层结构，支持不同的行为图，包括 Use Case（用例）图、时序图、协作图、状态图和活动图。模型管理包则定义了对模型元素进行分组和管理的语义，它描述了几种分组结构，包括包、模型和子系统。

2.3 UML 技术的主要内容

作为一种建模语言，UML 的定义包括 UML 语义和 UML 表示法两个部分。

(1) UML 语义　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，消除了因人而异的最佳表达方法所造成的影响。此外 UML 还支持对元模型的扩展定义。

(2) UML 表示法　定义 UML 符号的表示法，为开发者或开发工具使用这些图形符号和文本语法并对系统建模提供了标准。这些图形符号和文字所表达的是应用级的模型，在语义上它是 UML 元模型的实例。

统一建模语言 UML 的重要内容可以由下列五类图(共 9 种图形)来定义：

第一类是用例图，从用户角度描述系统功能，并指出各功能的操作者。

第二类是静态图(Static diagram)，包括类图、对象图和包图。其中类图描述系统中类的静态结构，不仅定义系统中的类，表示类之间的联系如关联、依赖、聚合等，也包括类的内部结构(类的属性和操作)。类图描述的是一种静态关系，在系统的整个生命周期都是有效的。对象图是类图的实例，几乎使用与类图完全相同的标识。他们的不同点在于对象图显示类的多个对象实例，而不是实际的类。

一个对象图是类图的一个实例。由于对象存在生命周期，因此对象图只能在系统某一时间段存在。包由包或类组成，表示包与包之间的关系。包图用于描述系统的分层结构。

第三类是行为图(Behavior diagram)，描述系统的动态模型和组成对象间的交互关系，包括状态图和活动图。其中状态图描述类的对象所有可能的状态以及事件发生时状态的转移条件。通常，状态图是对类图的补充。在实用上并不需要为所有的类画状态图，仅为那些有多个状态其行为受外界环境的影响并且发生改变的类画状态图。而活动图描述满足用例要求所要进行的活动以及活动间的约束关系，有利于识别并行活动。

第四类是交互图(Interactive diagram)，描述对象间的交互关系，包括时序图和协作图。其中时序图显示对象之间的动态合作关系，它强调对象之间消息发送的顺序，同时显示对象之间的交互；协作图描述对象间的协作关系，协作图跟时序图相似，显示对象间的动态合作关系。除显示信息交换外，协作图还显示对象以及它们之间的关系。如果强调时间和顺序，则使用时序图；如果强调上下级关系，则选择协作图。这两种图合称为交互图。

第五类是实现图(Implementation diagram)，包括部件图和配置图。其中部件图描述代码部件的物理结构及各部件之间的依赖关系。一个部件可能是一个资源代码部件、一个二进制部件或一个可执行部件。它包含逻辑类或实现类的有关信息。部件图有助于分析和理解部件之间的相互影响程度。配置图定义系统中软硬件的物理体系结构。它可以显示实际的计算机和设备(用节点表示)以及它们之间的连接关系，也可显示连接的类型及部件之间的依赖性。在节点内部，放置可执行部件和对象以显示节点跟可执行软件单元的对应关系。

2.4 UML 技术与面向对象的软件工程

2.4.1 面向对象概念

面向对象技术是一种围绕真实世界中的事物来组织软件系统的技术。在这种技术中，系统的基本构成要素是对象。从软件开发人员的角度来看，对象是一种将数据和处理该数据的操作合并在一起的程序单元；从用户的角度来看，对象是一种具有属性和行为的事物。对象可以是具体的也可以是概念性的。对象具有标

识唯一性、分类性、多态性、继承性和封装性等基本特性。面向对象技术是通过将存在于问题空间范围内的事物抽象成对象、建立对象和对象之间的通讯联系来进行软件开发的。

在面向对象技术中，除了引入对象这一最基本的概念之外，还引入了对象类和类继承性两个基本概念。对象、对象类和类继承性也就是数据抽象、抽象数据类型和类型继承，是面向对象的三大要素。用公式表示就是：

面向对象 = 对象 + 对象类 + 类继承性

在面向对象的分析中，常把具有相同或相似特性的一组对象抽象成类，对象就是类的实例。系统中的类很少单独存在，相反，大多数类以几种方式相互协作。因此，当对系统建模时，不仅要识别形成系统词汇的事物，而且必须对这些事物如何相互联系建模。

在面向对象的建模中，有 3 种特别重要的关系：依赖（dependency），它表示类之间的使用关系；泛化（generalization），它把一般类连接到它的特殊类，也称为超类/子类关系或父类/子类关系，就是通常所说的继承关系；关联（association），它表示对象之间的结构关系。其中的每一种关系都为组织对象提供了不同的方法。下面深入讨论这些关系的内涵：

1. 依赖

依赖(dependency)是一种使用关系，它说明一个事物（如类 Event）规格说明的变化可能影响到使用它的另一个事物(如类 Window)，但反之未必。在图形上，把依赖画成一条有向虚线，指向被依赖的事物，如图 2-1。当要指明一个事物使用另一个事物是，就使用依赖。

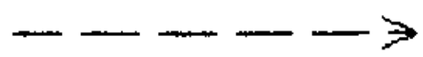

图 2-1

在大多数情况下，在类的语境中用依赖指明一个类把另一个类作为它的操作的特征标记中的参数。这的确是一种使用关系，如果被使用的类发生变化，那么另一个类的操作也会受到影响，因为这个被使用的类此时可能表现出不同的接口或行为。

2. 泛化

泛化（generalization）是一般事物（称为超类或父类）和该事物的较为特殊的种类（称为派生类或子类）之间的关系。有时也称泛化为“is-a-kind-of”关系：一个事物是更一般的事物的一个种类。泛化意味着子类的对象可以被用在父类的对象可能出现的任何地方，但反过来不这样。换句话说，泛化意味着子类可以替换父类。子类继承父类的特性，特别是父类的属性和操作。通常，子类除了具有父类的属性和操作外，还具有别的属性和操作。若子类的操作与父类的操作的特征标记（方法名、参数列表以及返回值等等）相同，则它覆盖（override）父类的操作，这称为多态性。在图形上，把泛化画为一条带有空心大箭头的有向实线，指向父类，如图 2-2。当要表示继承关系时，就使用泛化。在大多数情况下，用类和接口间的泛化指明继承关系。

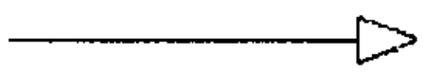

图 2-2

3. 关联

关联（association）是一种结构关系，它指明一个事物的对象与另一个事物的对象间的联系。给定一个连接两个类的关联，可以从一个类的对象导航到另一个类的对象，反之依然。关联的两端都连接到同一个类是合法的。这意味着，从一个类的给定对象能连接到该类的其他对象。恰好连接两个类的关联叫做二元关联。尽管不普遍，但可以有连接多于两个类的关联，这种关联叫做 n 元关联。在图形上，把关联画成一条连接相同类或不同类的实线，如图 2-3。当要表示结构关系时，就使用关联。

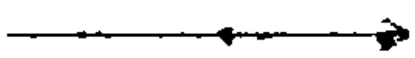

图 2-3

关联表示了对象间的结构关系。在很多建模问题中，说明一个关联的实例中有多少个相互连接的对象是很重要的。这个“多少”被称为关联角色的多重性，就是说明：在关联另一端的类的每个对象要求在本端必须有多少个对象。可以精确的表示多重性为 1、0 或 1（0..1）、很多（0..*）、一个或很多（1..*）。甚至可以精确的指定多重性为一个数值（如 3）。

两个类之间的简单关联表示了两个同等地位类之间的结构关系，这意味着这

两个类在概念上是同级别的，一个类并不比另一个类更重要。有时要对一个“整体/部分”关系建模，其中一个类描述了一个较大的事物（“整体”），它有较小的事物（“部分”）组成。把这种关系称为聚合，它描述了“has-a”关系，意思是整体对象拥有部分对象。其实聚合是一种特殊的关联，它被表示为在整体的一端用一个空心菱形修饰的简单关联，如图 2-4。

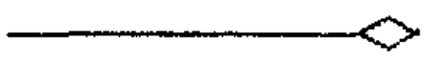

图 2-4

依赖、泛化和关联都是定义在类一级的静态事物。在 UML 中，通常在类图中对这些关系进行可视化。在解决对象的动态协作时，还会遇到其他两种关系：链（它是关联的实例，描述可能发送消息的对象间的连接）和转换（它是状态机中不同状态间的连接）。

4. 实现

实现（realization）是类元之间的语义关系，在该关系中一个类元描述了另一个类元保证实现的契约。在图形上把实现画成一条带有空心三角箭头并指向描述契约的那个类元的虚线，如图 2-5。

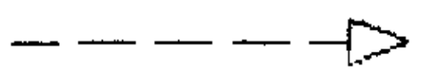

图 2-5

实现与依赖、泛化和关联很不相同，所以被看作一种独立的关系。实现是依赖和泛化在语义上的一些交叉，它的表示法是依赖和泛化的表示法的组合。在接口的语境中和在协作的语义中都要用到实现关系。

在大多数情况下，要用实现描述接口和（为其提供了操作或服务的）类或构件之间的关系。接口是一组操作的集合，而操作用于描述类或构件的一个服务。因此接口描述了类或构件必须实现的契约。一个接口可以由多个这样的类或构件实现，一个类或构件也可以实现多个接口。或许关于接口的最有趣的事情是它允许把契约（接口本身）的描述与实现（由类或构件完成）分离开来。此外，接口跨越了系统的体系结构的逻辑部分和物理部分。

2.4.2 面向对象的软件工程

软件开发的面向对象方法首先提出于 20 世纪 60 年代后期，然而，对象技术

花了几乎 20 年的时间才开始变得广为使用。在 20 世纪 90 年代，面向对象软件工程变成了很多软件产品建造者以及数量不断增长的信息系统和工程专业人员的首选模型。随着时间的流逝，对象技术正在替代传统的软件开发方法。

对象技术导致复用，而程序构件的复用导致更快的软件开发周期和高质量的程序。面向对象软件易于维护，因为它的结构是内在松耦合的，这样，当进行修改时，不会产生太多的副作用，对软件工程师和客户来说也不会轻易导致项目的失败。此外，面向对象系统易于进行适应性修改，并易于伸缩（即通过组装可复用子系统而可以创建大的系统）。

2.4.2.1 面向对象分析

在开始建造面向对象系统之前，我们必须定义表示待解决问题的类（对象）、类之间相互关联和交互的方式、对象的内部结构（属性和操作）以及允许对象在一起工作的通信机制（消息）。所有这些事情均是在面向对象分析（OOA）中完成的。

在对系统或产品有一个合理的理解前，是不可能建造软件的。OOA 为我们提供了具体的方式来表示我们对需求的理解，然后根据客户对待建造系统的感觉来测试该理解。

OOA 从用例的描述开始。类的静态和动态特征使用 UML 或其他方法建模。OOA 的成果是一个面向对象的分析模型。OO 分析模型包括图形的或文字的表示，它们定义了类的属性、关系和行为，并描述类之间的通信和类的行为随时间的变化。

OOA 以一组基本原则为基础。为了建立一个分析模型，OOA 使用了 5 个基本的原则：（1）信息域建模；（2）功能描述；（3）行为表示；（4）划分数据、功能和行为模型以表示更多细节；（5）早期模型表示问题的本质，后期模型提供实现细节。这些原则形成了讨论 OOA 方法的基础。

OOA 发生在很多不同的抽象层次，在业务或企业层，和 OOA 关联的技术可以结合业务过程工程方法，该技术通常称为领域分析；在应用层次，对象模型着重于特定的客户需求，因为这些需求将影响正在建造的应用。

OOA 过程从用例的定义开始，然后应用 UML 建模技术为类和类的属性与操作建立文档，它也提供了发生在对象间的协作的初始视图。OOA 过程的下一步是对

象的分类和类层次的创建，子系统可用于封装相关的对象，对象一关系模型提供了对象间如何相互关联的指示，而对象一行为模型指明了个体对象的行为和 00 系统的整体行为。

2.4.2.2 面向对象设计

面向对象软件的设计需要多层软件体系结构的定义、完成所需功能并提供基础设施支持的子系统规约、形成系统构造积木块的对象的描述以及允许数据在层间、子系统间和对象间流动的通信机制的描述。面向对象设计（OOD）完成上述任务。

面向对象系统依赖于从分析模型中导出的类定义来建立。有一些类定义必须重新建造，但如果采用适当的设计模式，很多类可以被复用。面向对象设计（OOD）建立一个设计蓝图，使软件工程师以复用最大化的方式定义 00 体系结构，以此来提高开发速度和产品质量。

OOD 被分为两个主要的活动：系统设计和对象设计。系统设计创建产品体系结构，定义一系列完成特定系统功能的“层次”，并标识由驻留在各层的子系统封装的类。此外，系统设计考虑三个构件的规约：用户界面、数据管理功能和任务管理设施。对象设计关注于个体类的内部细节，定义属性、操作和消息细节。

00 设计模型包含软件体系结构、用户界面描述、数据管理构件、任务管理实施以及系统中的每个类的详细描述。

OOD 包含如下四个层次：

（1）子系统层。包含每个子系统的表示，这些子系统使软件能够满足客户定义的需求，并实现支持客户需求的技术基础设施。

（2）类和对象层。包含类层次，它们使系统将能够用一般化及不断逼近目标的特例化机制来创建，这一层还包含了每个对象的表示。

（3）消息层。包含使每个对象能够和其协作者通信的设计细节，这一层建立了系统的外部和内部接口。

（4）责任层。包含每个对象的所有属性和操作的数据结构和算法的设计。

在分析建模中，用户模型和结构模型视图被表示，这些为系统的使用场景（用于指导行为建模）提供了洞察并通过标识和描述系统的静态结构元素建立了实现

和环境模型视图的基础。

UML 被组织进两个主要的设计活动：系统设计和对象设计。UML 系统设计的主要目标是表示软件体系结构。

对面向对象开发而言，概念体系结构涉及静态类模型的结构和该模型的构件间的连接。模块体系结构描述系统被划分成子系统或模块的方式以及它们如何通过移入和移出数据而通信。代码体系结构定义了程序代码如何被组织为文件和目录以及分组为库。执行体系结构关注于系统的动态方面及任务和操作执行时构件间的通信。

UML 对象设计着重于对象及其相互交互的描述。在对象设计期间，创建属性数据结构和所有操作的过程设计的详细规约；定义所有类属性的可见性，精化对象间的接口以定义完整的消息模型的细节。

UML 中的系统设计和对象设计被扩展以考虑正在建造系统的用户界面、数据管理的设计以及已经规约的子系统的任务管理的设计。用户模型视图驱动用户界面设计过程，提供一个设计场景，并迭代地为一组界面类。数据管理设计建立了一组类和操作，它们允许系统去管理永久数据。任务管理设计建立将子系统组织为任务的基础设施，并管理并发任务。

软件工程的观点强调 OOA 和 OOD，而考虑 OOP(面向对象编码)是重要的但非主要的活动，它是分析和设计的副产品。这一点的理由是简单的，当系统复杂性增加时，对终端产品的成功而言，设计体系结构比使用的程序设计语言对其成功有更强的影响。

2.4.2.3 面向对象测试

面向对象软件的体系结构导致了一系列分层的子系统，它们封装了协作的类。这些系统元素（子系统和类）每个都完成帮助达成系统需求的功能。有必要在各个不同的层次去测试某 OO 系统以努力发现可能发生在类相互协作时子系统跨体系结构层次通信时的错误。

OO 测试在策略上和传统系统的测试类似，但是，技巧上存在不同。因为 OO 分析和设计模型在结构上类似于最终 OO 程序的内容，“测试”从这些模型的评审开始。一旦代码生成，OO 测试开始“小规模”类测试。一系列测试被设计以检

查类操作并检查当某类和其他类协作时是否有错误发生。类被集成以形成子系统，基于线程的、基于使用的和集群测试结合基于故障的方法，被用于完全地测试协作类。最后，use-case（作为 OO 分析模型的一部分而开发）被用于发现在软件确认级的错误。

面向对象的测试将产生一组测试类，类的协作和行为的测试案例被设计和文档化，预期的结果被定义，实际的结果被记录。在测试时，应努力去“破坏”软件！以严格的方式设计测试案例并评审测试案例的完全性。

简单地说，测试的目标是在现实的时间跨度内应用可管理的工作量去发现最大可能数量的错误。虽然对面向对象软件而言，这个基本目标保持不变，但是 OO 程序的性质改变了测试策略和测试战术。

OO 系统的测试带给了软件工程师新的挑战。测试的定义必须被扩展以包括用于 OOA 和 OOD 模型的错误发现技术。OO 表示的完全性和一致性必须在其被建造时被评估。单元测试失去了本身的多数意义，集成测试策略也有了较大改变。总之，测试策略和战术必须适应 OO 软件的独特性质。

面向对象软件的构造从分析和设计模型的创建开始。因为 OO 软件工程的模型的演化性质，模型从对系统需求相对非正式的表示开始，逐步演化为详细的类模型、类连接和关系、系统设计和分配、以及对象设计(通过消息序列的对象连接模型)。在每个阶段，测试模型，以试图在错误传播到下一次递进前发现错误。

分析和设计模型不能进行传统意义上的测试，因为它们不能被执行。然而，正式的技术复审可被用于检查分析和设计模型的正确性和一致性。

用于表示分析和设计模型的符号体系和语法将是和为项目选定的特定分析和设计方法联系的，因此，语法正确性基于符号是否合适使用，而且对每个模型复审以保证保持合适的建模约定。

