分类号：　　单位代码：10019
密　级：　　学　　号：S08111418

中国农业大学

学位论文

交互式虚拟培训系统应用框架设计与实现

Design and Implementation of the Interactive Virtual Training System Application Framework

研　　究　　生：牛景波
指　导　教　师：朱虹　副教授
合 作 指 导 教 师：陈洪
申请学位门类级别：工学硕士
专　业　名　称：计算机科学与技术
研　究　方　向：虚拟现实
所　在　学　院：信息与电气工程学院

二零一零 年 十一 月

独 创 性 声 明

本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得中国农业大学或其它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。

研究生签名：　　时间：　　年　　月　　日

关于论文使用授权的说明

本人完全了解中国农业大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件和磁盘，允许论文被查阅和借阅，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。同意中国农业大学可以用不同方式在不同媒体上发表、传播学位论文的全部或部分内容。

(保密的学位论文在解密后应遵守此协议)

研究生签名：　　时间：　　年　　月　　日

导师签名：　　时间：　　年　　月　　日

摘要

虚拟培训作为一种理想的培训方式，目前已经被广泛应用于军事教学，体育训练，医疗实习，农业技能培训等各个领域中。现有虚拟培训系统由于使用底层渲染接口，系统针对性强导致复用性低。因此引入软件复用技术来开发具有较高复用性的虚拟培训系统应用框架是解决这个问题的一种有效途径。

本文首先研究框架开发方法，在总结现有方法的基础上，采用自顶向下的框架开发方法，本方法有利于框架的重构和演化，使设计出来的框架具有良好的复用性和扩充性。接着研究了开发虚拟培训系统中使用的设计模式，包括单件模式，访问者模式，命令模式等，并分析每种模式适用情况，结合虚拟培训框架符合 MVC 架构的特点，设计了视图、模型、控制模块相互分离的交互式框架，并实现了人物模块，组件模块，网络通信模块，背包系统，任务系统，UI 系统。其中组件模块通过抽象封装图形引擎的接口，实现了模型组件，动作组件，技能组件；UI 系统对 GUI 引擎接口进行封装，保证框架具有良好的扩展性。组件模块和人物模块使用访问者模式建立关联，使两模块独立地演化；网络通信模块使用 ACE 网络开发框架，网络命令的实现采用命令模式，有效解决网络消息管理混乱的问题。

最后通过开发一个基于该框架的典型应用，说明框架的开发流程，并证明框架具有较好的复用性。

关键词：框架，虚拟培训[1]，设计模式，ACE

Abstract

Virtual training as a kind of ideal training mode, currently has been widely used in military teaching, sports training, medical practice, agricultural skills training and other fields. However, the uses of underlying rendering interface of existing virtual training system have led to low reusability. Therefore the introduction of software reuse can provide an effective way to develop high reusability of virtual training system and solve the problem.

This paper studies the framework of development based on analysis of existing methods and proposes an top-down approach which can conducive to reconstruction and evolution of the framework and give a good solution for developing high reusability and scalability system.. The paper studied the design patterns used for development of virtual training system, including the singleton pattern, visitor pattern, command pattern, the application of each model. Combined with MVC architecture features, an interactive virtual training framework is designed in which the view, model, control module are separated from each other. Based on the framework, the character modules, components, modules, network communication module, backpack system, mission system and UI system are achieved. Moreover, the model components, animation, component, components, and other skills are implemented through the component module which encapsulates the abstract interface graphics engine. UI system completes the engine interface for GUI package and ensures that the framework has a good scalability. The component module is associated with the character modules through visitor pattern. They can evolve independently. In order to effectively solve the problem of chaotic management on network message, the ACE framework is used to implement the network command.

Finally, a typical application is given to explain the processes of development using framework and demonstrate its capability of reusability.

Keyword: Framework, Virtual training, Design pattern, ACE

目录

第一章 引言

1.1 研究背景

据统计，目前仅北京市的各类培训机构就达千家以上。但专家们说，70%的培训都是无效的，真正起作用的只有 30%，企业的培训费用很大一部分是在浪费！因此，未来培训技术的发展既有理论研究的价值，又有广泛的应用基础和巨大的市场[2]。

尽管近年来以网络培训为代表的现代培训技术取得了巨大成就，人们仍然在探求更高层次的培训方式。虚拟现实技术以其拥有的特征为人们提供认识自然的各种先进手段[3]。

虚拟培训是指利用虚拟现实技术生成实时的、具有三维信息的人工虚拟环境，学员通过运用某些设备接受和响应环境的各种感官刺激而进入其中，并可根据需要通过多种交互设备来驾驭环境、操作工具和操作对象，从而达到提高培训对象各种技能或学习知识的目的。

从本质上说，虚拟现实技术是人类认识自然、以自然方式与世界通信的一种工具，或者说虚拟现实是人与世界交互的计算机接口。虚拟现实技术的这些特性是开发虚拟培训系统[3]的核心技术和基础。

虚拟培训系统具有信息容量大、多向演示、模拟生动、身临其境等显著特征，这是有限空间、有限时间的其它传统培训方式无法比拟的。它可以非常方便地解决培训中的理论与实践问题，能瞬时反馈各种信息，有助于提高参训人员的概念化理解能力和综合技术水平，并检查培训成绩。对于同一项目可多次反复训练。虚拟培训的效益主要体现在提高教学效率、节约培训成本等方面。

在知识网上搜索“虚拟培训”文献多达 442 篇，从理论研究到实际应用，大部分集中在工业领域，如石油，航天，或者灾害逃生培训。虚拟培训所呈现出来的问题：

工业领域的应用没有框架支持，系统无法复用。 工业领域存在众多高危领域，需要熟练的操作，而新手操作易导致严重生产事故，虚拟培训最先应用于此。但是，系统在设计之初就没有考虑到复用和扩展，每个系统仅仅针对具体领域开发。比如油井操作培训[4]，车辆驾驶[5],坦克驾驶[6]。系统和行业结合紧密，让系统丧失了移植性，代码无法复用。一个成功的培训系统由于没有上升到框架，导致巨大的资源浪费。

1.2 国内外研究现状

1.2.1 国内研究现状

国内在虚拟现实，虚拟培训方面的研究起步较晚，大部分为实际应用开发，且主要集中在航天，工业，军事等领域，农业方面的应用较少。

常壮，邱金水，张秀山[7]利用虚拟现实技术开发了舰船虚拟消防培训系统，较好地解决了灾害模拟逼真度、仿真训练时效性等问题，为火灾模拟、消防训练、决策和评估提供了一种途

径。 其系统针对舰船设计，底层模型根据实体舰船构建，在舰船模拟上具有较高的逼真度，能够提升培训效果。其架构没有提供开放的接口，需要更换舰船模型都需要大量修改系统，根本无法迁移到其他交通工具的火灾训练上。

刘航，王春水，王积忠[8]利用视景仿真技术开发了某型装备的虚拟操作训练系统。其主要利用了 GL Studio， MultiGen Creator，VEGA 等模拟了装备的操作流程，瞄准镜绘制直接利用了 OpenGL。直接利用底层渲染接口，大大降低了系统复用性。

郭燕舞[9]将虚拟现实技术和神经外科的临床应用相结合，在临床手术的教学，培训中有重大意义。由于医学领域的专业性和复杂性，系统在迁移至医学领域其他科室时都极其困难。

战守义，郑宏，王涌天[6]将虚拟现实技术应用于坦克驾驶模拟训练中。通过研究新型坦克的驾驶规范，将其实现为三维培训系统，来对驾驶员进行培训，有效降低训练成本。系统中驾驶规范抽象程度不够，做不到不修改系统而迁移至其他车辆驾驶培训模拟。

刘刚[10] 将虚拟现实技术应用于培训集装箱起重机驾驶的仿真培训。同样存在系统无法复用的问题。

国内学者也将虚拟现实技术应在教育，农业等领域的应用做了探索性研究。

黄鑫[11]探索了虚拟现实技术在虚拟教学中的应用。将虚拟现实作为一种新型教育技术的手段，从教育技术学的角度进行了研究，提出了虚拟教学的概念。其主要对象是学生，是教育，不是技能培训。

胡林，周国民[12]对虚拟农业教育平台进行了研究，提出了虚拟农业教育软件的构架方法。其利用农业知识模型和知识库作为构架的一部分，很有意义。三维显示方面，对于农业环境的构建，过分依赖硬件设备，逼真度较高，却忽略了娱乐性，以及教育的侧重点问题。

张孝飞，张振国，刘星何[13] 从虚拟动物，虚拟植物，虚拟仪器等几个角度论述了虚拟现实技术在农业职业教育实验中的作用，可惜的是没有实现成相应的系统。

彭辉[14]；杨沛[15]；何喜玲，王俊[16]；探讨了虚拟现实技术在农业领域中可能存在的应用。

张杜鹃，杨安祺，单振芳[17] 将无线网技术和虚拟现实技术相结合，应用于构建虚拟农业。很具有前瞻性，详细讨论了技术指标，适用于经济发达地区的现代化温室农业。

1.2.2 国外研究现状

虚拟培训作为一种理想的培训方式，目前国外已经被广泛应用在军事教学、体育训练、医疗实习、农业技能培训和一些普通中学的实际教学中。美国是虚拟现实技术的发祥地，大多数研究机构都在美国，美国虚拟现实的研究水平基本代表了国际的发展趋势，而且美国也是最早将虚拟现实技术运用于教育培训的国家。

1992 年，在美国的东卡罗林纳大学成立了虚拟现实技术与教育实验室（VREL）。该实验室的目的是确认虚拟现实技术在教育方面的合适应用，评价虚拟现实的硬件和软件，考察虚拟显示技术对教育的影响以及其在真实世界中的应用,把虚拟现实与其他教育媒体的效果进行比较。

SimuLearn[18]在 2006 推出一款韩语版的 Virtual Korean Leader。通过三维虚拟环境来培训大企业员工的领导能力。三星公司采用了这一系统。

ForgeFX[19][20]其总部位于加州旧金山，是一个仿真和实时三维软件开发公司。ForgeFX 提供航空航天，生猪养殖，农民索赔等方面的培训软件。2009 年推出生猪养殖培训软件，利用虚拟现实技术营造了三维可视化的场景，对生猪的健康养殖，疾病控制提供了整套的培训，还提供了在线交流和积分系统提高系统的互动参与性。 在虚拟培训方面 Canon 和 Cisco 也在进行相关的研发。

1.3 研究目标与内容

本文将研究软件复用技术、设计模式、框架开发方法等技术，针对虚拟培训系统复用性差的问题，利用这些技术开发一个具有良好复用性的虚拟培训实框架。主要研究内容包括：

（1） 确定适合虚拟培训的框架开发方法，在保证复用性的同时提高开发效率。

（2） 利用设计模式等技术降低框架模块间的耦合度。

（3） 设计并实现框架的核心组件。

（4） 利用框架开发实例，验证框架的复用性。

1.4 论文章节安排

第一章介绍本文的背景，引出虚拟培训系统中存在复用性差等问题，随后介绍了国外虚拟培训的研究现状，提出了自己的研究目标和内容。

第二章介绍常用的框架开发方法，以及框架带来的高复用性，本文选用了自顶向下框架开发方法。并详细介绍了框架开发中常用的设计模式。

第三章是本文的核心，详细阐述培训框架的设计与实现。利用软件复用技术和设计模式，设计并实现框架中的世界框架，背包系统，人物系统，组件系统等关键模块。

第四章利用框架实现了一个简单的培训系统，验证了框架，证明利用框架可以明显降低开发难度，提高开发效率。

第五章为本文的结论部分，总结了文章的工作，指出了不足和待完善之处。

第二章 设计模式与框架开发方法

2.1 软件复用技术概述

软件复用[21]是在软件开发中避免重复劳动的解决方案，开发应用系统就不需采用一切“从头开始”的模式，而是以已有工作为基础，充分利用过去应用系统开发中积累的经验和代码，从而将精力集中在系统逻辑的构成上。

软件复用以复用为目的[22]。它是一种系统化的方法，是为了复用而进行的设计，为了复用而进行的开发。并需要有效组织和管理这些可复用的资源，方便使用者查询和使用。可复用现有软件资源是软件复用技术的核心[23]。

拷贝、粘贴、函数、方法、模块、系统函数库、通用类库等复用方式被认为是在软件复用发展初级阶段的表现形式，因为这几种复用方式局限于对代码的复用，而编码工作仅仅是软件生产中的一个环节，局限于代码复用充其量只能改进编码工作，而不能改进整个软件生产过程[24]。相比之下，设计复用[25]比代码复用有更大的优势，应用范围也更加普遍；并且，它在开发初期就被引入，对项目影响更大。众所周知，系统的编码仅仅占系统开发工作的 20%左右，而系统的分析、设计以及维护的工作量占总开发工作量的 70%左右[26]。

系统的设计凝结了系统架构师、领域专家的智慧和心血，是系统的灵魂所在，决定了一个系统的优劣程度和各种质量特性。

因此以下重点研究设计阶段的软件复用技术。一般这类技术主要包括：软件构件技术、领域工程、软件构架、软件在工程、开放系统、设计模式等[27][28]。这些因素相互联系，共同影响了软件复用的实现。

本文主要使用了其中的设计模式和框架开发方法，下面重点研究。

2.2 框架的定义

框架（Framework）是整个或部分系统的可重用设计[29]，表现为一组抽象构件及构件实例间交互的方法;另一种定义认为，框架是可被应用开发者定制的应用骨架。前者是从应用方面而后者是从目的方面给出的定义。

可以说，一个框架是一个可复用的设计构件，它规定了应用的体系结构，阐明了整个设计、协作构件之间的依赖关系、责任分配和控制流程，表现为一组抽象类以及其实例之间协作的方法，它为构件复用提供了上下文(Context)关系[30]。因此构件库的大规模重用也需要框架。

构件领域框架方法在很大程度上借鉴了硬件技术发展的成就，它是构件技术、软件体系结构研究和应用软件开发三者发展结合的产物。在很多情况下，框架通常以构件库的形式出现，但构件库只是框架的一个重要部分。框架的关键还在于框架内对象之间的交互模式和控制流模式。

框架比构件可定制性强。在某种程度上，将构件和框架看成两个不同但彼此协作的技术或许更好。框架为构件提供重用的环境，为构件处理错误、交换数据及激活操作提供了标准的方法。

应用框架[31]的概念也很简单。它并不是包含构件应用程序的小片程序，而是实现了应用领域通用完备功能（除去特殊应用的部分）的底层服务。使用这种框架的编程人员可以在一个通用功能已经实现的基础上开始具体的系统开发。 框架提供了所有应用期望的默认行为的类集合。具体的应用通过重写子类(该子类属于框架的默认行为)或组装对象来支持应用专用的行为。

应用框架强调的是软件的设计重用性和系统的可扩充性,以缩短大型应用软件系统的开发周期，提高开发质量，与传统的基于类库的面向对象重用技术比较，应用框架更注重于面向专业领域的软件重用。应用框架具有领域相关性，构件仅根据框架进行复合而生成可运行的系统。框架的粒度越大，其中包含的领域知识就更加完整。

2.2.1 框架的重要性

因为软件系统发展到今天已经很复杂了，特别是服务器端软件，涉及到的知识，内容，问题太多。在某些方面使用别人成熟的框架，就相当于让别人帮你完成一些基础工作，你只需要集中精力完成系统的业务逻辑设计。而且框架一般是成熟，稳健的，他可以处理系统很多细节问题，比如，事物处理，安全性，数据流控制等问题。还有框架一般都经过很多人使用，所以结构很好，所以扩展性也很好，而且它是不断升级的，你可以直接享受别人升级代码带来的好处。

框架一般处在低层应用平台（如 J2EE）和高层业务逻辑之间的中间层。

软件分层是为了实现“高内聚、低耦合”。把问题划分开来各个解决，易于控制，易于延展，易于分配资源，总之好处很多。

框架和设计模式

框架、设计模式这两个概念总容易被混淆，其实它们之间还是有区别的[32]。构件通常是代码重用，而设计模式是设计重用，框架则介于两者之间，部分代码重用，部分设计重用，有时分析也可重用。在软件生产中有三种级别的重用：内部重用，即在同一应用中能公共使用的抽象块;代码重用，即将通用模块组合成库或工具集，以便在多个应用和领域都能使用；应用框架的重用，即为专用领域提供通用的或现成的基础结构，以获得最高级别的重用性。

框架与设计模式虽然相似，但却有着根本的不同。设计模式是对在某种环境中反复出现的问题以及解决该问题的方案的描述，它比框架更抽象；框架可以用代码表示，也能直接执行或复用，对模式而言只有实例才能用代码表示;设计模式是比框架更小的元素，一个框架中往往含有一个或多个设计模式，框架总是针对某一特定应用领域，但同一模式却可适用于各种应用。可以说，框架是软件，而设计模式是软件的知识。

