摘 要

随着多媒体技术、计算机科学、计算机通信和网络技术的不断发展，现代远程教育已经成为人们关注的热点。远程软件教学系统是现代远程教育研究领域中的一个重要的课题。本文对 *EPSS*（*Electronic Performance Support System*）远程软件教学系统及其相关的决策分析问题在实现方法、实现技术和理论分析等方面做了详尽的论述。

本文首先对现有的远程软件教学系统进行了全面的分析，针对传统远程软件教学方式重论述、轻实践的缺点和软件教学强调用户操作的特点，提出了一个全新的远程软件教学模式，即用户实际操作目标应用程序，教学系统对用户的每步操作进行跟踪指导。

在此基础上，本文提出了基于分布式组件技术（*DCOM*）的 *EPSS* 系统框架模型，同时在对 *IEEE P1484.1* 标准草案（*LTSA*）进行深入研究的基础上，提出了基于该标准草案的客户端应用程序体系结构模型。

在分析了各种可能的技术实现方案的优缺点的基础上，本文提出了使用屏幕取词技术来截获用户对目标应用程序的操作的技术方案。屏幕取词技术中关键的 *Windows API* 函数拦截技术是涉及 *Windows* 操作系统核心的技术，本文对在 *Windows* 操作系统中实现 *API* 函数拦截的各种方法和技术实现手段进行了全面的总结。

接着，本文分析了在 *EPSS* 系统中进行决策分析的目的和意义，在此基础上提出了将粗糙集（*Rough Set*）理论应用于 *EPSS* 系统中进行决策规则提取和知识样本过滤的解决方案。属性约简是粗糙集理论的核心内容，针对传统 *Jelonet* 属性约简算法效率不高，本文提出了 α 算法，减少了算法的时间复杂度。

最后，本文给出了 *EPSS* 系统的改进方案，并明确了进一步的研究工作。所有这些，为现代软件远程教学系统的开发和研究提供了有益的参考。

关键词：远程教育、屏幕取词、*Windows API* 拦截、粗糙集、属性约简、*LTSA*、*COM/DCOM*

Abstract

With the development of multimedia technology, computer science, computer communication and network technology, the modern distant education has focused people's attention. The distant application learning system is an important subject in the research of modern distant education. In this paper, we discuss the methods, technologies and theories used in the implementation of *EPSS*（*Electronic Performance Support System*）distant application learning system and the problems of correlative decision analysis in details.

First, the article analyzes the current distant application learning systems entirely, and according to the fact that the practice should be emphasized in application learning course, a new method that the distant application learning system tracks the user's actions when he operates the target application and then provides the instructions for each step is submitted against the flaw of the traditional method that the statement other than practice is accentuated in distant application learning system.

On the basis of the above, the *EPSS* system architecture based on the *DCOM* technology is submitted in this article, through the deep research on the *IEEE P1484.1*（*LTSA*）draft standards, the client application architecture is also submitted according to such standards.

After analyzing the advantages and shortcomings of all kinds of possible implementation schemes, the scheme of capturing the user's actions by the technology of capturing word from screen is submitted as the choice in the *EPSS* implementation. Windows API interception is the key of the technology of capturing word from screen, which touches the kernel of windows operation system, the article synthesizes the technologies and methods used in windows API interception thoroughly.

Next, the article discusses the meaning and the intention of making decision analysis in *EPSS* system. Then Rough Sets theory is pointed out as the solution of extracting the decision rules and filtering the knowledge samples. Attributes reduction is the core of Rough Sets theory and the traditional *Jelonet* algorithm is inefficient, so an improved α algorithm is submitted which reduces the time complexity compared with the *Jelonet* algorithm.

Finally, the article describes the amendments of *EPSS* system and specifies the correlative future research. All of the above can be beneficial references for the research and development on modern distant application learning system.

Key Word: Distant Education, Capturing word from screen, Windows API Interception, Rough Sets, Attributes Reduction, LTSA, COM/DCOM

1 绪论

1.1 引言

现代远程教育是利用网络技术、多媒体技术等现代信息技术手段，通过音频、视频（直播或录像）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新型教育形式[1]。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 *Internet* 的出现，远程教育的特征发生了深刻的变化。

1.1.1 远程教育的发展[2][4][6]

在 20 世纪早期和中期，远程教育技术（如打印机、收音机和电视）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勾通的作用，仅实现了师生之间有限的交流。限于传输技术，远程教育还受到时间的限制（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，它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不受时间限制，可以将录制了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。

80 年代中期，个人计算机技术在远程教育系统中的应用标志着第三代远程教育技术的诞生，不久又出现了双向视频会议系统。第三代远程教育技术同以前相比，教员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、计算机模拟以及通过计算机磁盘、光盘和 *Internet* 等途径传播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，所需要时间变得更短。这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。

1.1.2 远程教育在中国

远程教育在中国的发展经历了三代：第一代是函授教育。这一方式为我国培养了很多人才；第二代是 80 年兴起的广播电视教育。我国的这一远程教育方式和中央电视大学在世界上享有盛名；90 年代，随着信息和网络技术的发展，我国产生了以信息和网络技术为基础的现代远程教育。

1.1.3 现代远程教育系统的开发模式

现代远程教育可利用的信息传输通道主要有以下三种：卫星广播网、邮电通讯网、计算机网络[3]。目前，利用计算机网络进行远程教育已经成为现代远程教育的基本模式。我们从教学和作业两个环节来看看现代远程教育的一般形式：

1) 教学环节。

现代远程教育的教学方式主要有两种：一种是使用计算机辅助教学（*CAI*）课件。即教师根据不同课程的内容、难度预先编制好一系列 *CAI* 课件，并在适当的时候将它们发送到互联网上。学习者不再受时空的限制，根据自己的学习能力和掌握程度，随时随地从网上点播适合自己的 *CAI* 课件，循序渐进地学习课程。

另一种是 *Web* 浏览器方式。*Web* 将世界各地极其丰富的信息资源以含有链接的超文本形式组织成一个巨大的信息网络，用户通过 *Web* 浏览器，在 *Web* 网页中用鼠标点击有关的文字或图形，就可以随心所欲地浏览他所感兴趣的内容。基于 *Web* 的学习就是把教学内容设计成网页，放在 *Web* 上，供学员以浏览网页的方式进行自主学习。

2) 作业环节

现代远程教育的作业方式主要也有两种：一种是 *CAI* 方式，教师预先将练习或测试题编制成 *CAI* 作业自测软件放置在计算机教学网上，学员按要求上网查询到该软件就可以在线做作业。当作业、自测完成时，教学系统会自动为其评分并将成绩通过网络“上交”给教师。

第二种就是 *E_mail*（电子邮件）方式。根据教师的统一要求，学员以统一的文件格式来书写所做作业。作业完成后，再将这个作业文件作为附件随 *E_mail* 发送给教师。教师对作业进行批改后，对作业中存在的问题进行解答，仍以 *E_mail* 的方式反馈给学员，针对学员中普遍存在的问题，还可在网上发布公共信息或组织网上实

时讨论。

1.1.4 远程教育面临的问题[5]

现代远程教育的突出特点是：真正不受时间和空间的限制；受教育的对象扩展到全社会；随着互联网的发展，将会有极丰富的教学资源供学习者选用；教学形式由原来的以教为主变为以学为主。

任何事务都有不足之处，基于 *Internet* 和多媒体技术的现代远程教育也是如此。首先，*Internet* 带宽是一个共享信道，目前其接入带宽远远小于传统远程教育的信息通道容量（比如有线电视），学生和教师在进行多媒体交互的时候，性能会受到影响甚至完全行不通。因此，如何减少教学课件的数据量成了首要研究的问题。其次，数据传输只是利用了 *Internet* 分布式计算能力很小的一部分。如何充分利用计算机强大的数据处理能力，将教学中涉及的系统管理和维护等工作实现信息化也是我们需要研究的课题。从硬件基础上讲，如何建设远程教学网络，实现高质量的教学环境，也是一个重要问题。除此之外，政策的制定、系统开发标准化、教学课件的制作等等都是现代远程教育研究必须面对的关键问题。

1.2 国内外现代远程教育标准化工作概况

现代远程教育以计算机网络（以及卫星数字通讯）技术为支撑，具有时空自由、资源共享、系统开放、便于协作等优点。世界各国在发展现代远程教育时深刻认识到，学习资源的可共享性和复用性对于网络远程教育的实用性和经济性具有决定性意义[7]。虽然目前的网络技术已为教育资源在低水平上的自治与共享（例如通过 *HTTP* 和 *HTML*），为学习活动的合作（例如通过各种通信工具）提供了基本技术条件，但是允许教学资源在课程知识和教学管理水平进行交换的标准却没有很好地制定，因此妨碍了教学资源的大范围共享与交流。

有鉴于此，国际上已有不少国家和组织致力于远程教育技术标准的研究。在美国有航空工业计算机辅助训练委员会（*AICC*）最早提出的计算机管理教学标准；有 *IMS* 全球学习联合公司提出的学习系统技术规范。在欧洲方面，有 *ARIADNE*（欧洲远程教育多媒体制作与销售网联盟），*CEN/ISS*（欧洲标准委员化/信息社会标准化系统）等组织进行多媒体和远程教学技术标准的研究、国际合作及本土化工作。目前

在国际电气和电子工程师协会学习技术标准委员会（简称 *IEEE LTCS*）的主持下，若干个工作小组正开展网络远程教育技术标准的制定和修订工作，将形成 *IEEE1484* 标准。国际标准化组织 *ISO* 于 1999 年成立了一个 *JTC1/SC36* 委员会，专门从事学习、教育、培训技术标准的征集、修订和批准工作，目前已有美国、英国、德国、日本、乌克兰等国提交了标准议案。世界上许多国家十分重视教育技术国际标准的采用，组织力量参与国际标准制定与本土化工作，其中法国、德国、西班牙已完成了部分 *IEEE LTSC* 标准的本土化工作，荷兰、希腊、意大利等国家即将推出他们的教育技术标准草案。

我国的远程教育技术已经开始进入以网络为基础的新阶段。在基础教育方面，近年来各地自发地涌现出一大批中小学教育网校；在高等教育方面，教育部已经批准 30 多所重点高校开办网络远程教育。大力发展现代远程教育，对于促进我国教育的普及和建立终生学习体系，实现教育的跨跃式发展，具有重大的现实意义。前几年由于我国目前还没有制定关于网络远程教育技术的标准，各网络教育系统的资源自成体系，无法实现有效交流和和共享，造成大量低水平的重复性开发工作，不但带来人力物力的浪费，而且将无法与国际网上教育体系相沟通。因此国家教育部成立了中国教育部现代远程教育标准化委员会，负责领导全国各高校的科研小组进行远程教育技术标准的研制工作。

我国的现代远程教育技术标准研制工作以国际国内现代远程教育的大发展与大竞争为背景，以促进和保护我国现代远程教育的发展为出发点，以实现资源共享、支持系统互操作性、保障远程教育服务质量为目标，通过跟踪国际标准研究工作和引进相关国际标准，根据我国教育实际情况修订与创建各项标准，最终形成一个具有中国特色的现代远程教育技术标准体系（*Distance Learning Technology Standards*，简称 *DLTS*）[7]。

通过分析国际上关于教育信息技术标准的研究线索，特别是参照 IEEE 1848 的框架，中国现代远程教育技术标准体系目前提出了 27 项子标准，分为总标准、教学资源相关标准、学习者相关标准、教学环境相关标准、教育服务质量相关标准五大类。此外，还设立了 4 个跟踪研究项目。这些标准的相互关联如图 1.1 所示[7]。

在 2002 年 2 月 6 日，教育部正式印发了《现代远程教育技术标准体系和 11 项试用标准 V1.0 版》，从此，我国远程教育系统的研发工作有了适用于本国的参考依据。

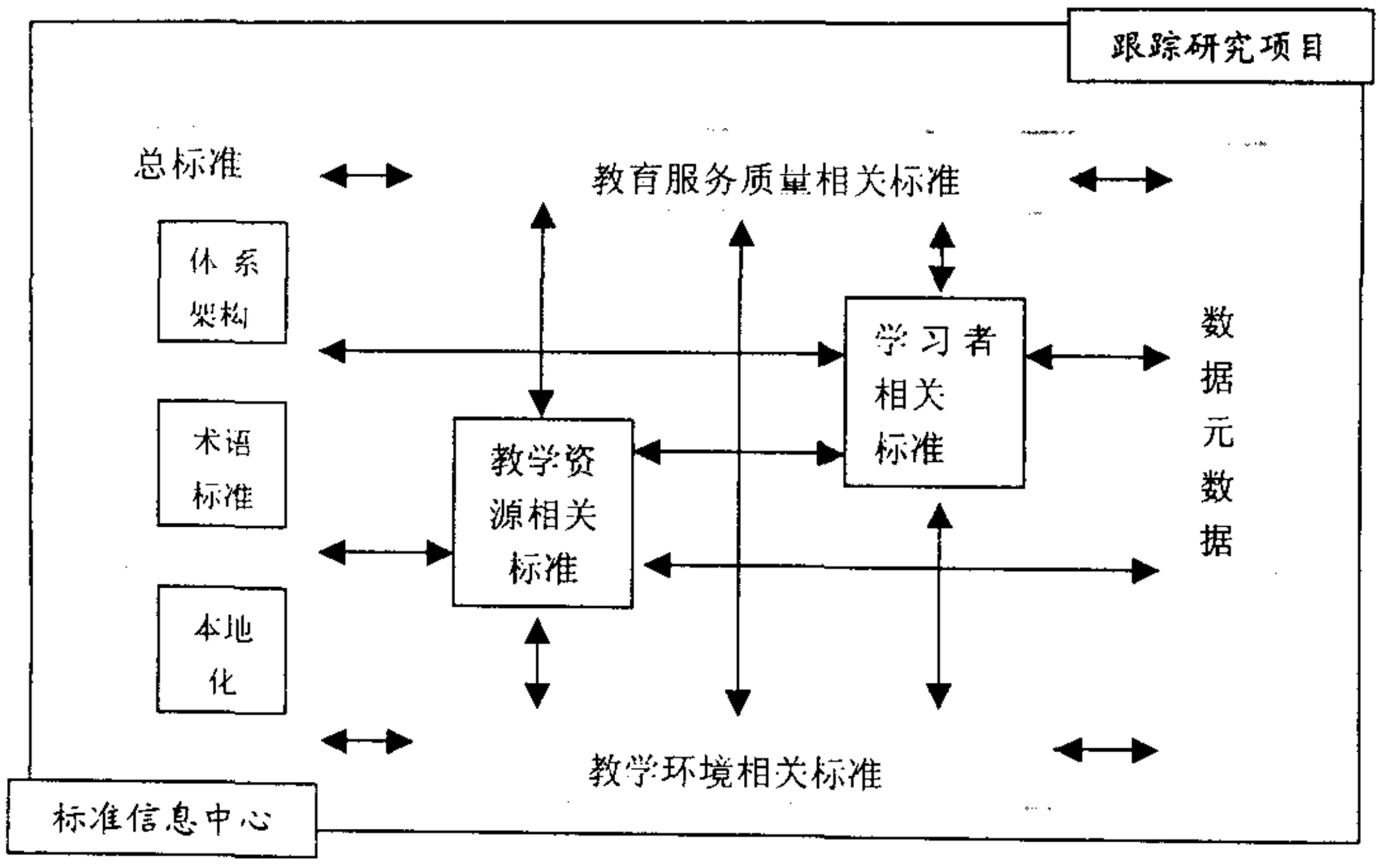

图 1.1 DLTS 的标准类型及其相互关联

1.3 课题的研究背景

本课题是华中科技大学电子与信息工程系图像教研室与美国FUTTON公司合作进行的开发远程软件教学系统的课题。该系统是在 *Web* 平台上，对 *Windows* 32 位应用程序进行专家指导的交互式远程教育系统，简称 *EPSS*（*Electronic Performance Support System*）。

1.3.1 应用软件远程教学的发展状况

目前应用软件的远程教学主要有以下几个途径：一是将软件的使用方法编写成 *HTML* 页面，配以图片、声音等媒介，用户通过 *Web* 浏览器访问这些页面来进行学习。这是目前使用最广泛的模式；另一个就是 *CAI* 方式（一般是记录操作的视频文件），由专门人员为学习对象编制教学课件，用户从网站上下载这些课件，在客户端运行来进行学习。这些方法的优势是，*Internet* 使得教学内容能够方便地更新，多媒体的应用使得教学更加生动。然而，应用软件的教学不同于其他理论知识的教学，它更加强调学习者的操作。当软件初学者看到一个步骤复杂的功能介绍时，由于对

软件的不熟悉往往会觉得不知所云。这时如果用户能亲自运行应用软件，由教学系统对用户的每步操作进行跟踪指导，将会大大的提高教学的效果。本软件教学系统的目的就是实现这一功能。

1.3.2 前景的预测

目前我国远程教育的网站有很多，但绝大多数都停留在 *HTML* 页面，配以音频、视频等媒介，重论述，轻实践。本课题中提出的软件远程教学体系结构根据软件教学的特点，以对用户的实际操作进行跟踪指导为指导思想，真正实现专家指导信息的实时反馈。另外传统 CAI 课件传输数据量很大，一个包含了音频和视频信息的教学课件的一般都是以 *M* 字节为单位计量大小；而本系统中，教学课件基本上是文字信息，再加上一些图片信息，其大小是以 *K* 字节单位计量的，在网络传输方面无疑具有很大优势。由于同类系统在技术上实现有相当的难度，目前在国内仍处于研究阶段，因此，本课题有着良好的应用前景。

1.4 多层软件结构

这一节我们介绍一下应用系统的软件结构。根据用户界面与后台数据之间层次数目的不同，可以把应用结构分为单层、两层、三层或多层软件结构。在 *EPSS* 系统中，我们将采用多层软件结构。首先我们从软件结构的发展过程讨论这三种结构的基本概念，然后我们对多层软件结构作进一步介绍，最后总结多层结构的一些优点。

1.4.1 应用结构的发展[8]

随着计算机技术和网络技术的飞速发展，计算机软件的复杂程度在不断增加，系统结构在软件设计和开发过程中所起的作用越来越重要。早期的单层应用软件通常包括了所有的用户界面、业务规则以及数据处理，应用的数据有可能存放在远程机器上，但访问数据的逻辑被包含在应用程序中。

单层应用软件由于包含了所有的应用逻辑，从后台最基本的数据处理，到前台对用户的响应以及处理结果的显示，所以应用软件往往比较庞大。给软件的设计、

开发、测试和维护以及版本更新都带来了很多不利因素。

对于网络应用或分布式应用，单层结构显然是不能满足要求了。于是很自然地，把应用程序分为了两个部分，客户端部分和服务器端部分，从而形成了两层结构，有时候也称为客户-服务器结构（*CS*）。在这种两层结构地应用中，用户界面和业务规则在应用的客户端；数据维护在服务器端实现，通常由另一个独立的应用程序来完成，比如，*SOL Server* 或 *Oracle* 等数据库系统。

从两层结构到三层结构或多层结构的演变也是个很自然的过程。在两层结构中，如果把业务逻辑放在客户端，往往使得客户端非常笨重；如果把业务逻辑放在服务器层，则往往难以在数据库系统中实现复杂的应用逻辑。因此在三层或多层结构中，把业务逻辑单独提取出来，构成中间的一层或多层，形成真正的分布式应用系统。

在多层应用结构（包括三层）中，客户层只提供应用的用户界面，它根据用户的操作调用相应的业务逻辑，它永远不会直接访问后台数据库，有时我们也把客户层称为表示层；业务逻辑是应用系统的关键所在，它负责处理所有的用户请求，并且把处理结果返回给表现层；服务器层仍然提供数据库支持，我们也可以用一些简单的存储过程来维护数据，这一层也称为数据层。三层应用结构如图 1.2 所示。