在分析和设计阶段，语义正确性必须基于模型对现实世界问题域的符合度来判断，如果模型精确地反应了现实世界(到这样一个细节程度：它对模型复审时所处的开发阶段是合适的)，则它是语义正确的。为了确定是否模型确实在事实上反映了现实世界，它应该被送给问题域的专家，专家将检查类定义和类层次以发现遗漏和含混。评估类关系(实例连接)以确定它们是否精确地反应了现实世界的对象连接。

一旦已经创建了设计模型，也应该进行对系统设计和对象设计的复审。系统设计描述了构成产品的子系统、子系统被分配到处理器的方式、以及类在子系统中的分配。对象模型表示了每个类的细节和实现类之间的协作所必需的消息序列活动。

通过检查在 OOA 阶段开发的对象—行为模型，和映射需要的系统行为到被设计用于完成该行为的子系统来进行系统设计的复审。也在系统行为的语境内复审并发性和任务分配，评估系统的行为状态以确定哪些行为并发地存在。

对象模型应该针对对象 — 关系网络来测试，以保证所有设计对象包含为实现为每张 CRC 索引卡片定义的协作所必须的属性和操作。此外，使用传统的检查技术来检查复杂操作细节的详细规约。

第 3 章 UML 技术在系统开发中的应用

3.1 概述

本章将把上一章中讨论的内容应用于虚拟实验仿真系统开发的各个阶段，本系统的开发采用面向对象的技术，大致分为如下几个阶段：需求分析、系统设计、对象设计、系统实现以及测试。下面将对系统开发的各个阶段进行详细讨论。

3.2 需求分析阶段

3.2.1 软件需求

3.2.1.1 软件需求的定义

IEEE 软件工程标准词汇表(1997 年)中定义需求为：

(1)用户解决问题或达到目标所需的条件或权能(Capability)。

(2)系统或系统部件要满足合同、标准、规范或其它正式规定文档所需具有的条件或权能。

(3)一种反映上面(1)或(2)所描述的条件或权能的文档说明。

3.2.1.2 需求的层次

软件需求包括三个不同的层次——业务需求、用户需求和功能需求（也包括非功能需求)。业务需求(business requirement)反映了组织机构或客户对系统、产品高层次的目标要求，在项目视图与范围文档中予以说明。用户需求(user requirement) 文档描述了用户使用产品必须要完成的任务，这在用例(use case)文档或方案脚本(scenario)说明中予以说明。功能需求(functional requirement)定义了开发人员必须实现的软件功能，使得用户能完成他们的任务，从而满足了业务需求。

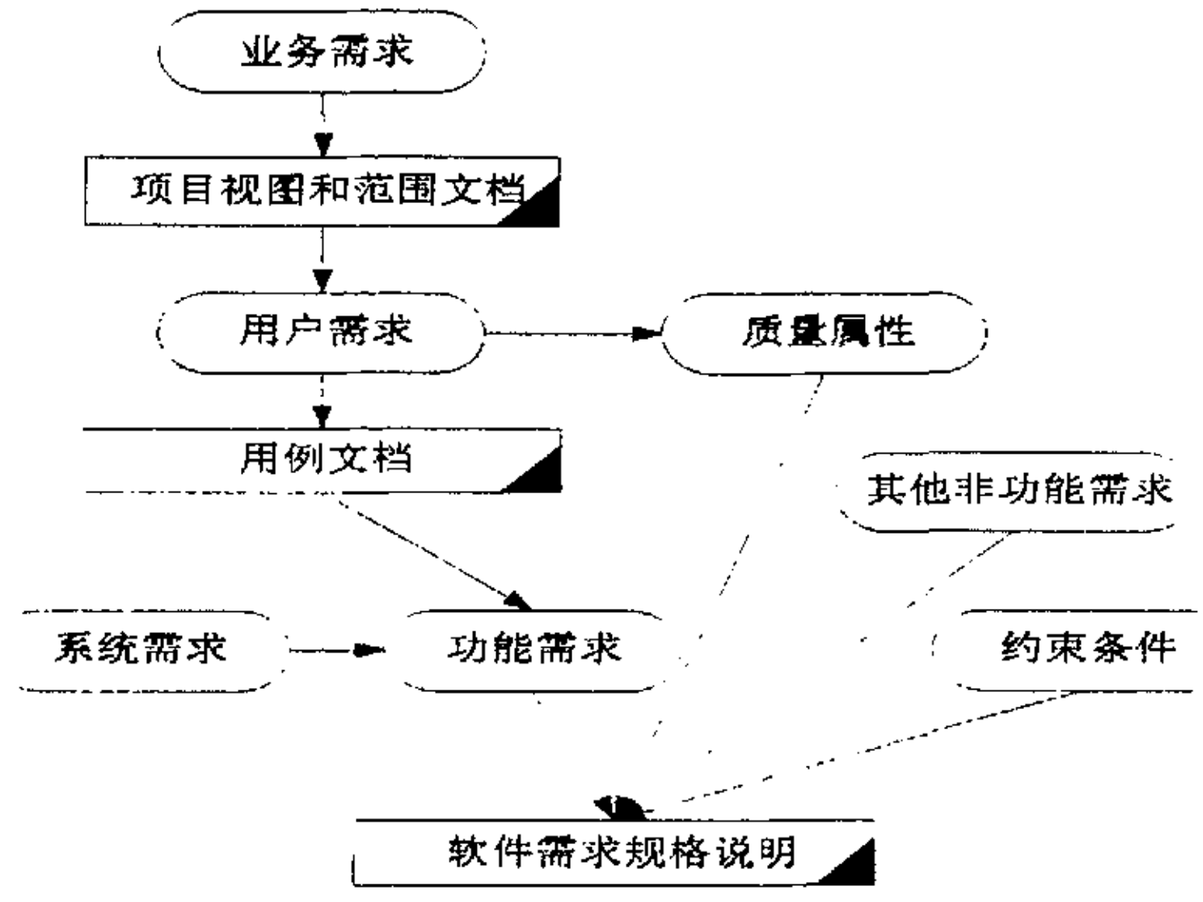

图 3-1

作为补充，软件需求规格说明还应包括非功能需求，用来描述系统展现给用户的行为和执行的操作等。它包括产品必须遵从的标准、规范和约定，外部界面的具体细节，性能要求，设计或实现的约束条件及质量属性。所谓约束是指对开发人员在软件产品设计和构造上的限制。质量属性是通过多种角度对产品的特点进行描述，从而反映产品功能。多角度描述产品对用户和开发人员都非常重要。值得注意的是，需求并未包括设计细节、实现细节、项目计划信息和测试信息。需求与这些没有关系，它关注的是充分说明你究竟想开发什么。

3.2.1.3 需求分析的重要性及任务

Frederick Brooks 在他的一篇经典的文章中说：开发软件系统最为困难的部分就是准确说明开发什么。最为困难的概念性工作便是编写出详细技术需求，这包括所有面向用户、面向机器和其它软件系统的接口。同时这也是一旦做错，将最终会给系统带来极大损害的部分，并且以后再对它进行修改也极为困难。

软件工程中包含需求、设计、编码和测试四个阶段，其中需求工程是软件工程第一个也是很重要的一个阶段， 首先我们必须了解需求工程和其他项目过程的关系：

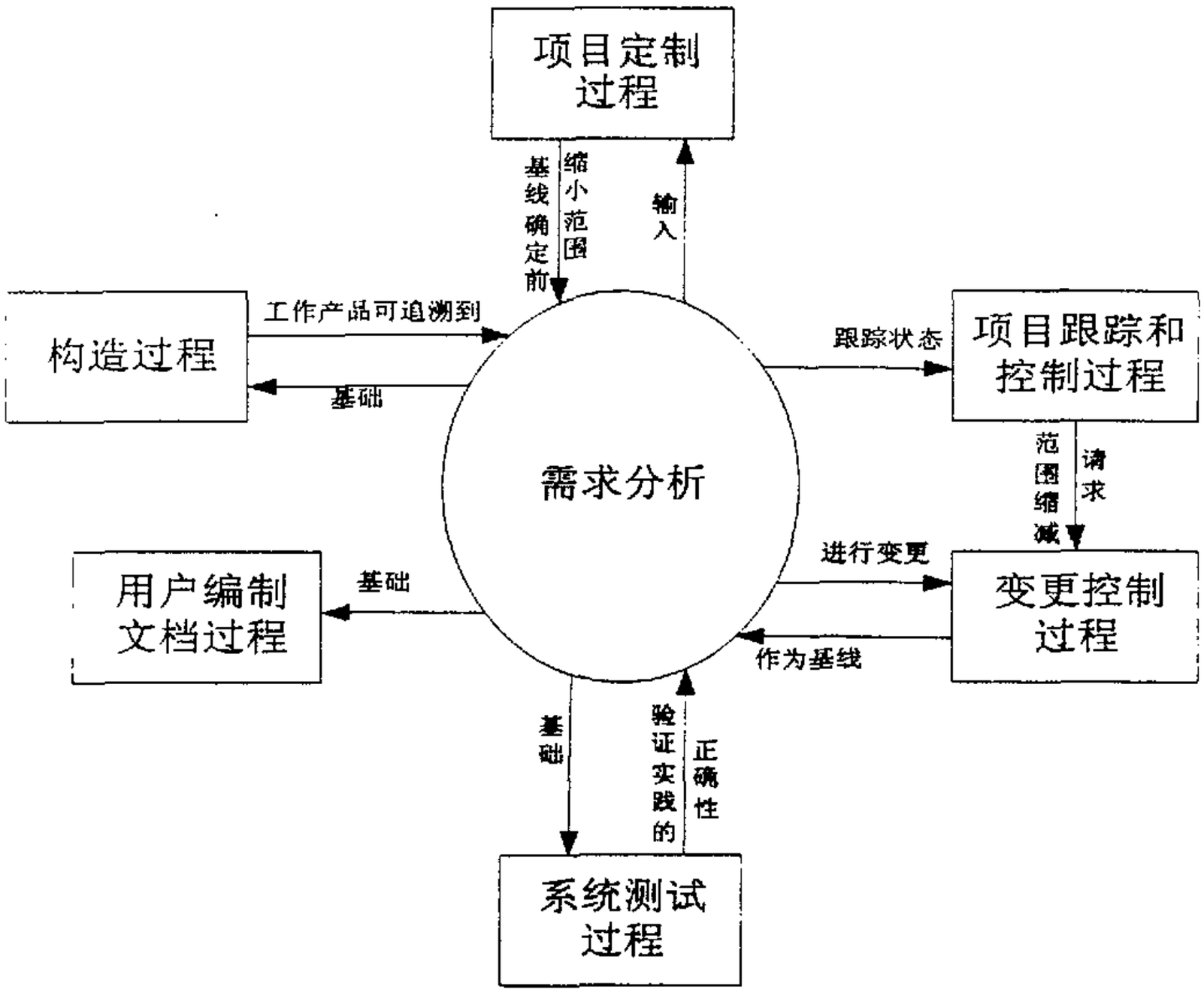

图 3-2

需求分析作为软件生命周期的第一个阶段，并贯穿于整个软件生命周期，其重要性越来越突出，到 80 年代中期，逐步形成了软件工程的子领域——需求工程。进入 90 年代后，需求工程成为软件界研究的重点之一。

需求分析是工程项目成功的关键，需求分析的目的就是要准确理解、明确定义用户的需要和系统功能。需求分析的任务是准确地描述系统应该做什么或不应该做什么的问题。需求分析所要做的工作是深入描述软件的功能和性能，确定软件设计的限制和软件同其它系统元素的接口细节，定义软件的其它有效性需求。需求分析的任务不是确定系统如何完成它的工作，而是确定系统必须完成哪些工作，也就是对目标系统提出完整、准确、清晰、具体的要求。需求分析的好坏直接关系到软件的成功与否，是软件生命周期中的关键一个环节。

3.2.1.4 使用用例捕获需求

软件开发通常使用典型场景(scenarios)来理解一个系统需要的是什么和系统是怎样工作的。但是这样做的同时，却很少用有效的形式将它归档。用例(Use Cases)就是将这些场景获取正式化、形式化的技术。基于用例来进行需求获取和

建模，克服了以往没有统一的格式、缺乏描述的形式化、随意性较大等缺点。

用例概念是 Jacobson 在面向对象的软件工程中提出的，但它实际上是独立于面向对象的。用例是指一个用户或其它系统与要设计的系统进行的一个交互，这个交互的目的是为了达到某个目标(goal)。随着 UML 的发展，用例本身也因其 UML 中的位置和地位再次得到强调。

用例用来获取需求，规划和控制项目。用例的获取是需求分析阶段的主要任务之一，而且是首先要做的工作。大部分用例在项目的需求分析阶段产生，并且随着工作的深入会发现更多的用例，这些都应及时增添到已有的用例集合中。用例集合中的每一个用例都是一个潜在的需求。

用例分析技术包括阅读和分析需求说明，此时需要与系统的潜在用户进行讨论。用例模型的主要构件是用例、角色和系统边界。用例用于描述每个功能需求，系统边界用于界定系统功能范围，而角色用于描述与系统功能有关的外部实体，它可以是用户，也可以是外部系统。

用例是获取业务过程和系统需求的一种有效方式。因为用例是从用户的角度看待系统，而不是基于程序员的角度，所以基于用例的方法给需求获取带来的好处来自于该方法是以任务为中心和以用户为中心的，同使用以功能为中心的方法相比，用例方法可以使用户更清楚地认识到系统要求做什么，并有助于分析者和开发者理解用户的业务和应用领域，用户的任何需求都能够在系统开发链中得到完整的体现。此外在技术上，基于用例的方法易于揭示于对象以及它们之间的责任，可以方便的利用面向对象的设计方法把用例转化为对象模型。同时用户和程序员间也能通过用例进行有效的沟通，从而更好地降低了项目失败的风险。

UML 对需求工程化的支持主要体现在它对用例的支持，用来理解并传递功能性的要求。使用 UML 对需求建模，并结合用例驱动的开发过程，便于跟踪从需求到设计这一过程。这时所说的跟踪能力，指的是确定设计中现有的元素，这些元素为制定需求的结果。在用例驱动的开发过程下，会创建特定的设计元素来满足用例。因此会隐式实现跟踪功能。所以需求分析的第一步是定义用例，以描述所开发系统的外部功能需求。

3.2.2 用例建模

在 20 世纪 90 年代早期，Ivar Jacobson 等人推广了运用用例理解功能性系统需求的做法。后来，用例符号表示法也就被纳入到统一建模语言（UML）中。从概念上看它很简单，但是却非常有用。特别当软件开发过程是以用例为驱动的过程时，它就变得尤为重要。用例不但被用于捕获需求，还用于提供从分析到测试的活动基础。

用例模型的建立是系统开发者和用户反复讨论的结果，表明开发者和用户对需求规格达成的共识。它将系统看作黑盒，从外部执行者的角度来理解系统并描述待开发系统的功能要求，同时驱动需求分析之后开发工作的各阶段和 UML 各个模型。

3.2.2.1 用例建模中的两个基本概念

执行者（Actor）和用例（use case）是用例建模中的两个基本概念。

执行者表示系统外的某事物（或某人），是同系统交互的所有事物，如，人、其他软件、设备、数据存储或者网络。执行者与系统打交道，引发系统执行某些动作。每一个执行者代表一个独一无二的角色。

用例封装了由代表执行者的系统所执行的一系列操作步骤，用例为执行者提供了某些值。

3.2.2.2 确定系统边界

需求主要是区分系统做什么不做什么。然而，这个边界非常难以确定，需求分析首先要做的是确定系统边界。也就是要找出系统中有什么，系统外有什么，划清内部和外部的界限。在使用 UML 进行系统分析的过程中，需要通过确定执行者和用例来确定系统边界。

1. 确定执行者

每一个执行者定义一种特定的角色。每一个系统之外的实体可以用一个或者多个执行者来代表。一个人可能会被多个执行者来代表，因为一个人在系统中可

能具有多种角色。另外，如果多个人对于系统来说具有相同的角色，那么这些人也可以用一个执行者来表示。

执行者必定是在系统之外的，而不是系统的一部分。

2. 确定用例

用例是系统的一种行为。从执行者的角度看，用例应该是一个完整的任务，而且一个用例行为通常是在一个相对较短的时间段内完成。在 UML 中，用例通常由执行者触发。

3. 描述执行者和用例

每一个执行者和用例都需要用一个名字和一两句简短的描述。

在确定和定义执行者和用例的过程中，同时也就定义了系统的边界，这些都需要记录下来，记录这些内容的就是用例图。在 UML 中，用例图是用来描述系统需求的一类重要图形。

用例图描述了系统提供的一个功能单元。用例图的主要目的是帮助开发团队以一种可视化的方式理解系统的功能需求，包括“执行者”（也就是将与系统交互的人或事物）与基本流程的关系，以及不同用例之间的关系。用例图一般给出了用例组或者是整个系统的全部用例，或者是一组分开的具有相关功能（例如，所有用户管理相关的用例）的特定用例组。

3.2.3 用例分析技术

3.2.3.1 归档用例

对项目完成了初步需求的获取后，抽象出来的用例图描述了系统的所有功能，但只有这些用例图还远远不够，紧接着应该是一个细化阶段。在这个阶段需要给项目的需求增加细节，对每个用例都应该有一个详细的描述。

用例描述可能包含这样一些信息：名称、标识符、参与者、状态、前置条件、后置条件、被扩展的用例、被包含的用例、基本操作流程、可选操作流程。这些信息有些是必须的，有些则是可选的。下面对这些信息分别做出说明：