框架的优势

框架的最大好处就是重用。面向对象系统获得的最大的复用方式就是框架，一个大的应用系统往往可能由多层互相协作的框架组成[33]。

由于框架能重用代码，因此从已有构件库中建立应用变得非常容易，因为构件都采用框架

统一定义的接口，从而使构件间的通信简单。

框架能重用设计。它提供可重用的抽象算法及高层设计，并能将大系统分解成更小的构件，而且能描述构件间的内部接口。这些标准接口使在已有的构件基础上通过组装建立各种各样的系统成为可能。只要符合接口定义，新的构件就能插入框架中，构件设计者就能重用构架的设计。

框架还能重用分析。所有的人员若按照框架的思想来分析事务，那么就能将它划分为同样的构件，采用相似的解决方法，从而使采用同一框架的分析人员之间能进行沟通[34]。

主要特点：

1) 领域内的软件结构一致性好； 建立更加开放的系统；
2) 重用代码大大增加，软件生产效率和质量也得到了提高；
3) 软件设计人员要专注于了解相关领域，使需求分析更充分；
4) 存储经验，可以让那些经验丰富的人员去设计框架和领域构件，而不必限于低层次的编程；
5) 允许采用快速原型技术；
6) 有利于在一个项目内多人协同工作；
7) 大粒度的重用使得平均开发费用降低，开发速度加快，开发人员减少，维护费用降低，而参数化框架使得适应性、灵活性增强。

2.2.2 框架开发方法

框架研究者们已经提出了一些框架开发方法。这些方法在它们各自的应用范围内是比较成熟的，为框架的开发方法的研究打下了坚实的基础。主要有以下几种常用的开发方法[35][36]。

1) 基于应用经验的框架开发方法

首先开发出几个问题领域的应用系统。识别出应用系统中的共性内容，将它们抽象出来作为该领域框架的内容。为了评估所抽取的内容是否适当，使用该框架重新开发这些应用系统。根据在开发中获得的经验改进最初开发的框架，达到新的改进版本的框架。如此迭代下去，直到框架的设计趋于成熟。这种开发过程能够区分框架的公共性和可变性，因而能够识别出框架的热点。但是，由于这种开发过程依赖于具体的应用，因而不能保证框架的热点能够真正满足用户的需要。

2) 基于逐步归纳的框架开发方法

这种开发过程包括两个步骤。第一阶段，叫做问题概括。它首先从框架应用领域中的一个有代表性的应用系统开始,通过阶梯式的步骤不断地将这种功能说明归纳为更为一般化的形式，直到达到对问题描述的最为一般化的形式。第二阶段，叫做框架设计，使用第一步得到的对问题领域不同抽象层次的说明，通过逆向顺序逐步实现这些层次。框架设计阶段的最后一步是将得到的框架应用于问题概括阶段的最初的应用系统中，这可以看作框架的测试活动。这种开发过程缺乏对问题概括阶段的指南；另外，开发出来的框架的可复用程度依赖于最初选择的应用能在多大程度上代表领域。

3)基于领域分析的框架开发方法

首先分析框架所应用的问题领域(一般需要分析领域中存在的应用系统)，理解并标识领域中的抽象概念和关键内容。领域中的重要概念被确定之后，开发框架并应用于某个测试系统中。如果发现问题，则修改框架。然后利用框架开发下一个系统。修正新发现的问题并对第一个系统进行更新。通过领域分析确定领域的共性，我们能够提取出定义良好的抽象概念，这对把握框架的整体特征是大有好处的。但是，进行领域分析需要投入大量的时间和预算。

从总体上讲，框架的开发过程可以分为自顶向下和自底向上两种。上述的三种框架开发方法中，前面两种应归属到自底向上的类别中，而后一种则归属到自顶向下的类别中。

本文将采用自顶向下的领域分析方法开发本框架，通过对这些系统的分析和抽象，设计一个具有较高复用性的交互式虚拟培训的通用框架。

2.2.3 实现交互式培训框架的难点

三维可视化技术最成功的应用是在游戏行业，本文将借鉴游戏行业中的框架设计的方法，来完成农民培训领域的框架。

需求变动性是实现框架的最大困难，不能以传统的标准软件工程方法对游戏进行建模。但游戏的框架具有某种程度上的模式可以借鉴。游戏的框架应该是一类游戏的框架而不是所有游戏，框架领域企图达到“万能”类的想法最终必然失败。因此本文专注于虚拟培训领域。

三维可视化应用软件需要用到底层渲染接口，要完成对这些接口的封装本身就是一个很大的工程。从底层开发根本不可能，并且容易硬编码，造成耦合度高，代码无法重用等问题。幸运的是开源的游戏引擎已经完成了对渲染层的封装，框架只需对部分图形引擎的接口进行抽象。然而存在大量的图形渲染引擎，如何抽象接口，抽象到何种程度，成为一个难点。

虚拟培训领域还处于探索阶段，本文将借鉴其他领域的经验克服上述问题和难点。

2.3 设计模式

设计模式[37]指的是对一些某个领域中的公共设计问题行之有效的设计结构和设计经验的系统抽象、明确预定义。其目的是复用那些成熟的设计技巧和设计经验，提高软件整体的灵活性，可复用性。 模式的思维源于 20 世纪 70 年代 Christopher Alexander 的一系列著作，这些著作中记录了模式在建筑方面的应用。而模式在软件界确立其地位的标志是 1995 年由 Erich Gamma, Richard Helm, Ralph Johnson 和 John Vlissides 四人合著的《Design Patterns Elements of Reusable Object Oriented Software》一书的出版，此书又称为 GOF 书，是第一次将设计模式提升到理论高度，并将之规范化。该书将设计模式的研究和推广推向了高潮，成为软件设计方法的经典书籍。

2.3.1 设计模式的描述

设计模式使人们可以更加简单方便地复用成功的设计和体系结构。将已证实的技术表述成设计模式也会使新系统开发者更加容易理解其设计思路。一般设计模式必须描述四个基本要素[38]：

1) 模式名称（pattern name）

一个助记名，它用一两个词来描述模式的问题、解决方案和效果。命名一个新的模式增加了我们的设计词汇。模式名可以帮助我们思考，便于我们与其他人交流设计思想及设计结果。找到恰当的模式名也是我们设计模式编目工作的难点之一。

2) 问题(problem)

描述了应该在何时使用模式。它解释了设计问题和问题存在的前因后果，它可能描述了特定的设计问题，如怎样用对象表示算法等。也可能描述了导致不灵活设计的类或对象结构。有时候，问题部分会包括使用模式必须满足的一系列先决条件。

3) 解决方案(solution)

描述了设计的组成成分，它们之间的相互关系及各自的职责和协作方式。因为模式就像一个模板，可应用于多种不同场合，所以解决方案并不描述一个特定而具体的设计或实现，而是提供设计问题的抽象描述和怎样用一个具有一般意义的元素组合（类或对象组合）来解决这个问题。

4) 效果(consequences)

描述了模式应用的效果及使用模式应权衡的问题。尽管我们描述设计决策时，并不总提到模式效果，但它们对于评价设计选择和理解使用模式的代价及好处具有重要意义。软件效果大多关注对时间和空间的衡量，它们也表述了语言和实现问题。因为复用是面向对象设计的要素之一，所以模式效果包括它对系统的灵活性、扩充性或可移植性的影响，显式地列出这些效果对理解和评价这些模式很有帮助。

2.3.2 设计模式的分类

在 GOF 书[39]中，根据两条准则对设计模式进行分类：目的准则和范围准则。目的准则，即设计模式是用来完成什么工作的。设计模式据此可分为；

1)创建型(Creational)设计模式：与类或对象的创建有关；

2)结构型(Structural)设计模式：处理类或对象的组合；

3)行为型(Behavioral)设计模式：对类或对象怎样交互和怎样分配责任进行描述；

范围准则，即指定设计模式主要是用于类还是用于对象。设计模式据此可分为：

①类设计模式：处理类和子类之间的关系，这些关系通过继承建立，是静态的，在编译时刻便确定下来了。

②对象设计模式：处理对象问的关系，这些关系在运行时刻可以改变，更具有动态性。

从某种意义上讲，几乎所有设计模式都使用继承机制，所以“类设计模式”是指那些集中于

处理类之间关系的设计模式，而大部分设计模式都属于对象设计模式的范畴。

2.3.3 设计模式的选择与使用

为了更加有效地使用设计模式，本文给出一些方法：

选择设计模式的关键在与鉴别某个设计模式能否解决你所需要解决的问题，需要考虑如下一些因素：

1)设计模式是怎样解决设计问题的。设计模式是怎样找到合适的对象。决定对象的粒度、指定对象的接口的。

2)某种设计模式的意图是否与你的问题相关。

3)模式是怎样互相关联的。研究设计模式的关系能指导你获得合适的模式或模式组。

4)目的相似的模式之间有什么共同点和不同点。

5)什么原因导致重新设计，哪些模式可以帮助你避免这些会导致重新设计的因素。

6)你的设计中哪些是可变的。考虑每一个设计模式允许哪些方面可以独立变化，你可以改变它们而不会导致重新设计。最重要的一点是封装变化的概念。

在游戏框架中常用的设计模式有单件，对象工厂、VISITOR、COMMAND[40]等模式。在本文中可以借鉴游戏框架中的设计方法，利用这些模式在规范层次接口、对象创建、对象管理以及相关功能调用上起到重要作用。

单件模式

对整个培训系统中只有一个实例的存在，惯例是使用单件模式，管理器就是这种模式的产物。该单件模式图描述了单件类（Singleton）的组成，其中包括实例接口 Instance（），单键操作接口 SingletonOperation()和单件数据提取接口 GetSingletonData()，实例句柄存储 uniqueInstance 和单件数据集合 singletonData。单件模式的静态类结构图如图 2-1 所示。

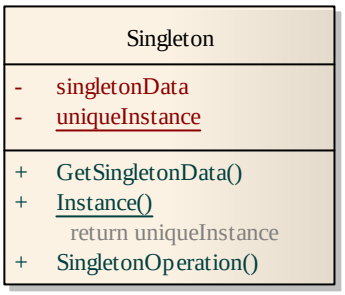

图 2-1 单件 UML 图:Singleton 模式的类图

Fig 2-1 Singleton UML

任何对单件模式的应用都是先通过 Instance（）获取实例后再进行类方法调用。Singleton 要求一个类有且仅有一个实例，并且提供全局访问点。

这就提出了一个问题：如何绕过常规的构造器，提供一种机制保证类仅有一个实例。客户程序在调用某一个类时，是不会考虑这个类有多少个实例等问题，这是类的设计者的责任，而不是类的使用者。

从另一个角度来说，Singleton 模式其实也是一种职责型模式。因为创建了一个对象，这个对象扮演了独一无二的角色，在这个单独的对象实例中，它集中了所属类的权力，同时肩负行使这种权力的责任。实现的时候采用静态变量是另一种简便的方法。

对象工厂模式

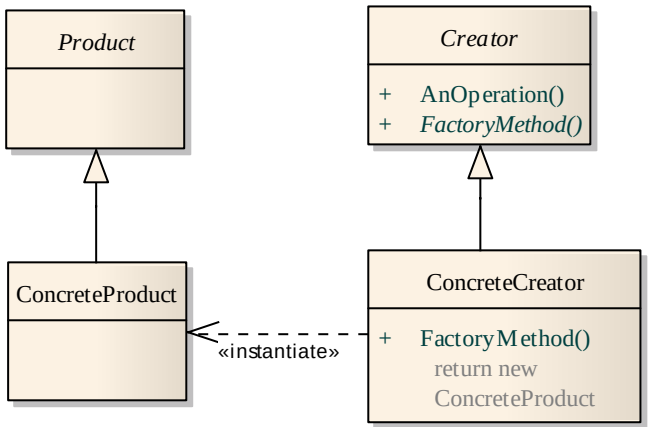

图 2-2 对象工厂 UML 图：对象工厂与被生产对象的协作方式描述

Fig 2-2 Object Factory UML

图 2-2 说明了对象工厂模式的静态类结构图，对象工厂用于创建具有灵活对象层次的实例。Product 是想要创建的对象类型，他包含了自己的操作，ConcreteCreator 是从 Creator 中派生的类型，ConcreteCreator 是用户自己的对象创建器，Creator 则负有创建 Product 的责任，它便是对象工厂通过对 CreateProduct 的调用，它创建了 ConcreteProduct 的实例，而每一个 ConcreteProduct 必须注册自己的对象类型的创建函数到 Creator 中去。

在虚拟培训系统中，经常会遇到一系列相互依赖的对象的创建工作。同时由于需求的变化，往往需要创建更多类型的对象。如何应对这种变化呢？如何才能替换传统的对象创建方法，提供一种封装机制来避免程序和“多系列对象的创建工作”的紧密耦合，这就是图 2-2 所示的对象工场模式。

以下情况应该考虑对象工厂模式：培训系统不应该依赖于产品类实例如何创建、组合和表达的细节，这对所有形态的工厂模式都是重要的。虚拟培训系统有多个类族，而系统只使用其中某一个类族。同属于一个类的对象在一起使用，这一约束在系统的设计中体现出来。虚拟培训系统提供一个产品类库，所有的产品以同样的接口出现，从而使客户端不依赖于实例。

访问者模式

访问者模式用于对象之间的信息通知[41]。Element 是需要被观察的对象模块，ConcreteElementA 和 ConcreteElementB 是从 Element 派生出来的类型，他们重载了 Accept 方法，该方法负责注册不同类型的 Visitor，Visitor 本身派生出不同的 ConcreteVisitor，ObjectStructure 引用 Element，同时为 Element 加载 Visitor。访问者模式的静态类结构图如图 2-3 所示。

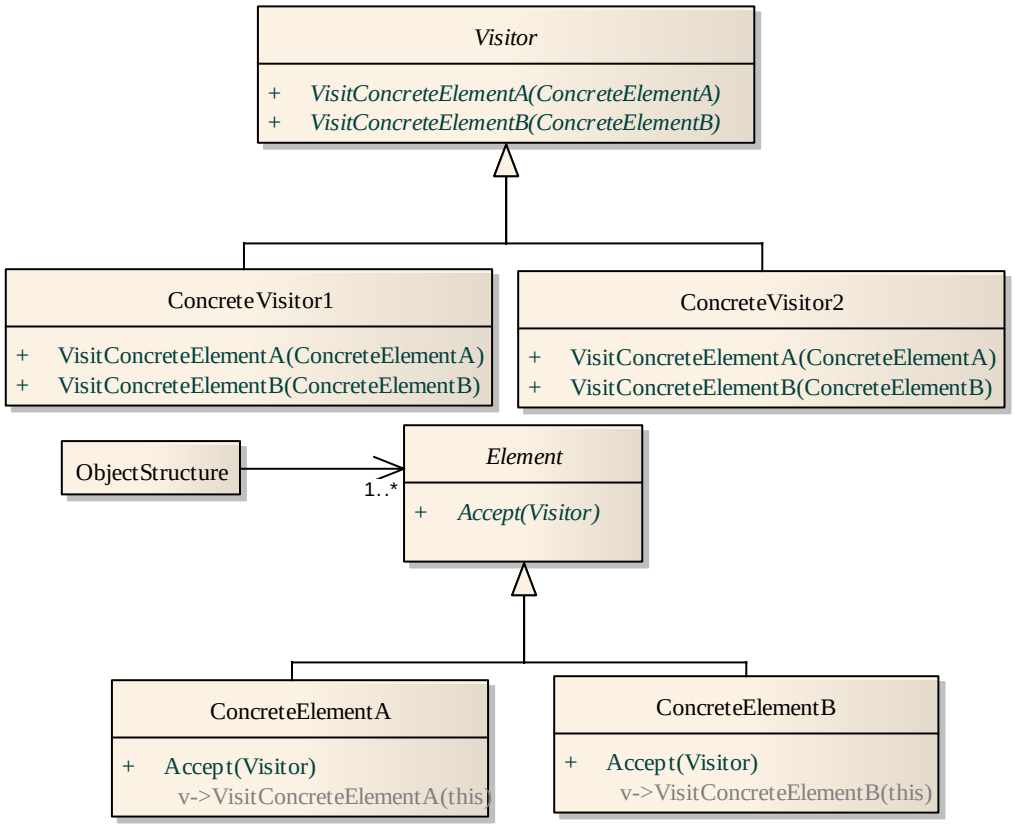

图 2-3 访问者模式 UML 图：描述访问者模式的基本框架以及交互方式

Fig 2-3 Visitor UML

在虚拟培训中构建逻辑框架时，需要为某些对象建立一种“通知依赖关系”，一个对象状态发生改变，所有依赖对象（观察着对象）都将得到通知。如果这样的依赖过于紧密，将来软件不能很好地抵御变化。使用此模式，可以将这种依赖关系弱化，形成一种稳定的依赖关系。从而实现软件体系结构的松耦合。