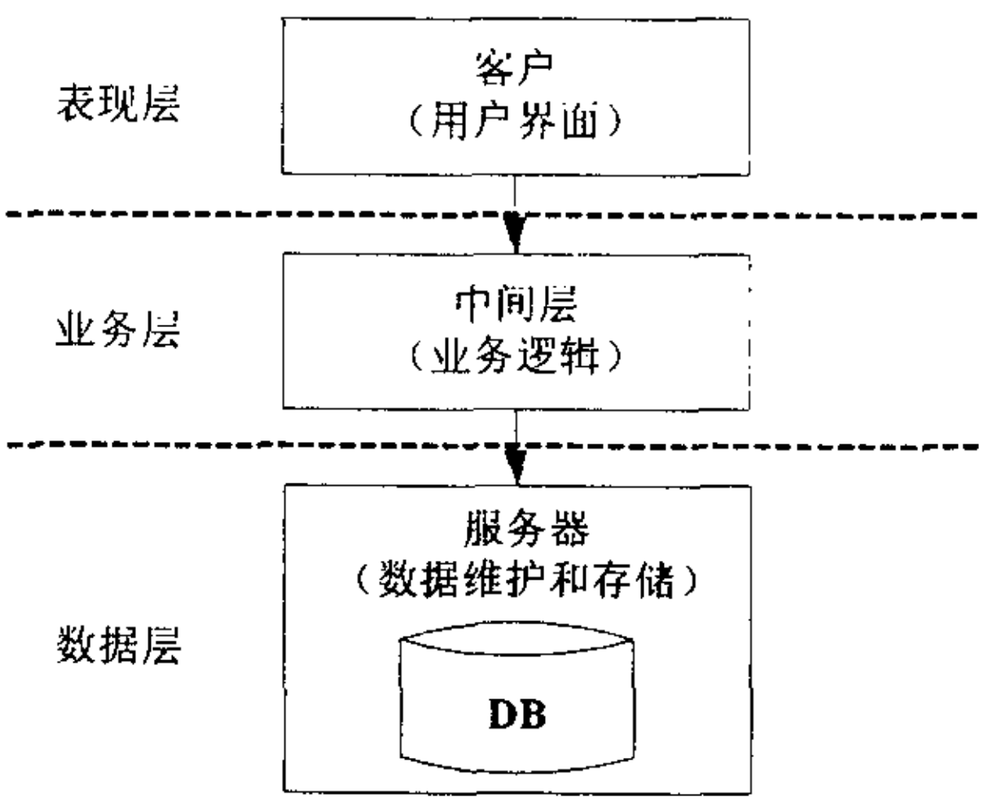

图1.2 三层应用结构示意图

多层应用结构要求层与层之间必须有明确的接口定义，从而保证多层之间可以协作完成应用任务。业务层作为表现层和数据层的中间服务层，必须保证所有的业务逻辑被正确处理。多层结构比两层结构具有更大的灵活性，首先，三层可以运行在不同的机器上，可以使用高配置的计算机来运行业务层；如果应用的数据量很大，我们可以采用分布式的数据库作为应用的数据存储结构。其次，只要层与层之间的

接口保持不变，那么某一层的变化不会影响到其他层。

当然，多层应用的开发需要各种工具来支持才可能实现。微软开发的 *COM*、*DCOM* 和 *MTS*（微软事务处理服务器）等技术为我们提供了一个非常好的多层结构的基础平台[8]，目前在 *Windows* 操作系统中，基于 *COM*、*DCOM* 的多层应用软件技术得到了非常广泛的应用。在基于 *COM*、*DCOM* 的多层结构平台中，层与层之间可通过 *COM* 接口联系起来，他们可以运行在不同的进程。甚至不同的机器上，*COM* 和 *DCOM* 提供了进程透明和位置透明特性。*MTS* 又为中间的业务层提供了统一的配置和管理环境，我们可以把业务逻辑封装到 *MTS* 对象中，然后由 *MTS* 负责运行和包装这些业务组件。

1.4.2 基于 *COM*、*DCOM* 的多层应用软件结构

多层结构建立了一种基本的软件建模思想，它主要是针对分布式应用软件系统。*COM* 为多层应用软件结构提供了强有力的支持，利用 *COM*、*DCOM* 和 *MTS*，我们可以给出一种典型的多层软件实现方案。如图 1.3 所示[8]。

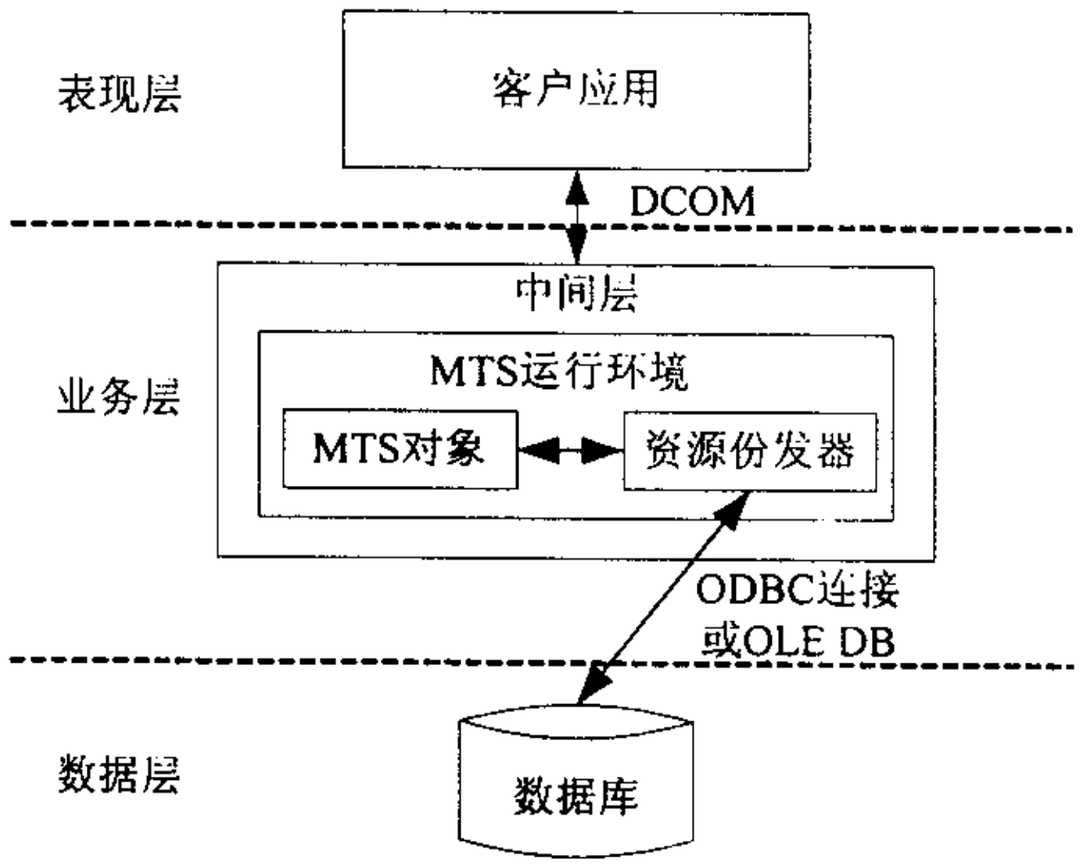

图1.3 基于COM的三层应用结构

在分布式应用系统中，直接面对客户的表现层实现的功能比较简单，它只是把用户的操作转化成一些业务指令，然后调用业务逻辑层的接口函数，当业务层返回结果后，它再把结果以可视化的方式展现给用户。因此，客户机上的应用程序往往比较简单，对机器配置要求也比较低，只要求客户机能够通过 *DCOM* 访问中间业务

层。

在中间层，*MTS* 可以把所有的业务组件管理起来，并提供运行环境。使用 *MTS* 作为中间层的基础平台，除了简化编程模型外，我们还可以获得广泛的灵活性[17]。首先，当应用的规模增长时，我们可以在多个机器上安装 *MTS*，并配置和运行 *MTS* 组件，无需修改代码就可以适应应用规模的增长。其次，由于组件本身的独立性，随着应用发展的需要，改变业务规则意味着只需改变有关的 *MTS* 组件即可，而不必改变整个应用。*COM* 和 *MTS* 的优势可以在中间层得到充分的体现，包括版本升级、应用维护等。

数据层通常由数据库系统来实现，在多层机构模型中，业务逻辑层与数据层不一定在同一台机器上，他们有可能分布在不同的服务器上。对于应用系统，数据库和访问接口的选择是设计的关键要素。*ODBC* 是访问数据库系统公认的标准接口，*MTS* 也把 *ODBC* 连接作为资源管理起来。目前微软推出了新的数据访问标准-*OLE DB/ADO*[10]。*OLE DB* 是一个比 *ODBC* 更为先进的数据访问接口，它以 *COM* 接口的形式统一了对各种数据访问的标准，包括关系数据库和非关系数据源；而 *ADO* 则是建立在 *OLE DB* 基础上的一套自动化接口，适用于高级语言或脚本语言访问各种数据源。*OLE DB* 和 *ADO* 构成了一致数据访问（*UDA*）机制。如图 1.4 所示。

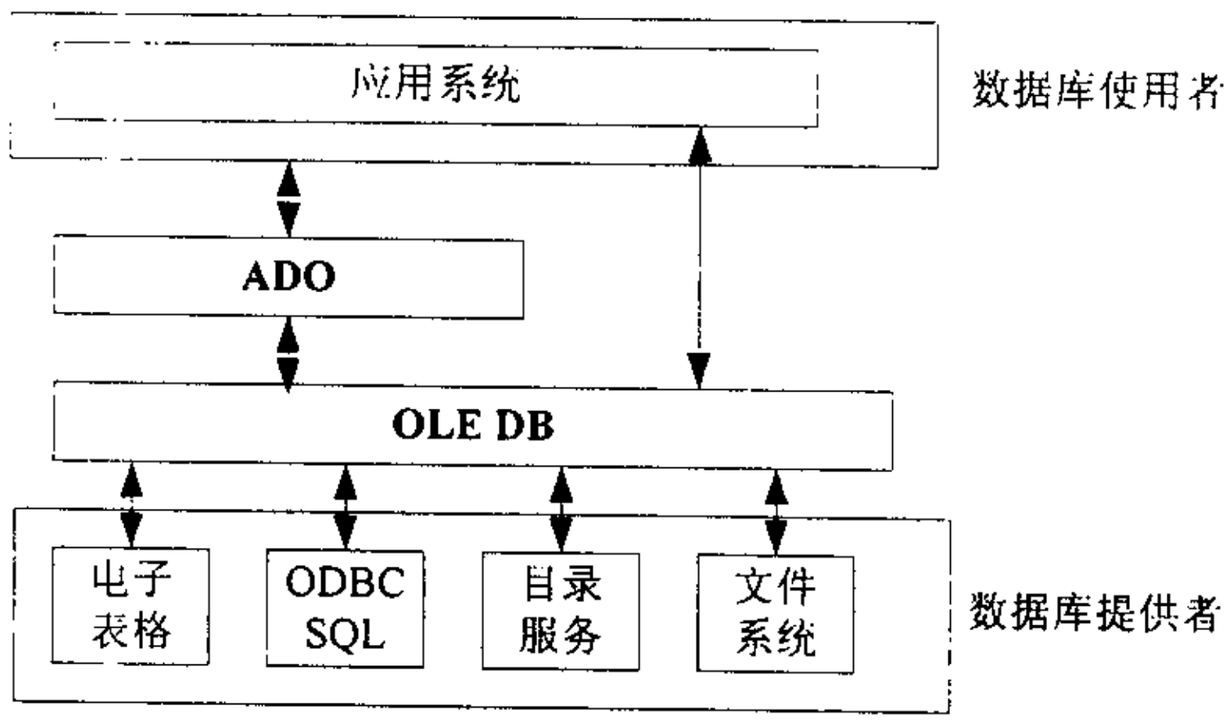

图1.4 OLE DB和ADO结构图

COM 在 *Windows* 平台上已经根深蒂固了，并且正在向其他操作系统扩展。在 *UDA* 和 *MTS* 的支持下，我们在开发基于 *COM* 的分布式应用系统时，既不需要考虑网络调用，也不需要考虑数据源的差异，大大方便了系统的开发。

1.4.3 基于 *COM*、*DCOM* 的多层应用结构的优点[9]

我们在了解了基于 *COM* 的多层应用的基本结构之后，现在结合 *COM* 和 *MTS* 的一些特性，对多层结构的特性作一个简要说明。

1) 可伸缩性。多层结构的伸缩性来自 *DCOM* 和 *MTS* 的可伸缩性，当应用的规模增大时，通过系统结构的配置而不必修改代码就可以适应新的应用要求，而且利用这种伸缩性我们可以在低档的计算机上实现复杂的分布式应用系统。
2) 可配置性。利用 *MTS* 管理工具，我们可以很方便地改变应用的配置，包括应用的执行功能信息、*MTS* 组件的属性等。
3) 可靠性。因为多层结构把一个复杂的应用拆分成一些小的组件，并且这些组件依赖于操作系统提供的支持，所以应用系统的可靠性易于得到保证。
4) *MTS* 事务和安全模型。多层结构的应用系统可以直接得益于 *MTS* 的事务和安全模型，从而简化应用的业务处理规则。
5) 充分体现了软件集成的思想。我们不仅可以设计和开发一些组件，而且可以使用大量已进入市场的组件产品，或者使用以前积累下来的组件库中的组件，从而缩短开发周期，提高组件的重用率。

鉴于以上优点，*EPSS* 系统从应用结构上将采用基于 *COM*、*DCOM* 的多层应用系统结构，其具体设计将在第二章详细介绍。

1.5 数据挖掘（*Data mining*）和知识发现（*KDD*）

近年来，随着科学技术的不断发展，数据库规模日益扩大，复杂程度不断增长，从大量数据中及时获取有利于系统改进的决策信息显得越来越重要。目前作为信息处理新发展阶段的决策支持系统尚处于初级阶段，由于在数据集成、数据建模、接口设计和数据分析等方面的局限，很多决策支持系统并不能很好地为决策者提供决策信息。90 年代初，数据挖掘技术的发展给决策支持系统的发展带来了新的契机。

1.5.1 数据挖掘（*Data mining*）技术简介

数据挖掘是在一些事实或观察数据集合中寻找模式的决策支持过程[11]。它具有以下特点：数据挖掘要求处理大量的数据，待处理的数据规模可能达到 *GB*、*TB* 甚

至更大；由于用户不能形成精确的查询要求，依靠数据挖掘技术为用户寻找他可能感兴趣的信息；它把大量的原始数据转化成有价值的知识，用于描述过去的趋势和预测未来的趋势；数据量增长快速，许多数据来不及分析就过时了，而数据挖掘能快速地做出响应，提供决策支持信息。

从应用深度上，我们将数据挖掘划分为三个层次空间[13]：（1）数据空间。它利用现有数据库管理系统地查询检索和报表功能，进行基于关键字的决策查询，实现联机事务处理（*OLTP*）。（2）聚合空间。利用聚合运算（相加、求平均、取最大值、取最小值等），结合多维分析和统计分析，实现在线分析处理（*OLAP*），以提供决策参考的统计分析数据。（3）影响空间。按照相似性的聚类、差异性的分类方法，发现关联性及结构模式、顺序模式，建立预测模型，从数据库或大量数据记录中发现隐含的有用信息，这是在更深层次上的知识发现，是数据挖掘实质性内涵。

以上数据挖掘的各个层次空间反映了不同级别的查询请求，这种划分有利于知识的逐步提取，知识的提取过程即为决策支持过程。在传统的决策支持系统中，知识库的知识和规则是由专家或程序人员建立的，由外部输入，而数据挖掘是从系统内部自动获取知识的过程。同数据库管理系统查询检索的信息相比，数据挖掘的知识是隐含的、精练的和高水平的。

1.5.2 知识发现（*KDD*）简介

数据挖掘的知识通常表现为概念、规则、规律、模式、约束和可视化等形式[12]。这些知识经过解释后可以直接在实际系统中应用以辅导决策过程，或者提供给领域专家以修正专家已有的知识体系，也可以作为新的知识转存到应用系统的知识库中。知识发现的过程是利用各种知识发现算法从数据库中发现、表达、更新和解释有关知识的过程，它主要包括一下几个方面的问题：

1) 知识的发现。数据关联是数据库中存在的一类重要的可被发现的知识[14]。若两个或多个变量之间取值之间存在某种规律性，则称为关联。知识发现目的就是采用关联规则归纳技术找出数据库中数据项（属性、变量）之间内在隐藏的关联网。
2) 不确定知识的表达。人们对事务的判断、预测和决策等是在问题域的信息不完全、不精确或者模糊的条件下进行的。粗糙集（*Rough Set*）理论作为一种智能数据决策分析工具，被应用于这种不确定性的知识获取和知识表达中[43]。
3) 知识的更新。决策分析必须不断地从样本模式中学习专家用于决策地定性的、经

验性的知识，从而保证系统不断地获取新的知识以及对系统中拥有的网络知识进行更新和完善。

4) 知识的表达和解释。为使用户能够理解所发现的知识，数据可视化采用直观的方式将信息模式、数据的关联或趋势多维地呈现给决策人员，使之能深入到数据的结构中了解数据的状况和数据的内在规律。

本文的第四章将针对在 *EPSS* 系统中进行决策分析进行详细的理论分析，并从实际应用的角度结合粗糙集理论讨论如何在 *EPSS* 系统中实现数据挖掘和知识发现。

2 *EPSS* 系统设计

本章主要介绍 *EPSS* 系统的方案设计。本章首先介绍了 *EPSS* 系统整个系统的框架，然后分别介绍了客户端应用程序，组件逻辑等主要部分的设计方案。其中客户端应用程序的体系结构设计参考了 *IEEE P1484.1* 标准草案，即 *Learning Technology Systems Architecture*（*LTSA*）；关键技术中提出了屏幕取词方案；对数据库的应用逻辑操作使用 *DCOM* 分布式组件技术。

2.1 *EPSS* 系统框架

EPSS（*Electronic Performance Support System*）作为一个远程软件教学系统，其主要功能是：用户通过互联网选择学习对象（即应用软件），*EPSS* 系统对用户的实际操作进行跟踪指导，以最直观的方式达到软件教学的目的。

EPSS 系统工作的基本原理是：用户通过客户端应用程序（内嵌 Web 浏览器）登录到指定的教学网站，选择学习对象和具体课件。当用户选定某学习对象后，客户端应用程序会在本地自动启动相应的应用软件（如 WORD 等），然后从远端学习资源数据库中获得课件的内容（即应用程序操作步骤），并通过客户端应用程序的向导窗口向学习者提供操作指示，用户按照辅导者的指导操作学习对象。用户对学习对象的每一个步操作，都是在客户端应用程序的监控下完成的，其操作的正确与否，系统会及时给出信息，使用户能按照正确的步骤对学习对象进行学习。

EPSS 系统采用三层应用模式，其系统框图如图 2.1 所示，整个系统主要由三部分组成：

1) 数据库服务器：负责存储各种数据资料，包括课程内容、用户信息、权限管理、访问统计等。

   课程内容按照章（Tutorial）、节（Lesson）、页（Page）的方式进行分类组织，一章对应一个软件学习对象（如 WORD 等），一节则对应一个能完成一定功能的操作步骤集（如打开文件、打印等），一页则对应一个具体操作步骤。用户信息包括作者（Author）和学习者（Learner）两部分，作者是课程的制作者，不同的课程设计者可能不同，通过权限管理可以有效的约束不同作者的权限，学习者是已经注册登记的软件学习用户。权限管理指定了作者、学习者以及数据库管理人员的权限。访问统计则负责统计用户的具体访问情况。