1. 名称：表明用户的意图或用例的用途，如“考勤管理”。

2. 标识符（可选）：唯一标识符，如 “UC091”，在项目的其他用例或其他元素中可用它来引用这个用例。

3. 目标说明。概括说明用例功能的几句话。

4. 参与者（可选）：与此用例相关的参与者列表。尽管这则信息包含在用例本身中，但在没有用例图时，它有助于增加对该用例的理解。

5. 状态（可选）：指示用例的状态，通常为以下几种之一：进行中、等待审查、通过审查或未通过审查。

6. 频率：参与者访问此用例的频率。这是一个自由式问题，如用户每次登录访问一次或每月一次。

7. 前置条件：一个条件列表，则这些条件必须在访问用例之前得到满足。

8. 后置条件：一个条件列表，如果其中包含条件，则这些条件将在用例成功完成以后得到满足。

9. 被扩展的用例（可选）：此用例所扩展的用例（如果存在）。扩展关联是一种广义关系，其中扩展用例连接并继续基用例的行为。这是通过扩展用例向基用例的操作序列中插入附加的操作序列来实现的。这在用例图中是用<<extend>>来表示的。

10. 被包含的用例（可选）：此用例所包含的用例列表。包含关联是一种广义关系，它表明对处于另一个用例之中的用例所描述的行为的包含关系。这在用例图中是用 <<include>>来表示的。

11. 基本操作流程：参与者在用例中所遵循的主逻辑路径。通常也称其为适当路径或主路径。

12. 可选操作流程。用例中很少使用的逻辑路径，或那些在变更工作方式、出现异常或发生错误的情况下所遵循的路径。

用例描述没有固定的格式，只要能够清楚表达项目需求所需要的所有信息就可以了，我们对要开发的网上办公系统也制定了自己的用例模板：

<用例编号><用例名称>

用例描述人员：

其他参加人员：

审核人员：

完成时间：

版本号：

用例目标：用例目标描述

范围：当前考虑的是哪个系统

前件：用例执行前系统应有的状态

成功后件：用例成功执行后的状态

失效后件：用例没有完成目标的状态

触发：启动该用例的系统动作

角色：与该用例有关的首要角色

主事件流：事件正常情况下的主要路径

扩展：从主要路径中扩展出的事件路径

变异：在出现异常或发生错误的情况下的路径

相关用例：此用例所包含用例

3.2.3.2 用例描述技术

在用例描述过程中，主要遇到的问题就是如何清晰完整的定义功能需求。用例描述应该尽可能的清晰，因为这一点是系统开发工作的基础和依据，是整个项目组协调统一的保证；用例描述应该尽可能的完整，因为这是系统开发成功和高效的基础和保证。如何编制清晰完整的用例描述文档是一个值得深入研究的问题，在网上办公系统的这部分工作中，我也经过了对此问题的反复推敲的一个阶段，体会并总结出这样一些用例建模技巧：

用例的目的是描述用户如何对系统进行操作，所以，应站在用户的角度并采用主动的语态描述来编写用例。用例描述的是对用户来说有价值的一系列行动，是描述用户如何与系统交互的。它不需要描述用户界面的外观，或者系统是如何工作的。用例既应该描述参与者是如何与系统交互的，也应该描述系统如何响应这些交互而不涉及内部细节，应该尽可能详细和明确的体现需要实现的目标。描述中要包含正常情况下使用的基本行动过程，也要包含备选过程。引入备选过程是为了描述潜在的使用错误以及应用逻辑错误和异常。用例既不是类规范，也不是数据规范。这是由概念性模型捕捉的一种信息，在对象世界中，它是通过 UML 类模型建模的。

原则上所有的用例都必须有一个描述文档，对于这个描述文档没有固定的格

式，但所有的用例描述文档的所涉及的内容都应包括：用例名、角色、事件流、前置条件（描写该用例执行所必须满足的条件）、后置条件（描写该用例执行完之后的条件）、扩展点。用例本身是指一个用户或其它系统与要设计的系统进行的一系列交互，这些交互是为了达到某个目标。用例名描写用例的名称，角色是与要设计的系统进行交互的人或外部系统，它强调了任何人或系统拥有目标的事实。事件流分为基本事件流和其他事件流，基本事件流描述用户操作该用例的基本的流程，其他事件流描述除基本事件流外的其他的事件流，可以有多个其他的事件流。前置条件描写该用例执行所必须满足的条件，后置条件描写该用例执行完之后的条件。

上述的几点，描述文档的书写过程中常常有所纰漏和错误，常见的纰漏和错误有以下几点：

1. 基本事件流描述步骤不清楚或描述中不能体现出明确对象

2. 其他事件流有所遗漏。

对于基本事件流描述步骤不清楚的问题，通常是由于系统分析者对该用例的流程不清楚，也就是说对用户的需求分析不够；对于基本事件流描述中不能体现出明确对象的问题，通常是由于系统分析者对系统的分析还没有做到用面向对象的方法来思考；其他事件流有所遗漏的问题，通常是系统分析人员的设计没有严格的按照面向对象分析与设计 OOAD（Object Oriented analysis and design）的思想进行分析，这对于分析与设计同属一个人的情况下是没有问题的，而这会有悖于 OOAD 的初衷。所以要求对于用例描述文档的事件流的描述，既要确保简洁，又不能有所遗漏。总之，对于用例描述文档的编写，只有经过切实的实践和对 UML 的理解，才可以运用自如，同时，具备 OO 的思想也很重要。

用例中包含（include）和扩展（extend）关联，以及旧版本 UML 中使用的（uses）和扩展（extends）关联的正确使用始终没有得到很好的定义。用例建模中往往在这些关联的正确应用上产生分歧，这个问题我个人认为在项目开发组内部达成一致意见或说自行规定这两种关联的用法就可以了。我们认为，扩展（extend）描述如何将一个用例描述插入或者扩充到另一个用例描述，当然，插入的用例也必须有一个完整的功能。包含（include）与使用（use）是一对相反的关系，“使用”关系是一种泛化关系，是一个用例使用一个用例，“包含”则从

相反的角度表达这种关系，即一个用例被另一个用例包含（使用）。

、另外，需要强调的是用例描述中不应该涉及任何内部结构，因为描述内部结构会给设计工作带来额外的限制。

开发计划是我们进行需求分析的重要依据。我们的需求分析工作是基于 UML 用例技术，运用 Rational Rose 工具，对于全新开发的 5 个模块，均制定了详尽的用例描述文件，对于依据原型开发而改动较大的模块也有详尽的用例描述（共有 32 个用例），该文件将成为后续开发工作的指导性文件。

3.2.3.3 图形化用例

在完成对用例的文本描述后，应该将文字描述的信息用图形表达出来，这也是 UML 的特点之一——可视化建模。

软件开发是一项复杂的工作。有不同角色的具有不同技术背景的代表不同人群的参与者在同一个软件系统的构建过程中，发挥各自的作用，同时还要彼此沟通与协作。而不同角色的参与者对系统的关注角度又是不同的，这就需要对系统在不同的方面进行描述与表现。例如系统的用户所关心的是系统将有什么样的功能，这些功能能不能很好的为他们服务。而系统架构人员关心的是系统的体系结构是什么样的，应该使用什么样的技术平台和开发模型。为了更好的对系统的各个方面进行描述，对需求可视化是一个理想的选择。需求可视化的好处在于它能够通过简单且易于理解的图形来描述系统的各个方面。即使没有受过专门培训的人也可以非常容易的理解。

UML 是一种业界公认的可视化建模的标准语言。通过使用 UML 可以非常准确的描述系统，同时也可以很好的解决项目成员之间的沟通问题。使用 UML 对系统需求进行建模，通过用例（use case）来表达系统应该具备的功能。开发人员可以完全理解 UML 的表示方法，这样开发人员也就可以准确的理解需求人员的分析结果了。

时序图是 UML 的一种动态建模机制。UML 时序图一般用于确认和丰富一个用例的逻辑。用例就是系统潜在的使用方式的描述。一个用例的逻辑可能是一个用例的一部分，或是一条扩展路径；一个贯穿单个用例的完整流程，例如动作基本过程的逻辑描述，或是动作的基本过程的一部分再加上一个或多个的扩展的逻辑

描述。或是包含在几个用例中的流程。在设计研究中，因为时序图提供了一种方式，可以用来可视化的调用类定义的操作。

时序图是对用例添加了一定的灰盒细节后创建的，从我的体会来看，它是对系统的分析与设计的一种承接。在时序图中，通常有三种分析对象：

1. 边界对象

每一个用例－参与者关系就是一个潜在的边界对象。边界对象把系统的其余部分与外界环境隔离和屏蔽起来。

2. 控制对象

每个用例都有一个或多个控制对象。控制对象用于系统内的行为，它不必实现行为，但可以与其他对象协作来完成用例的行为。

3. 实体对象

实体对象表示系统的重要信息，实体对象通常跨越多个用例，也就是说，它不是某一个用例所独有的，甚至可能存在于它们自身所在系统之外。

例如在前面引述的用例，可以得到如下的图3－10和图3－11所示的时序图，图中带有折叠角的矩形表示在符合条件的情况下进入的另一个用例，上方矩形从左到右依次表示边界对象、控制对象和实体对象。

3.2.3.4 电路虚拟实验仿真系统的需求分析

电路虚拟实验仿真系统的需求分析使用UML技术和用例分析技术，通过图形结合文字的形式进行描述。下面列出了系统的主要用例的用例图。系统总体需求如图3-3，使用系统的用户分为三类，分别是管理员、教师及学生；系统主要由四个大的用例组成，包括学籍管理、实验管理、学生管理以及公共模块。系统中三类用户，通过这四个用例来访问系统提供的功能。系统的具体功能，由这四个用例包含的用例来描述。如图3-4，学籍管理包括用户管理和班级管理，用户管理包括添加用户，修改用户和删除用户；班级管理包括添加班级、修改班级和删除班级。如图3-5，实验管理包含新建实验、发布实验、布置实验以及批改实验四个用例，其中新建实验包含仿真平台用例。如图3-6，学生实验包含必做实验和自主实验两类，必做实验包含试做已布置实验、进行已布置实验、提交实验报告以及查看批语及成绩四个用例；自主实验包含进行自主实验、添加自主实验和

删除自主实验三个用例；其中新建实验、试做已布置实验、进行已布置实验、进行自主实验和添加自主实验这五个用例还包含了仿真平台用例。仿真平台用例可以被系统中的其它用例所共享。如图 3-7，公共模块包含个人消息、在线论坛和实验帮助三个用例。

使用 UML 技术提供的用例图，我们清晰的描述了系统的功能需求，为下一步的开发打下了良好的基础。为了更加细致准确的描述系统的需求，在图形描述基础上引入了文字描述。在图形用例后面，附上了新建自主实验用例的文字描述。采用图形与文字相结合的方式，使需求既直观又准确，可以很好的指导后续的开发工作。