适用性： 当一个抽象模型有两个方面，其中一个依赖另一个方面。将这两者封闭在独立的对象中以使他们各自独立地改变和复用。当对一个对象的改变需要同时改变其他对象，而不知道具体有多少对象有待改变。当一个对象必须通知其他对象，而它又不能假定其他对象是谁。换言之，不希望这些对象是紧密耦合的。

通过 Visitor 模式，把一对多的对象之间的通知依赖关系的变化变得更为松散，大大地提高了程序的可维护性和扩展性，也很好地符合了开放-封闭原则[39]。

命令模式

Command 模式提供了很好的过程框架。Command 定义了一个 Excute 方法的框架，派生自 Command 的具体 ConcreteCommand 实现 Excute 方法，实际的执行是通过对 Command 的 Excute 的方法进行动态绑定而执行 ConcreteCommand 的方法。在主程序中，是通过 Command 的对象指针调用 Excute，所以 ConcreteCommand 无法绕过这一限制，这就体现了过程框架的意义。

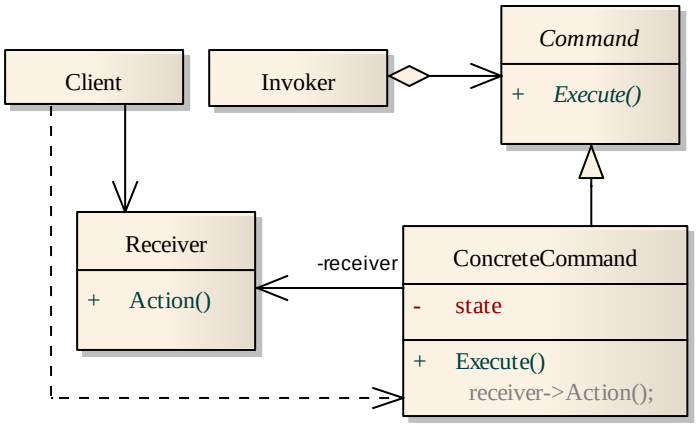

图 2-4 Command UML：描述命令框架类之间的交互

Fig 2-4 Command UML

图 2-4 是命令模式的静态类结构图。在虚拟培训框架中，“行为请求者”与“行为实施者”通常都是“紧耦合”的。但某些场合，比如要对行为进行“记录、 撤销/重做、事务”等处理，这种无法抵御变化的紧耦合是不适合的。在这种情况下，如何将“行为请求者”与“行为实现者”解耦。将一组行为抽象为对象，可以实现二者之间的松耦合，这就需要命令模式。

把一个请求封装成一个对象，从而可以依据不同请求对客户进行参数化。对请求排队或者记录请求日志，以及支持可撤销的操作。

效果以及实现要点：Command 模式将“行为请求者”与“行为实现者”解耦，在面向对象语言中，常见的手段是“将行为抽象为对象”。实现 Command 接口的具体对象 ConcreteCommand 有时候根据需求可能会保存一些额外的状态信息。

通过使用 Composite 模式，可以将多个命令封装成一个“复合命令”。Command 模式与 C# 中的 Delegate 有些类似。可两者定义行为的接口规范有所区别：Command 以面向对象中的“接口-实现”来定义行为接口规范，更严格，更符合抽象原则；Delagate 以函数签名来定义行为接口规范，更灵活，但抽象能力较弱。

使用命令模式会导致某些系统有过多的具体命令类。某些系统可能需要几十个，几百个甚至几千个具体命令类，这会使命令模式在这样的系统里变得不实际。

适用性：使用命令模式是作为“Callback”在面向对象系统中的替代。“Callback”讲的便是先将一个函数登记上，然后再以后调用此函数。需要在不同的时间指定请求、将请求排队。一个命令对象和以前的请求发出者可以有不同的生命周期。换言之，原先在请求发出这可能已经不在了，而命令对象本身仍然是活动的。这时命令的接受者可以在本地，也可以在网络的另一个地址。命令对象可以在串行化之后传送到另一台机器上去。系统需要支持命令的撤销。命令对象可以把状态存储起来，等到客户端需要撤销命令所产生的效果时，可以调用 undo（）方法，把命令所产生的效果撤销掉。命令对象还可以提供 redo（）方法，以供客户端在需要时，重新实施命令效果。如果一个系统要将系统中所有的数据更新到日志里，以便在系统崩溃时，可以根据日志里读回所有的数据更新命令， 重新调用 Excute 方法一条一条执行这些命令，从而恢复系统在崩溃前所做的数据更新。

2.4 本章小结

本章研究了软件复用技术，本文主要用到了框架开发和设计模式等技术。然后阐述了框架的定义，其是整个或者部分系统的重用。为了达到大粒度重用代码，设计之初就考虑使用软件复用技术，设计模式就是某领域中公共问题行之有效的解决方法，在框架设计中被大量采用。本文详细描述了设计模式使用的准则，并介绍了最常用几种模式，如单件模式，工厂模式。采用设计模式的目的就是在设计时就注重代码重用。

第三章 虚拟培训框架设计与实现

3.1 培训框架概述

三维可视化框架不言而喻，首先需要考虑如何利用三维可视化技术。虚拟现实技术在航空航天和工业领域应用时大部分选用了 OpenGL 图形开发库，OpenGL 是工业图形开发标准，被广泛使用，缺点是 OpenGL 是底层图形接口，容易造成系统开发效率低下，无法重复使用等问题。现在开源社区提供了多款图形引擎，集成众多三维应用程序中常用的功能，力求提高开发效率和代码复用。本框架选用图形引擎作为三维可视化的基础组件。

虚拟培训是一个互动的过程，搭建好了三维场景，参与培训的工作者，不仅仅是漫游场景，更需要在这个具有沉浸性的环境中进行交互。三维场景中的模型可以做各种运动，发出声音，可是大部分信息无法有效通过三维模型形态的改变传递给参训人员，文字信息就能很好解决这个问题。实现三维立体的文字，需要耗费大量工作，并且提供的信息量有效，内容的编辑也异常麻烦。因此，需要在三维场景中显示二维的文字，这就需要用到 GUI。网上有大量的 GUI 引擎，一部分是开源社区提供的，另一部分是商业产品，需要购买。本框架选用 GUI 引擎作为交互基础。

本框架设计的目的之一就是整合零散分布的农业资源，其中农业科学模型可以为培训提供精确的作物，家畜的生长模型，以及大量的农业气象、水文数据。在培训中，模型和数据是最为重要的资源，框架需要提供一个模型系统来方便这些资源的迁移。

仅仅有大量的农业模型，漂亮的三维场景依然不能构成一个引人入胜的培训。传统培训就已经暴露了单调、枯燥等问题，虚拟培训必须找到解决方法，这也是本框架的目的之一。游戏被公认为具有较高的娱乐性，其丰富而强大的任务系统，华丽的技能系统，以及在线交流系统等都有效地提高娱乐性。为了提高所开发系统的娱乐性，本框架将加入游戏中常用的系统。

图形引擎，GUI 引擎，农业模型，娱乐模块，再加上适当的逻辑控制就可以构成一个培训系统，而框架所做的就是建立这些模块之间的关联，然后把逻辑控制部分留给培训系统的开发人员。框架所用模块之间的关系非常符合 MVC 架构，结构如图 3-1 所示：

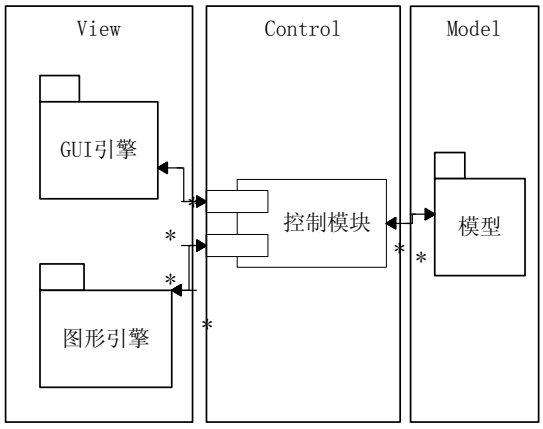

图 3-1 框架结构

Fig 3-1 Framework Structure

图形引擎和 GUI 引擎相当于视图(View)模块，培训模型和其他模型充当模型(Model)模块，最后的逻辑控制就是控制(Control)模块，三维可视化系统和 MVC 架构有原生的亲和性。视图(View)模块由外部程序负责，框架的工作就是提供底层的控制模块来控制视图模块的工作，模型模块需要兼容和扩展农业模型，可以利用继承树，基类与控制模块交互，子类扩展具体的功能。三维可视化程序是一个实时系统，系统的每个对象都需要不断更新，框架就需要在三个模块中都提供这种接口，才能保证系统的实时性。利用 C++的多态性，框架在底层设计好接口，应用层则具体实现，整个框架就可以自上而下统一起来。

视图模块的具体功能由外部程序实现后，框架的任务就集中在模型和控制模块上。控制模块需要“控制”模型，这种方法会导致控制模块权力太大，一方面封装视图模块的接口，另一方面使用模型模块。其实控制模块以组件形式提供服务，模型模块来调用服务，一方面可以降低两个模块之间的耦合度，另一方面控制模块就不会太庞大。这种情况符合 Visitor 模式描述的情况，Control 模块作为 Visitor，Model 模块是 Element，这两大模块就可以轻松解耦，相对独立地变化。Model 模块将来会有大量的对象，逐个对象的及时更新，非常困难，框架借用游戏中的管理器思想，为 Model 模块设计管理器，及时更新工作由管理器负责。实质上，依然是控制模块控制了整个框架，由于增加了网络功能，控制模块得到了扩展，一部分控制信息来自服务器端。框架结构进一步细化后，如图 3-2 所示。

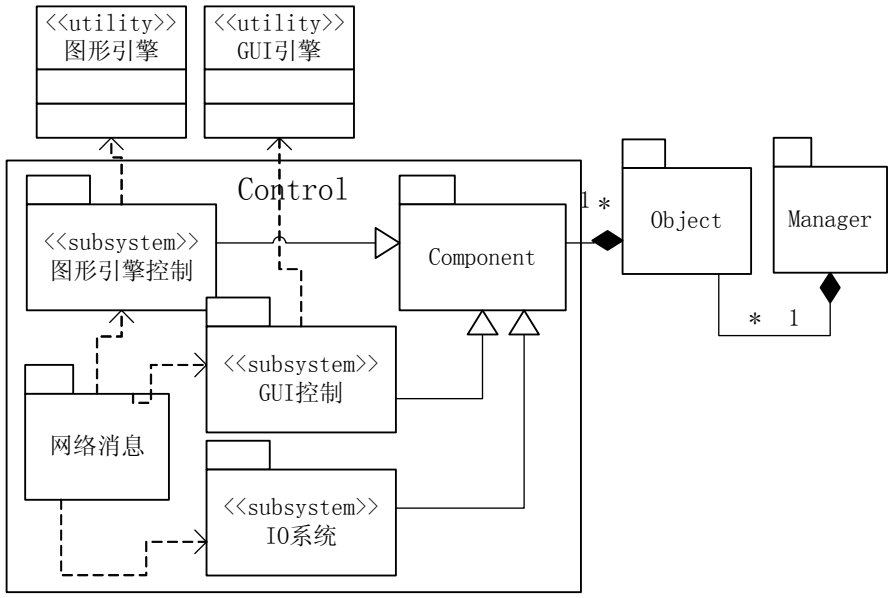

图 3-2 框架细化结构

Fig 3-2 Framework Structure details

管理器 Manager 会提供及时更新接口 AdvanceTime，Object 实现此接口，然后由管理器统一调用。Control 模块的及时更新接口有基类 Component 提供，子组件类实现。控制部分以组件的形式内聚于 Object，而 Object 又内聚于 Manager。管理器掌握了框架中所有的类，初始化时只需调用 Manager，然后逐级执行即可。I/O 系统主要考虑外设，如键盘鼠标等的控制和相应。

考虑到联网功能，增加了网络消息模块，其主要用于接收、发送控制消息。控制模块需要大量接口来控制其他模块，而网络之间的控制，只能传递数据。因此必须将控制模块的操作和数据分离，才能符合网络传输的要求。

Control 模块和 Model 模块已经完成了框架的核心，这两个模块之间使用 Visitor 模式建立联系，Control 模块的基类“组件”聚合到 Model 模块的基类 Thing 中，同时组件保留 Thing 的指针，Model 模块向自己内部的组件容器发送消息，经过解析后，相应的组件完成服务。

Model 模块需要解决两个问题，第一负责与组件的交互，即完成 Visitor 模式，并提供一系列接口；第二负责抽象有关三维模型创建，控制的接口。Thing 作为 Model 的基类，解决了第一个问题，提供了丰富的标识变量和大量的虚函数接口。SceneRole 封装了与图形引擎相关的接口，组件模块中的模型、移动、技能、动作组件是这些接口的实际执行者。

这两个问题解决后，Model 模块的骨架就确定了，以后系统开发人员需要对 Model 模块添加新的属性，就需要继承图 3-3 黑框标注的类 SceneRole，新的功能就放在 Control 模块中，继承组件，利用多态性重新定义接口。最后将新组件注册到新添加的 model 中即可。框架各个模块间的关系如图 3-3 所示。

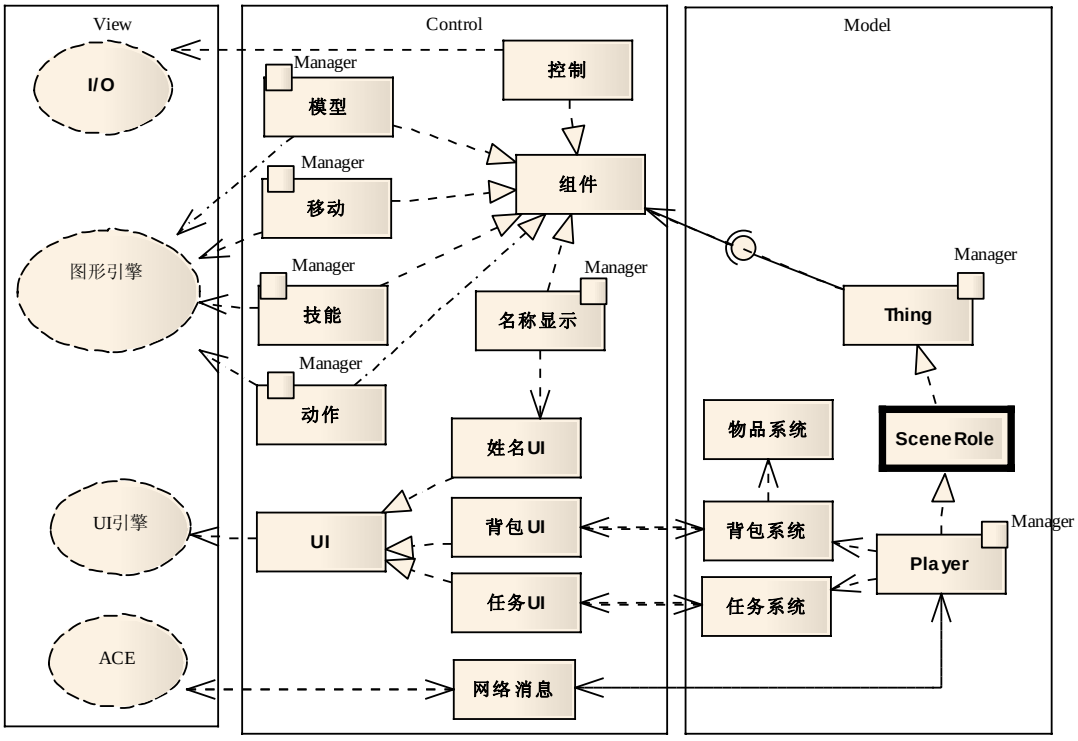

图 3-3 框架模块关系

Fig 3-3 Framework module relationship

框架中最重要的模块就是组件，Thing，UI 和网络消息了，组件和 Thing 在底层确定了关系；UI 是前端显示，需要文字信息提示的地方都需要用到；网络消息则是服务器用来驱动客户端的，其主要通过 Thing 的子类来控制组件，进而达到控制整个系统的目的。框架还有一个非常重要的模块，他无处不在，在图 3-3 上上角有标注的模块都提供了 Manager，这些模块实际运行时会有多个实例，如果没有 Manager，也需要开发人员自己提供控制部分。因此，框架索性提供 Manager，实例都需要注册，统一管理，开发中就可以忽略实例，而将更多的精力用在管理器接口的实现上。背包，物品和任务系统已经和 Player 建立了底层使用关系，但接口没有实现，上层逻辑需要系统开发人员来完成。框架个模块需要一个模块来负责加载和调用，见下文。