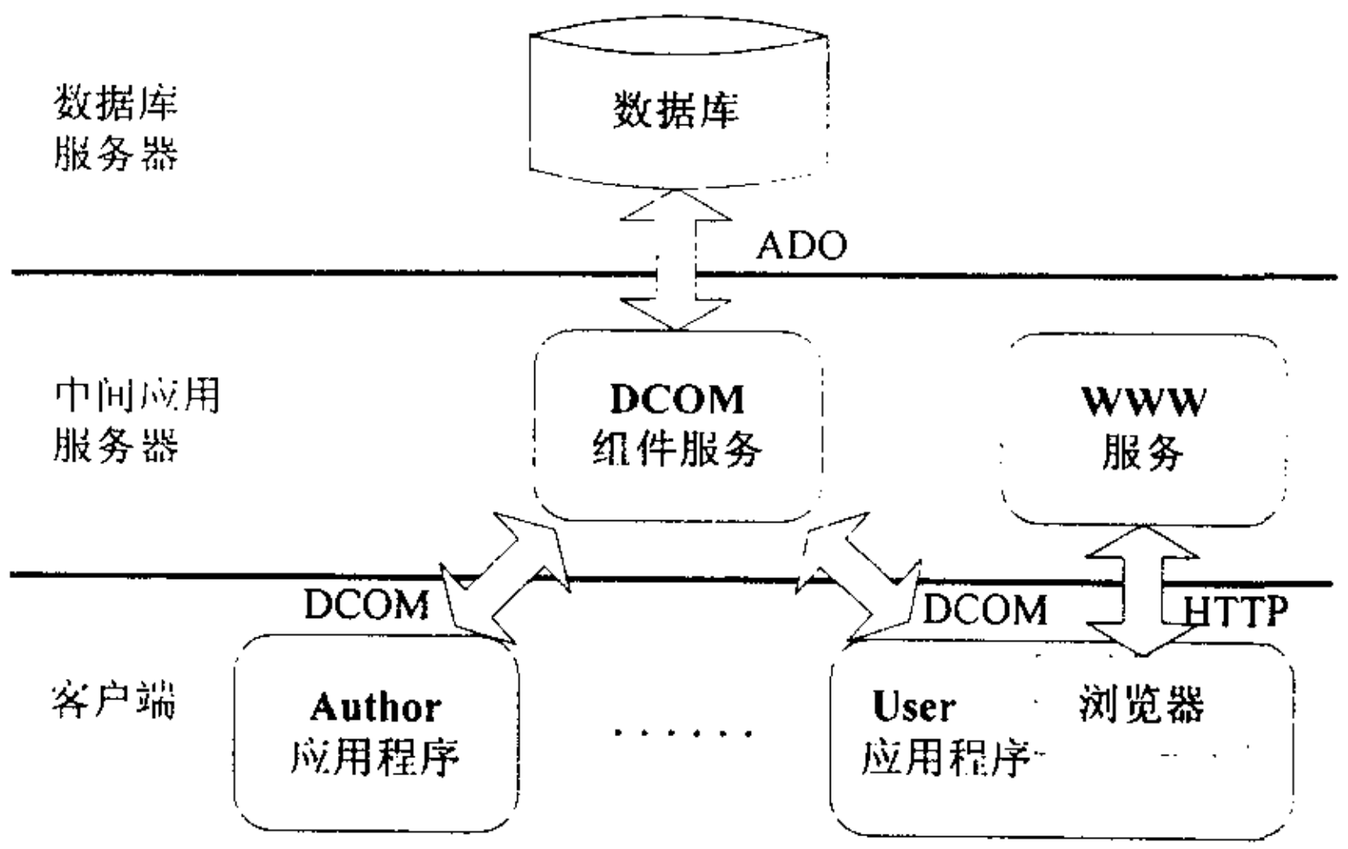

图 2.1 *EPSS*系统框图

2) 中间应用服务器：包括组件服务器和 WWW 服务器。

组件服务器：用于管理各种 *DCOM*（分布式 *COM*）应用组件，包括应用对象组件和数据库访问管理组件。用户端各种应用程序通过调用 *DCOM* 组件的方法来实现逻辑处理和数据访问，其中包括课件制作和设计、用户的远程数据库访问、权限的管理等，在下文中将作详细介绍。

WWW 服务器：提供服务网站的相关信息。

3) 客户端应用程序：主要包括 Author 应用程序和 User 应用程序。另外还包括报表应用程序等辅助应用程序。

Author 应用程序（Author Tool）：负责编辑排版数据库内的课程内容，包括每一个学习步骤的图片、文字提示信息等。在编排的过程中，课件制作者用鼠标、键盘操作实际的学习对象，Author 应用程序捕获每个操作步骤的鼠标、键盘信息，对某些重要的步骤还要截获相应的操作界面（压缩成 *JPEG* 格式），将这些数据通过 *DCOM* 组件存入数据库中，作为用户的学习内容。

User 应用程序（User Tool）：内嵌浏览器，访问 WWW 服务器，得到课程学习的相关信息。用户选定学习课件后，通过 *DCOM* 组件从数据库中获得相应的学习内容，当用户操作指定的学习对象时，监控用户的操作行为，获得鼠标键盘信息，并与数据库里的内容进行比较，根据比较的结果给出提示信息。

从通用性考虑，*EPSS* 系统的学习对象定位为任何 Win32 应用程序。根据下一章介绍的系统技术实现的介绍可知，*EPSS* 系统的实现是平台相关的，实际课题中要求

用户使用的 *Windows* 95/98 操作系统。所有客户端应用程序将使用 *Microsoft Visual* C++ 6.0 开发，某些底层处理函数使用汇编语言。下面我们从 User 端应用程序开始对整个系统做详细分析。

2.2 User 端应用程序设计

User 端应用程序在体系结构设计上参照了 *IEEE P1484.1* 标准草案：*Learning Technology Systems Architecture*（*LTSA*）；在技术实现上提出屏幕取词方案。我们首先介绍体系结构的设计。

2.2.1 *LTSA* 体系结构介绍

LTSA 标准草案的制定和修订工作是在 *IEEE Computer Society Learning Technology Standards Committee*（*LTSC*）主持下，由若干个工作小组开展进行的。由于其权威性和完备性，该标准被很多科研组织借鉴。该标准主要规范了基于信息技术的教育、学习和培训系统的高层体系结构和系统的组成模块，使研究人员能更好的理解这类系统及其子系统的工作模式，以及不同系统之间的相互关系。同时该标准的有很好的通用性，适用于各种教学方法、各种培训内容、各种不同的文化以及各种开发平台[18]。

除了 *LTSA* 标准，*LTSC* 还制定了 P1848.*一系列有关开发由计算机实现的教学系统的标准草案[18][19][20]，包括与学习者相关的 P1848.2、P1848.13 和 P1848.20 草案；与教学课件内容相关的 P1848.10、P1848.6 和 P1848.17 草案；与数据和元数据（*Metadata*）相关的 P1848.12、P1848.9、P1848.14 和 P1848.15 草案；有关系统管理的 P1848.11、P1848.18 和 P1848.7 草案。由于 *LTSC* 的一系列草案还在不断地修改完善，因此并没有成为国际标准。其中 *LTSA* 草案由于是从整个体系结构上进行规范化，因此是其他所有草案的基础，并且对 *EPSS* 系统有很好的借鉴作用。

LTSA 规定了任何按此规范设计的教学系统由 5 个抽象层次组成。表 2.1 对这 5 个层次进行了简要说明。

表 2.1 *LSTA* 抽象层次表

层次	名称	简要说明
1	**Learner and Environment Interactions**	负责传输、交换、格式化信息以完成与学习者交互的功能；
2	**Learner-Related Design Features**	负责处理学习者对系统的影响（学习者学习时对系统的要求等）；
3	**System Components**	描述了以模块为基础的系统结构；
4	**Implementation Perspectives and Priorities**	从各种不同的角度描述了系统结构中各模块的实现；
5	**Operational Components and Interoperability — coding、APIs、protocols**	描述具有互操作性系统接口，即程序代码、API 函数和通信协议等；

其中第三层介绍了一个教学系统的通用体系结构，对 EPSS 系统客户端应用程序的设计有很好的指导意义。该层次的具体结构如图 2.2 所示：

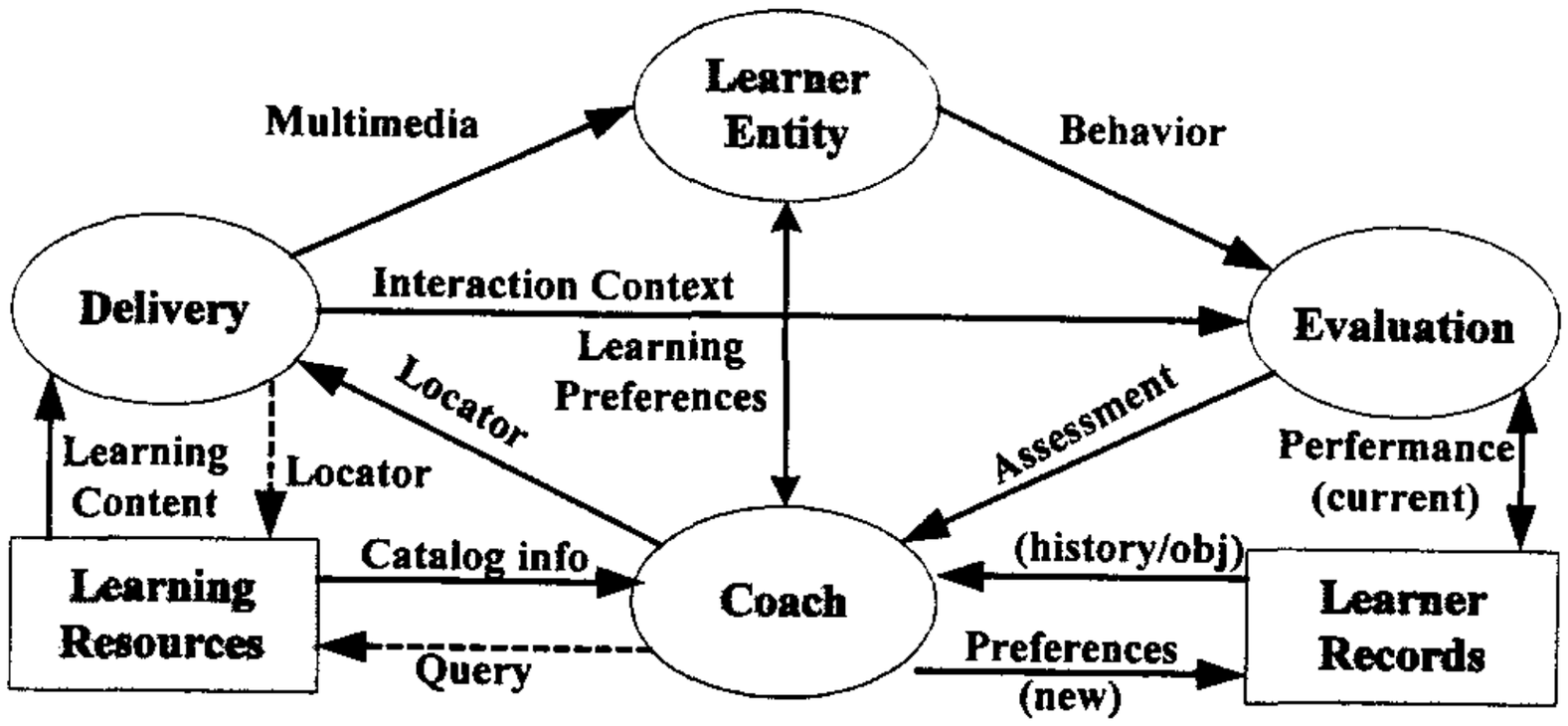

图2.2 LTSA系统模块结构图

在 LTSA 模块体系结构中定义了四个过程体：学习者、评估者、指导者和传递者（对应 *Learner Entity*，*Evaluation*，*Coach*，*Delivery*）；两个数据库：学习资源数据库和学习者信息数据库；以及这些模块之间的十三条数据流[18]。简单的说，其基本工作流程是：

1) 学习者和指导者之间协商学习的类型、方法和策略（称为学习参数，即 *Learning Preferences*）；

2) 评估者分析传递者发送的交互信息并且观察学习者的实际操作；

3) 评价者生成评价信息（即 *Assessment*）和学习者的当前学习情况信息（即 *Current Performance*）；
4) 学习情况信息被存储到学习者信息数据库；
5) 指导者综合学习者的评价信息、过去的学习情况和学习参数等各方面的信息；
6) 指导者在学习资源中查找恰当的学习内容；
7) 指导者根据查询结果索引决定学习的内容安排，并通知发送者学习内容在数据库中的位置（*Locator*）；
8) 发送者根据内容安排从学习资源数据库中取出学习内容，以多媒体的形式传给学习者。

其中各过程体、数据库和数据流在 *LTSA* 标准草案中都有详细的规范，这里不作详细介绍。由于作为一个 *IEEE* 标准，*LTSA* 必须要保证其通用性和完备性，针对不同的应用实际，其中某些模块可能是不必要的，因此在 *EPSS* 系统实际应用中，我们在遵循 *LTSA* 体系结构的一些基本原则的基础上，做出了相应的修改。

2.2.2 User 端应用程序体系结构

考虑到实际应用的要求，在 EPSS 系统中，User 端应用程序的体系结构如图 2.3 所示。

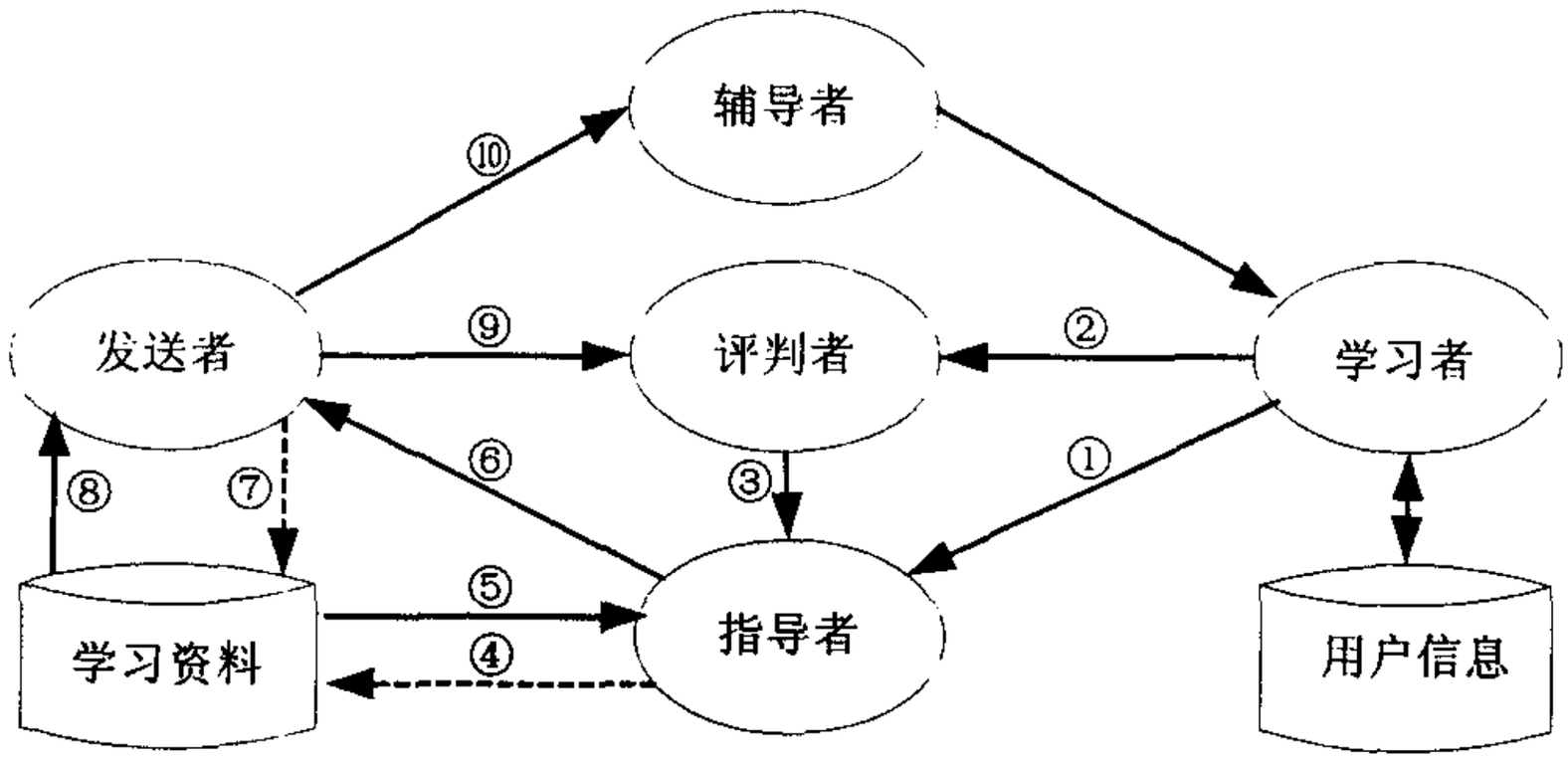

图2.3 User端应用程序体系结构图

这里定义了 5 个基本模块：

1) 学习者：表示单一的用户对象，通过 User 端应用程序的人机交互界面传达其学

习意图，同时按照辅导者提供的指导信息操作学习对象。

2) 评判者：这个模块接受用户的执行动作信息，同时分析判断用户操作正确与否，将评判结果发送给指导者模块。

3) 指导者：该模块是实现类似人类教师的教学功能。它在学习者开始学习之前，通过与学习者的一系列的简单对话交流，获得学习者的有关知识背景，形成一个相对比较适合于该学习者的学习内容和学习过程，以提高学习效率，在学习者的学习过程中，通过综合评判结果信息以及学习者的学习要求（例如：用户要求学习WORD 的打印功能等），调整相应的学习内容，选择最适合学习者的学习材料，并向发送者模块提交相应的信息。

4) 发送者：该模块是一个信息组织和传递模块，完成将学习内容从学习资源数据库中取出并以某种媒体表现形式发送给辅导者或评判者。

5) 辅导者：用一个类似 *Microsoft Agent* 的窗口对话框，将指导用户操作的信息传达给用户。

学习资料在学习资源数据库中，其内容由 Author 端应用程序定制，形成各个学习对象的章（Tutorial）、节（Lesson）和页（Page）。用户信息数据库记录学习者的注册信息、登陆访问信息等。

各模块之间主要有 10 条通信数据流，下面结合整个 User 应用程序的工作流程进行详细说明。其基本工作流程是：

1) 学习者和指导者之间协商学习的类型、方法和策略（例如：用户选择学习什么应用软件等），通知指导者要学习哪一节（Lesson），通过数据流①传递，目的是更好地促进学生的学习。

2) 评判者对学习者的实际操作进行跟踪和观察，即截获用户的对目标学习对象的操作，包括鼠标、键盘操作等，通过数据流②传递。

3) 评判者生成评判信息，即将用户当前的实际操作与正确操作步骤进行比较，判断是否正确，把比较结果通过数据流③传递给指导者。

4) 指导者综合评价结果信息以及学习者的学习要求形成对学习资料数据库的查询请求。

5) 指导者在学习资料数据库中查找恰当的学习内容，通过数据流④传递，查找下一步将要进行的操作。

6) 指导者根据查询结果索引（数据流⑤）决定下一步给学习者发送哪一页（Page），并通知发送者（数据流⑥）。

7) 发送者根据指导者的指示向学习资料数据库发出请求(数据流⑦),取出页(Page)内容（数据流⑧），然后将该页操作的正确结果通知评判者（数据流⑨），供评判者做下一步判断使用，最后把页内容传送给辅导者（数据流⑩）。
8) 辅导者接受到页内容后，做分离处理，将下一步操作的指导信息通过窗口对话框通知用户，同时修改用户应用程序内嵌浏览器的内容，显示该步操作的图例和详细说明。

2.2.3 User 端应用程序实现的技术方案

根据上一节介绍的User端应用程序的体系结构,其技术实现有两个主要的方面：一是数据库访问。前面已说明将使用 *DCOM* 组件技术来实现，后文将详细介绍；二是对用户当前的实际操作进行跟踪。在 *Windows* 操作系统中，用户操作主要包括鼠标操作和键盘操作，而监视鼠标操作并从中分析得出用户在进行什么操作是技术实现的难点。

首先，由于学习对象和 User 端应用程序是分属于两个进程，所以监视用户对学习对象的操作是跨进程的过程。这样只能使用 *Windows* 的 *HOOK*（钩子）才能截获鼠标或键盘在其他应用程序中的使用情况。*HOOK*（钩子）是 *Windows* 操作系统中非常重要的系统接口，因为用它可以截获并处理送给到任意应用程序的消息[21]。由于 *Windows* 操作系统是建立在消息驱动的机制上的,因此这是非常有用的应用接口。

在本系统中，要实现监视学习者对目标程序进行的操作，并且判断该操作正确与否，有以下几种可能的方式：

1) 截获鼠标的位置。用 *HOOK* 可以很方便地获得每次鼠标事件所发出的消息，并从中分析得到鼠标的坐标。因此在制作课件时,可以由课件制作者操作学习对象，记录下每步正确操作鼠标在屏幕上的点击位置，并根据该坐标确定一个正确区域；当用户实际操作学习对象进行某一步操作时，比较此时鼠标位置是否在相应的正确区域内，就可以判断用户操作是否正确。但是使用这种方法首先要保证学习对象在 User 端和 Author 端运行时各窗体对象的坐标位置要完全一致。这显然是不可能的，比如一般 *Windows* 应用程序的菜单栏等都是可移动的，因此无法保证坐标位置完全一致。另外对于某些对话框窗体（例如：有滚动条的组合框（*Combo box*）控件），根本无法确定正确的鼠标点击位置。所以这种方法是不可行的。