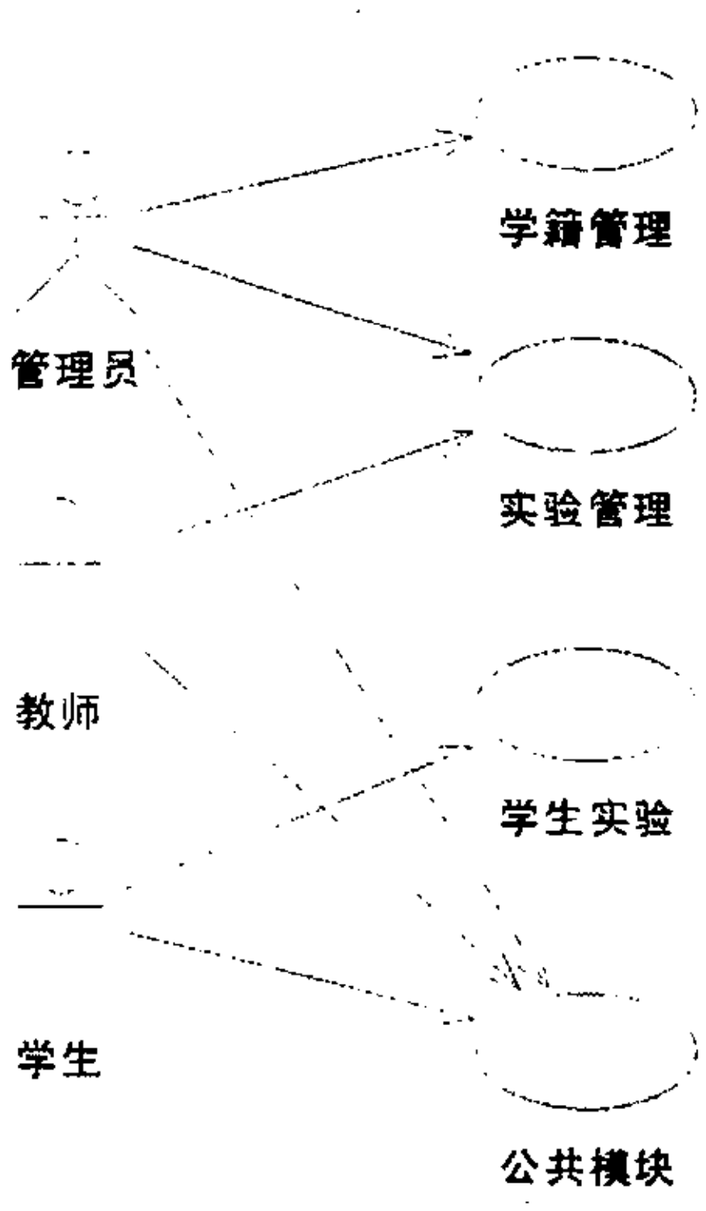

图 3-3 系统总体需求

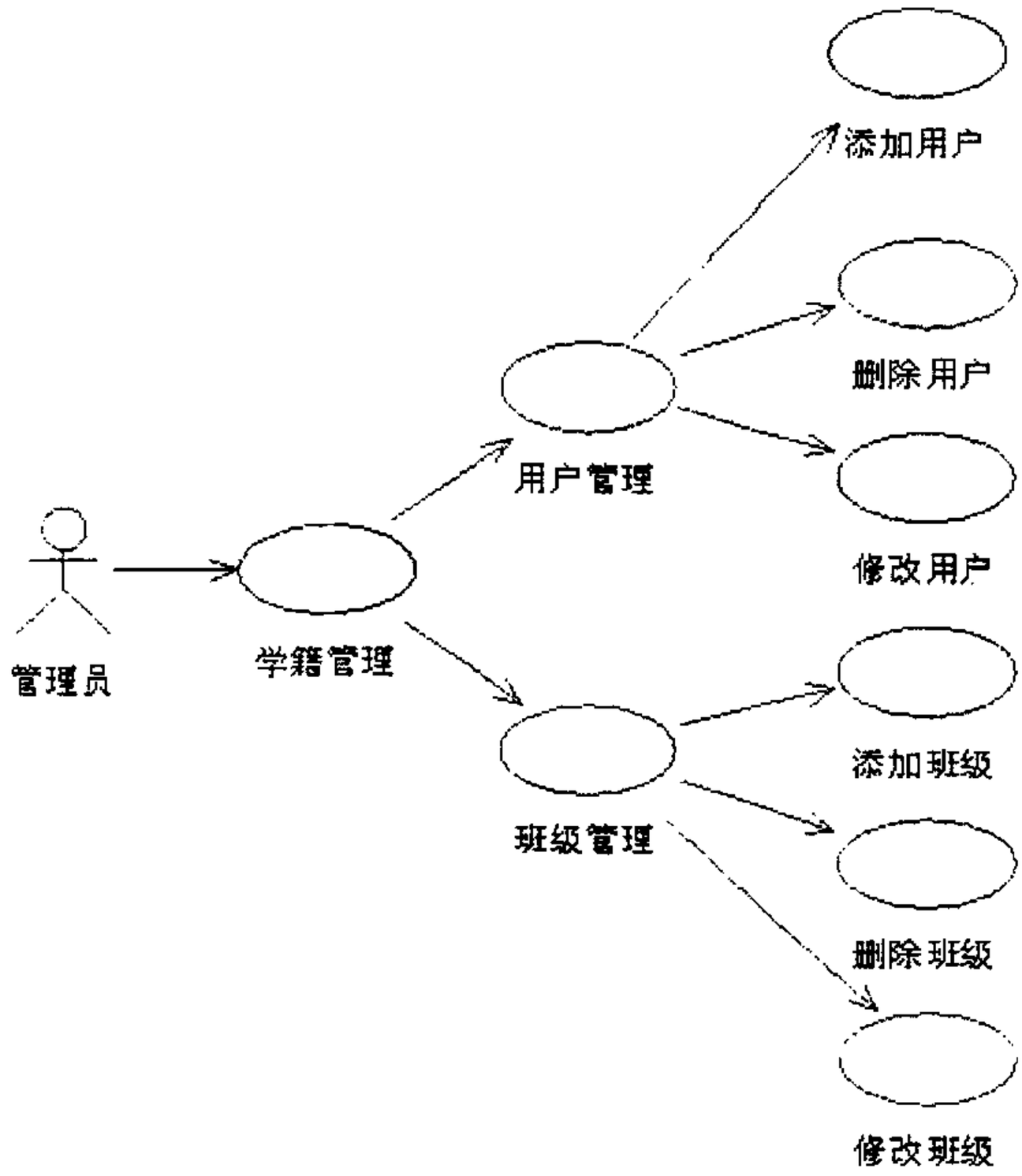

图 3-4 学籍管理用例

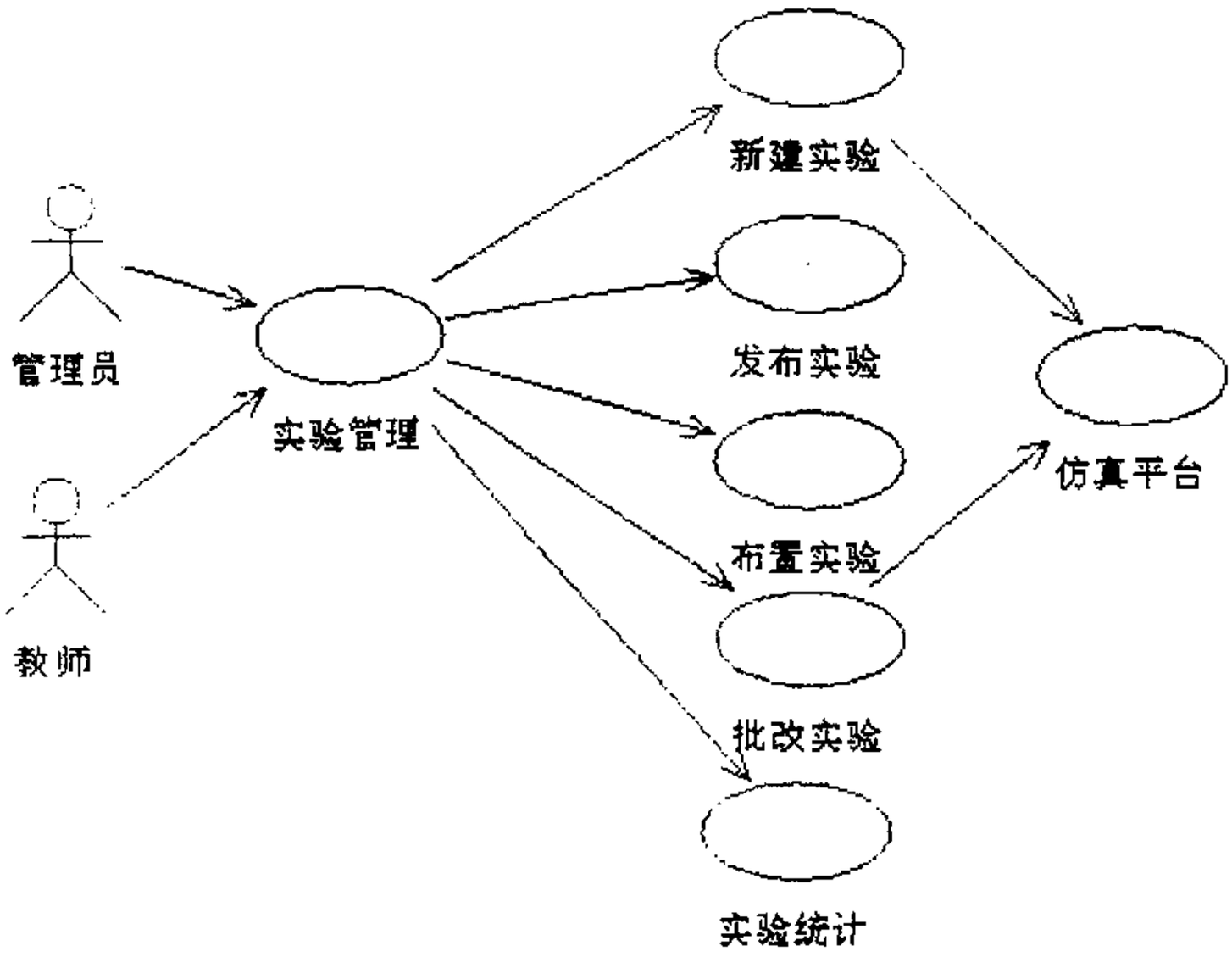
管理员
教师
实验管理
新建实验
发布实验
布置实验
批改实验
实验统计
仿真平台

图 3-5 实验管理用例

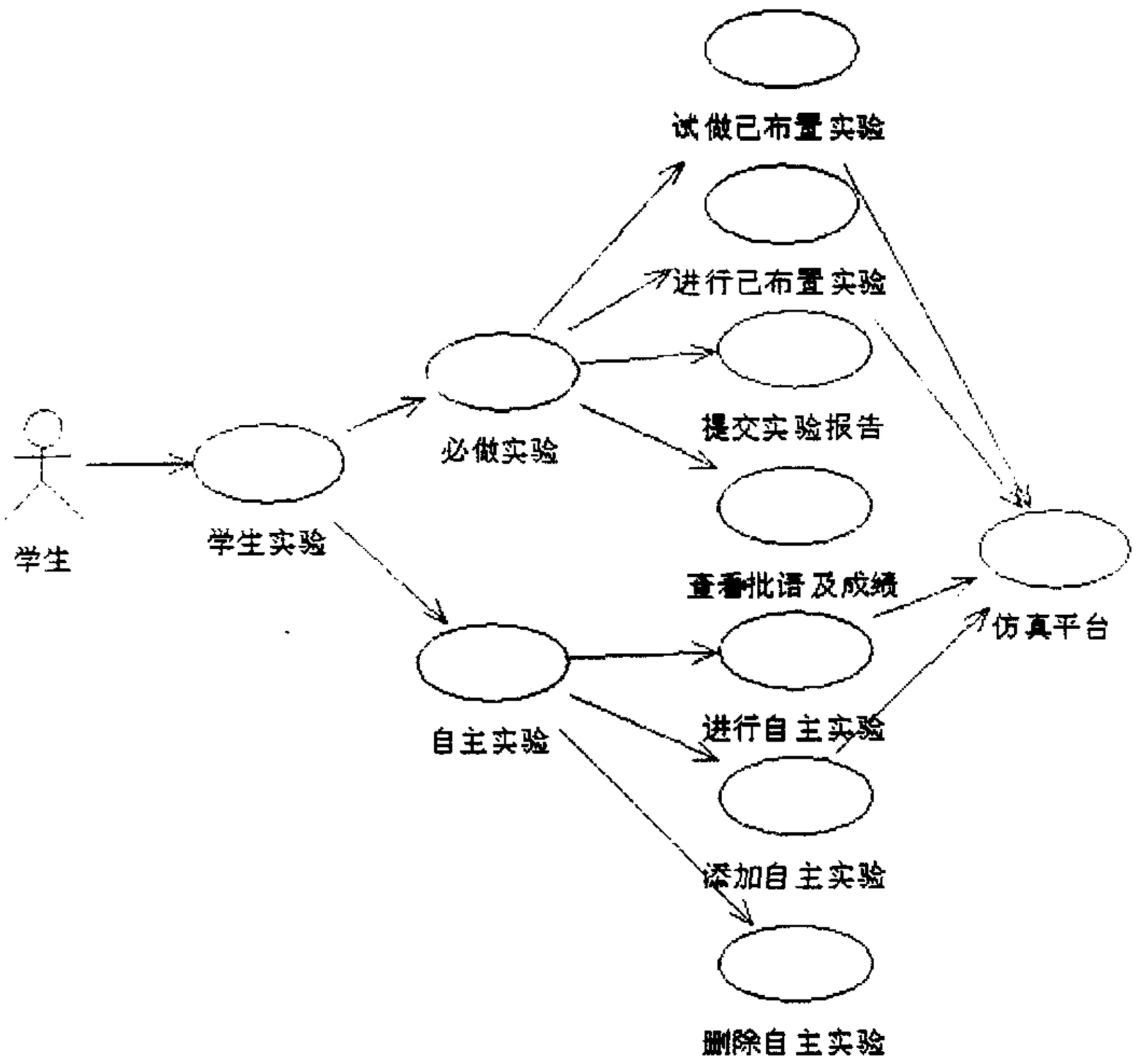
学生
学生实验
必做实验
自主实验
试做已布置实验
进行已布置实验
提交实验报告
查看批语及成绩
进行自主实验
添加自主实验
删除自主实验
仿真平台

图 3-6 学生实验用例

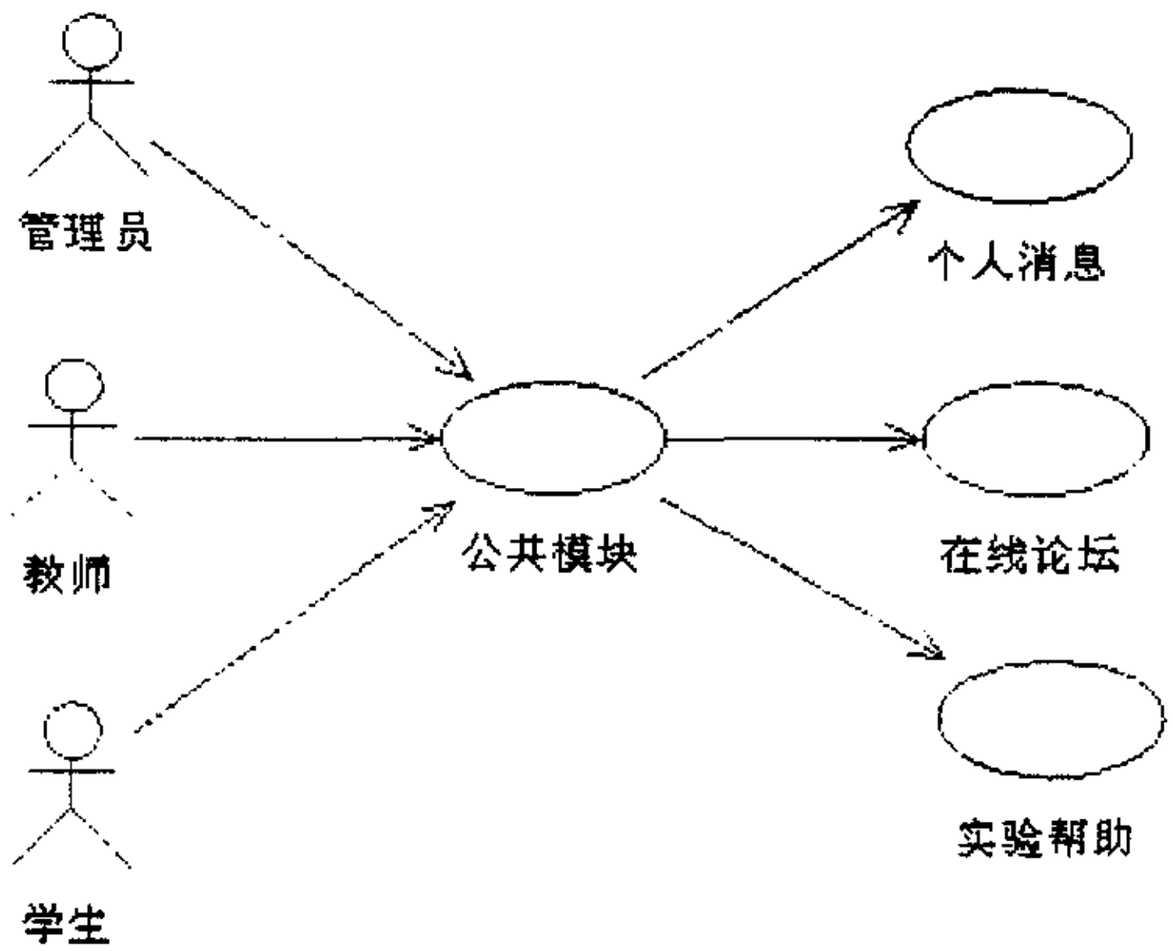

图 3-7 公共模块用例

以下是虚拟实验仿真系统中，“学生创建自主实验”用例的文字描述：

用例目标：创建自主实验

前件：进入自主实验列表用例

成功后件：无

失效后件：显示出错处理界面

触发：用户点击“自主实验列表”->＂创建自主实验＂入口

角色：学生

主事件流：

1. 点击＂填写实验基本信息＂按钮，填写包括实验名称，目的，要求，步骤等实验基本信息。

2. 点击＂创建实验＂按钮，显示虚拟实验平台，用户可以为实验添加器材，设定参数等（参考教师创建实验的过程）。

3. 点击＂保存＂按钮，保存新创建的实验。

扩展：

2a. 点击＂创建实验＂按钮，由于超出自主实验数量上限，显示消息框提示用户。

3a. 点击＂消息框确定＂按钮，回到步骤一或进入＂自主实验列表用例＂进行＂修改＂或＂删除＂操作。

说明：

学生创建自主实验有数量限制（3－5个），超出此限制则提示用户达到数量上限，用户可以删除实验满足创建实验的条件或修改实验（增删实验器材，调整参数等）完成创建新实验的目的。

虚拟实验仿真系统的管理部分有 45 个如上的用例，这些用例详细的描述了管理平台的功能，并通过测试，这些模块共同组成了一个闭包的功能集合，系统的边界清晰的展现，为后面的开发工作打下了良好的基础。

3.3 系统设计阶段

3.3.1 系统运行环境

操作系统采用 Windows 2000/Windows XP；浏览器为 IE5.0 以上版本，需要安装 directx9.0 插件。

3.3.2 系统配置

系统分为管理平台和仿真平台两个部分。前者实现了具体的实验管理流程，包括用户管理、实验管理等模块；后者提供了进行实验的仿真环境。

管理平台采用分布式计算环境，基于 B/S 架构，符合 J2EE 技术规范。它通过 Web 方式向用户提供服务，用户使用浏览器（如 Internet Explore）就可以访问系统。管理平台的服务器端还使用了引擎服务器、数据库服务器以及应用服务器来共同提供服务。引擎服务器采用 Resin，用来解析 JSP 文件；数据库服务器采用 MySQL，用来保存系统中的各种数据信息；应用服务器是可选的，可以用来部署可重用的组件以及优化对数据库的访问。

仿真平台采用 ActiveX 技术，仿真的实验环境以控件的形式嵌入到网页中。因此，客户端需要注册仿真平台控件，客户端浏览器可以看成是一个瘦客户。

管理平台的服务器既可以配置在同一台机器上，也可以分别配置在不同的机器上，用户通过局域网或 Internet 来访问系统提供的功能，系统配置图如图 3-8

所示：

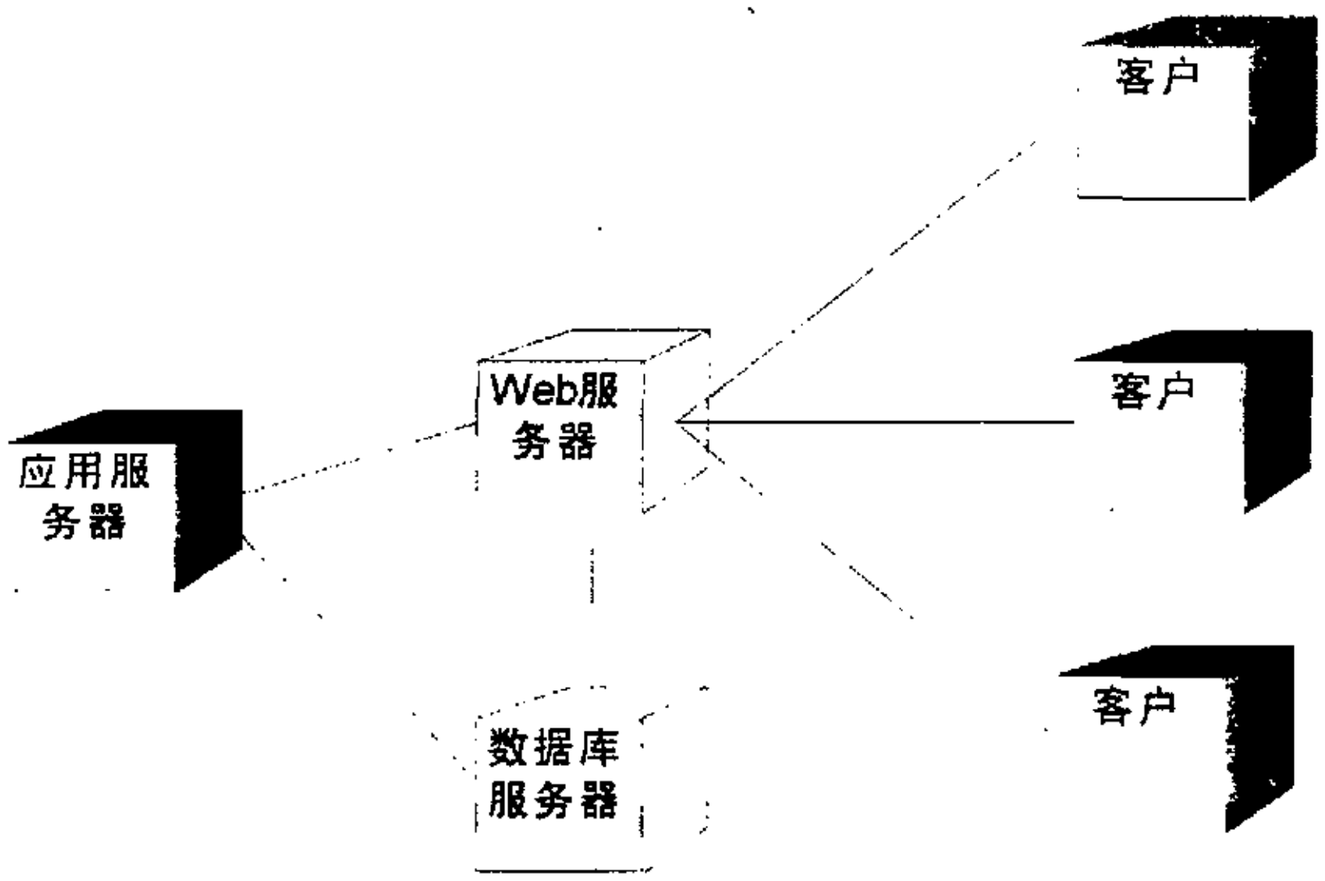

图 3-8

3.4 对象设计阶段

3.4.1 管理平台对象设计

3.4.1.1 管理平台对象建模

根据需求分析阶段得到的用例图，我们设计了管理平台需要的主要对象，分别是用户（包括管理员，教师和学生）、班级、实验、自主实验以及实验结果。由于教师和班级之间的关系表现为，教师为班级布置实验，而且布置实验的过程中包含了许多有用的信息，因此，我们对教师和班级之间的关联关系进行抽象，得到了布置实验信息对象。为了后面分析的需要，简要介绍这些主要对象包含的信息。