3.2 培训框架设计

虚拟世界是培训的主体，参训人员需要操纵“化身”在虚拟世界中完成相应的培训。传统的培训刻板、单调，参训人员只能硬性记下培训内容，这对知识水平本来就不高的参训人员来说，很容易造成知识的遗忘和培训效果的不理想。因此，本框架在培训中引入娱乐因素，增加培训的趣味性，后续开发人员可以营造一个寓教于乐的培训环境。

培训框架除了基本的世界框架、人物系统、物品系统、模型系统、背包系统外，还借鉴游戏框架的结构，增加任务系统、技能系统、动作系统、网络通信等框架模块[42]。来自游戏框架中的模块让框架更加完整，为系统开发提供更多功能。此外模块的设计以降低耦合，提高复用为目的。

3.2.1 虚拟世界框架结构

世界框架是整个框架的外层，本文设计的培训框架首先是一个应用程序，需要窗口化，并能进行三维可视化渲染。因此外层框架就要对基本的窗口操作，三维渲染的初始化部分进行封装。三维渲染部分有多款图形引擎可以采用，使用框架构建系统时有变动的可能。因此，窗口操作和三维渲染将设计成两个类，独立变化。

窗口类主要负责创建显示窗口，其中包括 windows 窗口创建的所有操作。图形引擎的初始化，框架主逻辑的加载都需要在这里完成。

设备类主要与图形引擎相关，需要对引擎中与底层设备相关的接口进行封装。主要负责调整渲染参数，设置视口大小，恢复设备等。

TrainApp 类就是窗口类，由于外层的 WinMain()函数，TrainApp 在初始化时，需要接收主函数的窗口实例 hinstance，即确定主函数和窗口类之间的关系。

DeviceEnv 类是设备类，其功能比较简单，主要是根据配置文件正确初始化渲染引擎。这两个类在框架中不允许有重复，都实现为单件模式，本文的单件没有采用继承的方式，而是采用宏来实现，实现部分会详细介绍。

3.2.2 任务系统

任务系统[43]是虚拟培训中必不可缺的一个部分，培训的绝大部分内容要以任务的形式体现出来，一个扩展性强，功能强大的任务系统可以提供丰富的培训内容。

通常，网络游戏会把任务系统的核心逻辑放在服务器端，这样可以增加作弊难度，保证游戏的公平性和长期收益，但增加了网络通信的负担和系统开发的难度。

使用虚拟培训框架开发的系统旨在培训参训人员获取特定的技能和知识，是公益性质，参加培训的人员不会出现作弊现象，也没有这个动机，所以本文不会采用网络游戏的做法。任务系统的核心逻辑将放在客户端，以任务文件的形式保存在本地。当有新的任务发布时，将通知客户端更新任务，这样参训人就可以及时获取最新的咨询，克服了传统培训中内容陈旧的问题。

任务系统是对培训中基本事件的一个描述，即在某处（人）获得一个任务，根据任务要求

去完成任务，形式有很多种，例如种植多少小麦，对小猪进行疾病预防等，完成之后要标记任务已完成。这是最基本的事件形式，复杂任务可以由多个简单任务组合而成。

根据上面分析，可以很快知道任务系统需要的数据，例如任务名称，从哪里接任务，任务需求，需求数量，实际完成数量，任务完成标志等，根据这些对任务系统进行详细设计。
类设计时，考虑到扩展性，舍弃了将所有任务内容集中在一个类中的做法，而是将其拆分成一个控制器 TaskManager 和 TaskData，控制器和任务数据分离，TaskData 存储上面提到的任务数据，TaskManager 封装对任务数据的操作，数据和操作可以独立变化。开发时只需定制系统的 TaskData 即可。

框架的主要逻辑都会使用任务系统，大量的 TaskManager 肯定会造成框架无法管理，必须提供一个唯一的接口来访问任务系统。因此，将 TaskManager 设计成单件模式。

静态结构如下：

TaskData：任务的数据。

eTaskState：任务的状态。

TaskManager：任务系统，任务的注册，更新和检测都由它来完成。和其他管理器相同。
任务系统的类静态结构如图 3-4 所示：

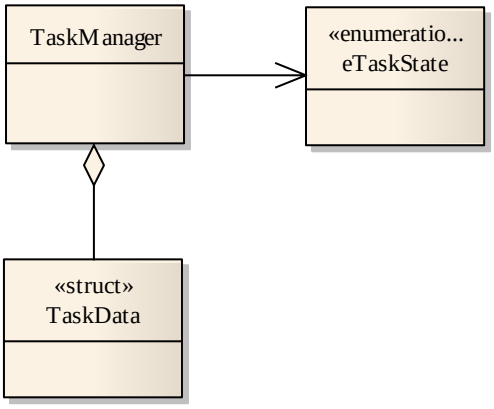

图 3-4 任务系统结构图

Fig 3-4 Task System

任务需要存储在本地，任务文件需要将 TaskData 中的数据保存下来，TaskManager 提供了这接口。

3.2.3 人物系统

人物系统在游戏中一般指玩家控制的角色，而培训框架中的人物系统更加广义，泛指一切需要逻辑控制的物体。参训人员在培训时可以有一个角色来增加亲和力，帮助完成一些操作，有些时候却不需要这样的主角。养殖动物或者田间管理时，主角的频繁移动反而会影响培训的流畅性，这时有逻辑驱动的动物或者植物就可以成为“主角”，农民只负责管理它们。

三维场景中的物体分为两类，一类是环境物品，比如树木，石头，流水等，其目的是装饰性的，增加场景的真实感；第二类是活动物体，即背后有相应的逻辑，比如 可以走动的“化身”，动物，会生长的植物，其他玩家等；第二类物体都可以纳入到人物系统中。

本节采用 Object+Components[43]的方式来设计，Object 负责提供一个基础框架，其他的功能由 Components 来提供，本节主要介绍 Object 的设计，Components 在组件模块会详细介绍。首先需要一个类来处理最底层的服务，与服务器通信，提供消息处理接口，三维场景中的 Object 就需要处理模型问题。底层服务和三维服务是最基本的服务，其他功能都是可以在此功能上进行扩展而实现。

静态结构如图 3-5 所示：

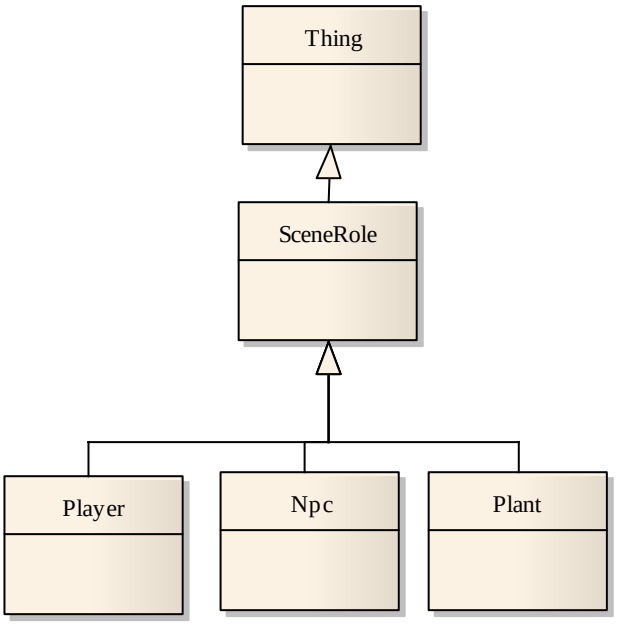

图 3-5 人物系统结构图

Fig 3-5 Player System

Thing： 基类，提供组件消息的处理接口。

SceneRole：场景角色，提供了 3D 模型管理的接口。

Player： 场景中的化身，可以使是其他在线玩家。

Plant：场景中需要逻辑驱动的作物。

Animal： 场景中需要逻辑驱动的家畜等。

MainPlayer：玩家自身，每个客户端只有一个此类的实例。

3.2.4 物品系统

网络游戏中，做任务、杀怪时会获得某种物品，其背后的逻辑控制就是由物品系统完成的。本文的培训框架中也要用到类似的系统，因为培训中需要的道具不比游戏中少，并且更加的专业化、科学化。

框架很庞大、复杂，物品系统只是其中一部分，关联部分还有 GUI 系统以及每个物品的定义，也就是物品属性，其中就包括了显示图标的名称。图标需要美工的协助，不在本文讨论范围。

物件是框架的主要组成部分，所有的物件继承于同一个基类，物件基类和派生类的主要功能是存储和区分自身所包含的各种属性信息。记录对自身包含的某类或某几类属性信息感兴趣的观察者，信息改变时进行通知。 物件类会包含一个或多个组件，用于对属性数据进行不同的

逻辑处理。 物件实体会有一个总的管理器，统一的 ID 标识。一些特定的物件的标识会被包含到特定的管理器中，用于逻辑控制。

物品系统就会携带更多的数据，最直接的方法就是设计一个物品管理器，然后将数据存放在硬盘上，加载框架时读入到内存中。最直接的设计不一定是最具复用性和扩展性的。上面的设计就存在扩展性不够的问题，将数据和方法放在一个类中管理将会是类肩负太多的责任，耦合度太高，提高框架的不稳定性。

在此将数据和方法拆分开来，设计成两个类，一个负责存储数据，另一个封装对数据的操作。

扩展时只用调整数据，或者重载处理函数即可，无需调整整个系统。为了方便管理，框架会提供一个单间模式的管理器。

背包系统与玩家密切相关，是玩家随身携带的物品。玩家获取的道具，物品都存储在背包之中。当然背包系统不是简单地提供一个物品的容器，还需要对物品进行管理，比如那些物品可以叠放在一起，如何拆分，使用物品等。

背包系统和物品系统相比，更加依赖 GUI，需要展示玩家拥有的所有物品，以及判断鼠标操作，相应键盘。设计中为背包系统提供统一的接口供其他模块调用。

背包物品系统静态结构如图 3-6 所示：

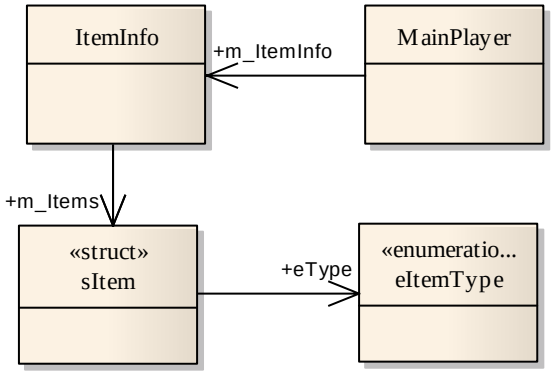

图 3-6 背包系统结构图

Fig 3-6 Package System

MainPlayer：玩家，聚合了一个背包

ItemInfo：背包 sItem： 物品

eItemType： 物品类型

3.2.5 组件模块

如果说人物系统是框架的骨架，组件模块就是框架的血肉，由于采用了 Object+Components 结构设计，Object 的功能完全被分离到了 Components 中，Object 只需负责向组件发送请求，组件就会完成相应的功能，组件部分完全独立了出来，能独立演化。

组件模块虽然独立了出来，和人物模块依然有密切的关系。组件模块顾名思义，其是人物模块的组件，人物系统的功能需要一个个组件来完成，最基本的三维模型创建需要模型组件，人物的动作需要动作组件的协助，甚至最基本的人物移动，键盘操控等，都有相应的组件。这

些功能的实现需要借助装饰模式和语言的多态，首先需要定义组件的基类，并确定与人物系统基类之间交互。组件不能脱离人物而单独存在，这样没有任何意义，必须内聚于人物系统中。内聚是一种单向关系，人物系统知道有相关组件，可组件依然不知道人物的存在。组件中需要一个指针来确保，组件知道自己是谁的组件。这就完成了两者之间的关系，组件就可以通过这个指针“指挥”人物活动起来。组件模块和人物系统的交互图如图 3-7 所示。

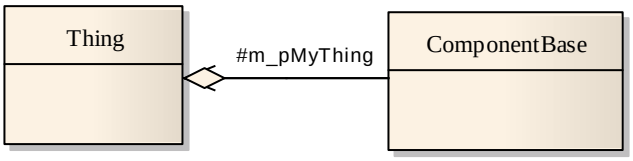

图 3-7 组件系统和人物系统结构关系

Fig 3-7 Component System

组件内部处理消息的函数依据具体项目的不同会有较大的变动，采用 visitor 模式将组件和数据分开，使得可以根据系统需求自由变动，增加新的消息处理函数将异常方便。

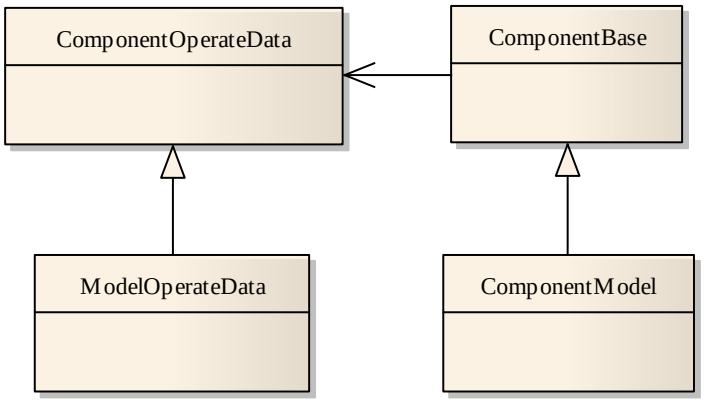

图 3-8 组件系统结构图

Fig 3-8 Component System detail

组件模块与组件数据之间的关系如图 3-8 所示。ComponentOperateData 是组件的操作数据，组件要完成的功能需要数据提供支持，设计的时候就考虑到了以后的变化和扩展，将每一个可能变化的部分都独立出来。考虑到组件模块是变化最大的，因为培训中需要农业科学模型的支持，这部分是非常复杂，每个模型都有不同的侧重点，基础参量也不尽相同，操作和数据分离后，可以方便模型的扩展。对于培训机构而言，此部分提供了更多扩展，可以让培训内容更加生动，有趣。

技能系统

技能系统按照常规的理解，只应该存在于游戏之中，和魔法战斗技能等联系在一起，和培训没什么关系。其实框架中只提供了基本操作，稍微复杂的操作或者新技术的动作规范等都需要通过技能系统来实现，并且技能系统是组件模块的一部分，具有良好的扩展性，提供配置文件，只需简单修改就可以完成复杂的功能。

技能系统在组建模块的架构下，提供丰富的扩展接口，降低使用难度。提供了配置文件，抽象出每个技能的属性。

模型系统

模型系统是三维虚拟程序中必不可少的部分，各种三维模型都需要通过模型系统加载到系统中。虚拟培训框架中需要加载的模型主要集中在人物系统部分，种类相对较少，比较符合简单工厂模式。

模型系统在内部封装游戏引擎或者图形引擎创建模型的接口，通过接受到的参数动态决定创建何种模型。

设计如下：

ComponentModel：提供创建模型的接口。

ModelOperateData：模型参数类，提供创建模型所需的各种参数。

动作系统

动作系统可以说是模型系统的附属系统，每个模型都有自己的独特的动作。考虑到动作文件的复用和 3D 美工的工作量，将动作部分从模型系统剥离，单独作为系统是必要的。

1 方便程序调试程序，即使发现问题；

2 将所有动作统一管理，提供统一的资源格式，有利于资源复用；

3 模型和动作部分和底层密切相关，动作系统充当了适配器的功能。

设计如下：

eActionPriority： 枚举类，枚举动作；

ActionOperateData：动作数据类，封装了动作常用的数据；

ComponentAction：封装了引擎创建动作的接口；

控制系统

框架需要对按键相应，游戏中通常会使用键盘上的 w，a，s，d 来控制角色，框架根据仿照游戏的做法设计了控制系统。系统主要的作用是检测键盘按键的状态，并据此来调用适当的接口。

首先需要一个结构来表示一个按键的组合，然后由管理器负责更新和维护这些注册的按键组合。

SRegKey： 组合键类，存储一组按键和对应的函数；

eSKeyState： 按键状态；

ComponentKey：按键的控制器，提供注册、销毁、载入、保存按键的接口；

移动系统

本框架的移动基于键盘操作，处理的原则是于给定时间为准则，通知其他客户端和服务器作为主角的客户端以何种方式走到当前的位置的，其他客户端按照这种方式移动到这个给定的位置上去。跳跃，骑乘，游泳等状态的改变也通过消息系统通知其他客户端[44]。