2) 使用 *HOOK* 程序拦截发往应用程序的消息，通过分析消息来判定用户操作了什么对象。判定方法是从消息中提取发消息对象的“特征信息”，所谓“特征信息”即能表征发消息对象的信息，如按钮的文字就是按钮对象的“特征信息”，菜单项的文字就是菜单项的“特征信息”，由“特征信息”来确定用户的每一步操作。但通过分析消息得到对象的“特征信息”这种方法是有很大局限性的，这是因为各个应用程序中传递使用的消息是大相径庭的。这里以菜单栏为例：在有的应用程序中，选中菜单，菜单窗体会发出 *WM_SELECT* 消息，从这个消息中可以分析得出菜单文字；而在 Word 中选中菜单，菜单对象发出的信息根本就没有菜单项的文字信息。考虑到 *EPSS* 系统的通用性，这种方法也是不可行的。

3) 截获鼠标点击处的文字。利用金山词霸等屏幕翻译软件所使用的屏幕取词技术我们可以得到鼠标点击处的文字，以此文字作为操作正确与否的判定依据。使用这种方法我们可以克服以上两种方法的缺点。首先它不需要保证应用程序运行时坐标位置完全一致，不管窗体对象在什么位置，只要鼠标点击在窗体的文字上，我们就能截获并做出比较；这种方法还避免了分析各种不同消息的繁琐步骤，任何鼠标点击处的文字，都使用同一种方法进行截获。综合以上因素，在 *EPSS* 系统中我们决定使用这种方法来截获学习者对目标程序的操作。

在实际应用中，考虑到对应用软件的操作主要集中在菜单和对话框中，屏幕取词这种方法可以适用于绝大多数情况。但也有例外，比如对点击工具栏等点击图标的操作就无能为力，这类操作 *EPSS* 系统还无法实现教学；另外，不排除在操作过程中，在某个对话框中存在两个窗体对象有相同的文字，这样就无法判断操作的正确性了，对于这种情况，在制作课件时由课件制作者给予相应的提示，以避免用户发生混淆。

屏幕取词技术是整个系统实现的难点，其具体实现原理和实现方法将在第三章给予详细介绍。

2.3 *DCOM* 组件模块设计

2.3.1 基于 *DCOM* 的系统模型

在传统的数据库信息处理应用程序中，其逻辑处理层处理的数据是直接来自数据库的多个离散信息，这种方式已经越来越不适合基于对象的应用程序的应用和开

发，基于对象的应用程序希望处理的数据以对象的封装形式出现，这种基于对象的数据封装使得应用程序具有更好的模块化、更好的独立性。

DCOM 是一种完全支持对象概念的组件对象技术，采用基于 *DCOM* 的组件对象技术来构建应用程序，可以使系统更加结构化、模块化，便于系统的小组开发和升级维护，同时 *DCOM* 使用一种基于远程过程调用（*RPC*）的标准，具有位置独立性、语言平台独立性、可伸缩性等技术优势[16]。因此，在 *EPSS* 系统中采用 *DCOM* 技术来实现各种逻辑应用。整个结构分成三层，从上到下分别为应用程序层、应用对象层和数据库访问管理层，如图 2.4 所示。

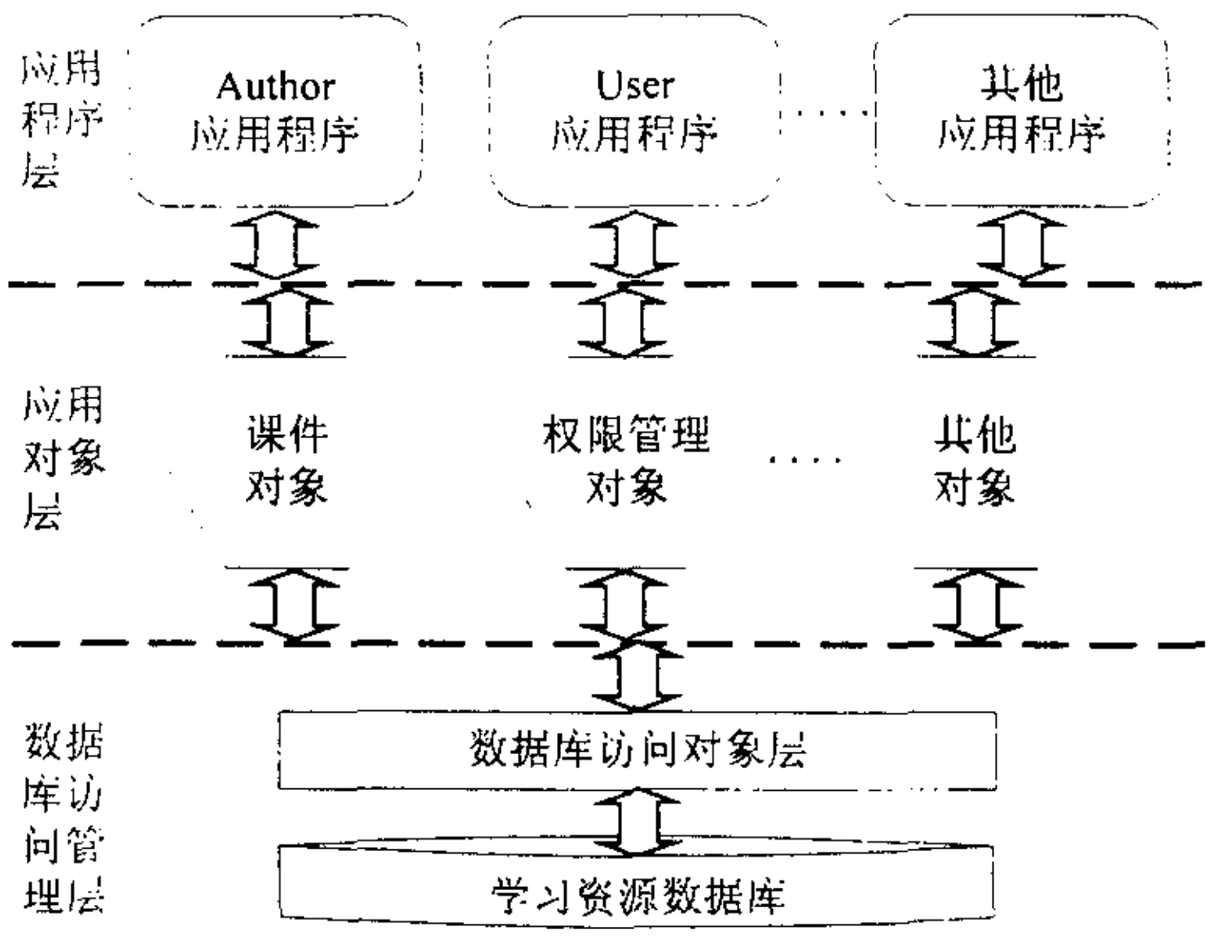

图2.4 基于DCOM的组件模型

应用程序层包括 Author 应用程序、User 应用程序以及报表应用程序等。Author 应用程序用于制作课件，User 应用程序供学习者学习课程使用。这些应用工具都是一种 *DCOM* 客户，它们通过 *DCOM* 规范访问应用对象层的接口。

应用对象层和数据库访问管理层采用 *COM/DCOM* 组件方式实现。

应用对象层包含多个应用对象（如课件对象、权限管理对象等），为了具有较好的封装性，对象由一组逻辑信息结构和操作接口组成，逻辑信息结构用于存放对相应数据库表操作的结果集，操作接口包含一组输入接口和输出接口，应用程序通过调用这些接口来访问对象，通过输入接口提交用户请求，通过输出接口访问逻辑信息结构以得到用户请求的结果，应用对象层对数据库的访问通过调用数据访问管理层对象接口来实现。

数据库访问对象层主要包括 CADOTier 对象，该对象负责与数据库的连接建立、对数据库的访问、事务处理管理以及具体的数据库参数配置（如服务器名、数据库名等）等，该对象通过 *ADO* 数据库访问技术直接对数据库进行 *SQL* 操作，并将执行操作后返回的结果递交给上层的应用对象，由应用对象对结果进行处理。由于上层所有对象均通过该对象访问数据库，为了高效地完成数据库访问工作，该对象采用基于 *MTS*（微软事务处理服务器）的 COM 组件技术来实现，*MTS* 具有以下一些重要特性[17]：

1) 并行管理：*MTS* 提供一个多线程同步管理机制，非常适合标准 *COM* 组件结构，通过这种同步管理机制，能够很好地管理与数据库的连接，降低与数据库地连接建立的时间，提高访问速度。
2) 安全性：安全性对于一个基于 *Internet* 的应用来说非常重要，*MTS* 提供了基于对象的安全模式，系统管理员可以方便的配置对象的访问权限。
3) 分布式事务处理：事务处理是保证数据一致性的重要处理，也是比较复杂的处理，*MTS* 提供了成熟的事务处理特性，可跨越不同的操作平台。通过 *MTS* 提供的 *DTC*（分布式事务协调器）系统服务，*MTS* 应用可方便地管理事务功能。

各组件均以动态连接库的形式存放在服务器或客户端，每台客户机在完成一次性 *DCOM* 组件注册后，便可以随时调用组件中的对象接口对远端数据库进行访问。*DCOM* 组件实现的方法和原理将在第三章进行详细介绍。

2.3.2 组件设计

在 *EPSS* 系统中，将编制 EpssAdmin.Author 等 15 个组件对象，在表 2.2 中列出并给予了简要说明。

表 2.2 *EPSS* 系统组件列表

编号	对象名称	功能说明
1	PageObject.Page	代表数据库中的一个页（Page），拥有该页的全部属性。实现对数据库页记录的读写功能。
2	PageObject.Pages	是 Page 对象的一个枚举集合；负责所有涉及多个页记录的数据库操作（如查询多个页记录等）
3	LessonObject.Lesson	代表数据库中的一节（Lesson），拥有该节的

		全部属性。实现对数据库节记录的读写功能。
4	LessonObject.Lessons	是 Lesson 对象的一个枚举集合；负责所有涉及多个节记录的数据库操作
5	TutorialObject.Tutorial	代表数据库中的一章（Tutorial），拥有该章的全部属性。实现对数据库章记录的读写功能。
6	TutorialObject.Tutorials	是 Tutorial 对象的一个枚举集合；负责所有涉及多个章记录的数据库操作
7	EpssObject.Right	代表数据库中的某一权限级别
8	EpssObject.Rights	是 Right 对象的一个枚举集合；负责所有涉及多个权限级别记录的数据库操作
9	EpssAdmin.Author	代表数据库中一个 Author 记录，拥有该 Author 的全部属性，实现对数据库 Author 记录的读写功能。
10	EpssAdmin.Authors	是 Author 对象的一个枚举集合；负责所有涉及多个 Author 记录的数据库操作
11	EpssAdmin.Learner	代表数据库中一个 Learner 记录，拥有该 Learner 的全部属性，实现对数据库 Learner 记录的读写功能。
12	EpssAdmin.Learners	是 Learner 对象的一个枚举集合；负责所有涉及多个 Learner 记录的数据库操作
13	CategoryObject.Category	代表课件的类别，比如（文字编辑类、系统工具类等）
14	CategoryObject.Categorys	是 Category 对象的一个枚举集合；负责所有涉及多个 Category 记录的数据库操作
15	Epsssvr.ADOTier	负责所有的数据库访问，提供对数据库进行 Insert、Delete、Update 和 Select 等基本操作的应用接口

2.3.3 使用 *ADO* 访问数据库

ActiveX 数据对象（*ADO*）是最新的数据库访问工具，它提供了访问 *OLE DB* 的

应用程序编程接口（*API*）。*ADO* 集成了数据访问对象（*DAO*）和远程数据对象（*RDO*）两者的优点，即轻量级客户端远程数据库访问（类似 *RDO*），同时提供对不是存储在传统关系数据库中的数据的访问（类似 *DAO*）[10]。在 *Visual C++*环境下开发应用程序大多使用 *ADO* 或 *OLE DB* 接口来访问数据库。

ADO 和 *OLE DB* 两者关系密切，但两者是不同的。严格的讲，*OLE DB* 是一套组件对象模型（*COM*）接口，它向应用程序提供一个统一的接口以访问存储在不同信息源中的数据，这些接口支持适用于数据源的数据库管理系统（*DBMS*）功能，使得它可以共享数据。*ADO* 是一个对象模型，它封装了访问 *OLE DB* 数据源的接口。*ADO* 可以看成是一个 *OLE DB* 生产者的客户。

如图 2.5 所示，*ADO* 对象模型并没有很强调层次结构，所有对象都可以独立地创建和使用。其中 Connection 对象负责到数据库地连接；RecordSet 对象为执行命令所返回的记录的集合；Field 为记录集的列；Command 为命令的定义，例如 *SQL* 语句；Parameter 为命令的参数；Error 为数据库操作所产生的错误信息。

在 *EPSS* 系统实际应用中，首先导入 *ADO* 的类型库，然后 Epsssvr.ADOTier 组件对象将实例化各 *ADO* 对象，实现对数据库的访问。具体实现细节这里就不再介绍了。

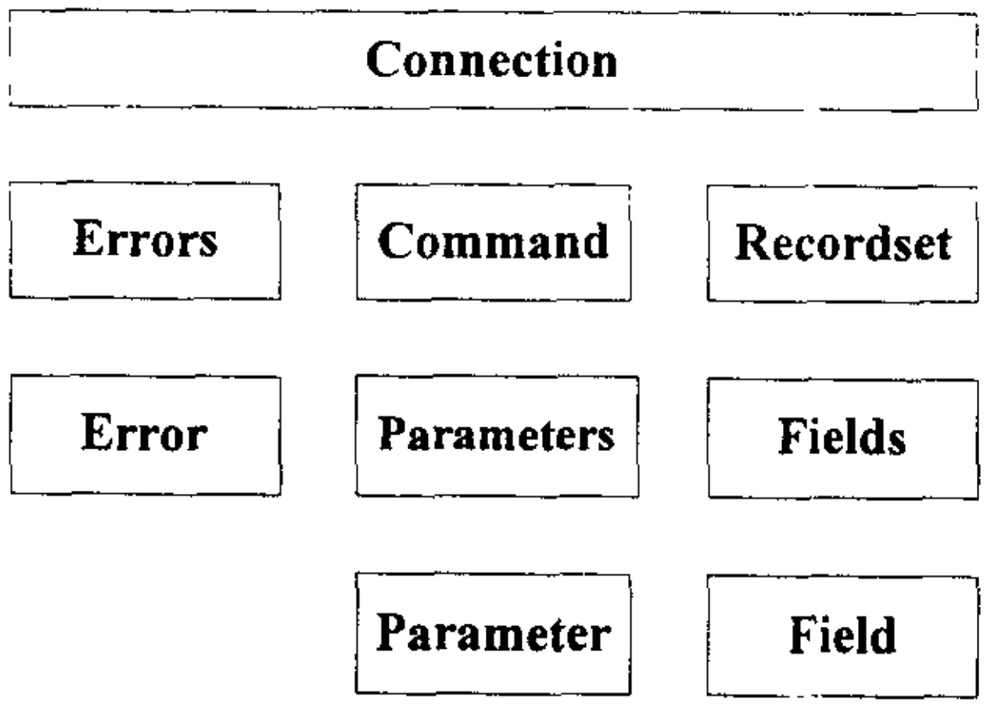

图2.5 ADO对象模型

2.4 Author 端应用程序设计

Author 端应用程序的功能主要是制作和编辑学习课件。在制作课件的过程中，课件制作者用鼠标、键盘实际操作学习对象，Author 应用程序捕获每个操作步骤的

鼠标、键盘信息，对某些重要的步骤还要截获相应的操作界面，操作完毕后，这些操作步骤将以列表的形式列在 Author 应用程序中，用户可以对这些步骤进行编辑，确认后即可将这些数据通过 *DCOM* 组件对象接口存入数据库中，作为用户的学习课件。符合权限的课件制作者还可以从数据库读取每个课件，修改课件对象的各属性值。

Author 端应用程序中同样要用到的屏幕取词技术来记录课件制作者的操作，这部分不用多作介绍了。下面介绍一下如何截获当前的操作界面：对于某些步骤，我们允许课件制作者截取相应的操作界面，以方便学习者进一步理解该操作步骤。*Windows* 操作系统提供了 *API* 函数接口使我们能获得屏幕的 *HDC*（设备描述表），根据屏幕 *HDC* 可获得屏幕各象素的 *RGB* 色彩值，这样可以拷贝得到无失真的屏幕位图图像[22][23]。对拷贝下来的位图图像，我们将使用 *JPEG* 标准进行压缩，在基本保证图像质量的情况下减少图像大小，以减小网络传输的负担。我们还考虑到，每步操作的操作对象窗口并不是占据全部屏幕的，因此我们截取的屏幕图像只需要包括操作对象即可，其他的背景图象都可去掉，这样可以进一步减小图像的大小。在 Author 应用程序中，我们设计了“二次裁剪”，即设定一个全屏 1/4 大小的窗口，当课件制作者截取了全屏图像之后（“一次裁剪”），再由该制作者将全屏图像中有关中当前操作对象或需要学习者关注的部分（基本都小于 1/4 屏幕）移动到该窗口中，进行“二次裁剪”，将“裁剪”后的图像存入数据库，这样图像的大小仅为原来的 1/4。

2.5 本章小结

本章主要介绍例了 *EPSS* 系统的体系结构，以及各子模块的方案设计。

本章首先先在介绍了 *EPSS* 系统工作原理的基础上，提出了以数据库服务、中间应用服务和客户端三层结构作为整个系统的框架。然后对系统的几个主要模块：User 端应用程序、Author 端应用程序和 *DCOM* 组件分别给予介绍。

User 端应用程序的体系结构的设计参考了 *IEEE P1484.1* 标准草案 *LTSA*，在介绍了 LTSA 模块结构图的基础上，详细阐述了 User 端应用程序的体系结构设计。User 端应用程序实现的技术难点在于截获用户对学习对象的操作，在分析了各种技术实现方法的不足后，提出了屏幕取词的解决方案。

EPSS 系统使用 *DCOM* 分布式组件来实现各种逻辑应用。整个结构分成三层，

从上至下分别为应用程序层、应用对象层和数据库访问管理层。对数据库的访问使用 *ADO*，各组件对象采用基于 *MTS*（微软事务处理服务器）的组件技术来实现。

最后介绍了 Author 端应用程序设计中用到的“二次裁剪”拷屏方案，其目的是减少课件的数据量。

3 屏幕取词和 *DCOM* 技术分析

本章详细介绍了 *EPSS* 系统中所用到的屏幕取词、*COM* 和 *DCOM*（分布式 *COM*）等关键技术的基本原理和实现方法。本章首先介绍了屏幕取词的原理，在深入分析了 *Windows* 操作系统的内存管理机制和系统 *API* 函数调用机制的基础上，分析比较了各种屏幕取词的方式，并最终将其应用到 *EPSS* 系统中。然后介绍了 *DCOM* 中的代理机制，以及如何应用 *ATL* 生成 *DCOM* 组件等方面的内容。

3.1 屏幕取词技术分析

3.1.1 屏幕取词的基本原理

屏幕取词（或者叫动态翻译）是指随着鼠标的移动，软件能够随时获知屏幕上鼠标位置的单词或汉字，并能截获出来提示用户。它对于上网浏览、在线阅读外文文章等很有帮助作用，因此许多词典软件都提供了屏幕抓词功能，比如著名的金山词霸。

屏幕抓词的关键是如何获得鼠标位置的字符串，*Windows* 操作系统的动态链接和消息响应机制为之提供了实现途径。概括地说，主要通过下面的几个步骤来取得屏幕上鼠标位置的字符串：