如图 3-9，是管理平台的用户信息，包括用户姓名、账号、密码、性别、出生日期、地址、籍贯、电子邮件信息和个人主页地址。这些是系统用户的必要的信息，通过这些信息来标识系统中的用户、与用户进行联系以及为统计模块提供信息。

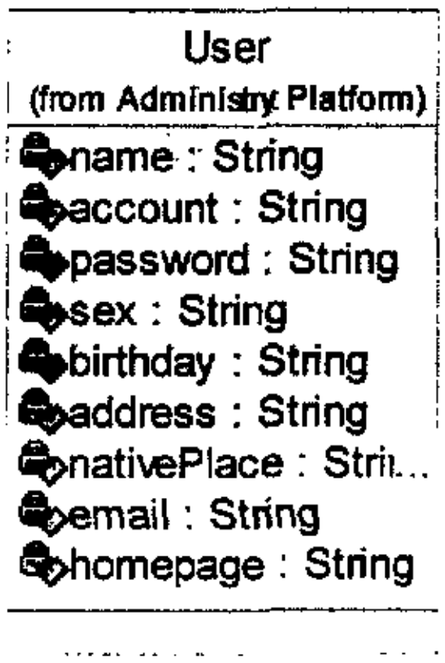

图 3-9

如图 3-10，是实验信息，包括实验名称、注释、创建者、创建时间、状态（已发布或未发布）、实验脚本（与仿真平台的接口）、空白实验报告和参考答案。

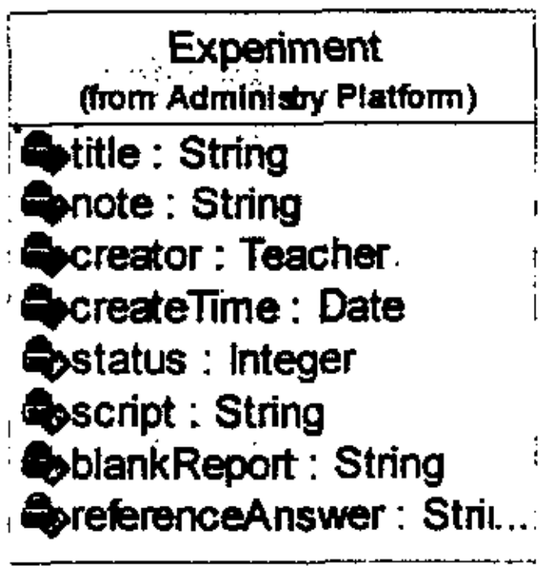

图 3-10

如图 3-11，是布置实验的信息，这个类是教师和班级关联关系的抽象，保存了教师对自己的班级布置实验任务的信息，包括布置实验的教师、接受实验任务的班级、所布置的实验、实验任务标识、实验结果提交最后期限、实验报告提交最后期限、成绩给出方式（百分制、优良中差或 ABCD 等级）以及成绩发布时间。

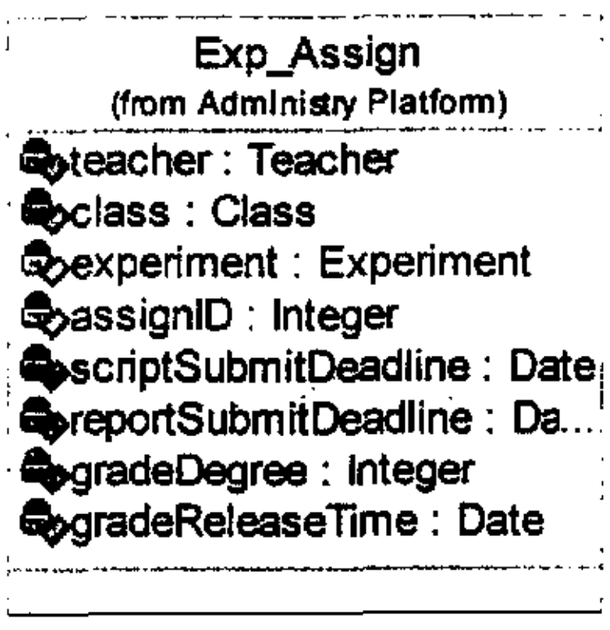

图 3-11

如图 3-12,是实验结果信息，包括实验结果（保存后的实验脚本）、实验报告、当前的实验步骤、成绩、教师批语以及所对应的实验任务。

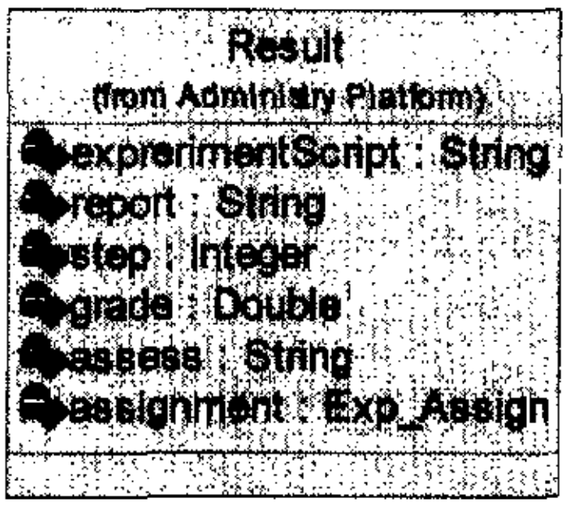

图 3-12

系统参照高校电路实验的流程，为学生实验规定了几个必须的步骤，学生必须按步骤进行实验。并对各个步骤规定了最后期限，如果到达最后期限仍没有完成当前步骤，则直接进入下一步，当前步骤的成绩按完成情况给出成绩。实验步骤的跳转，可以看作实验状态的变化，可以用 UML 技术提供的状态图来描述。如图 3-13 所示。

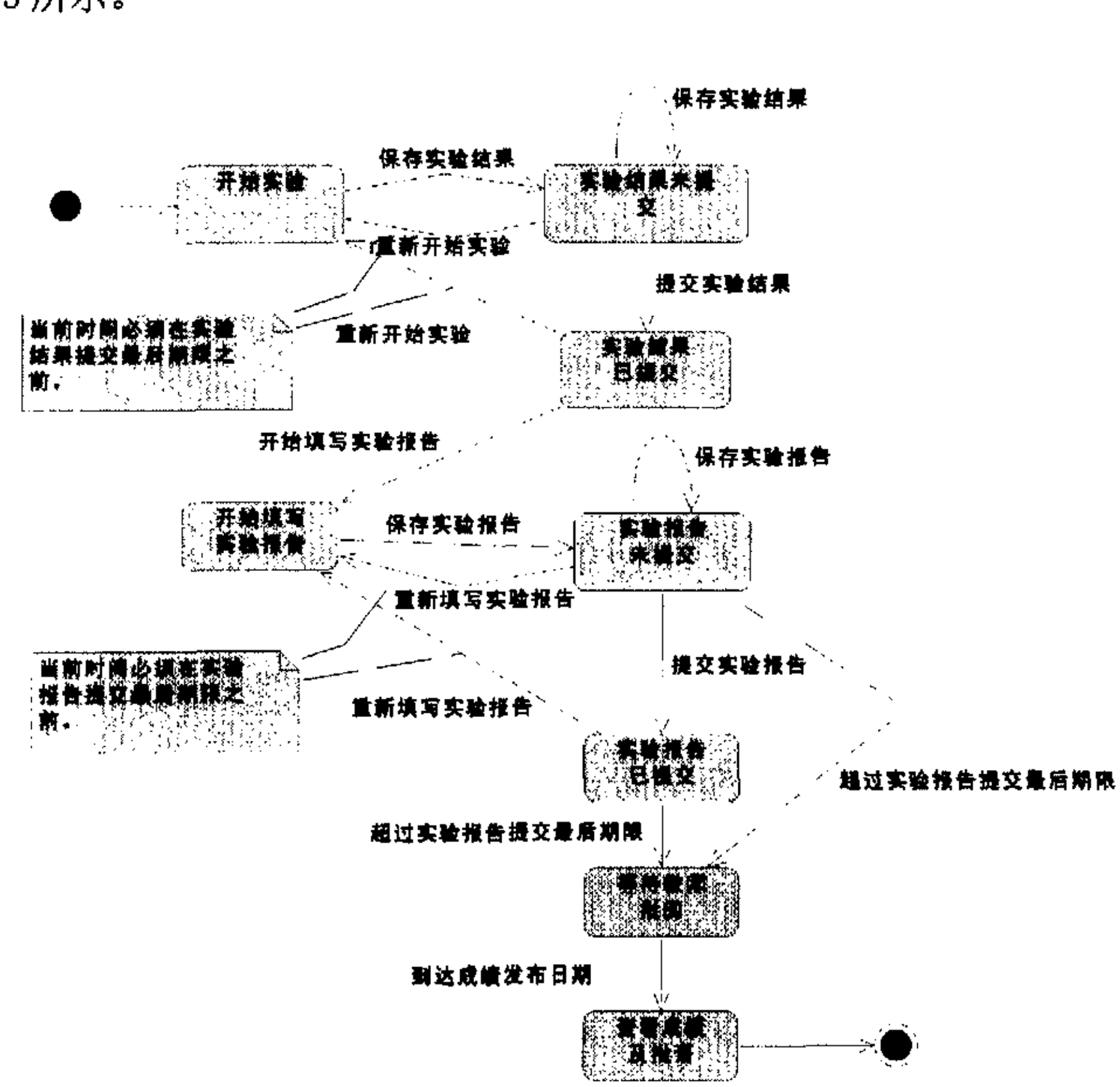

图 3-13

学生进行实验过程中要经历 8 个状态，分别为“开始实验”、“实验结果未提交”、“实验结果已提交”、“开始填写实验报告”、“实验报告未提交”、“实验报告已提交”、“等待教师批阅”和“查看批语及成绩”。各个状态之间的转换和触发条件，在图 E 中已经详细注明。下面介绍前四个状态之间的转换作，后面的状态的转换与此相似。

1.“初态”是状态机的起始状态，这里表示用户以学生身份登陆成功。学生选择某个实验任务，进入“开始实验”状态。

2.在“开始实验”状态中，学生可以在仿真平台上按实验要求搭建电路图，并保存实验结果（搭建的电路图），进入“实验结果未提交状态”。

3.在“实验结果未提交状态”中，可以进行三种操作：

1).取消以前所做的工作，重新开始实验，回到“开始实验”状态。进行这个操作的前提是，当前时间在实验结果提交的最后期限之前。

2).修改电路图，并保存实验结果，再次回到“实验结果未提交”状态。

3).提交实验结果，进入“实验结果已提交”状态。

4.在“实验结果未提交”状态中，可以进行两种操作：

1).取消以前所做的工作，重新开始实验，回到“开始实验”状态。进行这个操作的前提是，当前时间在实验结果提交的最后期限之前。

2).选择开始填写实验报告，进入“开始填写实验报告”状态。

其他状态之间的转换如图 E 所示，这里不再赘述。

3.4.1.2 管理平台对象关系建模

如图 3-14 所示，管理平台的三类用户，管理员、教师和学生，都是从用户类派生。这三类用户的区别在于对系统访问的权限不同：管理员可以访问“学籍管理”模块、“实验管理”模块和“公共模块”；教师可以访问“实验管理”模块和“公共模块”；学生可以访问“学生实验”和“公共模块”。

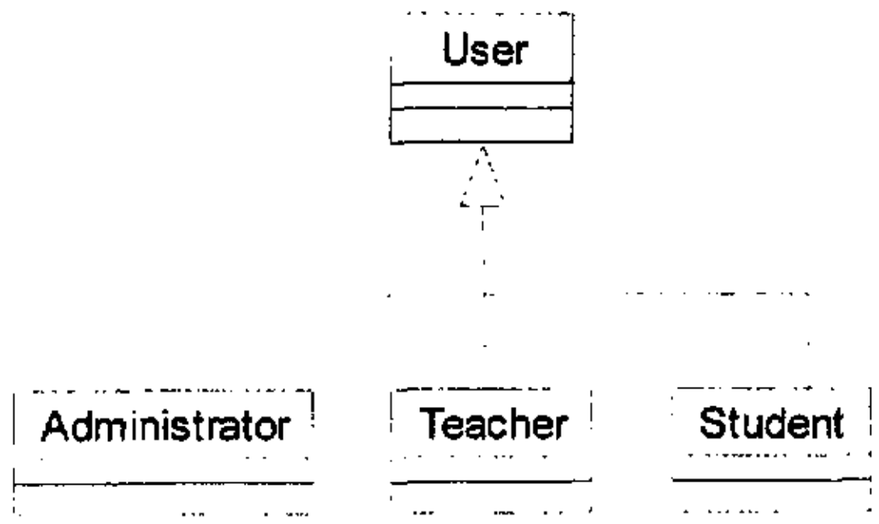

图 3-14

和图 3-15 所示，管理员可以为系统添加不同身份的用户（教师或学生），并可以对学生分班（创建班级）；教师可以管理实验，包括创建、修改、删除和发布实验等操作，而且，教师可以为自己的班级布置实验任务，表现为教师和班级之间的关联关系。此关联关系含有较多信息，抽象为类 Exp_Assign，每个实验任务对应 1 个或多个实验结果，具体数量和该班级的学生人数相同。班级和学生之间是包含关系，一个班级可以包含一个或多个学生，一个学生只能属于一个班级。学生可以创建自主实验，一个学生最多可以创建 5 个自主实验，并可以对自己创建的实验进行修改、保存和删除等操作。通过图 3-15 可以清楚的看到系统中各个对象之间的关系，不仅如此，利用对象之间的数量关系可以直接映射到数据库的设计，例如，实验任务对象和实验结果对象之间是一对多的关系，可以方便的用于数据库设计中实验对象表和结果对象表的关联。