控制系统主要在客户端，而移动部分则需要和服务器交互，里面涉及到网络通信部分和协

议部分，放在后面章节。移动系统会提供一系列接口供其他模块和系统使用。

姓名显示系统

姓名显示系统[45]其实是 UI 系统的一部分，放在组件模块是因为每个玩家都要显示自己的姓名，关系紧密。

系统内部的逻辑由 UI 部分来控制，在此只提供基本的接口，比如设置姓名，删除等操作接口。

3.2.6 UI

UI 界面用于显示框架中的信息，并提供一些操作功能。UI 界面不会与框架的其他部分发生直接关系。当一个界面创建时会向他所关注的物件实体注册他感兴趣的数据，当相关数据发生改变时，物件实体会将数据通过消息接口发送到界面类中，界面根据数据进行自身所需的操作。

背包 UI

1. 背包格
   这里只能容纳背包，人物可以使用 12 个背包格子。
2. 物品格
   这里可以放任何物品，包括未装东西的背包。
   其它 UI 包括 生命条 ，体力条，名字，小地图等，不再详细介绍。

具体设计

背包实现概要说明:

在背包 UI 栏中每一个背包格子都是一个 BUTTON, 在 BUTTON 下挂背包 ICON,物品 ID。当物品的拖拽与背包发生关系时,需要注意的事项:

如果所拖拽的物品是进入背包，会发生如下几种情况：

1) 物品被拖入一个空的背包格子内，这种情况下可以直接将物品放入背包中；
2) 物品被拖入的是一个以经放有物品的格子，在这种情况下就需要做出如下判断：首先，被拖拽的物品是否可以重叠，如果可以，那么是否和目的地的物品相同，如果相同，是否达到重叠的最大上限，如果没有达到最大上限，物品成功放入背包。如果物品的类型不相同，则替换背包中目的地的物品，如果物品不可重叠，则替换背包中的物品。

在背包内部可对物品进行操作, 分三种情况:

1) 鼠标左键单击物品
2) 鼠标右键单击物品
3) 拖拽

物品的删除过程：

从被包中直接将物品拖出到其他无确定的目标出点击左键,会出现提示"是否销毁该物品"
在背包中装备物品或者使用物品:

在背包中点击鼠标右键, 会自动装备物品或者使用物品,

如果物品可以装备,或者使用。

管理器

一个无法管理的框架是难以使用的，因此，本框架为要为每类都设计一个对应的管理器，统一管理对象。管理器本身要避免数量过多造成无法管理的情况，设计时都将其设计为单件模式。管理器的职能：

1 用于对同一类型的物件进行统一的管理（创建，删除，逻辑循环，等）

2 用于对某些特定的功能提供统一的操作接口。比如场景的加载，特效的创建删除，网络模块的运转，UI 的管理和逻辑调用，姓名版渲染的收集等。

3 对于一些公共数据的保存。比如技能的数据信息，玩家的设置信息，基本动作名的信息，等。

管理器和数据之间的使用关系如图 3-9 所示：

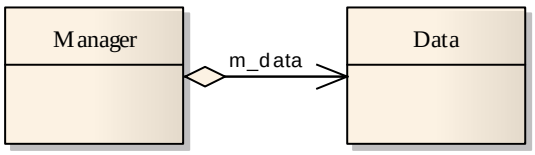

图 3-9 管理器基本结构图

Fig 3-9 Manager

管理器在内部提供一个容器,类似 vector 或者 map,将需要管理的对象 Data 放入容器。Vector 和 map 都提供了丰富的接口供管理器来查询，修改，删除这些对象。管理器还可以利用内存池技术，以后开发人员只需要从管理器中提取使用对象，而无需担心内存泄露等细节问题。

3.2.7 网络通信模块

网络化应用软件的开发比较昂贵、费时、而且容易出错。这些开销和困难源于软件开发过程中不断变化和增长的需求，以及整个软件行业对核心软件和实现制品的“持续的重新发现和发明”。而且，硬件构架的异种性、操作系统和网络平台的多样性，以及严酷的商业竞争使得从头开始构建网络化应用软件变的日益困难[48]。

ACE 针对网络化应用领域提供了系统化的复用框架。它使得网络化应用中类的关键角色和关系得以具体体现，从而增加了可复用的代码数量，减少了重写的代码量。

ACE 框架是半完成的应用。通过对框架中的类进行继承和实例化，开发者可以对这些“半完成”的应用进行定制，形成完整的应用。继承使得框架基类的特性能够被子类有选择地共享。如果基类提供了方法的缺省实现，对这些缺省行为不符合要求的虚方法，应用开发者只需要重定义（Override）这个方法就可以了。所以开发者能够进行大规模的软件复用，其规模超过了

复用独立的类和函数所能达到的规模。框架嫩巩固集成应用定义的类和不依赖于应用的类，从而增加复用的数量。使用 ACE 的好处包括[49]：

1） 增强可移植性： 在 ACE 组件的帮助下，很容易在一种 OS 平台上编写并发网络应用，然后快速地移植到其他各种 OS 平台上。

2） 更好的软件质量：ACE 的设计使用了许多提高软件质量的关键模式，这些质量因素包括通信软件灵活性、可扩展性、重用性和模块性。

3） 更高的效率和可预测性：ACE 经过仔细设计，支持广泛的应用服务质量（QoS）需求，包括延迟敏感应用的低响应等待时间、高带宽应用的高性能，以及实时应用的可预测性。

在 C/S 模式网络应用开发中，服务器端通信模块是开发中最为复杂的模块之一。服务器端要应对大量客户端连接请求及数据传输的处理，直接选用最基本的 I/O 模型管理这些 Socket 上的活动有很多局限。

服务器在设计之初就考虑到要被动地与客户端建立连接，连接过程采用异步 I/O 方式，当连接建立后，由接收进程负责处理数据接收和连接状态修改和消息执行等操作，还有多线程的管理，内存管理等都可以交给 ACE 来处理。

本框架主要使用 ACE 中的两个框架 Proactor 框架提供了前摄式的接收器-连接器模式[50]，他利用了 OS 对异步连接建立的支持。Task 框架提供了强大而可扩展的面向对象并发的功能，可应用在一些关键模式的开发中，在系统中负责应用层消息的处理。

本框架中的通信，主要指逻辑服务器与客户端、登录服务器与客户端之间的通信。本框架的主要目的是培训，在登录验证环节不做过多验证，降低使用门槛。本文采用的服务器模型如图 3-10 所示。

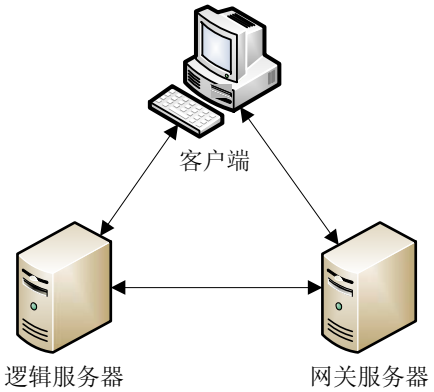

图 3-10 服务器通信结构

Fig 3-10 Server Connection Structure

在与客户端通信时，选用 ACE Proactor 框架中的 ACE_Asynch_Acceptor 模式。这样对服务器的请求和请求的处理就可以分开，并提供了对处理方式的定制能力。ACE_Asynch_Acceptor 在端口启动一个或多个异步接收请求，被动的接收客户端的连接。客户端通信类关系如图 3-11。

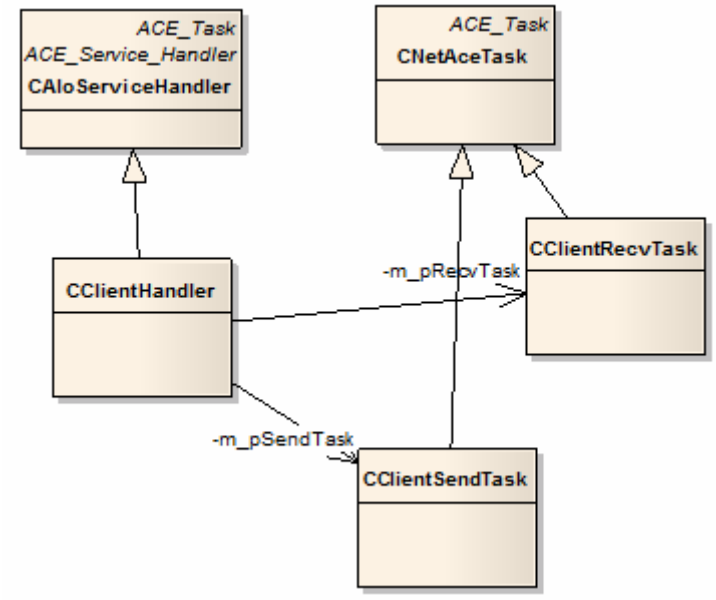

图 3-11 客户端通信结构图

Fig 3-11 Client Net Structure

CClientHandler 是一个客户端连接的抽象，其功能是由其基类 CAioServiceHandler 实现的。CAioServiceHandler 是异步通信接收器将要使用的服务处理器，用于把接收到的数据进行分析并封装要发送的数据。CClientRecvTask 是客户端的接收器，负责监听指定端口，主要功能有其父类 CNetAceTask 完成，CNetAceTask 是 ACE_Task 的子类。CClientSendTask 是客户端的发送器，负责将要发送的消息打包，并加入发送队列中。

逻辑服务器、网关服务器与客户端之间的通信主要有 CNetAceTask 来管理。实际上 CNetAceTask 是一个线程池，其继承自 ACE_Task，保证了其现成安全的消息处理能力。Activate() 方法由 ACE_Task 提供，它使用 thr_mgr()访问其所返回的 ACE_Thread_Manager 指针来派生一个或者多个运行在任务中的线程。这个方法将任务转换为主动对象，使其线程引导自身的执行，对事件作出响应，而不是借由调用者的线程被动执行。

为了在面向对象的程序中有效使用生产者/消费者并发模式，各个线程需要与消息队列及其他任何与服务有关的消息关联在一起。因此，为了保证模块性和内聚性，并降低耦合，最好能将一个 ACE_Message_Queue 和其他数据方法封装在一个其服务线程能直接访问的类中。NetMessage 使用 ACE_Thread_Manager 类来激活任务，这样，任务就会像一个主动给对象而运行，并调用响应的方法来驱动服务器和客户端之间的互动。

CNetAceTask 和 NetMessage 在上层定义了两者之间的交互接口，如要扩展消息任务只需在继承 NetMessage，添加新的消息即可。

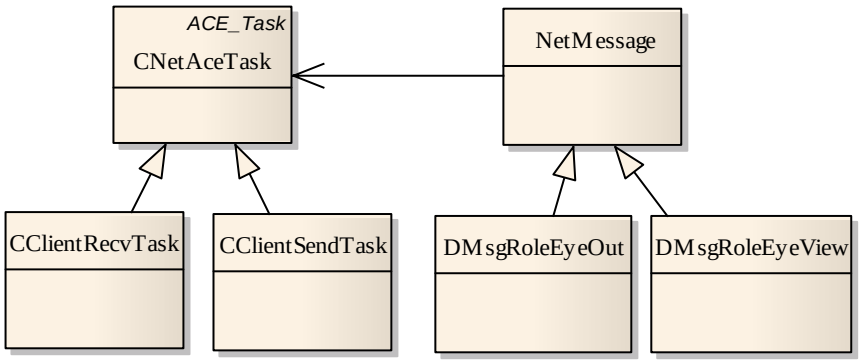

图 3-12 网络模块和消息体交互关系

Fig 3-12 Net and Message

框架通信模块与客户端的通信相关类如图 3-12 所示。

服务器端发送、接收模块与客户端类似，不在详细介绍。

3.2.8 场景管理设计与实现

服务器集中处理客户端发送的请求，同步各个在线的客户端，作用重大。虚拟培训中，服务器中的逻辑处理在客户端也进行了实现，即可以实现单机培训，也可以在线培训，更新培训任务等。逻辑处理部分见客户端部分，服务器部分不再介绍，此部分着重介绍服务器端最重要的模块-场景管理模块，其负责同步各个在线客户端，本框架并对同步模块进行了改进，可以有效降低网络负载。

玩家在线培训时，客户端和服务器需要通信，玩家的一举一动都要通过服务器同步到其他在线的客户端去，这一切都是由场景管理器实现的。

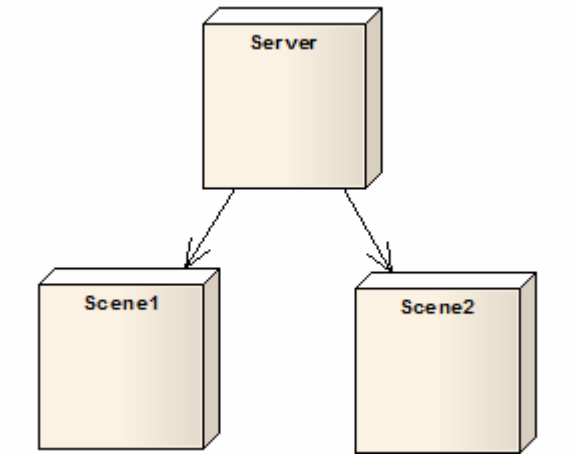

图 3-13 服务器端场景管理结构

Fig 3-13 Scene Structure

常用的服务器端场景管理模块结构如图 3-13 所示。客户机首先给服务器发送消息，服务器接到消息后判断消息的有效性，如果为无效消息则直接丢掉，有效的消息，根据类型对服务器上所有玩家进行广播。这种设计最直接，实现也最简单，但会浪费大量网络资源，在三维虚拟世界中，由于所有人并不是集中在一起，所以并不是每个人都应该得到收到广播。

本框架对上面的设计进行改进，改进后的场景结构如图 3-14 所示：

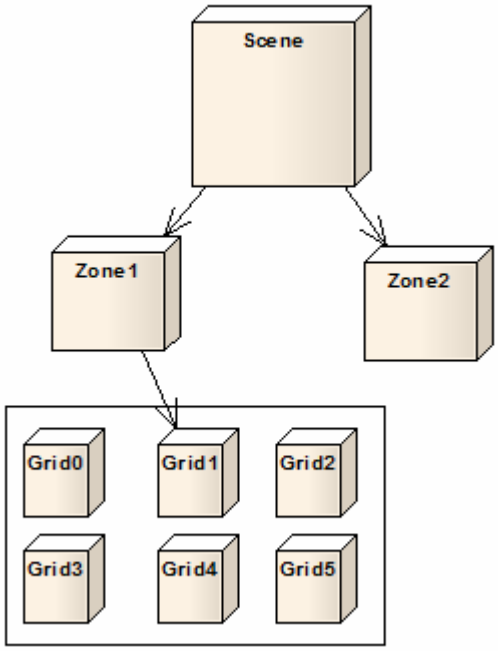

图 3-14 优化过的场景管理结构

Fig 3-14 Optimized Scene Structure

场景被分成多个正方形地块（Scene），每个地块再细分为区域（Zone），每个区域细分为更多的网格（Grid），每个网格负责更新和维护在此网格内的玩家发送的消息。

这种层次结构，可以精确控制每个网格内的玩家数量，也就控制了网络消息的最大并发量。每个玩家发送给服务器端的消息只是发送给同一网格内的其他玩家，降低了框架对网络性能的要求。

这种结构需要将 Scene，Zone，Grid 连接起来，所需的数据结构也就更为复杂，场景靠管理器 CSceneManager 是场景类的管理器，一个单件类，管理服务器端所有的场景，但并不负责具体功能的实现；CScene 是场景类，代表了一个场景，提供了场景操作的所有上层接口；CSceneZone 是场景中的区域类，内聚与 CScene 中，提供了访问上层接口的指针；CSceneGrid 是最基层的，负责具体的操作，保存在此网格内的所有玩家信息；CObjBase 与客户端的 Thing 类似，都是最基础的类，每个物体都记录了所在的网格和场景。

3.3 虚拟培训框架实现

框架采用 C++语言进行实现，并根据语言特点，实现过程中对部分设计进行了优化处理，充分发挥 C++语言的优势。

3.3.1 虚拟世界框架实现

世界框架主要由 TrainApp 和 DeviceEnv 实现，这两个类实现了框架的窗口和渲染引擎的默

认初始化。客户端的主函数必须调用这两个接口，才能继续客户端的启动。

根据详细设计，知道世界框架的功能主要如下：

1） 注册窗口类，创建 windows 窗口等。

2） 启动客户端逻辑处理功能。

3） 处理 windows 消息，进行框架主循环，更新各模块。

4） 初始化图形引擎，设置资源路径等。

5） 加载场景，问题，模型等资源。

作为 TrainApp 的设计基础首先应该研究 windows 编程中的窗口创建部分，因为 TrainApp 将窗口创建过程进行了封装，要想创建出窗口，就必须先了解如何创建窗口。

创建窗口包含三个部分，首先要创建窗口类，窗口类决定了窗口的风格，标题名称，鼠标指针，最大化，最小化按钮风格等，这些参数需要一一赋值；然后注册这个窗口类，windows 系统就知道上一步定制的窗口风格；最后调用系统 API 创建窗口[46]。创建完成后，还要处理 windows 的消息循环。