1) 代码拦截：*Windows* 以 *DLL* 方式提供系统服务，可以获取 *Windows* 字符输出 *API* 函数的入口地址，修改其入口代码，拦截其他应用程序对这些函数的调用。
2) 鼠标 *HOOK*：安装 *WH_JOURNALRECORD* 类型的全局鼠标 *HOOK* 过程，监视鼠标在整个屏幕上的移动以及点击事件。
3) 屏幕刷新：将鼠标周围一块区域“弄脏”，以强制鼠标位置的应用程序的窗口刷新鼠标周围的区域。应用程序响应 *WM_NCPAINT* 和 *WM_PAINT* 消息，调用 *ExtTextOut/TextOut* 等字符输出 *API* 函数刷新被“弄脏”区域里面的字符串[27]。由于这些函数的调用已被我们拦截，可以得到被拦截 API 函数的调用参数，如字符串地址、长度、输出坐标、*Hdc*、剪切域等信息。从中分析得出鼠标位置处的文字。

以上提到了很多内容涉及到了 *Windows* 操作系统的核心运行机制，在下文中将做详细的介绍。在整个取词的过程中，拦截 *ExtTextOut/TextOut* 等字符输出 *API* 函数

是最关键的部分，而且是技术实现最复杂的部分。首先我们要谈谈 Windows 操作系统的内存管理机制。

3.1.2 *Windows* 内存管理机制简介

一般来说，80X86*CPU* 在三种模式下工作：实模式，保护模式和 *V*86 模式。实模式就是“古老”的 *MS*－*DOS* 运行环境。*Windows* 32 位操作系统主要运行在保护模式下（下文所提的 *Windows* 操作系统除特别说明外，均为 *Windows* 32 位操作系统）。只有在保护方式下，*CPU* 才能真正发挥更大的作用，因为在保护方式下，全部 32 条地址线有效，可寻址高达 4*G* 字节的物理地址空间，而运行在实模式下的 16 位程序最多只能存取 1*M* 的内存。

虽然与 8086 可寻址的 1*M* 字节物理地址空间相比，80X86 可寻址的物理地址空间可谓很大，但实际的微机系统不可能安装如此大的物理内存。所以，为了运行大型程序和真正实现多任务，*Windows* 操作系统采用虚拟存储器。虚拟存储器是一种软硬件结合的技术，用于提供比在计算机系统中实际可以使用的物理主存储器大得多的存储空间。另外 *Windows* 操作系统还要对存放在存储器中的代码及数据的共享和保护提供支持[26]。

保护模式下的虚拟存储器由大小可变的存储块构成，这样的存储块称为段。80X86 采用称为描述符的数据来描述段的位置、大小和使用情况。虚拟存储器的地址（逻辑地址）由指示描述符的选择子和段内偏移两部分构成，这样的地址集合称为虚拟地址空间。80X86 支持的虚拟地址空间可达 64*T* 字节。

显然，只有在物理存储器中的程序才能运行，只有在物理存储器中的数据才能被访问。在 *Windows* 操作系统中，每一个 32 位任务都有一个虚拟地址空间。为了避免多个并行任务的多个虚拟地址空间直接映射到同一个物理地址空间，操作系统采用线性地址空间隔离虚拟地址空间和物理地址空间。

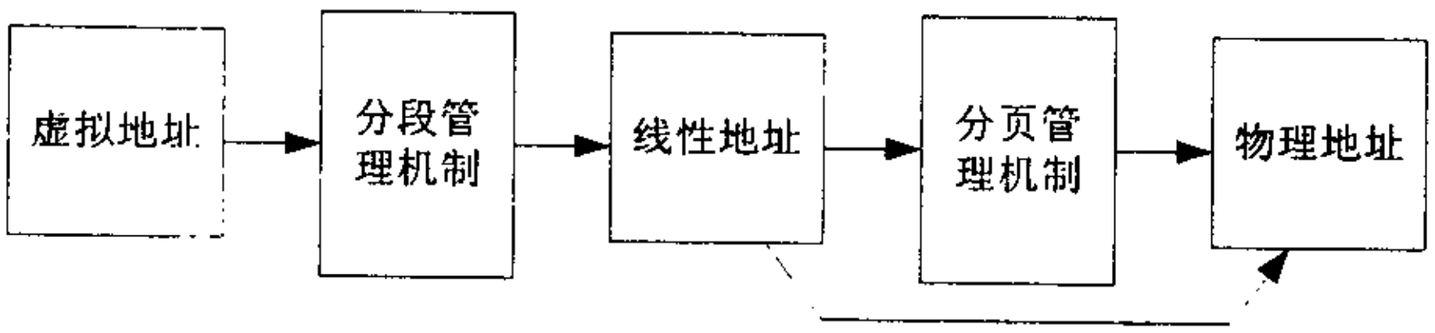

图3.1 Windows操作系统地址映射转换示意图

线性地址空间由一维的线性地址构成，线性地址空间和物理地址空间对等。线

性地址 32 位长，线性地址空间容量为 4*G* 字节。图 3.1 为在 80X86 中实现地址之间映射转换的示意图[26]。 通过描述符表和描述符，分段管理机制实现虚拟地址空间到线性地址空间的映射，实现把二维的虚拟地址转换为一维的线性地址。分页管理机制实现线性地址空间到物理地址空间的映射，它是可选的，在不采用分页管理机制时，线性地址空间就等同于物理地址空间。虚拟地址、线性地址和物理地址是 *Windows* 系统编程中最重要的概念。

在 *Windows 95/98* 操作系统中，每一个 32 位进程拥有 4*G* 线性地址空间。每个进程的虚拟地址空间都要划分成各个分区。下表列出了不同 *Windows* 操作系统的进程地址空间的分区情况[24]：

表 3.1 不同操作系统进程地址空间分区表

分区	*Windows* 2000/*NT*4.0	*Windows* 95/98
NULL 指针分配的分区	0x00000000-0x0000FFFF（64K）	0x00000000-0x00000FFF（4K）
*DOS*16 位 Windows 程序兼容分区	无	0x00001000-0x003FFFFF（4M）
进程私有地址空间	0x00010000-0xBFFFFFFF（3G）	0x00400000-0x7FFFFFFF（2G）
共享内存映射文件	无	0x80000000-0xBFFFFFFF（1G）
操作系统内核文件	0xC0000000-0xFFFFFFFF（1G）	0xC0000000-0xFFFFFFFF（1G）

在 *Windows*95/98 系统中，低 2*G* 是进程的私有地址空间，各进程的私有地址空间是相互独立的，无法互相读、写或已其他任何方式互相访问，应用程序的 *exe* 和其他 *DLL* 模块加载到这个分区中；高 2*G* 是所有进程的共享地址空间，主要的 *Win*32 系统 *DLL*(Kernel32.dll、User32.dll 和 GDI32.dll)均在这个分区中。所有 *Windows API* 函数的代码都在系统 *DLL* 内，系统启动时会将这些 *DLL* 映射到 2*G* 到 3*G* 的进程共享线性地址空间里，因此任何进程都能自由调用这些 *API* 函数。了解以上内容后，我们就可以讨论 *API* 函数拦截技术了。

3.1.3 *Windows API* 函数拦截原理

准确的说，在 *Windows*32 位操作系统中，完成字符输出的 *API* 函数共有 4 个：*ExtTextOutA*、*ExtTextOutW*、*TextOutA* 和 *TextOutW*，其中以 *A* 结尾表示函数输出 *ASCII* 字符(2 字节)；以 *W* 结尾表示函数输出 *Unicode* 字符（4 字节）。这四个函数就是我们的拦截目标函数。我们首先看看 *Windows* 操作系统函数调用的机制。

当进程以 *C* 或 *Stdcall* 调用方式调用一个 *API* 函数时，编译器对函数的参数以及返回地址在堆栈中的处理如图 3.2 所示：即进入 *API* 函数的代码前，程序先将函数的参数以从右至左的顺序入栈（如果用 *WINAPI* 调用方式，参数从左至右入栈），再将返回地址入栈。值得注意的是堆栈是由高位地址向低位地址扩展的，而且函数的返回值是在 *EAX* 寄存器中。

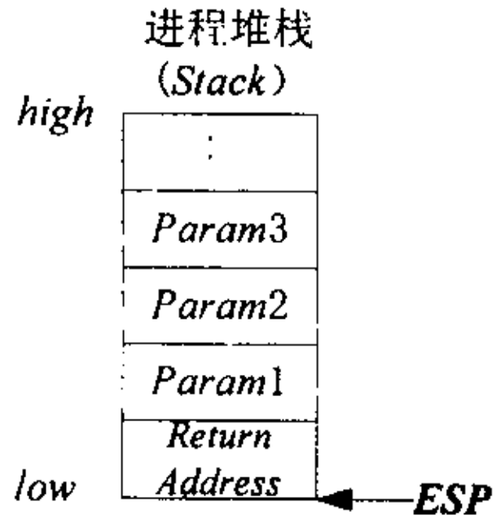

图3.2 *API* 函数参数入栈示意图

为了实现对 *ExtTextOutA/TextOutA* 等 *API* 函数的拦截，需要修改被拦截的 *API* 函数入口地址的代码，使该函数一旦被调用，就首先转入我们设置的拦截函数。拦截函数负责从堆栈中获取参数，计算字符串坐标，分出鼠标位置的字符等工作。执行完成后，拦截函数再调用原来的被拦截函数，完成正常的字符输出，然后返回。图 3.3 以 *TextOuA* 为例示意了正常情况下应用程序对其调用的流程和被拦截后应用程序对其调用的流程。

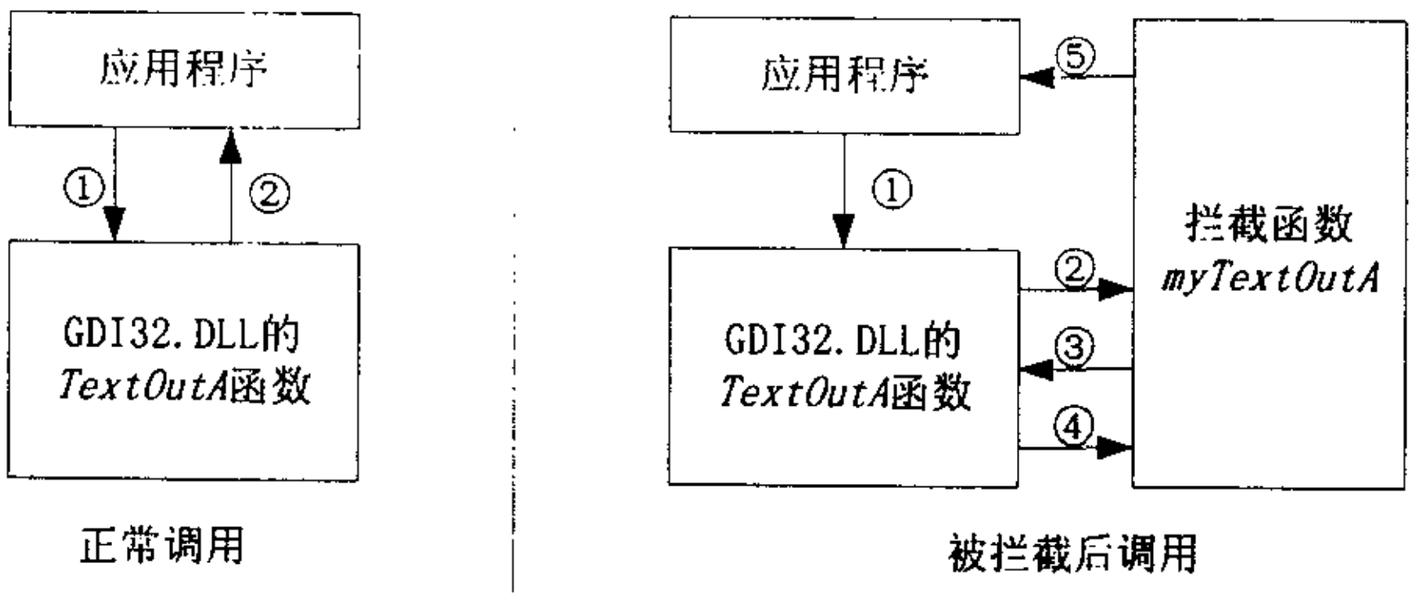

图3.3 不同状态下应用程序对*TextOutA*函数的调用流程

在正常情况下调用：

①应用程序调用 *TextOutA* 函数；

②从 *TextOutA* 函数返回应用程序；

当 *TextOutA* 函数被拦截后调用：

①应用程序调用 *TextOutA* 函数（其入口地址指令已被修改）；

②转入拦截函数 *myTextOutA*，从函数参数中分析得到鼠标位置的字符；

③恢复入口地址指令，从 *myTextOutA* 函数调用原来的 *TextOutA* 函数；

④从 *TextOutA* 函数返回 *myTextOutA* 函数，修改入口地址指令；

⑤从 *myTextOutA* 函数返回应用程序；

3.1.4 *Windows API* 函数拦截的方法研究

实现 *Windows API* 拦截的方法有很多，其中又包括系统级拦截和进程级拦截等，但是无论是在何种 *Windows* 操作系统下拦截何种 *API* 函数，可以归纳为以下三种典型的方法：

方法 1：修改应用程序模块的 *IAT* 表（*Import Address Table*）。在 *Windows* 32 位操作系统中，可执行模块(包括 *EXE*、*DLL* 文件)的文件格式是 *PE*（*Portable Executable*）格式[28]。*PE* 文件在内存中的结构与磁盘上的静态文件结构是完全一样的。每个 *PE* 文件的文件头都包含了该模块的输入函数入口地址表，即 *IAT* 表。当模块调用一个输入函数时，先从 *IAT* 表中找到该输入函数的入口地址，再跳转到该地址执行函数代码。*IAT* 表的内容是在输入函数对应的 *DLL* 被加载后，由系统来负责填写的，这样程序可以不受由于 *DLL* 加载时基址不定造成的影响而正常运行。如果修改模块 *IAT* 表中被拦截的 *API* 函数的入口地址使其指向自己的拦截函数，就可以获得控制权并实现拦截。文献[28]一书中就用该方法实现了一个 *API SPY* 程序。但是这种方法仅对单进程有效，而且只有模块“显式”调用的 API 函数才会被记录在 *IAT* 表中，如果程序通过 *GetProcAddress* 函数获得 *API* 函数地址，并使用函数指针进行 *API* 调用，这种方法就无能为力了。

方法 2：修改系统 *DLL* 的 *ET* 表（*Export Table*）。从方法 2 中我们得到启发，每个系统 *DLL* 的 *PE* 头中不但包含了 *IAT* 表，而且还有一个重要的 *ET* 表[24]，其中指定了 *DLL* 所有输出函数的代码入口地址。通过修改系统 *DLL* 的 *ET* 表中被拦截 *API* 函数的代码入口地址，使其指向自己的拦截函数，同样可以实现拦截。这种方法没有了方法 1 的各种限制，可以拦截系统所有进程对 *API* 函数的调用。

方法 3：直接修改 *API* 函数的入口地址的代码。用方法 2 你必须对 *PE* 文件格式

有一定了解，并且通过比较复杂的步骤才能找到系统 *DLL* 的 *ET* 表中 *API* 函数入口地址的位置。而用方法 3，我们只需使用 *GetProcAddress* 函数就可得到被拦截 *API* 函数的入口地址，然后保存入口地址的头几个字节的指令，用一个 *JMP* 或 *INT* 指令替换掉这几个字节，这样当该 *API* 函数被调用时会直接或通过中断方式跳转到我们的拦截函数。这里需说明的是：使用 *JMP* 指令和使用 *INT* 指令是有一定区别的。在 *Windows* 32 位操作系统中，一个 *JMP* 跳转指令至少要 5 个字节，如果被拦截的 *API* 函数与相邻的 *API* 函数的入口地址偏移少于 5 字节，就不能直接进行拦截，否则会影响相邻 *API* 函数的调用而造成潜在的危险。这种情况是实际存在的，用 *SoftIce*（*windows* 下的 *debug* 工具）察看 *TextOutA* 和 *TextOutW* 的入口地址就会发现两者仅相差 4 个字节。而一个 *INT* 中断指令一般只要 2 个字节就行了，可以基本避免地址冲突。但是 *Windows* 保留中断（如中断 4 和中断 5 等）的个数是有限的，如果要拦截多个 *API* 函数，所有这些 *API* 函数最好共享同一个中断向量，由中断处理程序根据堆栈中的中断返回地址来判断是哪个 *API* 发出的中断请求，再跳转到对应的拦截代码来实现拦截。这样将会涉及较多的堆栈操作而且不得不使用一些汇编代码。在实际应用中，用 *JMP* 指令更简便些。

以上是常用的几种 *API* 拦截方法，其中方法 2 和方法 3 有很好的实用性，在一些应用程序中已得以应用。在 *EPSS* 系统的实际应用中，我们主要使用方法 3 的来拦截字符输出函数。

3.1.5 *Windows API* 函数拦截的实现

Windows API 拦截的实现是一个比较复杂的过程。在不同的操作系统下、使用不同的编程语言实现方式是各不相同的，但是需要解决的主要就是两个问题：一是如何修改系统 *DLL* 的代码；二是如何加载拦截函数到高端内存。在 *EPSS* 系统中我们应用 *VC++*实现在 *Windows* 95/98 中拦截 32 位和 16 位 *API* 函数。其过程和原理将在下文详细介绍。

3.1.5.1 在 *Windows* 95/98 下实现拦截 32 位 *API* 函数

在 *Windows* 95/98 下实现 32 位 *API* 函数拦截是相当复杂的。

首先，*Windows*32 位操作系统中，进程的保护是通过优先级来实现的，在 80X86 系统中，优先级的概念是用环（*Ring*）来表示：0 环（*Ring0*）代表最高优先级；3

环（*Ring3*）代表最低优先级，即用 *Ring0* 表示核心态，用 *Ring3* 表示用户态。在 *Windows* 95/98 中，绝大部分的用户进程运行在 3 环，而系统高 2*G* 的共享地址空间对 3 环的进程是只读的，并且任何 *Windows API* 函数都不能改变其只读属性。如果要修改共享地址空间的代码，我们必须要获得 0 环控制权。解决的方法有三种：

1) 是使用 *VxD* 技术。这是 *Windows* 95/98 中唯一公开的编制 0 环程序的技术[30]。

虽然 *VxD* 技术是稳定的且被推荐的，但它需要专门编写，而且用它仅仅实现读、写内存的操作未免有些小题大做，因此在某些情况下，用别的方法进入 0 环是更有效的。

2) 另一种方法是使用 Kernel32.dll 中未公开的 API 函数 *VxdCall*[29]。它可以调用并执行工作在 0 环的代码，但由于这是未公开的函数，没有比较详细的用法说明。有些程序如 *BO2K*（黑客程序）就是用到了这个函数。

3) 第三种方法是修改系统表。*Windows* 32 位操作系统中，*CPU* 用两种方法处理优先级转换：一是通过中断/意外门；二是通过调用门(*Callgate*)。中断门被放在系统 *IDT* 表中，调用门被放在系统局部描述符表 *LDT* 或全局描述符表 *GDT* 中。在 *Windows* 95/98 下这些重要的系统表(*IDT*、*LDT*、*GDT*)可以在 3 环下被自由修改(注意：在 *Windows* NT 中这种方法是不行的)，这虽然是安全上的重大隐患，但这给我们一个机会：可以通过编写中断服务程序或调用门程序获得 0 环控制权来修改 *API* 函数的入口地址代码。

在 *EPSS* 系统中，我们应用方法 3 修改 *IDT* 表获得 0 环控制权。下面介绍一下修改方法：当一个中断触发时，*CPU* 会从 *IDT* 表中找到该中断对应的中断描述符，然后再转移至该描述符指定的中断处理程序。这个中断描述符是一个 64 位的结构体，其中包含了中断处理程序的段选择子及偏移量[29]。具体结构如下：

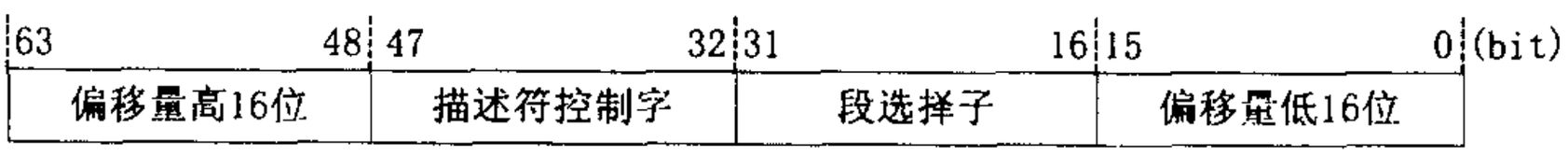

图3.4 中断描述符结构示意图

其中第 0 到 15 位为中断处理程序地址(32 位)的低 16 位，第 48 到 63 位为中断处理程序地址的高 16 位，第 16 到 31 位为处理程序所在段的选择子，第 32 到 47 位确定了此描述符为中断门，并确定其优先权。在 *SoftIce* 中用 *IDT* 命令可以察看系统 *IDT* 表的全部信息。我们可以修改某个中断描述符，使其转入自己的中断处理程