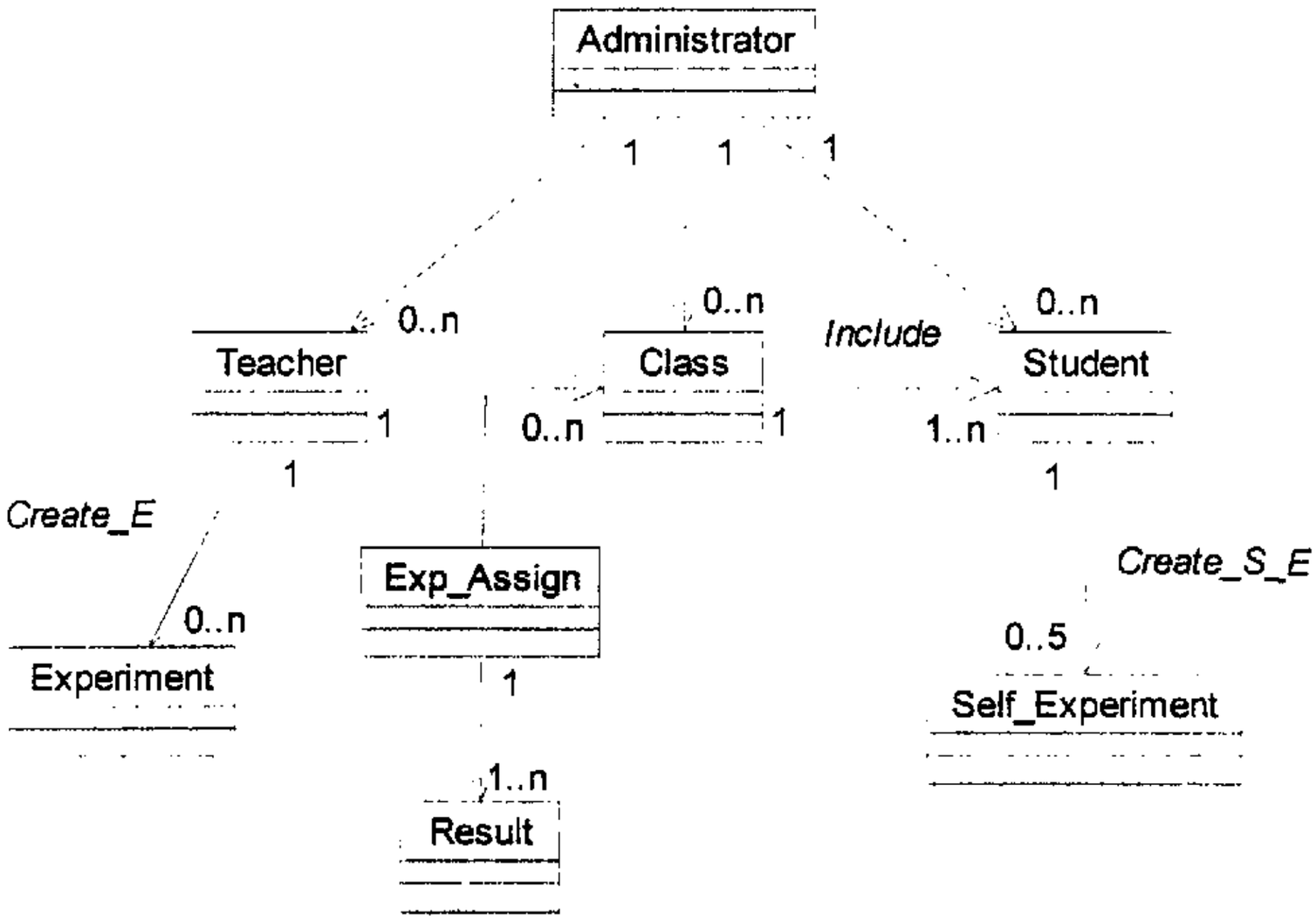

图 3-15

3.4.2 仿真平台对象设计

3.4.2.1 实验器材建模

仿真平台仿真了电路实验中需要的大部分器材，系统中对器材建模时，采用了面向对象的继承机制，对象间主要是泛化关系，对器材建模的类图如图 3-16 所示。

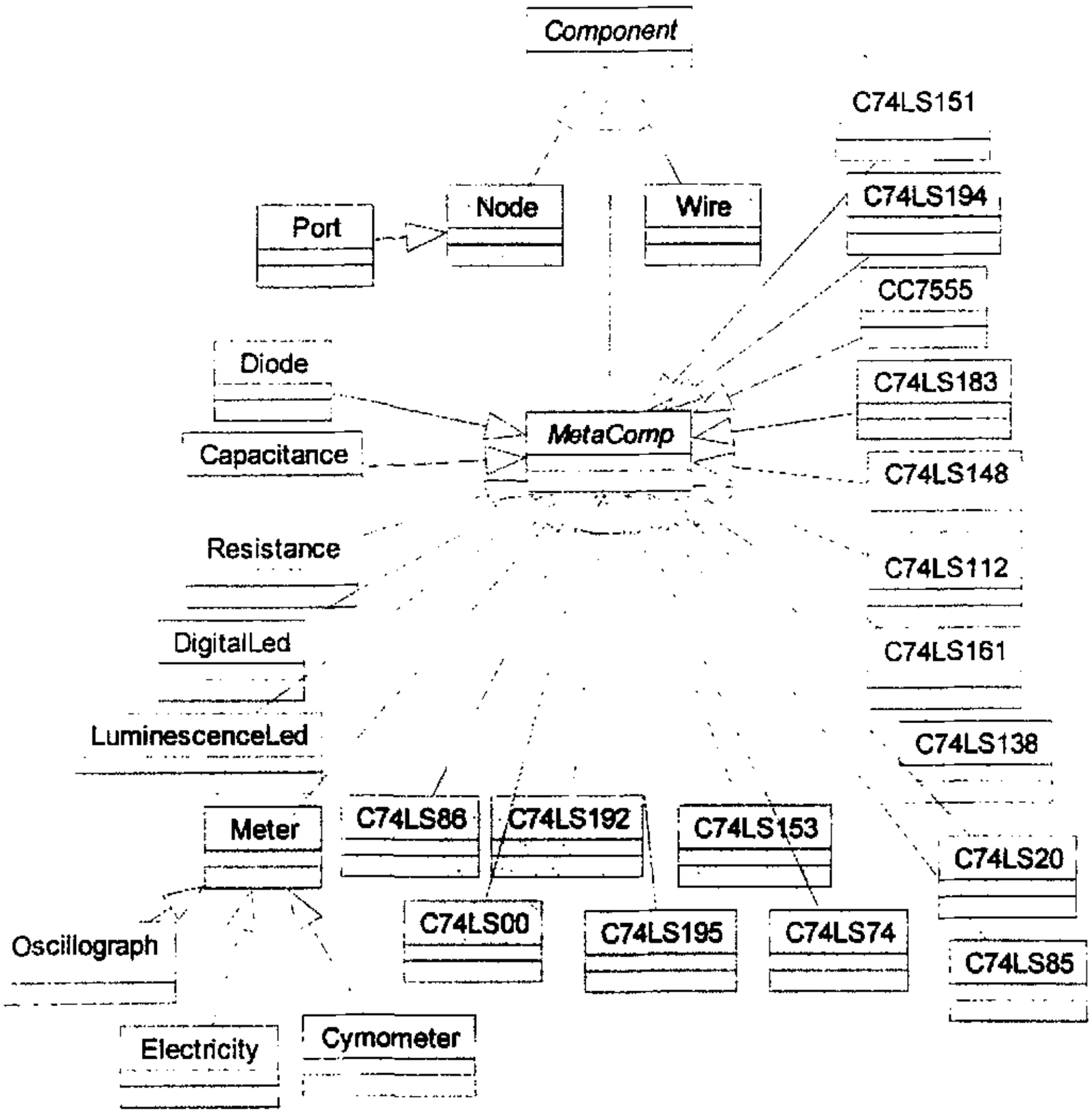

图 3-16 器材类图

系统所有需要显示的内容，全部从类 Component 继承，类 Component 包含三个子类，分别为 Node、MetaComp 及 Wire。类 Node 的对象和类 Wire 的对象之间关系十分紧密，连线的两端必须连接在节点上。同时类 MetaComp 的对象和类 Node 的对象共同完成了器材的仿真实现，器材通过节点与其它器材相连组成各种各样的电路图。

仿真现实中的电路实验器材是从类 MetaComp 继承的，主要分为三部分：首先是基本器材，包括类 Diode（二极管）、类 Capacitance（电容）、类 Resistance（电阻）、类 DigitalLed（数字二极管）、类 LuminescenceLed（发光二极管）；其次是仪表器材类 Meter，它包含三个子类 Oscillograph（示波器）、类 Electricity（）、类 Cymometer（），这也是系统所实现的三种仪表，通过使用

这三种仪表，可以测量电路中的电流，器材两端的电压，并能显示相关的波形图；还有系统实现了 15 种常用数字芯片，通过使用这些芯片可以进行教学大纲要求的数字电路实验。

对于系统中所有显示的内容都从同一个基类继承，这是为了充分的利用面向对象技术中的多态性，这样系统中的所有对象都可以看成是类 Component 的对象，这使在后面的控制器设计中把所有器材用相同的方式管理成为可能。

3.4.2.2 控制器建模

仿真平台共设计了三种控制器，它们是鼠标事件控制器，键盘事件控制器和命令控制器。顾名思义，前两种控制器分别用来处理鼠标事件和键盘事件，命令控制器主要用来响应菜单命令，以及处理需要撤销/重做功能的事件。

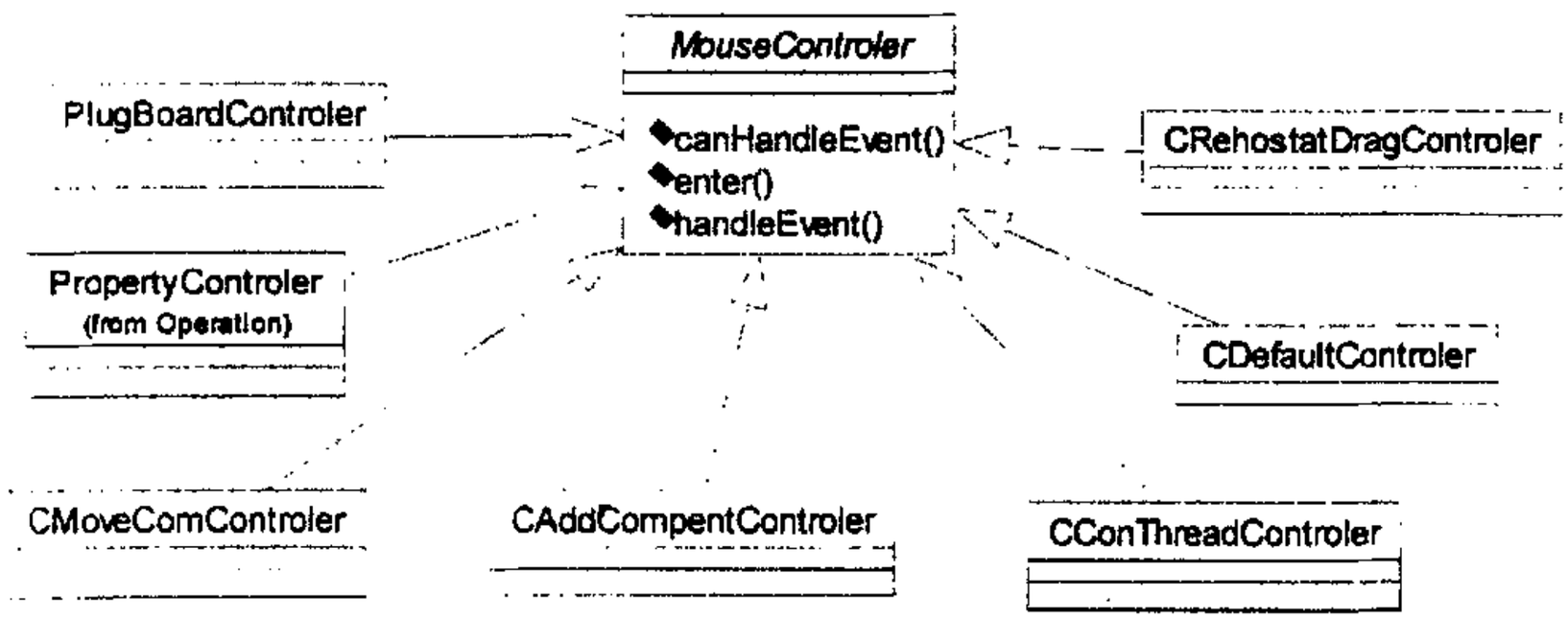

图 3-17 鼠标事件控制器

如图 3-17 所示，系统中处理具体鼠标事件的类 PropertyController（属性控制器）、类 CmoveComController（移动器材控制器）、类 CconTreadController（连线控制器）、类 CdefaultController（缺省控制器）都从基类 MouseController 继承。类 MouseController 声明了三个方法，其中的 canHandleEvent()方法很重要，系统中实现的所有控制器在同一时刻只能有一个在工作，也就是说当一个事件发生时只有一个控制器能够处理该事件。这就要求每个实现具体功能的控制器的 canHandleEvent()方法返回 true 的条件必须互斥，否则系统可能产生错误的行为。而其它两个方法，enter()用来初始化控制器的属性，handleEvent()用来处理事件。系统中实现的所有鼠标事件控制器在

仿真平台初始化的时候，在类 ControleMng 中进行注册，通过类 ControleMng 的对象来处理从系统捕获的鼠标事件。

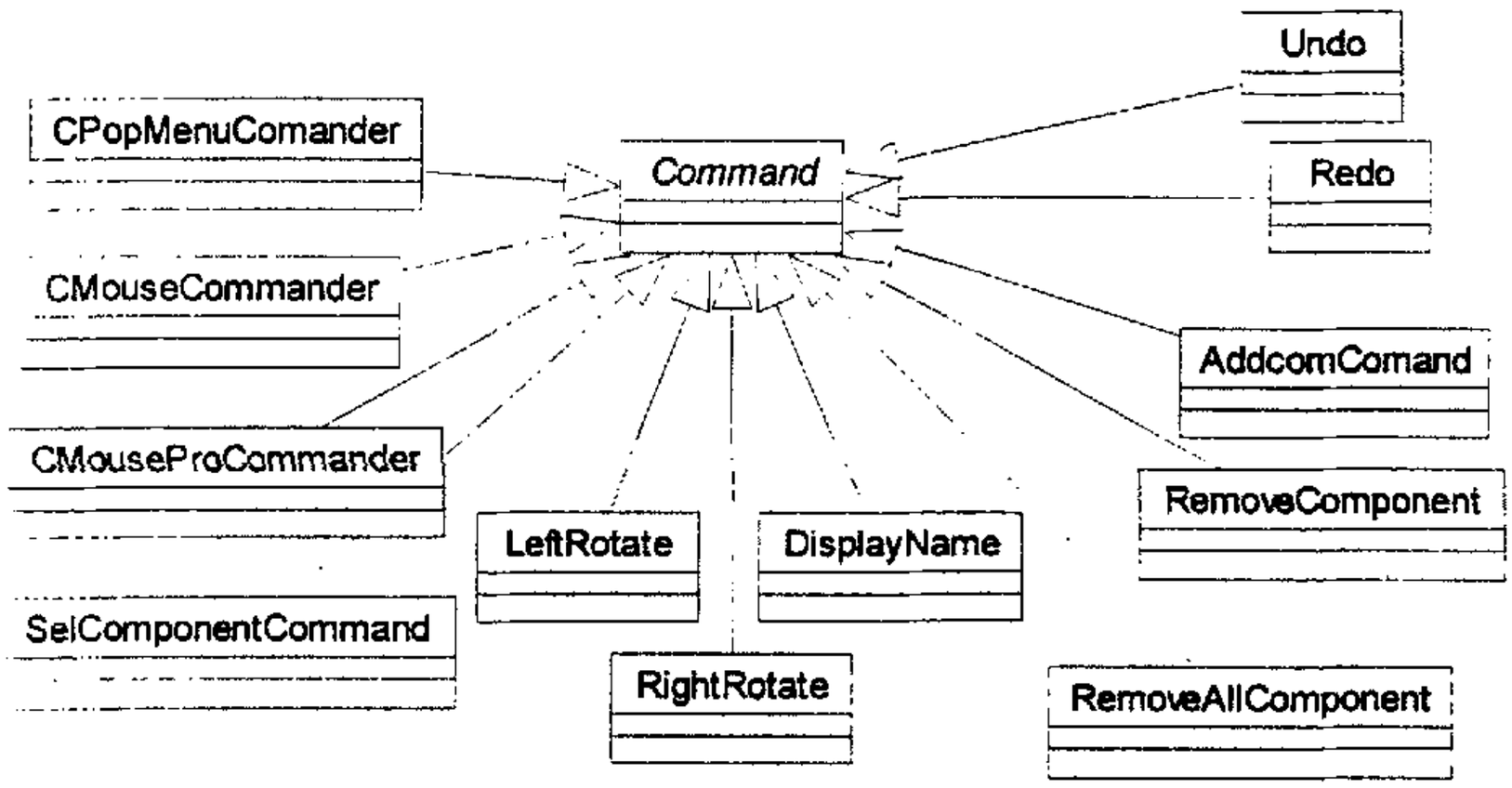

图 3-18 命令控制器

如图 3-18 所示，系统中的命令控制器从基类 Command 继承，类 CMouseCommander、类 SelComponentCommand、类 RemoveComponent、类 RemoveAllComponent、类 Redo 和类 Undo 等都是用来处理菜单命令。这里我们使用了 Command 模式，系统中实现的命令控制器所执行的操作可以实现“撤销/重做”（在系统实现部分会详细介绍）功能，提高了使用系统的便利性。撤销/重做功能的具体实现在后面有详细的讨论。系统中实现的所有命令控制器在仿真平台初始化的时候，在类 ControleMng 中进行注册，通过类 ControleMng 的对象来处理菜单命令。

3.4.2.3 绘制功能建模

如图 3-19，基类 RenderSystem 声明了绘制系统应该提供的服务，它有两个子类：类 NullRenderSystem 和类 Direct8_Sys。其中类 NullRenderSystem 用来处理当系统软硬件配置不符合要求的情况，类 Direct8_Sys 提供了系统上层对象需要使用的基本的绘制功能。类 InforMation 和类 FontUtil8 共同支持类 Direct8_Sys 提供的功能，它们之间是结构关系，类 Direct8_Sys 包含这两个对

象的引用来控制绘制的参数。而类 Dx8FontUtil、类 Dx8_RenderUtil 和类 CacheUnit8，依赖于类 Direct8_Sys 它们之间体现了一种使用关系。这三个类的操作需要使用类 Direct8_Sys 的对象作为参数，来实现特定的功能。类 Direct8_Sys 实现的改变会影响到这三个类。在进行系统维护和更新时，要特别注意这一点。我们不仅要保证修改后的类 Direct8_Sys 是正确的，同时要对类 Dx8FontUtil、类 Dx8_RenderUtil 和类 CacheUnit8 进行测试，这样才能保证修改后设计的正确性。