TrainApp 的函数如下：

1) RegWndClass(HINSTANCE hInstance)

创建窗口类，并将此类注册到系统。

2）CreatWindow(HINSTANCE hInstance)

首先设置窗口格式，从 DeviceEnv 获取窗口大小，窗口显示模式（全屏或者窗口），最后根据上面要求创建窗口。

3）StartTraining()，RunTraining(),EndTraining()

首先 StratTraining 调用引擎初始化接口，完成引擎的启动，然后加载客户端的培训场景和人物模型；RunTraining 处理 windows 消息循环，并更新其他模块；EndTraining 负责释放资源。

DeviceEnv 类与图形引擎关系紧密，但由于图形引擎较多，实现上又不完全一致，无法逐个介绍。因此，本文抽象了图形引擎初始化的接口。首先从配置文件获取引擎参数，根据程序模块地址，分别加载资源目录。最后根据相应参数启动引擎。

TrainApp 和 DeviceEnv 关系密切，之间存在调用关系，TrainApp 调用了 DeviceEnv 的引擎初始化接口，两者相互协作，完成了世界框架的基本功能，框架加载流程如图 3-15 所示。

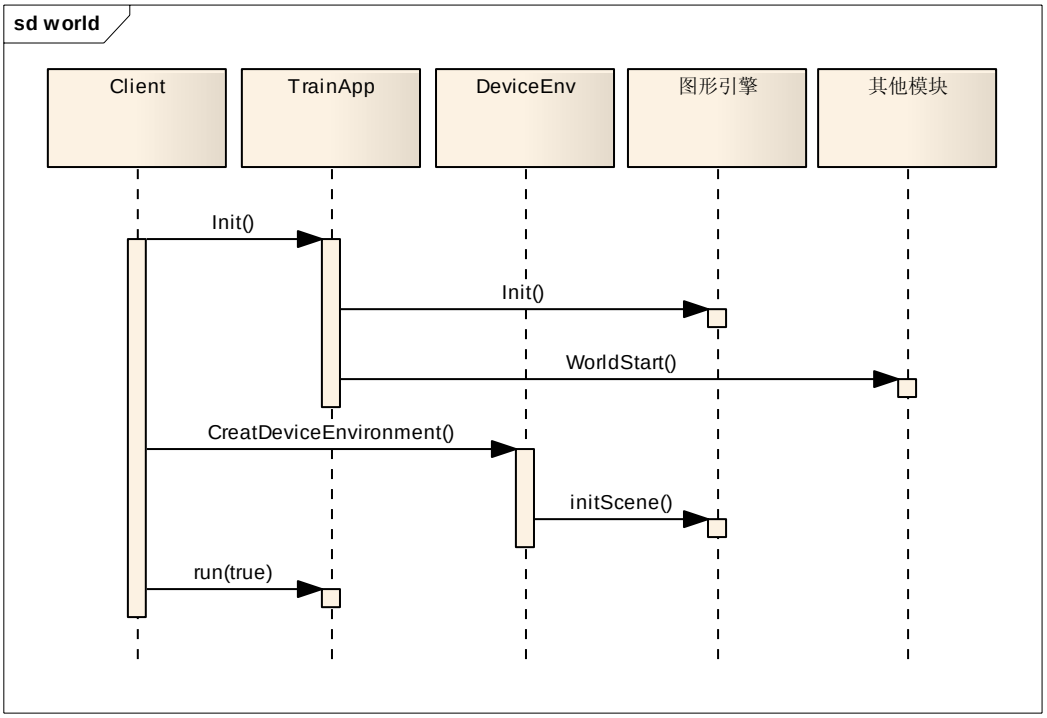

图 3-15 客户端启动时序图

Fig 3-15 Client Start Sequence

客户端的启动首先要创建窗口，调用 TrainApp 的 Init()创建并注册了窗口，同时完成了对引擎的初始化工作，接下来调用其他模块的初始化，保证整个客户端需要使用的模块都正确初始化。DeviceEnv 负责从配置文件读取引擎对视口的一些设置参数，引擎底层的初始化已经在前面完成，剩下的就是设置窗口大小，加载场景等工作。这一切完成之后就可以开始培训框架的主循环 run()了。

3.3.2 任务系统实现

任务系统主要对外提供接口来访问其中的数据，而 TaskManager 封装了这些接口，下面主要介绍管理器的接口

eTaskState 定义了任务的状态，扩展时只需添加相应的类型。TaskData 定义了任务的基本数据，以及在任务完成时的提示语。任务数据的管理则有任务管理器提供。

任务管理器是一个单件类，负责统一管理框架中的任务，封装了任务的相关操作，TaskManager 的核心功能接口如下：

```
void GetTask(int nTaskId);   ///接任务
void DelTask(int nTaskId);   ///删任务
void EndTask(int nTaskId);   ///交任务
```

其中，GetTask 负责获取一个任务，并从任务列表中删除该任务；DelTask 从玩家的任务列表中删除任务；EndTask 将一个任务列表中的某个任务标记为完成状态。

```
eTaskState GetTaskState(int nTaskId,TaskData& task);      ///任务当前状态
void TriggerTaskArea(zPoint2D<float>& pos);
```

```
void setTaskFinish();
bool IsTaskFinished(int index);
bool IsAllFinished();
```

GetTaskState 通过任务 ID 获取任务的数据，并返回任务状态；TriggerTaskArea 设置区域触发的任务，传入参数为任务发生地点，触发半径为默认值。setTaskFinish 强制任务完成；IsTaskFinished 判断任务是否完成；IsAllFinished 判断接受的任务是否全部完成。

```
void GetObject(int nObjectId);                    ///获取一个物品
bool GetPlayerUiInfo(uint nIndex,TaskData& task);
///获得人物身上的任务信息没有则返回 false
```

GetObject 是供外部使用的接口，任务要求获取物品，玩家获取相应物品时就调用此接口；GetPlayerUiInfo 获取玩家当前的任务 UI 显示信息。

```
void ReadFile();    ///读取任务文件
void WriteFile();    ///写任务文件
```

任务文件的读写接口，在任务管理器初始化时，需要读入任务文件，加载入框架，供系统实时调用；在系统退出或者需要保存时，就调用写任务文件接口。

任务系统的加载需要数据文件，其主要用来保存 TaskData 中的任务数据。文件存储有多种格式，xml，txt 是其中最常用的两种，为了开发方便选用 txt 格式。直接将 TaskData 的数据写入文件，下次启动时，由 TaskManager 负责载入。

任务文件选用 txt 格式，虽然操作简便，方便开发，可是当任务文件随着框架越来越复杂时，txt 就会成为一个瓶颈，反而会降低开发速度。鉴于文件时 TaskData 的一种硬盘保存形式，开发人员可以自行替换文件保存格式

3.3.3 人物系统实现

人物系统是培训框架的骨架，背包系统，组件系统都是为人物系统提供服务的。其中 Thing 和 SceneRole 是人物系统的核心和基础，Thing 提供最底层的服务，SceneRole 处理场景模型等，子类都是做了些逻辑上的扩展。

Thing 中有大量函数接口来处理组件部分，等到组件部分再详细介绍。Thing 是框架中的基类，所有的人物都从这里派生，数量众多，不便管理。因此，每个人物都要分配一个 ID，这个 ID 由服务器产生，同步到客户端，保证了每个人物都有标识。同时 ThingManager 也有 ThingIndex 来管理，SceneRole 则为每个需要模型的人物分配了 SceneObjID，每个人物有三个 ID。用 map 可以方便地实现任意两个的相互查询。

ID，SceneObjID，ThingIndex 三个主要 ID 的产生过程如图 3-16 所示。

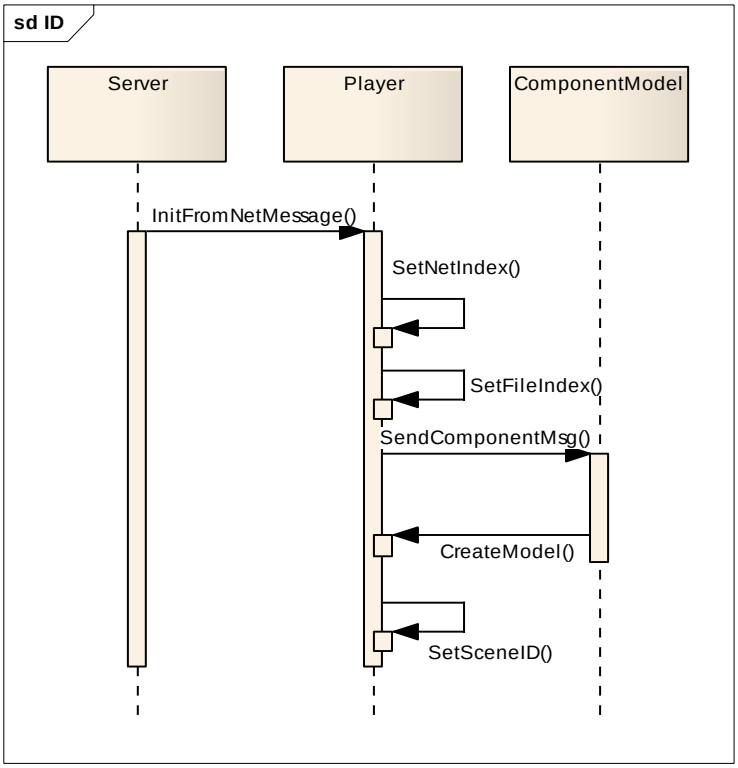

图 3-16 ID 产生时序图

Fig 3-16 ID produce sequence

服务器给客户端发送创建人物消息，消息中就包含了人物的基本信息，包括 ID，属性，模型类型等，客户端接到消息后，通过解析消息内容，调用 SetNetIndex()设置人物的 ID，以后与服务器通信就靠这个 ID，模型信息通过 SetFileIndex()设置，这个信息在调用模型组件创建三维模型时需要用到。Player 使用 SendComponentMessage()通知模型组件创建模型，完成后返回 SceneObjID，Player 调用 SetSceneID()完成 SceneObjID 的设置。

在框架使用中，需要经常查询某个人物，或者通过人物 ID 获取其他信息，而这些信息散落在各模块，很难组织。框架通过这三个 ID 的转换查询，很容易获取想要的结果。Map 是一种效率查询较高的结果，框架将 ID 和人物指针放入其中，核心代码如下：

```
std::map<ThingIndex, Thing*> m_mapThing;
std::map<ID, ThingIndex> m_mapThingInNet;
std::map<SCENEOBJID, ID> m_mapThingInScene;
```

这三个 map 实现了 ID，ThingIndex，SceneObjID，Thing*之间的相互转换查询，有效地组织了框架中的信息。

3.3.4 物品背包系统

背包系统是物品系统的一个具象化的使用，物品系统负责管理框架中所有的物品，而背包系统仅仅负责被 Player 使用的物品。通常还要与 UI 系统协作来完成于用户的互动。

在框架初始化时，物品系统就需要加载配置文件，读入系统中的物品属性。背包系统需要读入物品的 ID，恢复上次背包状态。背包系统的功能如下：

1) 以 UI 的方式呈现玩家随身物品，容易辨识。

2) 背包中的物品随身携带，方便使用。

物品系统中最重要就是物品属性的定义以及物品的使用函数，这关系到物品如何使用。背包系统实现时需要 UI 模块，在此简略介绍其实用，后面会有详细介绍。背包中物品的使用流程如图 3-17 所示。

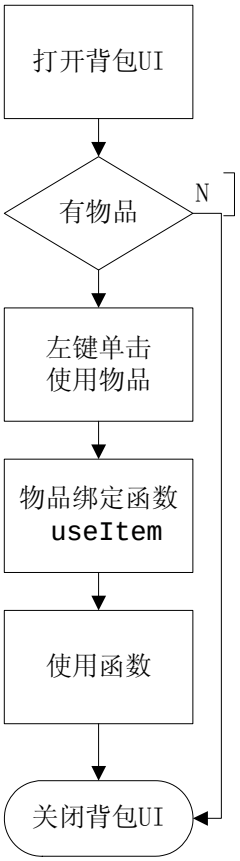

图 3-17 物品使用流程

Fig 3-17 Item Use

人物系统会调用 loadPlayerItem 加载背包资源，其中 LoadItemTable()从硬盘上载入物品系统，信息保存在 item.txt 中，通过 dExcelReader 类解析内容，最后加载入物品系统；接下来背包系统要加载背包中的物品，因为物品系统已经完整保存了物品属性，背包系统只需保存物品 ID 即可找到对应物品，故背包系统加载只需从物品系统中查询物品 ID 即可获取属性，然后深拷贝一份到背包中。接下来 UI 系统会加载物品的小图标，图标信息保存在物品属性中。

当使用物品时，会判断背包中是否有物品，如果有物品，可以在物品栏中移动物品，移除物品栏再放下物品，视为丢弃。使用物品时，只需左键单击物品图标，就会触发与之相绑定的

回调函数，物品 ID 会作为参数被传递，useItem 接收到物品 ID 后，就可以查询到物品类型，根据类型，调用对用的函数。

3.3.5 组件模块实现

组件模块以后最大的挑战就是变动，所以在框架中不能做太多的工作，这样会限制框架的发展，这里所做的就是建立组件模块和人物系统之间的关系，具体就是人物系统的基类 Thing 和组件模块的基类 ComponentBase 之间的关系。

首先要实现 ComponentBase 内聚于 Thing 中，常用的容器有 vector 和 map，vector 和 map 最的区别就是，map 支持键值。在人物系统中，组件具有唯一性，即一个人物不能同时具有两个相同的组件，框架中的可以选用枚举变量标识不同组件。因此，选用 map 作为容器。关键代码如下：

```
std::map<eComponentType, ComponentBase*> m_mapComponents;
```

枚举变量 eComponentType 作为键值，基类 ComponentBase 的指针作为 value，利用 C++的多态性，子类组件可以同样可以放进容器中。

组件模块标识人物就很容易实现，只需一个人物的指针即可。

人物系统和组件模块通过呼叫服务的方式来调用相关功能，图 3-18 是其呼叫的时序图。

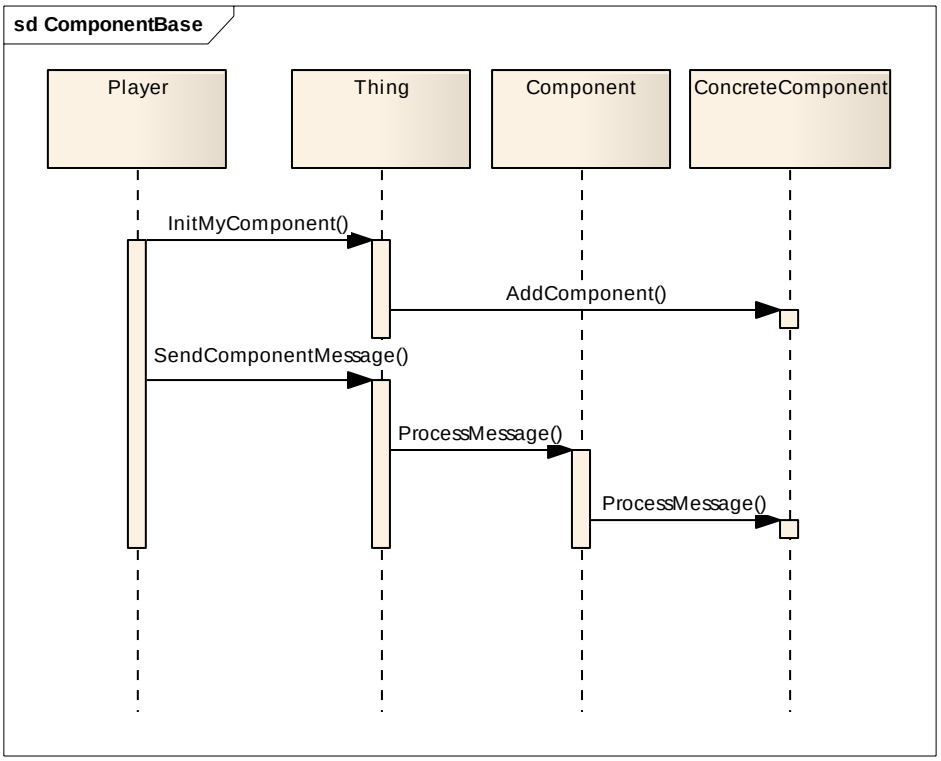

图 3-18 组件交互图

Fig 3-18 Component interaction diagram

Player 是 Thing 的子类，首先调用 InitMyComponet 加载需要的组件，创建新的组件对象，然后 Thing 的 AddComponent 将组件对象作为参数，在函数内部，通过组件对象的指针获取组件类型，加入 map 容器。这是人物系统加载组件的过程，调用服务时，SendComponentMessage