序，由于中断处理程序运行在 *Ring0* 下，所以可以实现对系统共享地址空间的写操作。修改 *IDT* 表可以用下面代码实现：

```
/*************************************************************************/
BYTE IDTTAB[6]; //IDT 表 6 个字节，低 16 位为界限，高 32 位为中断表基地址
_asm sidt IDTTAB; //存 IDT 表
WORD *Offset =（WORD*）(*（DWORD*）(IDTTAB+2) +IntNum*8);
*Offset =（WORD）newProc;
*（Offset+3）=（WORD）(newProc>>16);
/*************************************************************************/
```

其中 sidt 是 80386 指令，功能是将系统 IDT 表存入相应的操作数中；Offset 是中断号为 IntNum 的中断描述符的首址，newProc 为自己的中断处理程序的地址。

接着讨论如何在 *Windows* 95/98 中加载拦截代码。由于任何进程都可能调用我们要拦截的字符输出 *API* 函数，所以拦截代码必须能被所有的进程访问。根据前面介绍不难想到，拦截码必须被加载到高 *2G* 共享地址空间。如果拦截码在某个进程的私有空间，由于进程私有空间的独立性，其他进程调用被拦截的 *API* 函数必将导致异常错误。

将拦截码加载到高端内存常用的方法主要有两个：一是用创建内存映射文件。首先利用 *CreateFileMapping* 函数拦创建一块内存区域，并用 *MapViewOfFile* 函数将其映射到共享地址空间，然后将在进程私有空间中的拦截代码用 *MemCopy* 等函数拷贝到这个内存映射文件里，这样就实现了拦截代码的搬移[31][32]。这种方法有个很大的弊端就是需要修正重定位。拦截代码在被加载到进程私有空间时已经做了重定位，再将其移到高端内存时必须要修正搬移对重定位的影响，否则不能正常运行。这项工作相当麻烦，所以拷贝到内存映射文件的拦截代码一般都用汇编代码编写以减少重定位的工作量，但这样拦截代码的灵活性就受到了很大的限制。

第二种方法是创建动态链接库。即将拦截代码制成 *DLL* 动态地加载到共享地址空间。*DLL* 被映射到虚拟内存空间的什么地方是由其基地址决定的，而 *DLL* 的基地址是在链接时由链接器决定的。当新建一个 *Win32* 工程时，*VC++* 链接器使用缺省的基地址 0x00400000，你可以通过链接器的 *base* 选项改变模块的基地址。我们希望该 *DLL* 被映射到高 *2G* 的共享地址空间，则指定其 *base* 为 0x80000000（*2G*），除此还必须在 *DLL* 的 *.def* 文件加上以下定义信息将 *DLL* 的可写数据段都设成共享：

```
SECTIONS
    .data   READ WRITE SHARED
    .idata READ WRITE SHARED
    .bss    READ WRITE SHARED
```

这样生成的 *DLL* 文件运行时会被自动映射到高 2*G* 的共享地址空间。用这种方法你丝毫不用考虑重定位的问题，拦截码可以自由编写，有很强的实用性。因此在 *EPSS* 系统中，我们使用这种方法加载拦截函数。

需要注意的一点是，有很多地方提到可以用 *Windows* 全局钩子实现拦截代码注入：即将拦截代码写入 *Windows* 全局钩子的 *DLL* 里，钩子函数会自动把拦截代码映射到每个进程的地址空间内。钩子函数确实会把整个 *DLL* 文件的代码在每个进程的空间里都做一个映射，但是 *DLL* 是被映射到每个进程的私有空间里，并且在每个进程私有空间里的基地址都不一样，因此我们无法用一个统一的跳转指令来改写被拦截 *API* 函数入口地址代码。所以这种方法在 *Windows* 95/98 下是行不通的。

3.1.5.2 在 *Windows* 95/98 下实现拦截 16 位 *API* 函数

此时，读者不禁要问我们有必要讨论在 *Windows*95/98 中拦截 16 位 *Windows API* 函数吗？*Windows* 9X 不是 32 位操作系统吗？这是因为在实际取词中我们发现这样的问题，按照上述方法拦截了 32 位的字符输出 API 函数后，我们在应用程序中经常取不了词，在排除了程序代码出错的可能性，并参阅了一些资料以后，我们发现了以下事实：

虽然微软声明过在 *Windows*95/98 操作系统中，16 位的 *API* 只是为了兼容性才保留下来，程序员应该尽可能地调用 32 位的 *API*，但实际上根本就不是这样！*Windows* 95/98 内部超过 80%的 32 位 *API* 都利用“*thunk*”技术调用了 *Windows* 3.1（16 位操作系统）中同名的 16 位 *API*[33]。在 *Windows* 的 *system* 目录下你会看到 Krnl386.exe、GDI.exe、User.exe 三个文件，里面包含了几乎全部的 16 位 *API* 函数代码，用 *SoftIce* 查看你会发现这三个文件都被加载到高 2*G* 的共享地址空间，没有这几个文件 *Windows* 95/98 立即就会崩溃。我们没有取到词是因为这些应用程序的某些模块在刷新屏幕的字符时，绕过了 32 位的字符输出函数，直接调用了 16 位的字符输出函数，所以为了能在绝大多数应用程序中都能实现屏幕取词，我们还必须拦截 16 位的字符输出函数 *TextOut* 和 *ExtTextOut*。

由于 16 位程序运行时，CPU 工作在实模式下，进行内存代码修改是容易实现的。

可以用 Krnl386.exe 里未公开的函数 *AllocCStoDSAlias*，这个函数为给定的代码段选择符分配一个具有相同线性基址和尺寸的数据段别名（*DS alias*）。通过 *DS* 别名可以对给定的代码段进行修改,这样就可以修改被拦截函数的入口地址代码了。根据前面的介绍不难想到要在 *Windows* 95/98 下拦截 16 位字符输出函数，拦截代码必须被制成 16 位 *DLL* 然后加载到高 2*G* 共享地址空间（如同 GDI.exe、User.exe 等）。显然我们没法对一个 16 位 *DLL* 指定 32 位基址，好在 *VC++*编译器解决了这个问题，用 *VC++*1.5 及以下版本的编译器生成的 *Release* 版 16 位 *DLL* 会被自动加载到高端内存。这样实现在 *Windows* 95/98 下拦截 *TextOut* 和 *ExtTextOut* 函数也就没什么困难了。现在的问题是 *EPSS* 系统的应用程序是 32 位应用程序，而拦截代码在 16 位 *DLL* 中。如何在 32 位应用程序中加载并调用 16 位的动态链接库中的函数，即所谓的“*thunk*”技术，就是我们下一节要讨论的内容。

3.1.6 在 32 位应用程序中调用 16 位 *DLL*

微软并不希望程序员在 32 位操作系统中调用 16 位 *DLL*，所以我们要使用许多未公开的 *API* 函数才能实现这一功能。这部分可以说涉及到 *Windows* 操作系统核心的内幕了。

首先我们要用未公布的函数 *LoadLibrary*16, *GetProcAddress*16, *FreeLibrary*16 进行加载、取得函数的地址和卸载 16 位 *DLL*，有了这些函数的帮助我们就可以载入一个 16 位 *DLL*，并取得我们所需函数的地址。但是我们要使用这些函数还应解决存在几个问题：

1) 由于是未公开的 *API* 函数，Kernel32.DLL 的函数导出表中并没有声明这些函数的名称和序号，这样我们只能分析 *DLL* 文件的 *PE* (*portable executable*)文件格式的结构，并从中的函数导出表（*export table*）中查找我们所需的函数，获得函数的地址。
2) 16 位程序使用的是段地址:偏移地址格式，而我们现在却是使用线性地址，因而我们需要“模拟”成一个 16 位的系统。

现在我们可以载入 16 位 *DLL* 并取得所需函数得地址，而我们要调用它还须建立一个 16 位的堆栈并做一些工作。这些工作我们可以用一个未公开的 Kernel32.DLL 函数，其名称是“*QT_Thunk*”来处理[34]。这时我们需要插入一些汇编程序，将 16 位 *DLL* 中函数的参数依次入栈并 *call QT_Thunk*。但是调用该函数要注意以下几点：

1) 保留最少 60 个字节的堆栈，可以申明一个最少 60 的字符的局部变量，因为声明的局部变量是在堆栈中分配空间的。
2) 函数的参数将使用汇编“*push*”指令入栈，一般情况下使用 *Pascal* 编译的函数，是从左往右依次推入的，而使用 *C/C++*编译的则是从右往左推入堆栈的，而我们 *Windows* 提供的函数一般是须用 *Pascal* 格式调用。
3) 返回格式--般是这样的：字符型、整数型返回的值在 AX 寄存器中；长整数、各种指针等的值在 DX：AX 处。
4) 如果调用 16 位函数得参数中有指针的还须将 32 位指针映射成 16 位的指针，这就需要使用未公开的 Kernel32.DLL 函数 *SMapLS_IP_EBP_8* 和 *SUnMapLS_IP_EBP_8*，来进行映射和解除映射，调用 *SMapLS_IP_EBP_8* 时，先将要映射的 32 位地址入栈，再进行调用，不过因为之后还需要这个 32 位地址进行解除映射，所以并不能马上做出栈处理，而是调用 *SUnMapLS_IP_EBP_8* 后才出栈。

这里面比较复杂的步骤是分析 Kernel32.DLL 的 *PE* 文件格式，找到未公开函数的地址。下面做简要分析介绍：

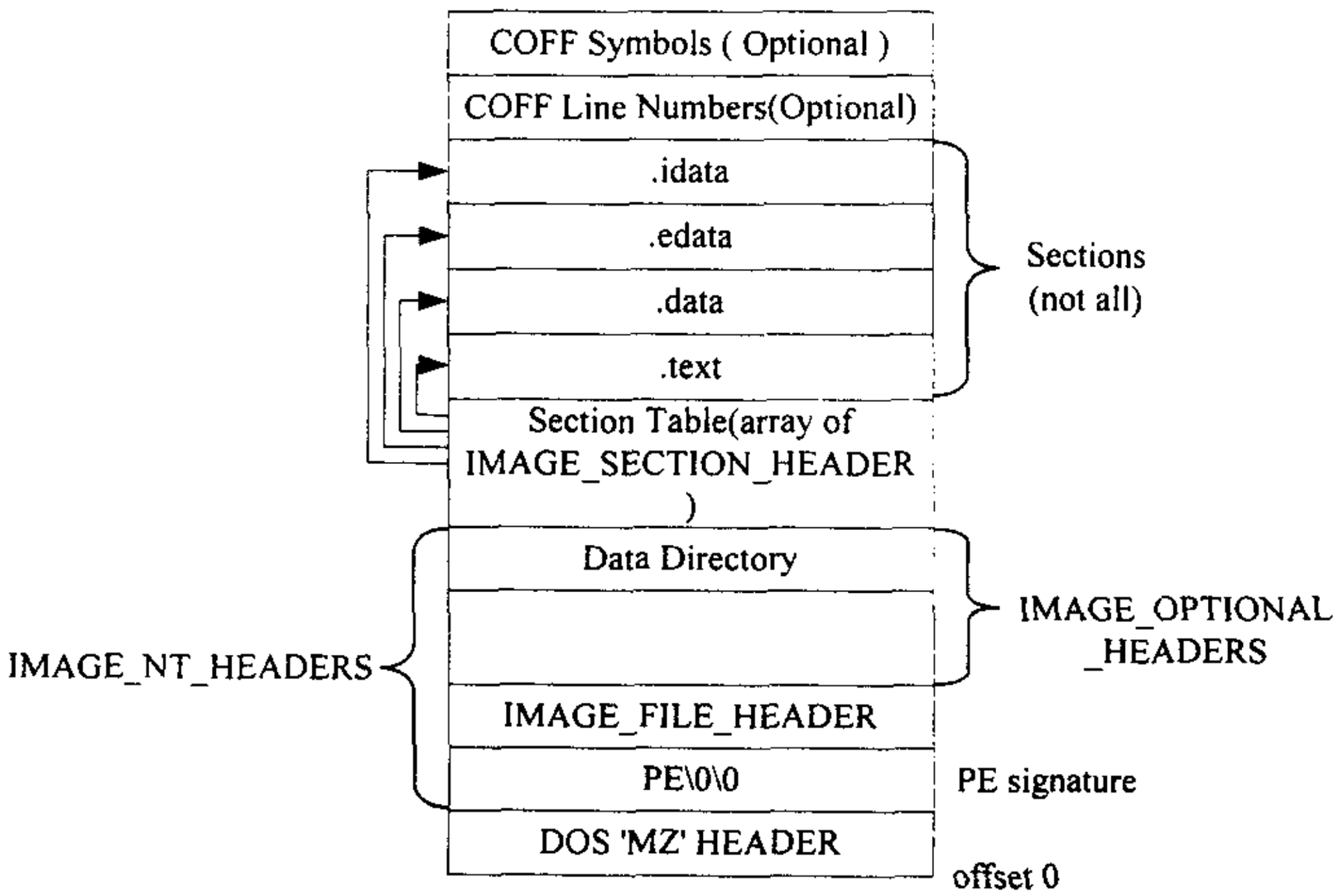

图3.5 PE文件的大致布局

PE 文件中的每个模块的内容被映射到内存空间后，称其为模块的“*image*”，这

个声明在 *PE* 文件格式中随处可见。*PE* 文件中相对虚拟地址的概念（*Relative Virtual Address*，*RVA*）很重要：*PE* 文件中许多重要模块的内容都以 *RVA* 表示，一个 *RVA* 是 *PE* 中某一模块相对于 *PE* 文件载入内存空间的基址的地址偏移值（*offset*）[28]。例如：*Windows* 载入器把一个 *PE* 文件映射到虚拟地址空间的 0x400000 处，如果 *PE* 文件中的某个 *image* 有个表格位于 0x411464，那么这个表格的 *RVA* 就是 0x11464。图 3.5 是 *PE* 文件中模块的大致布局。*PE* 文件的格式是比较复杂的，这里不作详细介绍，我们所关心的是 *PE* 文件的函数导出表。从上图可以看出 *PE* 文件的核心内容是以 Sections 的形式组织起来的。.text Section 内含有一般的程序代码；.data Section 是程序初始化数据（包括全局变量和静态变量）的存放区；.edata Section 包含 PE 文件输出函数的相关信息；.idata Section 包含 PE 文件输入函数的相关信息。可见我们需要了解一下 .edata Section 的结构。.edata Section 开始处是一个 IMAGE_EXPORT_DIRECTORY 结构体，之后是出该结构体中每个成员所指向的内容。

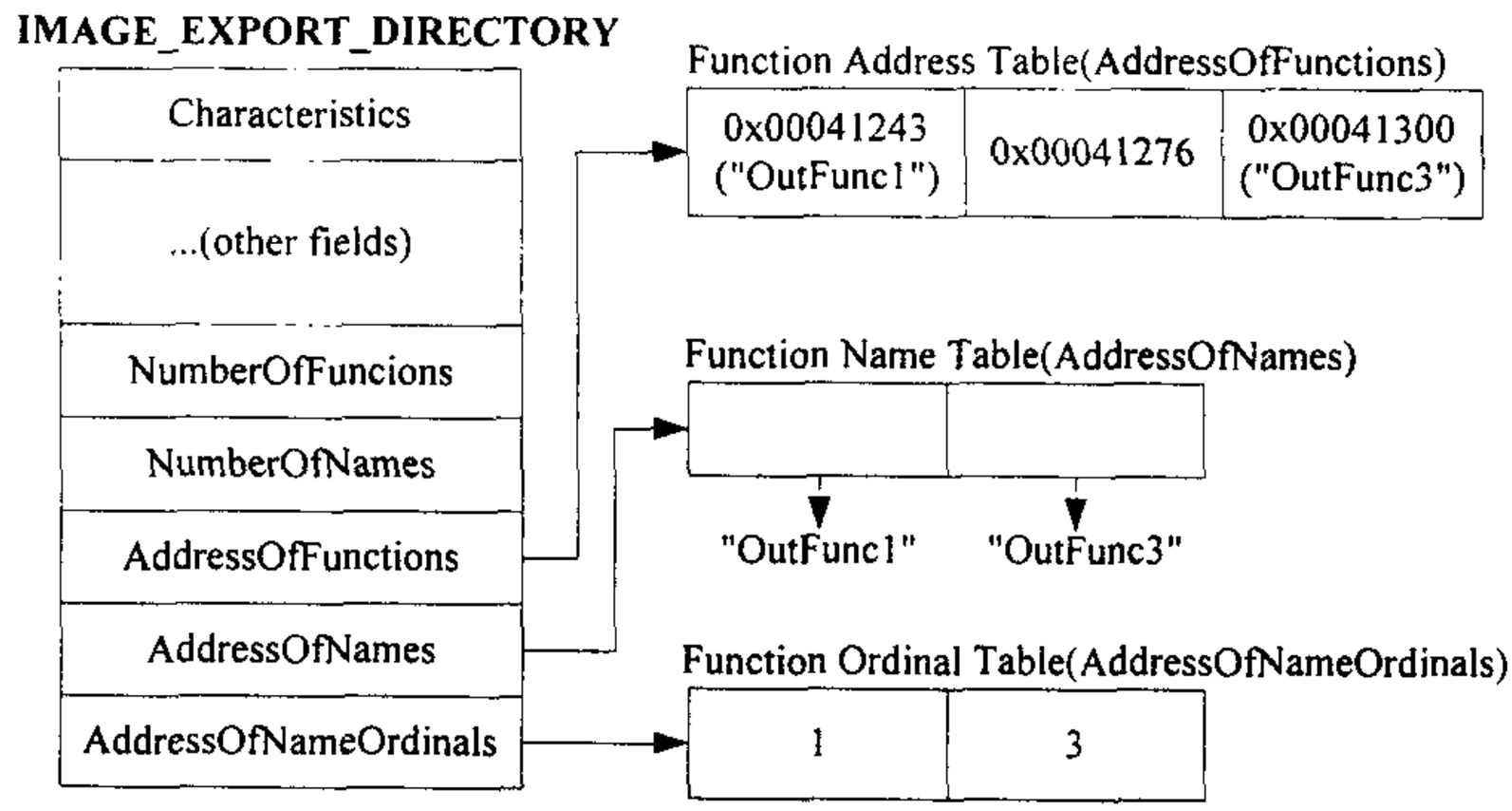

图 3.6 一个典型的DLL export table

IMAGE_EXPORT_DIRECTORY 结构体中这样几个成员变量是我们需要的：

DWORD Base

此 *PE* 文件输出函数的起始序号。

*PWORD *AddressOfFunctions*

这是一个 *RVA* 值，指向一个由函数地址所构成的数组。这个数组存放了该 *PE* 文件的每个输出函数相对于基址的 *RVA* 值。

*PWORD *AddressOfNames*

这是一个 *RVA* 值，指向一个由字符串所构成的数组。这个数组存放了该 *PE* 文件的每个输出函数的名称。

DWORD NumberOfFunctions

AddressOfFunctions 数组中的元素的个数。此值即表示 *PE* 文件输出函数的个数。

图 3.6 是一个典型的 *DLL* 文件的 *export table*[28]，其中有 3 个函数被输出（*exported*），其中两个以名称输出（序号为 1 和 3）。

了解以上内容后，我们不难知道如何从 Kernel32.DLL 的函数导出表中查到未公开函数 *LoadLibrary*16（35）、*GetProcAddress*16（37）、*FreeLibrary*16（36）相对于 Kernel32.DLL 基址的 *RVA* 值（括号内的编号是该函数在 Kernel32.DLL 函数导出表的输出序号）。

3.1.7 拦截函数设计

拦截函数插入被拦截 *API* 的进程中执行，需要有和被拦截 *API* 相同的参数和返回值原型。以对 16 位 *TextOut* 函数进行拦截为例，下面是拦截函数函数 *myTextOut* 的伪代码：

```
BOOL myTextOut(HDC hdc,int x,int y,LPSTR lpstr,int cbstr)
{
    DoSpy(hDC,x ,y,lpstr,cbstr);        //保存参数，分析坐标取词
    RestoreCode();                      //恢复 TextOut 入口原来的指令
    TextOut(hDC,x,y,lpstr,cbstr);       //调用原来的 API
    SpyCode();                          //再次放入 JMP 指令
    return TRUE;
}
```