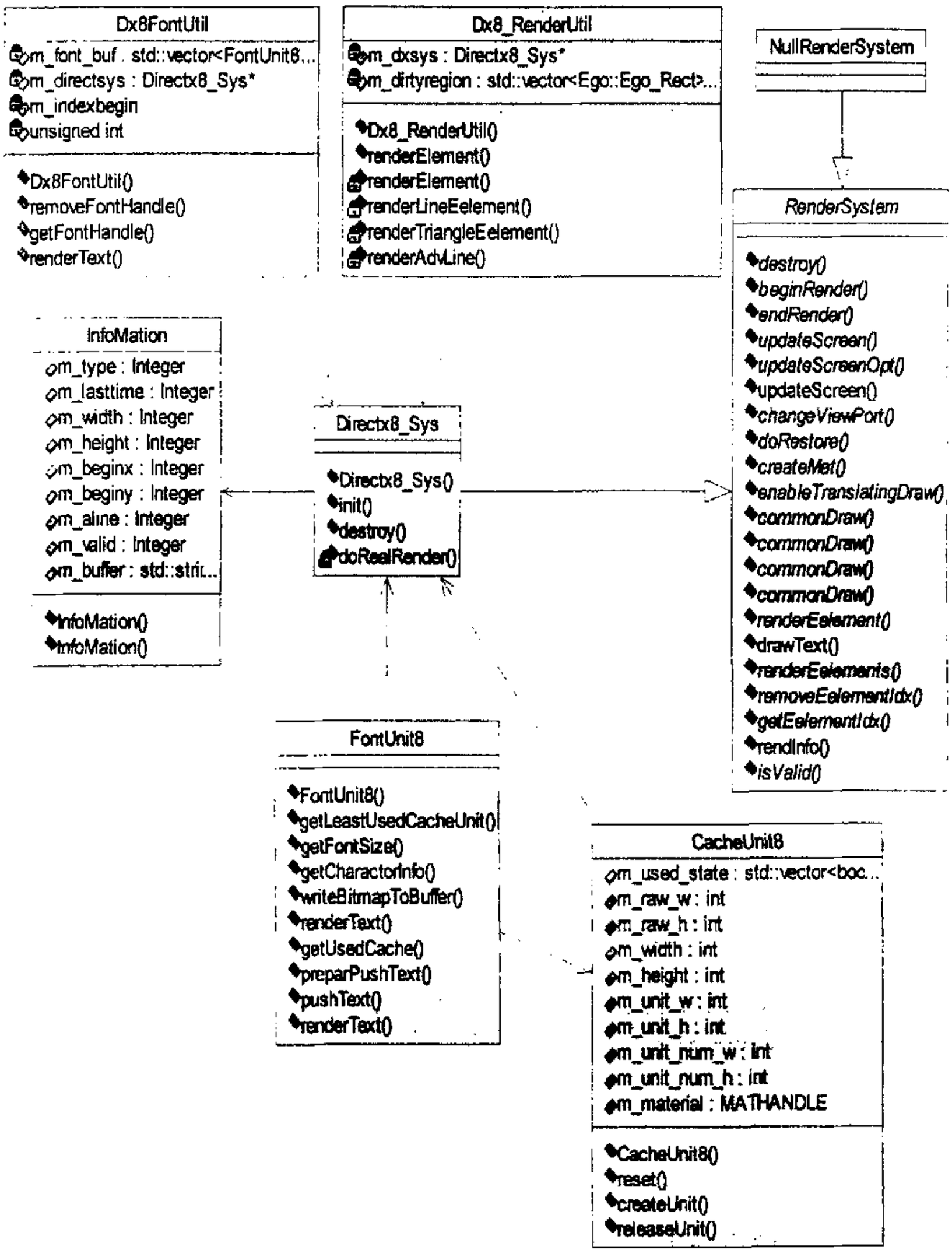

图 3-19 命令控制器

3.4.2.4 仿真平台对象关系建模

器材只是作为仿真平台的模型（Model），为使仿真平台可以运行起来，可以在其上搭建电路图，进行实验，仅仅对器材建模是远远不够的。

因此，除了器材之外，系统对其它逻辑概念进行了建模，它们之间的关系如

图 3-20 所示。

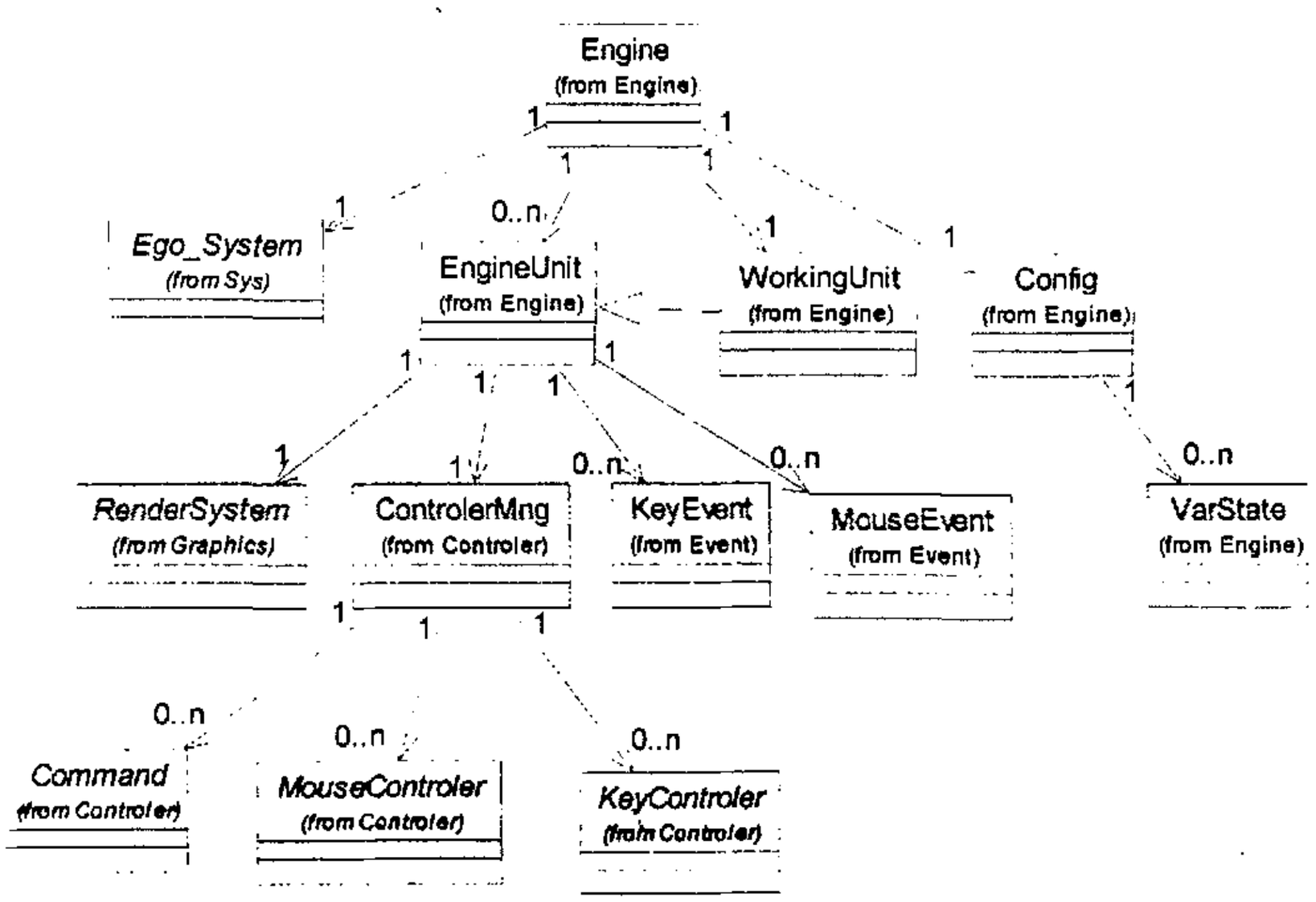

图 3-20 系统关系图

仿真平台的所用功能作为静态方法封装在类 Engine 中，也就是说类 Engine 不能实例化。类 Engine 包含类 Ego_System 的一个对象引用，来保存操作系统信息；包含类 Config 的对象的引用，保存需要提供给外部接口的配置信息；还包含了类 EngineUnit 的对象的集合，其中的一个类 WorkingUnit 的对象是当前系统中实际提供功能的引擎单元。

类 EngineUnit 包含一个类 RenderSystem（绘制系统）对象的引用；一个类 ControleMng（控制器管理器）对象的引用；一个类 MouseEvent（鼠标事件）对象的集合；一个类 KeyEvent（键盘事件）对象的集合。类 ControleMng 包含了一个类 Command（命令）对象的集合；一个类 MouseController（鼠标事件控制器）对象的集合；一个类 KeyController（键盘事件控制器）对象的集合。其中类 Command 和类 MouseController 作为基类规定了命令控制器和鼠标事件控制器所必须提供的服务（方法）。仿真平台截取操作系统产生的各种事件，保存在类 EngineUnit 的两个事件集合中，在固定的时间周期到来时发送给类 EngineUnit 的类 ControleMng 对象的引用，ControleMng 对象轮询自己包含的命令集合和控

制器集合，找到相应的控制器，并由该控制器处理类 EngineUnit 的对象发送来的事件。上述过程的时序图如图 3-21 和 3-22 所示。

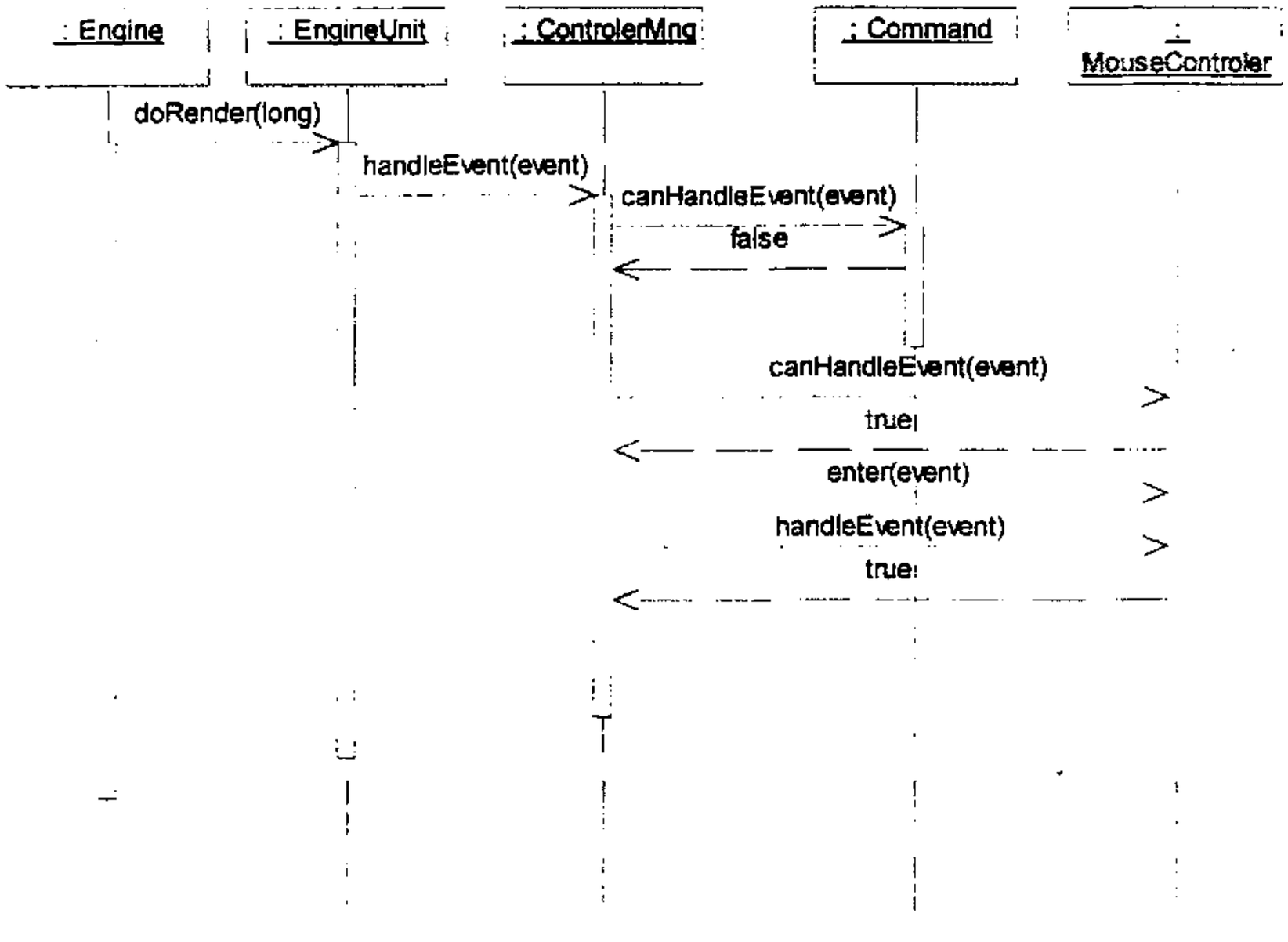

图 3-21 控制器时序（1）

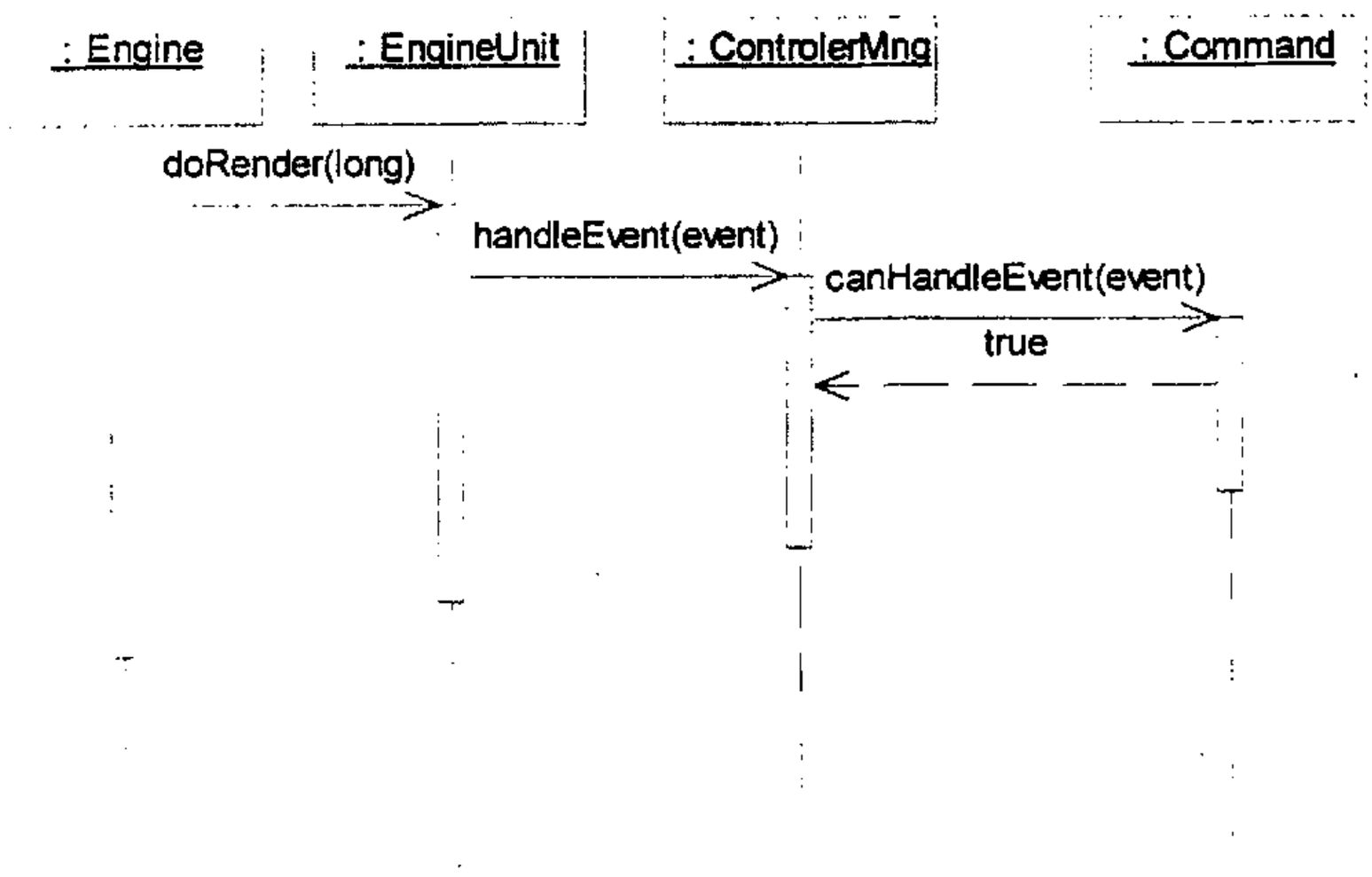

图 3-22 控制器时序（2）

下面深入分析上面两个图的含义。如图 3-21 所示，在固定时间周期到达后，类 Engine 向类 EngineUnit 对象发送 doRender()消息，要求更新仿真平台显示的内容；类 EngineUnit 的对象从自己保存的事件队列中，选取一个事件 event 为参数的 handleEvent 消息，发送给自己保存的类 ControleMng 对象；该 ControleMng 对象轮询自己保存的 Command 对象集合，并向其中的每一个命令发送包含 event 参数的 canHandleEvent 消息，若 Command 对象不能处理该对象，返回 false；ControleMng 对象开始轮询自己保存的 MouseController 对象的集合，并向其中的每一个鼠标事件控制器发送包含 event 参数的 canHandleEvent 消息，若可以处理该事件，返回 true；ControleMng 对象依次向 MouseController 对象发送包含 event 参数的 enter 消息和包含 event 参数的 HandleEvent 消息，MouseController 处理完该事件后，向 ControleMng 对象返回 true。EngineUnit 对象从自己的事件集合中删除该事件，并继续向 ControleMng 对象发送下以一个事件为参数的 handleEvent 消息。重复以上过程直到 EngineUnit 对象的事件集合为空为止。

图 3-22 所示的时序与图 3-21 类似，在这里 Command 对象可以处理 event 事件，事件由 Command 对象处理，不需要再发送给 MouseController 对象。

如上所示的时序过程反映了仿真系统中所采用的消息转发机制，系统上层对象如（Engine，EngineUnit）不直接调用处理具体事件的对象（Command，MouseController）的方法，而是通过 ControleMng 对象进行转发，降低了耦合性，使系统快速灵活的增加控制器和命令成为可能。增加新的控制器只需把新控制器在控制器管理器中注册即可，不需要改变类 Engine 和类 EngineUnit 的实现。

3.5 系统实现阶段

3.5.1 软件配置管理

3.5.1.1 软件配置管理（SCM）概述

当建造计算机软件时，变化是不可避免的，并且，变化使得共同工作在某一项目中的软件工程师之间的彼此不理解程度更加增高。当变化进行前没有经过分

析、变化实现前没有被记录、没有向那些需要知道的人报告变化、或变化没有以可以改善质量及减少错误的方式被控制时，则不理解性将会产生。

协调软件开发把不理解性降到最小程度的技术称为配置管理。配置管理是对正在被一个项目组建造的软件的修改标识、组织和控制的技术，其目标是通过最大限度地减少错误，来最大限度地提高生产率。