把组件类型和组件数据作为参数，利用组件类型查询到对应的组件的指针，把组件数据作为参数调用组件的 ProcessMessage 函数，ProcessMessage 是组件最重要的函数，每个组件都需要重载这个函数，组件的功能主要在这个函数中完成。

对象组件技术被广泛使用于各种游戏引擎中，将对象的功能以组件的形式分离出来，组件使用的数据和处理数据的方法分离，有效地降低了两个模块间的耦合性。

技能系统

技能系统在框架中作为组件模块的子组件，ComponentSkill 继承自 ComponentBase，所以其重要的事情就是实现 ProcessMessage 函数。

三维培训中，遇到动作系统，控制系统完成不了的功能才需要技能系统，比如在培训中需要粒子特效，果树修剪时特殊的效果等，这些动作，控制系统都解决不了。本文也无法预测具体的培训需要的特殊效果，技能系统只提供基本框架，不实现具体功能。

动作系统

动作系统和后文的控制系统都是作为组件系统的子组件，ComponentAction 类顾名思义，主要控制三维模型中动作。三维模型是美工通过专业建模软件 3D MAX 制作的，模型中绑定了动作。三维模型和动作需要借助图形引擎提供的插件导出成指定格式才能使用，具体操作过程已超出本文研究范围，请查阅相关图形引擎。ComponentAction 对引擎的关于控制模型动作的接口进行了封装，框架在更换图形引擎的时，无需改动框架代码。

由于三维模型需要专业建模软件开发，使用最广泛的就是 MAX 和 Maya，所以只需了解这两款软件中模型动作的基本参数，就可以抽象出控制参数。建模软件的动作都以骨骼动画的形式实现，基本控制参数有帧数，动作播放时间，播放速率。下面是 ComponentAction 抽象的动作播放函数

```
void advanceplay( const zName& nameSequenceName,
    zAnimationPlayMode ePlayMode = EPM_Loop,
    float fPlayTime = -1,
    int nStartFrame = 0,
    float fPlaySpeedRate = 1.0f
    );
```

其参数需要传递给图形引擎，用于骨骼动画的播放，nameSequenceName 是动作名称，动作导出时会有名称用于唯一标示这个动作；ePlayMode 说明了动作播放模式，默认为循环播放，只需播放一次的动作，需要将数值改为 ONCE；fPlayTime 为播放时间，骨骼动画导出时有标识，引擎会提供函数获取此值。本框架采用的图形引擎使用如下函数获取：

```
fPlayTime = G_GetTerObjPlayTime(m_pSceneRole->GetSceneId(),nameSequenceName);
```

G_GetTerObjPlayTime 是一个全局函数，封装了图形引擎获取动画播放时间的函数。nStartFrame 为动作的起始帧，也就是一个动作开始的位置，默认值为 0，即第一帧；fPlaySpeedRate 为播放速率，默认为 1，因为骨骼动画制作时以美观、流畅为目的，改变速率肯定影响美观性，框架

中默认改动，如果系统需要快进或者慢动作播放可以修改此值。

其他重要接口如下：

1) addBkAction(zName nameAct, eActionPriority ePriority, float fTimestop);

用于添加后备动作，当动作加载失败，或者播放失败时，就需要可以使用后备动作。如果后备动作播放失败，框架就会给出错误提示。

2) useBackAction();

用于播放后备动作，重新调用 advanceplay 函数，传入后备动作的数据。

3) killAction(zName nameAct);

结束一个动作。

动作系统在接到人物系统发出的消息后，会在 ProcessMessage 中调用 advanceplay 函数，如果执行失败，会使用后备动作，调用 useBackAction。

控制系统

框架提供的输入设备主要键盘和鼠标，角色移动，查看背包物品等都需要键盘的操作。框架需要获取鼠标和键盘的状态，并决定调用哪些功能函数。ComponentKey 提供了回调机制，时刻监视键盘。框架开发人员可以利用 ComponentKey 定制自己的控制系统，快捷键等。

ComponentKey 的主要功能就是保存、修改、添加、删除键盘组合，并和固定功能函数进行绑定。按键组合以配置文件形式进行存储，初始化时进行加载，框架退出时，更新文件。

控制系统最重要的函数就是：

int BindKey(int key1,int need1,int key2,int need2,int fun);

键绑定函数最多可以绑定由两个键组成的组合键，参数 key1 为按键的虚拟码，need1 为此按键需要组合的功能键，默认为-1 即不需要，第二组参数为备用键，即一个功能有两组按键都可以触发。fun 为按键组合触发的函数。BindKey 在内部构造了一个 SregKey 结构来存放按键组合。SregKey 结构如下：

```
class SRegKey      ///用于绑定按键的结构
{
    int m_nKey1;                ///定义第一个按键
    int m_nKey2;                ///定义第二个按键
    int m_nNeedKey1;            ///定义第一个按键需要组合的功能键
    int m_nNeedKey2;            ///定义第二个按键需要组合的功能键
    eSKeyState m_eKeyState;     ///当前按键的状态
    eSKeyState m_eKeyStateOld;  ///上次按键的状态
    int m_nFun;                 ///要调用的功能函数的 ID
};
```

每个按键组合都会对应一个 SregKey 对象，并保存在 ComponentKey 中容器 m_listKey 内，关键代码 ：

```
m_listKey.push_back(pNewKey);
```

其他重要接口：

1) void CheckKeyState(SRegKey* pTempKey);
   需要在每个的时间循环中检测已经注册的按键注册状态，这需要 windows 底层的 API，GetKeyState 来协助完成，如果两个键的状态都为按下时，就触发绑定的函数。
2) AdvanceTime(float fTime)
   是框架的时间循环函数，需要时刻更新的函数就放在时间循环中，检查按键状态的函数就在其中调用，
3) virtual bool RunFun(int opNum,float fDeltaTime,eSKeyState eCurrentKS,eSKeyState eOldKS);

键盘的控制不需要在 ProcessMessage 中，这个函数是消息触发的，无法实时监视键盘的状态。AdvanceTime 是系统的循环函数，每个类都有实现，键盘监测放在这里最合适。

3.3.6 UI 实现

GUI（Graphical User Interface）即图形化用户接口，是和用户玩家交互的一种重要手段。用户在使用中，大部分的信息是通过 GUI 获取的。GUI 中可以展现文字和图片，在辅助理解三维虚拟培训中有很大优势，三维场景和模型便于展示，知识性的内容就需要借助 GUI。

- **背包系统的实现：**

GUI 具有一定的专用性，背包 GUI 只提供给玩家使用，培训中只有一个玩家，所以将背包 GUI 实现为单间类。

功能：背包 GUI 是背包系统的前台显示，根据背包系统中的物品，管理前端显示，当添加物品时，会在 GUI 中加载对应的物品图标，允许同类物品的叠加存放，并叠加数量。

重要接口：

1) GUI 中提示窗口的显示/隐藏控制
   virtual void showTipWindow(void);
   当鼠标停靠在物体图标上时，就会有提示窗口显示，说明此物体的用途或者属性，showTipWindow 就负责显示提示窗口，内部则是调用了 GUI 引擎的显示接口。
   virtual void hideTipWindow(void);
   实现原理同 showTipWindow。
2) 窗口的显示/隐藏
   void ShowWindow();
   首先调用 GUI 引擎的 enableTopMost()，将窗口移到最上层，GUI 引擎有一个显示队列，只有队首的 UI 可以显示，需要显示某个 UI 时就将其提到队首，然后调用 Show 即可。
   void HideWindow();
3) 刷新，传入参数为背包系统信息
   void UpdateItemInfo(const ItemInfo& info);

用于刷新背包信息，UI 内部信息改变时，界面上不会显示，需要调用刷新函数，重绘 UI。

zFrameWindow 是 UI 系统的基类，封装了 GUI 引擎常用的接口，派生其他 UI，UI 系统的实现和人物系统类似，都有一个庞大的继承树，子类只需实现接口。这种设计保证了 UI 的实现和底层 UI 引擎的分离。

3.3.7 服务器通信模块实现

NetMessage 是通信模块的核心，在客户端和服务器端都要实现，以后应用框架时，这部分的扩展时不可避免的，在这里详细介绍其实现情况。

NetMessage 的核心接口如下：

1. 发送和接收数据

   SendMsg(NetMessage* pMessage, CNetAceTask* pSendQueue);

   virtual const char* GetDataInfo();

2. 处理消息

   virtual void Process();

3. 获取相关数据

   SOCKET GetSocket()

   SetSocket(MY_SOCKET socket)

   ServerID GetServerInfo()

   etServerID(INTER_SERVER idServer)

   GetServerID()

   SetSize(unsigned int uSize)

   SetType(unsigned int type)

   GetSize()

   GetType()

4. 创建消息体

   CreateMessageInfo(MESSAGE_STRUCT* pInfo);

   Create(const char* pMsgBuf, unsigned int dwSize);

   GetMessgeBlock(NetMessage* pMessage)

NetMessage 通过 CreateMessageInfo 来创建一个消息体，然后 GetSocket()会获取当前的 socket，即发送或者接收的目的地，服务器通过 socket 来区分不同的客户端，SendMsg 会负责将消息发送给消息队列，底层通信机制由 ACE 完成。收到需要处理的消息时，会先判断消息的有效性，对有效的消息，会在 Process()中进行处理，并调用相关的功能接口来响应消息。NetMessage 是基类，扩展消息时需要重载 Process()来完成不同的功能。

以 DMsgRoleEyeOut 为例来看一下如何扩展。每个消息主要有三个函数，Create(), Send(),Process()。其中 Create 负责填充消息体，也就是封装消息的内容，为了方便计算结构体的大小和统一接口，将消息实体封装在结构体 MSG_INFO 中；消息填充完成后由 Send()发送，

主要实现有 NetMessage 的 SendMsg()完成；Process()负责处理消息，其会接收到对应的 MSG_INFO 结构体指针，具体处理因消息而已，不再详细介绍。

代码如下：

```
void create(GAMEOBJID id, MY_SOCKET socket)
{
    m_pDataInfo->nObjID = id;
    SetSocket(socket);
}
void DMsgRoleEyeOut::send(GAMEOBJID id, MY_SOCKET socket)
{
    static DMsgRoleEyeOut msg;
    msg.create(id, socket);
    msg.SendMsg(&msg, RoleServerSendQueue());
}
```

GAMEOBJID 是由服务器端产生，用来标识服务器端物体使用的，主要用在网络通信和消息中。

服务器发送接收队列实现

服务器端发送队列是基于 CNetAceTask 派生而来的，发送接收队列的基本功能则由 ACE_Task 来完成，结构图如下：

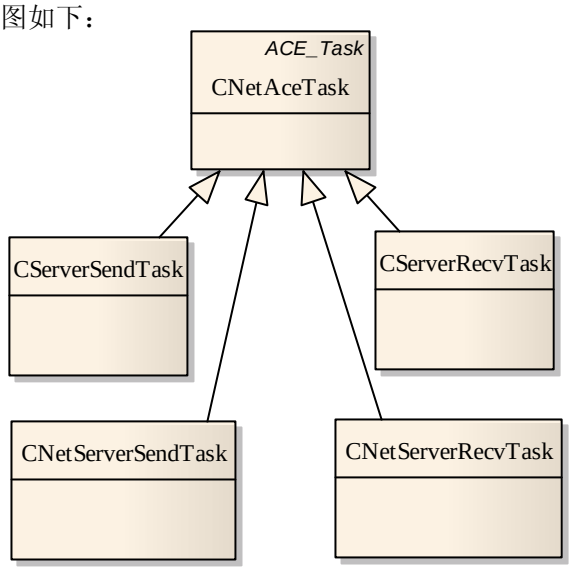

图 3-19 发送接收队列

Fig 3-19 Send and Receive Class

CServerSendTask 和 CServerRecvTask 是负责发送接收的类，另外两个类则是用于网关服务器消息的接收和发送，网关服务器主要负责登录消息的验证，在此不详细介绍。

由于基本功能都集中在基类 CNetAceTask 中，本文将详细介绍。在 ACE 中已经大量应用模板技术，ACE_Task 就内置了用来保存信息的队列，使用时只需调用接口即可。主要接口如下：

1. Open(int nThreads = DEFAULT_THREADS, void* pVoid = 0)
   开启线程，打开接收发送队列
2. PutMsgBlock(ACE_Message_Block *mblk, ACE_Time_Value *timeout = 0)
   将消息体放入队列
3. GetMsgBlock(ACE_Message_Block *&mblk, ACE_Time_Value *timeout = 0)
   从队列取出消息
4. int SvcRun(void)
   负责处理消息

以上三个接口完成了主要工作，子类中会重载 SvcRun(void)来完成消息的处理工作。为了方便开发，框架中进行了进一步的封装，

```
typedef ACE_Singleton<CServerSendTask, ACE_Thread_Mutex> SERVER_SEND_QUEUE;
#define ServerSendQueue()          SERVER_SEND_QUEUE::instance()
```

应用 ACE 的模板将发送器封装成单件类，并且线程安全，为了简化开发，使用时只需使用 ServerSendQueue()宏即可，接收器进行了同样的处理。

OBJ 对象进入场景

场景服务器收到客户端发送的创建消息（dCreateMonsterLTOS，dCreatePlayer）后，调用相应管理集的 CreateNew***（）接口，若创建成功，返回一个新对象。然后，过调用相应管理集的 EnterMapServer（）接口，使 Obj 对象进入地图。

以玩家对象为例说明：

场景逻辑服务器收到 dCreatePlayer 消息后，PlayerManager()->EnterMapServer(pPlayer)。PlayerManager()调用 AddObj(pPlayer)将 Player 对象加入管理集。

OBJ 对象调用 EnterScene() 接口进入场景，pPlayer->EnterScene()。（此接口为虚接口，需要执行特殊逻辑时，可以重载。）判断 Obj 对象进入场景是否存在，做存在，调用场景 EnterScene（）接口，使 Obj 进入场景，并将场景对象指针保留在 Obj 对象。

通过 Obj 对象坐标，查找其对应场景的地图块 SceneBlock，若存在，调用 SceneBlock 相应接口 EnterBlock（），令 Obj 对象进入 SceneBlock。将 Obj 对象添加入 SceneBlock 的 Obj 管理集，并向其周围 SceneBlock 中的玩家同步外形消息。

Player 移动

场景服务器收到 dMoveLTOS 消息后，通过 ID 找到 Player 对象后，调用 MoveTo（）接口移动。（pPlayer->MoveTo（））。判断玩家是否在场景中，调用场景对象 CScene 的 ChangeBlock（）接口，判断是否切换 CSceneBlock。判断新坐标是否在当前 Block 中，若不在，则离开当前的 SceneBlock（LeaveBlock（））、切换入新的 SceneBlock（EnterBlock），并向新进入视野的 SceneBlock 广播外形消息、并同步切换 SceneBlock 消息（CMsgToBlock）至 ChatServer。最后，广播移动消息至周围区域。

Player 对象离开场景

场景服务器收到 dDeletePlayer 消息后，调用 PlayerManager() 的 LeaveMapServer（）接口离开场景服务器。通过 ID 查找相应 Player，若存在，调用 Player 对象的 LeaveScene（）接口，若玩家所处场景存在，调用场景的 LeaveScene（）接口。若玩家所处 SceneBlock 存在，调用 SceneBlock 的 LeaveBlock（）接口。从 SceneBlock 的 Obj 集中删除 该 Player 指针。向周围玩家广播离开消息 dMsgLeave。

3.4 本章小结

本章是此次设计的核心部分。首先概述了这个框架的设计思想，然后详细描述了培训框架的模块分类以及各模块的结构。人物系统和组件系统是框架的重点，采用 Object+Components 思想设计，分离了 object 的功能，有效降低框架的耦合度，提高复用。管理器采用单件模式，实现时采用静态变量。UI 从设计之初就以分离底层 UI 引擎为目的，网络通信部分采用 ACE 作为底层通信框架，并对服务端的场景管理做了优化，降低了网络负载。在框架设计中大量采用了 C++动态绑定机制和类派生机制来完框架对象的描述。框架模块间由于使用设计模式等技巧降低了耦合度。

第四章 实例

本章以一个典型的应用来说明框架的开发过程。这是一个简单 demo，其中包含几个任务，以此来说明框架的开发流程，并验证框架中各模块的功能。

4.1 系统设计

第一个任务以司马光砸缸的故事为原型，第二个任务是收集草药。首先需要分析任务需要实现那些功能，框架中有哪些现成可用的模块。这两个任务都以小故事的形式体现，很明显需要任务系统的支持。任务系统的有两种逻辑,第一种是去 A 那里接一个任务，到 B 那里完成; 第二种是去 A 那里接一个任务，要求收集指定数量的某种物品，收集完成后，回到 A 那里完成任务。上面两个任务可以验证以上两种逻辑。