函数首先调用 DoSpy()来作取词的具体工作，然后 RestoreCode()函数恢复被拦截函数入口的代码，再调用 TextOut()执行正常的字符输出，接下来 SpyCode()在被拦截函数入口再次放入 JMP 指令，最后返回调用进程。

当应用程序刷新鼠标所在区域的屏幕时，很可能要刷新输出很多字符串，因此会调用很多次字符输出函数。要取得正确的字符，DoSpy（）必须做这样的处理：以 *TextOut* 函数为例，其 *x,y* 参数表示相对于设备描述表 *hdc* 的设备逻辑坐标。首先

将设备逻辑坐标转换为屏幕坐标作为起始点，然后计算输出字符串的长度和宽度，与起始点联合构成一个矩形区域，取得鼠标当前位置的屏幕坐标，判断鼠标是否在这个矩形区域内，如果在，这个字符串就是鼠标所在位置的字符串。

按照上述方法分析取词并不是万无一失的，有些软件（例如 *Word* 的某些窗口）在刷新屏幕时，为了消除闪烁，不直接把字符串输出到屏幕 *DC*（设备描述表），而是创建一个兼容的内存 *DC*，先将字符串写到内存 *DC*，再复制到屏幕上去。由于内存 *DC* 坐标和屏幕 *DC* 坐标的不同，不能直接依赖内存 *DC* 的坐标来计算字符串的屏幕位置。这时我们做如下处理：拦截 TextOut 函数时，调用 32 位 *API* 函数 *GetObjectType*（利用“*thunk*”技术，是上一节内容的“逆”过程，具体方法较复杂，不详细叙述了）考察 *hdc* 参数，判断字符是输出到屏幕 *DC* 还是输出到兼容的内存 *DC*，如果是输出到兼容内存 *DC*，则拦截 *bitblt* 函数（该函数是 *Windows* 操作系统中完成 *DC* 之间拷贝的 *API* 函数），根据 *bitblt* 函数的下 *x,y* 参数来计算内存 *DC* 中字符所在的屏幕坐标，再结合鼠标坐标取得鼠标所在处的字符串。

通过上面讨论的方法可以实现对 *Windows*95/98 操作系统中绝大部分应用程序的菜单和对话框窗口进行屏幕取词，但对网页或 pdf 文件等刷新机制复杂的窗口并不能实现 100%取词，不过考虑到 *EPSS* 系统的教学实际，这样就已经足够了，毕竟我们并不是做翻译软件，不需要像金山词霸那样强大的取词功能。

3.1.8 *Windows* 全局钩子（*HOOK*）监视鼠标动作

Windows 操作系统是建立在事件驱动的机制上的，即整个系统都是通过消息的传递来实现的。而 *HOOK*（钩子）是 *Windows* 操作系统中非常重要的系统接口，用它可以截获并处理送给其他应用程序的消息。下面简单介绍一下 *Win32* 全局钩子的运行机制。

钩子实际上是一个处理消息的程序段，通过系统调用，把它载入系统的进程空间。每当特定的消息发出，在没有到达目的窗口前，钩子程序就先捕获该消息，即钩子函数先得到控制权。这时钩子函数即可以加工处理（改变）该消息，也可以不作处理而继续传递该消息，还可以强制结束消息的传递。对每种类型的钩子由系统来维护一个钩子链，以后来先服务原则进行调度。要实现 *Win32* 的系统钩子，必须调用 *SDK* 中的 *API* 函数 *SetWindowsHookEx* 来安装这个钩子函数[35]。如果是全局钩子函数，即可拦截其他应用程序的消息，则函数代码必须包含在 *DLL*（动态链接库）

中，同时要将钩子句柄等数据全局共享。得到控制权的钩子函数在完成对消息的处理后，如果想要该消息继续传递，那么它必须调用另外一个 *SDK* 中的 *API* 函数 *CallNextHookEx* 来传递它。钩子函数也可以通过直接返回 *TRUE* 来丢弃该消息，并阻止该消息的传递。

在 *EPSS* 系统中，将使用 *WH_JOURNALRECORD* 日志钩子来监视鼠标的动作。我们预先创建一个小的自定义窗口，一旦鼠标在应用程序的某个窗口上产生点击事件，则全局钩子截获 *WM_LBUTTONDOWN* 消息进入处理程序，此时让自定义窗口在鼠标位置上显示一下，然后隐含掉，这样就会使鼠标所在位置的窗口产生无效区域，强制应用程序刷新这块区域。然后就可以按照前几节讨论的方式进行屏幕取词了。

3.2 *DCOM* 技术分析

3.2.1 进程内组件和进程外组件

在 *EPSS* 系统中，客户端程序对数据库的访问都是通过 *DCOM* 来实现的，在介绍 *DCOM* 的运行机制之前，必须先了解 *COM* 组件的实现方式。根据上面的介绍，我们知道如果我们用动态链接库的方式实现组件程序，则客户程序调用组件程序的服务时，会把组件程序加载到自己的进程中，所以客户程序和组件程序运行在同一个进程空间中，我们把这种组件程序称为进程内组件[8]。实现组件程序的另一种形式是 EXE 程序，这种组件程序在被调用时有其自己的进程空间，所以客户程序和组件程序运行在不同的进程空间中，我们把这种组件程序称为进程外组件[8]。这里我们关注的是进程外组件，因为它相当于一个优化了的 *DCOM* 实现。

因为进程外组件程序和客户程序位于不同的进程空间中，它们使用不同的地址空间，所以组件和客户之间的通信必须跨越进程边界，这就涉及下面的问题：

1) 一个进程如何调用另一个进程的函数。
2) 参数如何从一个进程被传递到另一个进程中。

Windows 平台上不同进程之间进行通信的办法很多，包括动态数据交换（*DDE*），以及在屏幕取词中用到的使用进程的共享内存空间等等，*COM* 采用了本地过程调用（*local procedure call*，*LPC*）和远程过程调用（*RPC*）的方法进行进程间的通信，其中 *LPC* 用于同一机器上的不同进程之间的通信，而 *RPC* 用于在不同机器上的进

程之间进行通信。*LPC* 和 *RPC* 的机理相同，但由于 *RPC* 需要通过网络传递消息，因此 *RPC* 更复杂。

我们分析一下在本地客户程序是如何调用进程外组件的。如图 3.7 所示，组件对象实际上是调用一个 *DLL* 模块，这里称之为存根（*stub*）模块，因为它是动态链接库，所以这个调用是直接进行的。客户进程只与同一进程中的代理（*proxy*）对象打交道，组件进程中的组件对象只与同一进程中的存根 *DLL* 打交道，*LPC* 调用只在代理对象和存根 *DLL* 之间进行，当客户程序需要调用组件提供的功能服务时，需要执行图中所示的六个步骤才能完成一个函数调用，这六个步骤分别是：

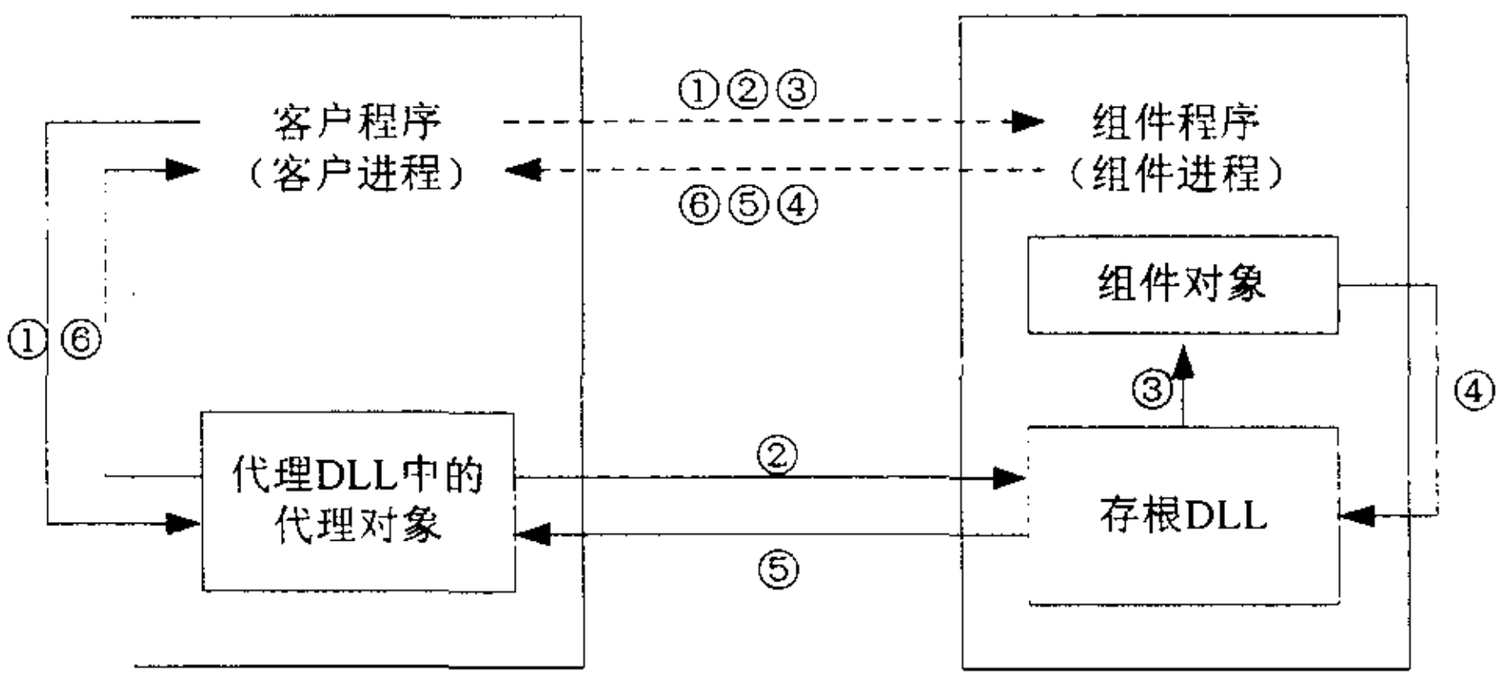

图3.7 进程外组件与客户程序协作的结构图

步骤①：客户调用接口成员函数；

步骤②：代理对象通过 *LPC* 调用组件存根；

步骤③：调用组件对象接口成员函数；

步骤④：服务完成后返回；

步骤⑤：存根 *DLL* 通过 *LPC* 返回结果；

步骤⑥：代理对象返回最终结果；

代理 *DLL* 和存根 *DLL* 除了完成 *LPC* 调用之外，还需要对参数和返回值进行翻译和传递，客户程序调用的参数，首先经过代理 *DLL* 的处理，它把参数以及其他的一些调用信息组装成一个数据包传递给组件进程，这个过程称为参数列集（*marshaling*）[8]；组件进程接受到数据包之后，要进行解包操作，把参数信息提取出来，这个过程称为散集（*unmarshaling*）[8]；然后再进行实际的接口功能调用。

如果我们只考虑客户程序和组件程序，那么就简化得多了，客户程序调用接口成员函数就好像是直接进行得，如图 3.7 中得虚线所示，*COM* 正是通过这种方式实现进程模型的透明特性的。如果组件程序运行在不同的机器上，则代理 *DLL* 和存根 *DLL* 就通过 *RPC* 方式进行网络上的过程调用，从而实现分布式的组件对象模型（即 *DCOM*），所有这些特性的实现可以完全建立在操作系统提供的 *LPC* 和 *RPC* 模块基础上，而不需要开发人员去考虑这些底层的细节技术问题。

3.2.2 分布式 *COM*（*DCOM*）及其实现

我们知道 *COM* 作为组件软件的基本模型，它所建立的软件具有许多优点，比如多种语言混合编程、可重用性、软件可配置性好等。*DCOM* 作为 *COM* 的扩展，它不仅继承了这些特性，针对分布式环境它也提供了些新的特性，比如位置透明、网络安全性、跨平台调用等。这里我们主要讨论 *DCOM* 的基本结构和运行机理。

在上一节我们介绍了 *COM* 中进程外组件通信是如何实现透明传输的。对照 *DCOM* 组件与客户程序的通信过程[16]（如图 3.8 所示），对于客户和组件分别在

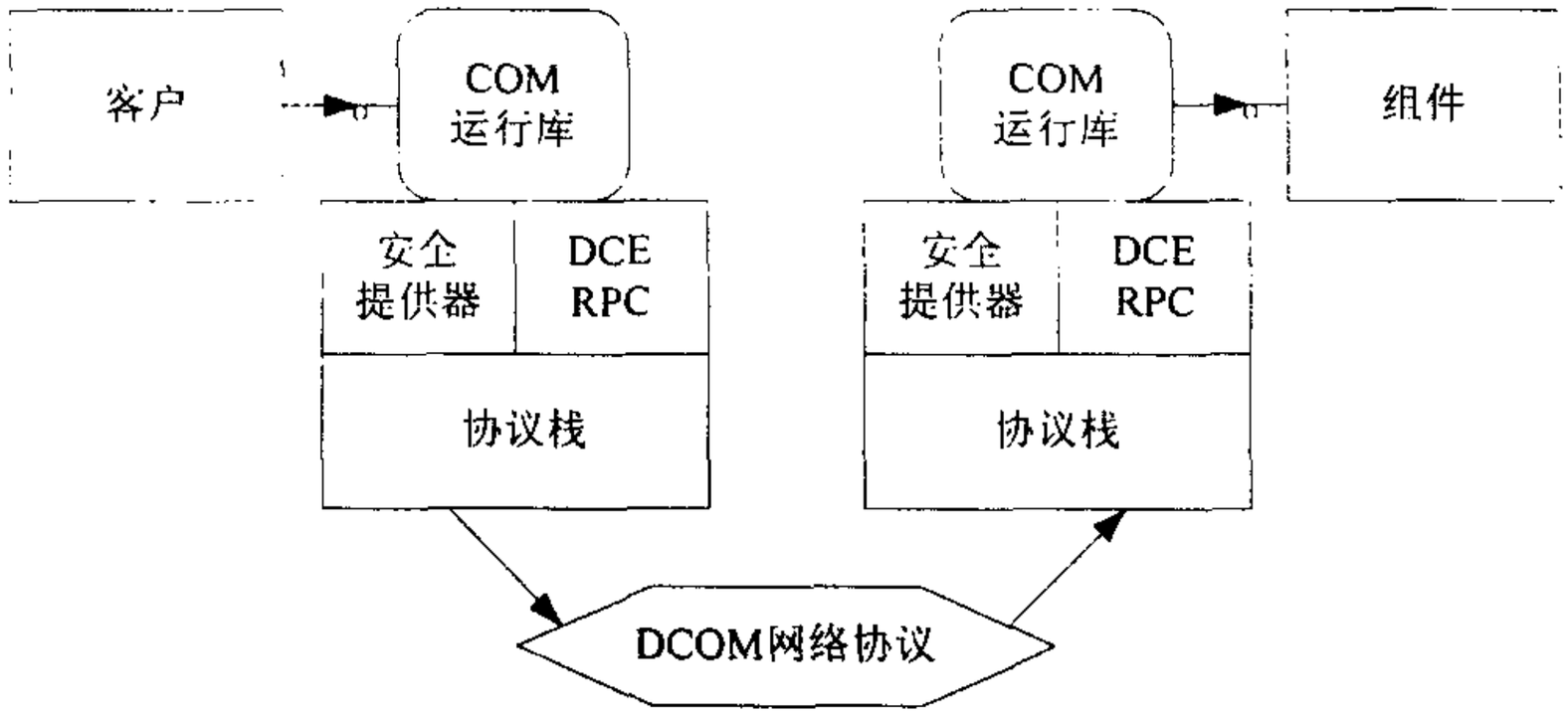

图3.8 客户与DCOM组件的通信示意图

不同的机器上的情形，*DCOM* 只是简单地把本地跨进程通信用一个网络协议传输过程来代替，客户代码和组件对象都感觉不到中间发生的过程，只是中间数据传递的路线更长一些。*DCOM* 的网络协议是基于 *DCE*（分布式计算机环境）*RPC* 的，所以在一个可以使用 *DCE RPC* 的平台上实现 *DCOM* 系统是比较容易的。*DCE RPC* 定义了经过证实是有效的标准来将存储器中的数据结构和参数变换为网络包。它的网络数据表示标准（*NDR*）是与平台无关的，并且提供了一套丰富的可用数据类型。

上面我们看到的是 *DCOM* 调用是如何从客户程序传到组件对象的，现在分析一

下客户程序如何创建一个 *DCOM* 对象。在 *COM* 体系中，每个组件对象都有一个 *CLSID* 作为对象的唯一标识。但创建一个 *DCOM* 对象，仅仅一个 *CLSID* 信息并不够，我们还必须指定待创建的对象位于哪个机器上。一旦机器名和对象 *CLSID* 给定了，那么我们可以调用 *COM* 库的基本创建函数（如 *CoGetClassObject*）创建远程组件对象。

当远程组件对象被创建之后，它在返回到客户机器的途中，还要经过列集（*marshaling*）和散集（*unmarshaling*）的处理，包括创建代理对象和装载存根代码等，这些处理与本地进程外组件对象的处理完全一致。一旦组件对象被创建完成之后，客户与组件之间的通信直接通过代理对象和存根对象以及 *COM* 库提供的底层传输机制来完成。

在 *EPSS* 系统中，客户端应用程序访问数据库的工作由 EpssAdmin.Learner 等 15 个组件对象来完成，*COM* 对象是使用 *Visual C++* 6.0 中 *ATL* 开发的。*ATL*（*Active Template Library*）是 Visual C++提供的一套基于模板的 C++类库，利用这些模板我们可以方便建立 *COM* 组件程序。它的内部模板类实现了 *COM* 的一些基本特征，比如一些基本 *COM* 接口 IUnknown、IDispatch 等，也支持 *COM* 的一些高级特性，如双接口、*ActiveX* 控制等。利用 *ATL* 开发组件的具体过程和代码这里就不详细介绍了。这里需要说明的是如何生成代理和存根 *DLL*。

生成代理和存根 *DLL* 并不需要程序员自己编写代码，我们可以借助于 *COM* 提供的使用 *IDL* 语言的接口描述文件，使用 *MIDL* 编译器即可生成代理和存根 *DLL*。*IDL* 语言同 *RPC* 规范一样是从 *OSF*（开放软件基金会）的 *DCE*（分布式计算环境）借用过来。它的语法和 *C*、*C++*一样，可以细致地描述接口和组件所共享的接口和数据。在用 *IDL* 对接口和组件进行描述后，可以通过 *MIDL*（*Microsoft* 的 *IDL* 编译器）来编译它，生成相应的代理和存根 *DLL* 代码，对这些代码进行编译和链接之后，即可得到相应的代理和存根 *DLL*。

使用 *ATL* 编制组件程序，编译成生 *DLL* 时会自动使用 *MIDL* 编译组件的 *IDL* 文件。以组件对象 EpssAdmin 为例，使用 *ATL* 模板创建工程后，会自动生成 EpssAdmin.idl 文件，程序员只需编写接口函数代码。当编译 *COM* 代码生成组件动态链接库 EpssAdmin.dll 时，*ATL* 会自动使用 *MIDL* 编译 EpssAdmin.idl 文件，生成 EpssAdmin.h、EpssAdmin_i.c、EpssAdmin_p.c 和 dlldata.c 等几个文件。将这几个文件和 EpssAdmin.def 文件一起使用 *Visual C++*编译器编译就可以得到该代理和存根 *DLL* 了，这里我们命名为 EpssAdminps.dll。然后我们在客户端和远端机器上使用

Regsvr32.bat 注册这个代理存根 *DLL*，这样，我们在编程时只需指定组件模块所在机器的 *IP* 地址，就可以使用 *COM* 库的基本函数在本地调用远端机器提供的组件服务。

3.3 本章小节

本章对 *EPSS* 系统中所用到的屏幕取词、*COM* 和 *DCOM* 等关键技术的原理和实现做了深入的分析。本章的内容很多都是涉及 *Windows* 操作系统的核心的技术，是笔者在编程实践中对相关问题所作的分析总结。在阐述的过程中本章注重原理分析，对代码编制没有做详细介绍。