软件配置管理(SCM)是贯穿于整个软件过程中的保护性活动。因为变化可能发生在任意时间，SCM 活动被设计来(1)标识变化，(2)控制变化，(3)保证变化被适当地实现，以及(4)向其他可能有兴趣的人员报告变化。

明确地区分软件维护和软件配置管理是很重要的。维护是发生在软件已经被交付给客户，并投入运行后的一系列软件工程活动，而软件配置管理则是当软件项目开始时就开始，当软件运行结束后才终止的一组跟踪和控制活动。

软件配置管理的主要目标是可以更容易地适应改进的变化，并减少变化发生时所需花费的工作量。软件过程的输出信息可以分为三个主要的类别：

(1)计算机程序(源代码和可以执行的程序)，

(2)描述计算机程序的文档(针对技术开发者和用户)，

(3)数据(包含在程序内部或在程序外部)。

这些项包含了所有在软件过程中产生的信息，总称为软件配置。

3.5.1.2 系统中的软件配置管理

在仿真平台的开发中，我们使用 VSS（Visual Source Safe）进行源代码的管理。开发小组的所有成员，能够看到整个系统的源代码。但对于每一个源代码文件，在某个时刻只有一个授权的用户可以进行修改，当对修改后的系统测试无误后，将新版本的源代码提交到配置管理服务器上。小组中的其他成员从配置服务器上下载新版本，来观察程序的变化。同时，VSS 可以记录在过去多个时间点上提交的程序，这样我们可以方便的控制程序的版本，并根据需要提交给不同用户不同版本的程序。

3.5.2 算法设计与实现

以“撤销/重做”功能为例，介绍系统算法的设计与实现。系统实现了“撤

销/重做”功能，这样当用户进行了误操作后，可以通过一个简单的快捷键撤销该操作。增加这个功能后用户可以连续查看以往进行的操作，只要用户没有进行新的操作，就可以浏览以前的一定数量的操作，以及这些操作带来的变化。

“撤销/重做”功能的实现要借助于 Command（命令）模式

如前控制器建模部分所示，可进行撤销操作的控制器，都是类 Command 的子类，Command 的定义如下：

```
class Command
{
public:
    /// 处理事件
    virtual void execute(const MouseEvent& evt) {};
    virtual void execute(const KeyEvent& evt) {};
    /// 直接运行
    virtual void execute() {};
    /// 撤销操作
    virtual void unexecute() {};
};
```

方法 execute()完成命令控制器要实现的功能，方法 unexecute()执行方法 execute() 的逆操作，撤销命令控制器的操作，使系统恢复到该命令控制器操作执行之前的状态。这是实现“撤销/重做”功能的基础。

系统中的其他命令控制器，如“添加器材控制器”、“删除器材控制器”、“删除全部器材控制器”、“顺时针旋转器材控制器”和“逆时针旋转器材控制器”都实现了基类 Command 中定义的这两个方法。因此，对这些控制器也可以进行撤销和重做操作。

同时，系统中建立了两个堆栈 undoStack 和 redoStack。undoStack 为“撤销”操作服务，记录了前 N 次执行了命令控制器对象；redoStack 为“重做”操作服务，记录了前 N 次“撤销”的命令控制器对象。

算法如下：

1) 当用户执行了命令控制器的 execute()操作之后，就将该命令控制器对象压入 undoStack，并清空 redoStack 中的所有命令控制器对象。

2) 当用户执行“撤销”操作时，就从 undoStack 中弹出一个命令控制器，并将该命令控制器对象压入 redoStack 中，然后，执行它的 unexecute()的操作。

3) 当用户执行“重做”操作时，就从 redoStack 中弹出一个命令控制器，将该命令控制器对象压入 undoStack 中，然后，执行它的 execute()操作。

具体实现代码如下：

1.“撤销”命令控制器

```
class Undo : public Ego::Command
{
    ......
public:
    virtual void execute()
    {
        if (!Ego::Engine::s_working_unit->undoStack.empty())
        {
            /// 从 undoStack 中取出命令控制器对象
            Ego::Command* Command =
                Ego::Engine::s_working_unit->undoStack.top();
            /// 执行该命令控制器的 unexecute()方法
            Command->unexecute();
            /// 将该命令控制器对象压入 redoStack 中
            Ego::Engine::s_working_unit->redoStack.push(Command);
            /// 从 undoStack 中该命令控制器
            Ego::Engine::s_working_unit->undoStack.pop();
        }
    }
};
```

2.“重做”命令控制器

```
class Redo : public Ego::Command
{
    ......
public:
    virtual void execute()
    {
        if (!Ego::Engine::s_working_unit->redoStack.empty())
{
    /// 从redoStack中取出命令控制器对象
    Ego::Command* Command =
        Ego::Engine::s_working_unit->redoStack.top();
    /// 执行该命令控制器的unexecute()方法
    Command->execute();
    /// 将该命令控制器对象压入undoStack中
    Ego::Engine::s_working_unit->undoStack.push(Command);
```

```
            /// 从redoStack中该命令控制器
            Ego::Engine::s_working_unit->redoStack.pop();
        }
    }
};
```

Undo 命令和 Redo 命令的 unexecute（）方法并没有实现。因为 Redo 命令的 execute（）操作相当于 Undo 命令的 unexecute()操作；同样，因为 Undo 命令的 execute（）操作相当于 Redo 命令的 unexecute()操作。

我们没有把 Undo 命令和 Redo 命令的功能放在一起形成一个控制器，是因为逻辑上，这是两个不同的操作。但是，与其他所有的命令一样通过 execute()方法实现具体功能。这样的设计可以保证系统中所有的命令控制器的操作采用相同的代码来执行，这也符合面向对象设计中的多态性要求。

3.6 测试阶段

3.6.1 基本测试概念

3.6.1.1 白盒测试

白盒测试，有时称为玻璃盒测试，是一种测试用例设计方法，它使用程序设计的控制结构导出测试用例。使用白盒测试方法，软件工程师能够产生测试用例来完成下列工作：

（1）保证一个模块中的所有独立路径至少被使用一次；

（2）对所有逻辑值均需测试 true 和 false；

（3）在上下边界及可操作范围内运行所有循环；

（4）检查内部数据结构以确保其有效性。

3.6.1.2 黑盒测试

黑盒测试注重于测试软件的功能性需求，也即黑盒测试使软件工程师派生出执行程序所有功能需求的输入条件。黑盒测试并不是白盒测试的替代品，而是用于辅助白盒测试发现其他类型的错误。

黑盒测试试图发现以下类型的错误：

(1) 功能不对或遗漏；

(2) 界面错误；

(3) 数据结构或外部数据库访问错误；

(4) 性能错误；

(5) 初始化和终止错误。

白盒测试在测试的早期执行，而黑盒测试主要用于测试的后期。黑盒测试故意不考虑控制结构，而是注意信息域。

3.6.2 单元测试

3.6.2.1 单元测试概念

单元测试完成对最小的软件设计单元——模块的验证工作。使用过程设计描述作为指南，对重要的控制路径进行测试以发现模块内的错误。测试的相关复杂度和发现的错误是由单元测试的约束范围来限定的。单元测试通常情况下是面向白盒的，而且这个步骤可以针对多个模块并行进行。

3.6.2.2 OO 语境中的单元测试

当考虑面向对象软件时，单元的概念发生了变化。封装驱动了类和对象的定义，这意味着每个类和类的实例(对象)包装了属性(数据)和操纵这些数据的操作(也称为方法或服务)。而不是个体的模块。最小的可测试单位是封装的类或对象，类包含一组不同的操作，并且某特殊操作可能作为一组不同类的一部分存在，因此，单元测试的意义发生了较大变化。

我们不再孤立地测试单个操作，而是将操作作为类的一部分。作为一个例子，考虑一个类层次，其中操作 X 针对超类定义并被一组子类继承，每个子类使用操作 X，但是它被应用于为每个子类定义的私有属性和操作的环境内。因为操作 X 被使用的语境有微妙的不同，有必要在每个子类的语境内测试操作 X。这意味着在面向对象的语境内在真空中测试操作 X(即传统的单元测试方法)是无效的。

对 OO 软件的类测试等价于传统软件的单元测试。和传统软件的单元测试不一样，它往往关注模块的算法细节和模块接口间流动的数据，OO 软件的类测试

是由封装在类中的操作和类的状态行为所驱动的。

3.6.2.3 系统中的单元测试

对系统中存在的对象，在开发过程中进行了大量的单元测试，从而保证了系统开发的高效。以下为系统中连线控制器及其测试类的实现。

```
class CConThreadControler:public Ego::MouseControler
{
public:
    ///判断是否可以处理当前鼠标事件
    bool canHandleEvent(const Ego::MouseEvent& event);
    ///对控制器进行初始化；
    void enter(const Ego::MouseEvent& event);
    ////处理当前鼠标事件；
    bool handleEvent(const Ego::MouseEvent& event);
private:
    int m_mouse_ldown_number;///鼠标左键按下的次数；
    Node* m_node;///被选中的节点；
    Wire* m_wire;///生成一条线的实例；
    Node* m_org_node;///指向导线的第一个节点指针；
    Node* m_dest_node;///指向导线末的节点指针；
    bool changexy;///是否要横线与竖线之间变化；
};
```

测试类实现如下：

```
class TestCConThreadControler
{

public:
    bool canHandleEvent(const Ego::MouseEvent& event)
    {
        cete->canHandleEvent(event);
    }

    void enter(const Ego::MouseEvent& event)
    {
        cete->enter();
    }

    bool handleEvent(const Ego::MouseEvent& event)
    {
        cete->handleEvent();
    }
```

```
private:
    CConThreadControler *pCtrl = new CConThreadControler;
};
```

3.6.3 集成测试

3.6.3.1 集成测试概念

集成测试是通过测试发现和接口有关的问题来构造程序结构的系统化技术，它的目标是把通过了单元测试的模块拿来，构造一个在设计中所描述的程序结构。集成测试是面向黑盒的。

3.6.3.2 OO 语境中的集成测试

因为面向对象软件没有层次的控制结构，传统的自顶向下和自底向上集成策略就没有意义，此外，一次集成一个操作到类中(传统的增量集成方法)经常是不可能的，这是由于“构成类的成分的直接和间接的交互”。

对 OO 软件的集成测试有两种不同策略，第一种称为基于线程的测试(thread—based testing)，集成对回应系统的一个输入或事件所需的一组类，每个线程被集成并分别测试，应用回归测试以保证没有产生副作用。第二种称为基于使用的测试(use—based testing)，通过测试那些几乎不使用服务器类的类(称为独立类)而开始构造系统，在独立类测试完成后，下一层的使用独立类的类，称为依赖类，被测试。这个依赖类层次的测试序列一直持续到构造完整个系统。序列和传统集成不同，使用驱动器和桩(stubs)作为替代操作是要尽可能避免的。

3.6.3.3 系统中的集成测试

面向对象系统的集成测试，通过测试用例进行指导。

OO 测试用例设计的整体方法如下：

1). 每个测试用例应该被唯一标识，并且和将被测试的类显式地相关联。

2). 应该陈述测试的目的。

3). 对每个测试应该开发一组测试步骤，应该包含：

a. 将被测试的对象的一组特定状态。

b. 将作为测试的结果使用的一组消息和操作。

c. 当测试对象时可能产生的一组例外。

d. 一组外部条件，即为了进行测试而必须存在的软件的外部环境的变化。

e. 辅助理解或实现测试的补充信息。

集成测试需要编制严谨的测试用例，以及测试用例进行测试。测试用例应尽可能合理的遍历系统的每一个功能点，并考虑到使用过程中可能出现的每一种可能的情况。以下为虚拟实验系统管理平台中“学生创建自主实验”功能的测试用例。

项目/软件	电路分析虚拟实验	模块名称	学生自主实验		
用例编号	UC030	编制人	陆 明	测试人	陆 明
相关用例	自主实验列表	编制时间	2004/09/07		
功能特性	学生创建自主实验				
测试目的	验证系统功能是否完整和正确				
预置条件	用户以学生身份成功登录	特殊规程说明			
参考信息	电路虚拟实验系统用例描述文件				
测试数据	设定学生能够创建自主实验最多为五个。				

步骤	操作描述	数据	期望结果	实际结果	测试状态（P/F）
1.	点击菜单“创建自主实验”菜单入口。		屏幕显示电路分析虚拟实验仿真平台。。	实现预期结果	p
2.	在仿真平台上搭建电路图。		仿真平台显示搭建的电路图，各种仿真器材工作正常，可以正确读取参数，与实际电路结果相同。	实现预期结果	p
3.	点击仿真平台下方的“保存”按钮。		保存搭建好的电路图，并返回到“自主实验列表”页面。刚刚创建的实验已经显示在列表当中，创建自主实验	实现预期结果	p

			成功。		

对于创建自主实验考虑到用户创建实验时种遇到的各种情形，由此确保创建自主实验功能的完备和正确。

系统的测试用例包括学生实验模块功能测试用例、实验管理模块功能测试用例、用户管理模块功能测试用例等。这些测试用例考虑了系统运行中出现的各种情况，并在不同的硬件环境、不同的操作系统下，不同的在线人数情况下进行了测试。测试结果表明我们开发的系统已经实现了开发计划中确定的所有功能，对测试中发现的问题进行修改和调整后，系统功能是完整和正确的。

3.6.4 压力测试

3.6.4.1 压力测试概念

在较早的软件测试步骤中，白盒和黑盒技术对正常的程序功能和性能进行了详尽的检查。压力测试(strees testing)的目的是要对付非正常的情形。在本质上说，进行压力测试的人应该这样问：“我们能够将系统折腾到什么程度而又不会出错？”

压力测试是在一种需要反常数量、频率或资源的方式下执行系统。从本质上来说，测试者是想要破坏程序。

3.6.4.2 仿真平台的压力测试

我们在测试阶段对仿真平台进行了压力测试，仿真平台上显示的所有器材，以及器材当前的状态，都是通过 XML 格式的配置文件存储和管理。所以对于仿真平台的压力测试可以通过配置文件进行观察。在进行压力测试时，我们给仿真平台添加了大量的器材，直到不能添加为止。这时，我们观察到配置文件的长度为500k，这也是配置文件的最大长度。

这个数值是具有指导意义的，系统中其它需要使用仿真平台用例的其它用例，在设置配置信息接口以及数据库设计中，可以根据这个值更为合理的编写程序和设计数据库。

第 4 章　论文总结

虚拟仿真实验系统真实的模拟了现实中的各种实验器材，其中许多器材是十分昂贵的。虚拟实验仿真系统与其他远程教学方式相辅相成，丰富了远程教育的形式，降低了实验的成本。同时使学生可以在进行真实的实验前，更深入的了解需要使用的器材，提高了学生的实验效率。

面向对象的分析和设计有很多优点，是现在广泛使用的软件开发技术。它是软件组件化成为可能，大大提高了软件的可重用性。本系统的开发采用面向对象的分析，面向对象的设计以及面向对象的编程，构建了一个符合需求的实验仿真系统。

UML 技术是当前最为流行的建模语言，对面向对象的系统开发具有很大的帮助。而且，该技术可以贯穿系统开发的各个阶段，其所提供的丰富的工具，统一的描述方式，降低了开发人员之间交流的成本，提高了系统开发的效率。

本论文着眼于 UML 技术在网络虚拟系统中的应用，介绍了本仿真系统开发的各个阶段的工作，面向对象的设计方法和 UML 建模技术在系统中的具体应用。

第 5 章 参考文献

1. 《UML 用户指南》 Grady Booch 等著，邵维忠等译。机械工业出版社，2002。

2. 《用例分析技术》 Geri Schneider, Jason P. Winters 著，姚淑珍 李巍等译。机械工业出版社，2002。

3. 《统一软件开发过程》Ivar Jacobson, Grady Booch, James Rumbaugh 著，周伯生 冯学民 樊东平译。机械工业出版社，2002。

4. 《设计模式—可复用面向对象软件的基础》 Erich Gamma 等著，李英军等译。机械工业出版社，2000。

5. 《企业应用架构模式》 Martin Fowler 著，王怀民 周斌译。机械工业出版社, 2004。

6. 《软件工程—实践者的研究方法》Roger S. Pressman 著，梅宏译。机械工业出版社，2002。

7. 《虚拟实验系统调研报告》虚拟实验系统项目组。北京邮电大学现代网络教育技术研所，2003。

8. 《虚拟实验系统需求说明》虚拟实验系统项目组。北京邮电大学现代网络教育技术研究所，2003。

9. 《虚拟实验系统研制报告》虚拟实验系统项目组。北京邮电大学现代网络教育技术研究所，2004。

致谢

衷心感谢导师勾学荣教授在我研究生学习阶段对我的帮助和教导。

在这篇论文的写作过程中，勾老师提出了很多宝贵意见，使我的论文从内容上更加丰富和完善，并进一步提高了论文的理论水平。两年来，通过向勾学荣教授请教，以及学习和实践的锻炼，使我获得了很多启示，专业知识和技能也得到了充实和提高。同时，勾学荣教授严谨的治学态度，渊博的学识，平易近人的作风，给我留下了深深的印象，是我今后工作和学习的榜样。

在此，我要感谢文福安教授、上官右黎教授，他们在我论文研究期间，给了我很多指导，帮助我提高了实践能力，养成了良好的软件开发习惯，使我受益非浅。

同时要感谢虚拟实验课题组的陈美松、温季睿、孙燕莲、魏代一、马书惠等同学，正是在他们的帮助下，在与他们的合作中，我完成了研究生阶段的项目工作。