司马光砸缸的故事以任务系统的逻辑整理过后，司马光去村长家接到一个任务，内容是去救小孩，司马光到出事地点后，灵机一动用石头砸碎了缸，小孩出来，司马光向小孩交任务，然后任务完成。收集草药的故事为，司马光找村长接任务，村长要 9 株草药，司马光去采集不少于 9 株的草药，回到村长那里交任务，任务完成。

这两个任务都用到了任务系统，人物系统，模型系统，动作系统，控制系统。网络通信模块实现了基本的同步功能。

4.1.1 操作流程

通过系统分析，系统流程如下图 5-1 所示。在一个框架循环中系统需要完成三个过程，首先系统以三维的方式把故事呈现给玩家，并通过 GUI 做相应的操作提示；玩家这时可以进行操作，从村长那接收任务等；然后这些操作被转化成数据传递给控制组件，其结果直接反映到三维场景中，可能是玩家的走动，或者草药被收集等。至此一个循环结束，在系统中随着时间的推移，不断重复迭代这个循环，直到完成所有任务，任务循环如图 5-1 所示。

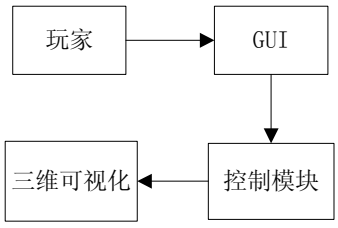

图 5-1 循环中的一个基本操作流程

Fig 5-1 Basic Operation Process in Cycle

框架是半成品的系统，其中大部分模块都已封装好。开发时只需根据系统需求，扩展框架无法实现的功能。上面的任务没有超出框架的功能，无需扩展。

4.2 详细设计与实现

任务虽小，可依然是三维可视化系统，有良好的交互性和娱乐性，实现的过程需要大量的美工工作来完成三维建模和 GUI 的创建。

4.2.1 任务文件

任务已经按照系统的要求进行了整理，接下来需要将任务逻辑以任务文件的方式保存下来。任务内容如表 5-1 所示。

表 5-1 任务内容

Table 5-1 Task Content

任务 ID	1	2
任务名称：	救小孩	采草药
接任务 NPC 的 ID	村长	村长
交任务 NPC 的 ID	小孩	村长
需求物品类型 ID	0	草药
需求数量	0	9
已经获取数量	0	0
接到任务时对话内容	村边刚才有人喊救命，我走不动了，你去看看吧	帮村长采 9 株草药吧
完成任务时的对话内容	谢谢你救了我	好孩子
未完成任务时的对话内容	时间宝贵	时间宝贵

表 5-1 为任务文件里的内容，当任务系统加载时，任务管理器会将这些数据填充进 TaskData 中。

4.2.2 模型制作

三维模型需要用 3D MAX 等建模软件来制作，建模完成后需要用图形引擎提供的模型导出插件来将 3D MAX 制作的模型进行转换。任务中需要至少 6 个模型，司马光，村长，小孩，草药，石头，大缸。司马光还需要走路，跑，砸缸三个动作。具体制作过程已经超出本文研究范围，不予介绍。制作好的模型如图 5-2 所示。

图 5-2 demo 模型图

Fig 5-2 demo model

4.2.3 GUI 制作

GUI 是和培训人员交互的重要途径，系统中提示信息以及大部分操作都需要 GUI 参与。但其制作却需要二维美工人员，本系统中由非专业人员进行制作，旨在验证框架。效果图如下：

图 5-3 背包系统 UI

Fig 5-3 Package system UI

图 5-3 是背包界面，背包有 15 个格子可以存放物品，最上面的那个用来存放和玩家绑定的物品，消耗类的物品可以通过鼠标左键单价直接使用。

4.2.4 逻辑处理

逻辑处理是设计部分的实现，需要实现模型，并将 GUI、三维显示部分和逻辑处理建立关联。首先需要实现将动作组件和模型组件有关的文件进行加载。三维场景就会载入相关的模型。

鼠标右键单击村长模型时，显示任务界面，这个过程需要将 UI 显示函数挂载到鼠标事件中，代码如下，接收任务的过程如图 5-5 所示。

图 5-4 接受任务前后变化

Fig 5-4 Before and After took task

模型移动时使用控制和移动模块，仅使用走动功能，无需扩展。当捡起石头时，需要在背包中增加物品，走到缸旁边时，鼠标左键单击使用背包中的“石头”就可以砸缸了，这时首先从背包中去掉“石头”的图标，调用“石头”的回调函数，该函数需要调用动作播放函数，来播放“砸缸”的骨骼动画，并替换缸的模型，生成小孩的模型。砸缸的过程如图 5-6 所示。

图 5-5 砸缸前后效果对比

Fig 5-5 before broke and after

草药任务界面和砸缸的基本相同，不再介绍，两个任务最主要的区别在于，拾取草药时，背包中增加一样物品草药，由于需要多株草药，背包中会堆叠显示，如下图所示。同时需要删除场景中的那株草药，更改任务系统中已经获取草药的数量。采草药的过程如图 5-7 所示。

图 5-6 采草药

Fig 5-6 take plant

4.3 本章小结

本章通过实现两个简单的小故事，使用了框架中的人物系统，组件系统中的模型组件，动作组件，控制组件，并使用了背包系统，网络通信等模块。整个实例中，工作量最大的是美工人员，因为框架已经提供了绝大部分成型的系统，可以大大提高开发效率。

第五章 结论

5.1 总结

本文利用自顶向下的框架开发方法，使用虚拟培训系统开发中常用的设计模式，通过封装图形引擎，GUI 引擎的接口，分离 MVC 模式中的模型和控制模块，设计并实现了交互式虚拟培训系统应用框架。

1. 首先研究软件复用技术，主要研究设计模式和框架开发技术。研究了虚拟培训系统开发中常用的设计模式，以及设计模式的使用，在框架开发中应用 Singleton，Factory 等模式，提高了框架的灵活性、可复用性。
2. 采用自顶向下的方法，设计并实现了客户端的基本功能模块，主要模块包括：

   ①世界逻辑框架，该框架是三维培训场景的逻辑主循环，实现了引擎层接口，应用程序接口，使得图形引擎和应用程序得以分离，此框架以单件模式实现。

   ②人物系统框架，人物系统是框架的核心，培训中需要控制的物体都派生自人物系统，实现时采用 C++面向对象的技术，以继承树的形式实现，叶子节点是场景中主要用到的类，如 Player，Plant，Animal 等。本论文完成了此部分的编码。

   ③物品系统，集合了物品和背包两个系统，为了方便系统的扩展，将物品系统的数据和操作进行了分离，物品管理器采用单件模式实现，提供了对物品的管理接口。

   ④组件系统框架，是框架中最具扩展性的部分，框架在底层预支了 ComponentBase 和其他模块的交互，扩展新功能只需继承基类，实现相关接口即可。
3. 服务器部分采用了一些现有成果（如 ACE），明显降低框架开发难度，通信层和应用层分离，使服务器端框架的结构化更加明晰；并根据需求对场景管理模块进行了局部广播的优化处理，有效降低网络负载。
4. 最后通过开发一个典型的三维虚拟培训应用来验证框架的正确性，以及开发中带来的便捷。

5.2 展望

为使虚拟培训框架的功能更加完善，具有更高的复用性，还需要做进一步的研究和完善工作，主要包括以下几个方面：

1. 加强对框架的应用验证

   每一个系统都有隐含的问题，本框架仅实现了一个培训系统，不能保证框架的稳定性。如果能使用框架开发更多的系统，肯定会发现系统的问题，并能促进框架的完善。
2. 增加数据库模块

   虚拟培训系统中需要对大量的数据进行管理，本框架中的数据管理采用管理器的方式，如果能用数据库来代替数据管理器，会提高框架对数据的管理能力。
3. 增加内存管理模块

框架采用传统 C++的默认内存管理模式，随着框架的扩展，使用更多底层框架，就会出现内存管理冲突的问题，需要为框架提供可钩挂自定义内存管理模式的内存管理模块。

参考文献

[1] 崔蔚，徐铁钢，韩卫华．虚拟培训技术及其系统开发．成都信息工程学院学报，2003,18(4)：361~365

[2] 汪成为．灵境（虚拟现实）技术的理论、实现及应用．北京：清华大学出版社，1996,40~50

[3] J．R．Ggalvao．Production and Teaching by Virtual Environment．Proc．Of ICAT’ 97，1997,4(07)：1067~1072

[4] 吴锋，侯平智．采油仿真培训系统的设计与实现．辽宁石油化工大学,2008,28(4): 70~72

[5] 周慎． 基于虚拟现实的汽车驾驶模拟器建模技术研究：[硕士学位论文]．武汉：武汉理工大学，2005

[6] 战守义，郑宏，王涌天．虚拟现实技术在坦克驾驶模拟训练中的应用．北京理工大学学报，1995，15(5): 95~99

[7] 常壮，邱金水，张秀山．基于虚拟现实技术的舰船虚拟消防训练系统体系架构研究．中国舰船研究，2009,4(3): 57~60

[8] 刘航，王春水，王积忠．基于视景仿真技术的某型装备虚拟操作培训系统．指挥控制与仿真，2007,29(2): 79~83

[9] 郭燕舞．虚拟现实技术在神经外科的临床应用研究：[硕士学位论文]．广州：南方医科大学，2008

[10] 刘刚．基于虚拟现实技术的集装箱起重机驾驶方丈培训器研究：[硕士学位论文]．武汉：武汉理工大学，2002

[11] 黄鑫．基于 VR 技术的虚拟教学应用研究：[硕士学位论文]．武汉：华中师范大学，2005

[12] 胡林，周国民．虚拟农业教育平台研究．农业网络信息，2007，(3): 6~9

[13] 张孝飞，张振国，刘星何．虚拟现实技术在农业职业教育中的应用．安徽农业科学，2007，35（7）：2092~2093

[14] 彭辉．虚拟现实技术及其在农业中的应用．农业网络信息，2004,(5): 15~18

[15] 杨沛．虚拟现实技术及其在农业中的应用．安徽农学通报，2008,14(23): 195~197

[16] 何喜玲，王俊．虚拟现实技术及其在农业中的应用．中国农机化，2004,2(1)：22~25

[17] 张杜鹃，杨安祺，单振芳．采用无线网建立农业虚拟现实技术的研究．农机化研究，2006，(7)：19~21

[18] http://www.simulearn.net/leadership_training/leadership_training.html

[19] http://www.forgefx.com/

[20] http://www.virtualwalkingpens.com/

[21] Kmeger C W．Softwam Reuse．ACM Computing Surveys,1992, 24(2):71~75

[22] Carma McChre，王亚沙．软件复用标准指南．北京：电子工业出版社, 2004, 103~106

[23]Fayad．M．E，Schmid．D．C，Johnson．ILE． Building Application Frameworks．Boston :Addison-wesley,1999, 65~67

[24] H．A Schrnid．Systematic framework design by generalization. Communications of the ACM,1997,40(101)：48~51

[25] K．Koskinmies，H．Mossenbock．Designing a Framework by Stepwise Generalization．ACM Computing Surveys, 1997,41(11):65~78

[26] 郑人杰，殷人昆，陶永雷．实用软件工程．北京：清华大学出版社，1997,89~93

[27] 孙大烈，杨静，王子立．基于对象的软件重用技术研究．哈尔滨建筑大学学报,2001,(03)：32~54

[28] Nadia Bouassida，Hanene Ben-Abdallah．Stepwise framework design by application unification．IEEE SMC, 2002，(8):212~324

[29] E．P． Brooks，The Mythical Man-Month．Boston：Addison-Wesley,1975,146~151

[30] 王丽娟，孙西超，底松茂．软件复用与基于面向对象框架的软件开发方法．郑州大学学报(工学版),2001,9(3):24~28

[31] 丁忠俊，徐政权．软件重用技术和方法．小型微型计算机系统,1998,(11)：56~60

[32] Ivar Jacobson，韩柯．软件复用：结构、过程和组织．北京：机械工业出版社，2000,241~253

[33] Erich Gamma，Richard Helm，Ralph Johnson．Design Patterns：Elements of Reusable Object-Oriented Software．Boston：Addison-Wesley,1995,169~176

[34] Mohamed Fayad，Ralph Johnson，and D C．Schmidt．Building Application Frameworks，Object-Oriented Foundations of Framework Design．New York：John Wiley & Sons Inc, 1999,263~278

[35]Mohamed Fayad，Ralph Johnson，and D C Schmidt．Implementing Frameworks：Object-Oriented Frameworks at work．New York：John Wiley & Sons Inc,1999, 369~378

[36] Erich Gamma，etc(美)著．李英军等译．设计模式．北京：机械工业出版社，2003,172~180

[37] Grady Booch，James Rumbaugh，Ivar Jacobson．The Unified Modeling Language User Guide．Boston：Addison-Wesley Professional，1998,249~256

[38] D．C Schmidt and P Stephenson．Experiences Using Design Patterns to Evolve System Software Across Diverse OS Platforms in Proceedings of the European Conference on Object-Oriented Programming．ACM,1995,(8):34~65

[39] Marter Flower．UML Distilled A．London：Brief Guide，2000,143~167

[40] Julian Gold(美)著，陈为、周骥等译．面向对象的游戏开发．北京：电子工业出版社,2004,234~265

[41] Dante Treglia(美)编．沙鹰等译．游戏编程精粹 3．北京：人民邮电出版社,2004,197~200

[42] Andrew Kirmse(美)编．沙鹰等译．游戏编程精粹 4．北京：人民邮电出版社,2005,158~163

[43] Mat Buekland(美)著，沙鹰等译．游戏编程中的人工智能技术．北京：清华大学出版社，2004,102~108

[44] 云风著作．游戏之旅——我的编程感悟．北京：电子工业出版社，2005,258~260

[45] (美)RobertB．Murray 著，王夕译．C++编程惯用法一一高级程序员常用方法技巧．北京：中国电力出版社,2004, 297~300

[46] Renaud P.E. Introduction to Client/Server systems．New York :John Wiley & Sons Inc,1998,171~173

[47] 苏羽，关沫，许研． Visual C++网络游戏建模与实现．北京：科学出版社, 2006,161~162

[48] Douglas C．Schrmdt,Stepheu D．Huston．CH Network Programming Volume l：Master Completxity with ACE and Patterns(影印版)．北京：清华大学出版社, 2003,267~281

[49] Douglas C．Schmidt,Stephen D．Huston．C++网络编程卷 2：运用 ACE 和框架的系统化复用．北京：电子工业出版社，2004,267~270

[50] Dante Treglia, 张磊译．网络游戏编程精粹 3．北京：人民邮电出版社，2003,149~154

[51]

致 谢

值此论文完成之际，谨向在我硕士研究生学习期间给予我指导、关心、支持和帮助的所有老师、同学、亲人和朋友们致以衷心的感谢！

首先感谢我的导师朱虹副教授！在这两年多的研究生涯中，她的细心指导和无私关怀不断鼓舞着我，让我在科研的旅途中始终充满信心，勇敢的面对各种挑战，克服一个又一个难关。

感谢陈洪老师和王庆老师！作为项目负责人，陈老师的理论指导和王老师的正确指引是我完成此项课题的关键；两位老师渊博的知识、宽广无私的胸怀、夜以继日的工作态度、对事业的执著追求、诲人不倦的教师风范和对问题的敏锐观察力，都将使我毕生受益。陈老师在百忙之中，仍时时关心我的研究进展，不遗余力地为我的研究工作提供帮助，在课题研究中提出了许多宝贵的意见和建议。

感谢同学赵琛、任兴超、王凯在工作、生活、学习上的帮助，是他们在严谨却又枯燥的科研中给我带来了欢乐，在这论文完成之际，我向他们表示最诚致的感谢。

感谢我的父母，感谢他们二十多年的养育之恩，感谢他们对我的关爱和支持，感谢他们不求回报的付出，在以后的人生道路上，我将做出更好的成绩，以报他们的养育之恩。

感谢所有关心和支持我的亲人！

最后感谢所有关心、帮助和支持过我的人们！

作者简介

牛景波：男，1984 年 11 月 15 日出生于河南省。2004 年 9 月考入中国农业大学信息与电气工程学院学习，于 2008 年 7 月毕业获得工学学士学位。2008 年 9 月保研进入中国农业大学信息与电气工程学院计算机应用专业学习，师从于朱虹副教授和陈洪老师。

攻读硕士期间发表的论文：

1. 牛景波，陈洪，王庆，朱虹，赵琛. 基于农业模型的农业培训系统的设计与实现. 微计算机信息. 2011.1

2. 牛景波，王庆，陈洪，朱虹，赵琛，朱德海. 基于虚拟现实技术的农民培训系统框架设计及实现. 计算机应用. 2010.6