本章首先介绍了屏幕取词的基本原理，提出对 *ExtTextOut/TextOut* 等字符输出 *API* 函数进行拦截是取词技术实现的关键。在介绍实现屏幕取词的具体方法前，先简要介绍了 *Windows* 操作系统的内存管理机制，其中包括虚拟地址、线性地址、物理地址和进程私有地址空间以及共享地址空间等在 *Windows* 系统编程中非常重要的概念。

在介绍了 *API* 函数拦截的基本原理后，分析比较了实现 *API* 函数拦截的几种方法，提出在 *EPSS* 系统中，使用修改被拦截函数入口地址代码的方法进行拦截。

Windows API 函数拦截的实现在技术上是较复杂的，在不同的操作系统中，拦截不同类型的 *API* 函数，实现手段各不相同。本章详细介绍了在 *Windows* 95/98 操作系统中拦截 32 位和 16 位 *API* 函数的实现步骤和方法，其中用到了修改系统中断/意外 *IDT* 表等未公开的技术。

在 *API* 函数拦截中最复杂的技术就是在 32 位函数和 16 位函数互相调用的问题，即“*thunk*”技术。本章在分析了可执行模块的 *PE* 文件和 *DLL* 函数输出表的格式的基础上，介绍了如何使用 *Windows* 未公开的 *API* 函数“*QT_Thunk*”等实现“*thunk*”技术。

最后介绍了在屏幕取词中如何监视鼠标的动作，即使用 *Windows* 全局钩子（*HOOK*）。

对于 *COM* 和 *DCOM* 技术，本章主要介绍了其实现机制。首先介绍了进程内组

件和进程外组件的概念，指出代理和存根 *DLL* 是实现进程外过程调用的关键。然后结合 *EPSS* 系统实际介绍了如何使用 *ATL* 开发实现 *COM* 组件 *DLL* 和代理存根 *DLL*，最终实现分布式组件。

4 基于粗糙集理论的 *EPSS* 系统决策分析研究

本章在理论上介绍了如何在 *EPSS* 系统中进行决策分析，如何将粗糙集（*Rough Sets*）理论应用于其中进行知识发现（*KDD*），同时根据粗糙集理论提出了相关具体算法。本章首先分析在 *EPSS* 系统中决策规则提取的目的，然后介绍粗糙集理论方法的基本原理，针对 *Jelonet* 属性约简算法效率不高，提出了改进的α算法，并且结合 *EPSS* 系统，详细阐述了决策分析的全过程和具体算法。最后介绍了如何利用粗糙集理论进行知识过滤。

4.1 知识表达系统

4.1.1 问题分析

回顾第二章 *EPSS* 系统体系结构和工作模式介绍，在 *EPSS* 系统中，用户学习课件的主要方式是通过 User 端应用程序提供的用户接口（UI）通知 *EPSS Server* 要学习的 Tutorial（章）和 Lesson（具体课程），然后进行该 Lesson 操作步骤的学习。但是我们必须考虑到这样一个事实：对于有经验的用户来说，对于 Tutorial（章）中 Lesson 的选择可能是有针对性的，而对于一个对目标应用程序（Target Application）不熟悉的用户来说，进行 Lesson 选择就有盲目性了，这类用户学习的对象就不能以 Lesson 为单位，而应以 Tutorial 为单位。

鉴于以上原因，*EPSS* 系统应该提供这样的工作模式：由用户选择要学习的 Tutorial（章），由系统安排要学习的 Lesson（具体课程）形成一个 Schedule（课程安排），然后按此 Schedule 学习每一个 Lesson。

如何安排 Schedule 中的 Lessons 是讨论的核心。我们的最终目的是 Schedule 的 Lesson 安排能使用户最快掌握目标应用程序的操作。考虑到用户对 Tutorial 中每个 Lesson 的关注程度不同（例如：有一个 Lesson 教用户如何在 Word 中打开文件，一个熟练的电脑用户对这个 Lesson 毫不感兴趣，而一个电脑初学者可能想学习这个 Lesson），对于所有的用户安排同样的 Lesson 显然是不科学和低效的。因此最好能够根据不同的用户群确定一个 Lesson 集合，里面每个元素为该用户群最关注的 Lesson，这样我们指定 Schedule 时，将这些 Lesson 作为必学课程，其他课程由用户选学，以达到提高用户学习效率的目的。我们做如下数学定义：

定义用户群 G 关注的 Lesson 为 MCL_G（Most Concerned Lesson）；

定义 U_T 为课件集合，包含该 Tutorial 的所有 Lesson；

定义集合 $L_G=\{a \mid a \in U_T,\ a \text{ is } MCL_G\}$，本章所有的讨论都是基于用户群 G 的，所以简称 L；

L 集合的获得有两种方式：一种是由课间的制作者根据专家经验在制作课件时规定一个 Tutorial 的 L 集是什么。从上面的讨论可以看到不同的用户对 lesson 的关注程度是不同的，即 MCL 是相对于不同的用户群而言的，让课件制作者针对每个用户群都指定某个 Tutorial 的 L 集显然是不现实的。另一种方法是根据用户提供的统计信息形成知识库，通过对知识库进行规则提取的过程来获得 L 集。由于统计样本集是由用户群提供的，规则提取的过程由计算机来完成，不但大大降低了课件制作者的工作量，同时也避免了专家经验在某些方面与实际情况相比可能产生的偏差。这种工作方式实际就是数据挖掘（*Data mining*）过程，数据挖掘是知识发现（*KDD*）的核心部分，知识发现是在积累了大量数据后，从中识别出有效的、潜在的、有用的及最终可以理解的知识，人们利用这些知识改进工作，提高效率和效益。本章的中心内容就是讨论如何对知识库进行数据挖掘得到集合 L。

4.1.2 知识库和决策表

知识库是进行数据挖掘（*Date mining*）和数据库知识发现（*KDD*）的基础。得到目标集合 L 的过程可以用下面图 4.1 表示。

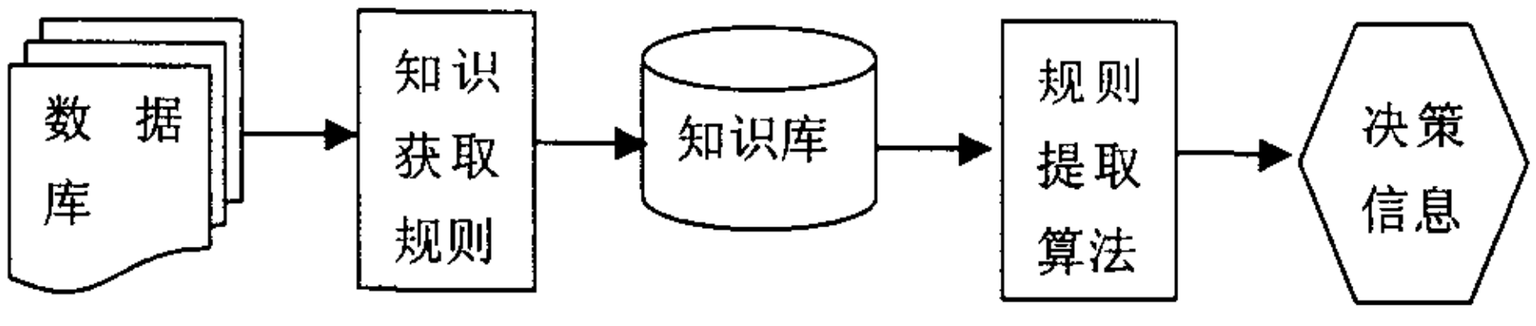

图 4.1 知识发现模型

首先需要从数据库中依靠知识获取规则得到知识库，然后用规则提取算法得到决策信息。获取的决策规则应该是变化的，即 L 集合的元素会随着用户要求的不断变化而变化，这就需要从数据库中的变化信息中得到可变的知识库。

知识表达系统的数学模型定义如下[43]：

定义 **4.1**　四员组 $S=(U, A, V, f)$ 是一个知识表达系统，其中

U：对象的非空有限集合，称为论域；

A：属性的非空有限集合；

$V=\bigcup_{a\in A} Va$，Va 是属性 a 的值域；

f：$U\times A\to V$ 是一个信息函数，它为每个属性赋予了一个信息值，即

$\forall a\in A,\ x\in U, f(x,a)\in Va$

知识表达系统的数据以关系表的形式表示，关系表的行对应要研究的对象，列对应对象的属性，对象的信息是通过指定对象的各属性值来表达的。

决策表是一类特殊而重要的知识表达系统，决策表可以根据知识表达系统定义如下[43]：

定义 **4.2**　设 $S=(U, A, V, f)$ 为一知识表达系统，$A=C\cup D, C\cap D=\phi$，C 称为条件属性集，D 称为决策属性集。可简写为 $S=(U, C\cup D)$，具有条件属性和决策属性的知识表达系统称为决策表。

实际上各种规则提取算法都是应用于根据知识库形成的决策表中的。下面介绍在 *EPSS* 系统中设计怎样的决策表得到集合 ***L***。

4.1.3 决策表设计

根据如上定义我们设计如下决策表：决策表的条件属性 C 是某个 Tutorial 的 Lesson 课件集合，即其中每个元素对应每个 Lesson（用 *L1*，*L2*，*L3*....表示）；决策表的决策属性 D 有一个元素，称之为满意度，用 i 表示；集合 C 中元素的取值是 1 或 0；集合 D 中属性 i 取值是 1 或 0。

决策表产生的过程是：用户对 C 集合中的每一个 Lesson 都学习一遍，然后进行测试，将测试结果作为 C 集合元素的值，测试通过用 1 表示，测试没通过用 0 表示。用户进行完学习和测试后，有的课程学会了，有的没学会，这时让用户对这次的学习结果做出评价，如果对学到的内容满意则将 i 设为 1，否则设为 0。

决策表的设计基于以下考虑：理想决策表的 C 集合条件属性应该是某 Tutorial 的 Lesson 集合，D 集合决策属性应该是用户的学习效果，对该决策表进行规则提取能得到影响用户学习效果的 Lesson 子集，我们有理由认为这个 Lesson 子集中的元

素应该是用户所关注的 Lesson（即 *MCL*），因此可以把这个 Lesson 子集作为该 Tutorial 的 ***L*** 集。在实际决策表中我们用满意度代替学习效果，由用户在学习、测试完毕后对自己的学习结果给出评价。满意度是一个主观的评价，而学习效果似乎更应该是一个客观的评价，之所以选择满意度作为决策属性是因为针对 *EPSS* 系统的实际应用给出一个客观评价比较困难，例如：一个客观的评价应该是出一个类似试卷的考题，试题由课件的制作者给出，在 *EPSS* 系统中，设计这样的试题和让每个用户都去做它是不现实的。在满意度是如实反映用户真实感想的情况下，针对某一用户群的满意度进行知识发现得到的 Lesson 子集同样是有意义的，我们有理由把这个子集作为该 Tutorial 的 ***L*** 集。当然不排除用户给的满意度是无意义的（比如用户把所有 lesson 全学会了，他仍然选学习效果不满意），这种情况下就要对这样的样本进行过滤，有关知识过滤的问题将在第 4.3 节详细阐述。下面的讨论都是基于样本有意义的情况进行的。表 4.1 是我们设计的决策表的一个例子，需要说明是的，受实验条件限制，该决策表并非来自真实数据，它仅供我们进行理论分析时使用，以下介绍的各算法分析都是基于这个决策表进行的。为了不使运算过于复杂，这里取 8 个 Lesson（*L1*－*L8*）、8 个样本（*U1*－*U8*）进行讨论。为方便，下面讨论中以 *a,b,…,h* 代替 *L1*，*L2,...,L8*。

表 4.1　决策表例子

条件属性(*C*)								决策属性(*D*)
User	*L1*(*a*)	*L2*(*b*)	*L3*(*c*)	*L4*(*d*)	*L5*(*e*)	*L6* (*f*)	*L7*(*g*)	*L8*(*h*)	满意度（*i*）
U1	1	0	1	1	1	0	1	1	1
U2	0	1	0	1	0	1	1	0	1
U3	1	1	0	1	0	0	1	1	0
U4	0	0	0	1	1	0	0	1	1
U5	0	1	1	0	0	1	1	1	0
U6	1	1	0	1	1	0	1	1	1
U7	1	1	0	0	1	0	1	1	0
U8	0	0	0	0	0	1	0	1	0

如何对表 4.1 决策表进行规则提取得到 ***L*** 集合，将应用下面介绍的粗糙集理论，以及根据粗糙集理论提出的具体算法。

4.2 粗糙集（*Rough Sets*）理论及相关算法研究

4.2.1 粗糙集（*Rough Sets*）理论的基本概念[36][37][39][43]

粗糙集理论是一种新的处理模糊和不确定性知识的数学工具。其主要思想就是在保持分类能力不变的前提下，通过知识约简，导出问题的决策或分类规则。其主要特点是不需要预先给定某些特征或属性的数量描述，如统计学中的概率分布、模糊集理论中的隶属度或隶属函数等，而是直接从给定问题的描述集合出发们，通过不可分辨关系和不可分辨类确定给定问题的近似域，从而找出该问题中的内在规律[36]。由此看出粗糙集理论很适合用来解决 *EPSS* 系统中面临的决策分析问题。粗糙集理论作为一种集合论，有很多的抽象定义和定理。除参考上面定义的知识表达系统外，这里还归纳整理出一些基本定义，作为理解粗糙集理论的基础：

定义 **4.3**　设 $U \neq \phi$ 是我们感兴趣的对象组成的有限集合，称为论域。设 R 是 U 上的等价关系（在集合中是自反、对称且可传递的关系），U/R 表示 R 所有等价类（或在 U 上的分类）构成的集合。一个知识库就是一个关系系统 $K=(U, \boldsymbol{R})$，$\boldsymbol{R}$ 是 U 上的一族等价关系。实际上知识库关系表中的每个属性都是一个等价关系，$\boldsymbol{R}$ 即相当于属性集。

定义 **4.4**　设 $\boldsymbol{R}$ 是 U 上的等价关系族，若 $\boldsymbol{P} \subseteq \boldsymbol{R}$，且 $\boldsymbol{P} \neq \phi$，则 $\cap \boldsymbol{P}$（$\boldsymbol{P}$ 中所有等价关系的交集）也是一个等价关系，称为 $\boldsymbol{P}$ 上的不可区分关系（*Indiscernibility*），记为 *ind*（$\boldsymbol{P}$）。$U/$ *ind*（$\boldsymbol{P}$）为 U 基于等价关系 *ind*（$\boldsymbol{P}$）的分类，表示与等价关系族 $\boldsymbol{P}$ 相关的知识。

ind（$\boldsymbol{P}$）的等价类称为知识 $\boldsymbol{P}$ 的基本概念或基本范畴。令 $X \subseteq U$，R 为 U 上的一个等价关系。当 X 能表达成某些 R 基本范畴的并时，称 X 是 R 可定义的；否则称 X 为 R 不可定义的。R 可定义集是论域的子集，它可在知识库中精确的定义，也称为 R 精确集；而 R 不可定义集不能在这个知识库中定义，也称为 R 粗糙集（*Rough Sets*）。

定义 **4.5**　对于粗糙集可以近似地定义，我们使用两个精确集，即粗糙集的上近似（*upper approximation*）和下近似（*lower approximation*）来描述[42]。

给定知识库 $K=(U, \boldsymbol{R})$，对于每个子集 $X \subseteq U$ 和一个等价关系 R，定义两个子集：

$$\underline{R}X=\bigcup\{Y \in U/R \mid Y \subseteq X\} \tag{4.1}$$

$$\overline{R}X=\bigcup\{Y \in U/R \mid Y \cap X \neq \phi\} \tag{4.2}$$

分别称为 X 的 R 下近似集和上近似集。

集合 $\mathrm{bn_R}(\mathrm{X}) = \overline{R}X - \underline{R}\mathrm{X}$ 称为 X 的 R 边界域；

集合 $pos_R(X) = \underline{R}X$ 称为 X 的 R 正域；

集合 $neg_R(X) = U - \overline{R}X$ 称为 X 的 R 负域。

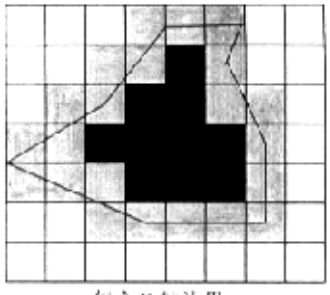

图4.1 粗糙近似

X 的 R 下近似集是那些根据知识 R 判断肯定属于 X 的 U 中元素组成的集合；X 的 R 上近似集是那些根据知识 R 判断可能属于 X 的 U 中元素组成的集合；$bn_R(X)$ 是 U 中那些根据知识 R 既不能判定肯定属于 X 又不能判断肯定属于 $U-X$ 的元素；$neg_R(X)$ 是那些根据知识 R 判定不属于 X 的 U 中元素集合。图 4.1 给出了粗糙近似的抽象图形表达。

定义 4.6 在决策系统中 $S=(U, C \cup D)$ 中，设 $R \subseteq C$，f 为决策属性 D 所决定的 U 的划分 $\{X_1, X_2, \ldots, X_n\}$，$f$ 的 R 下近似和上近似分别定义为：$\underline{R}f=\{\underline{R}X_1, \underline{R}X_2, \ldots, \underline{R}X_n\}$ 和 $\overline{R}f=\{\overline{R}X_1, \overline{R}X_2, \ldots, \overline{R}X_n\}$。定义一个量度为根据 R，f 的近似分类精度：

$$\alpha_R(f)=\frac{\sum_{i=1}^{n}|\underline{R}X_i|}{\sum_{i=1}^{n}|\overline{R}X_i|} \tag{4.3}$$

其中 $|A|$ 表示集合 A 的基数。

近似分类的精度描述的是当使用知识 R 对对象分类时，可能决策中正确决策的百分比。在下面论述中，f 可省略。

结合以上定义，将粗糙集的概念与普通集合论相比较，可以看出粗糙集的基本性质（如元素的成员关系），与不可区分关系所表示的论域的知识有关。因此，一个元素是否属于某一集合，不是该元素的客观的性质，而是取决于我们对它的了解程度。

4.2.2 属性约简（*reduct*）和核（*core*）[40][41]

知识约简（属性约简）是粗糙集理论的核心内容之一。众所周知，知识库中知识（属性）并不是同等重要的，甚至其中某些知识是冗余的。所谓知识约简，就是在保持知识库分类能力不变的条件下，删除其中不相关或不重要的知识[41]。

知识约简中有两个基本概念：约简（*reduct*）和核（*core*）。

定义 4.7　令 $\boldsymbol{R}$ 为一族等价关系，$r\in\boldsymbol{R}$，如果 $ind(\boldsymbol{R})=ind(\boldsymbol{R}-\{r\})$，则称 r 为 $\boldsymbol{R}$ 中不必要的；否则称 r 为 $\boldsymbol{R}$ 中必要的。如果每个一个 $r\in\boldsymbol{R}$ 都为 $\boldsymbol{R}$ 中必要的，则称 $\boldsymbol{R}$ 为独立的；否则称 $\boldsymbol{R}$ 为依赖的。

设 $\boldsymbol{Q}\subseteq\boldsymbol{P}$，如果 $\boldsymbol{Q}$ 是独立的，且 $ind(\boldsymbol{Q})=ind(\boldsymbol{P})$，则称 $\boldsymbol{Q}$ 为 $\boldsymbol{P}$ 的一个约简。显然 $\boldsymbol{P}$ 可以有多种约简。$\boldsymbol{P}$ 中所有必要关系组成的集合称为 $\boldsymbol{P}$ 的核，记做 $core(\boldsymbol{P})$。

核与约简的关系是：$core(\boldsymbol{P})=\cap red(\boldsymbol{P})$，其中 $red(\boldsymbol{P})$ 表示 $\boldsymbol{P}$ 的所有约简。

在决策系统中，一个分类（C 集合）相对于另一个分类（D 集合）的关系十分重要，因此我们要使用知识的相对约简（*relative reduct*）和相对核（*relative core*）的概念。

定义 4.8　令 P 和 Q 为 U 中的等价关系，Q 的 P 正域记为 $pos_P(Q)=\bigcup_{X\in U/Q}\underline{P}X$，$Q$ 的 P 正域是 U 中所有根据分类 U/P 的信息可以准确地划分到关系 Q 的等价类中去的对象集合。

令 $\boldsymbol{P}$ 和 $\boldsymbol{Q}$ 为等价关系族，$R\in\boldsymbol{P}$，如果 $pos_{ind(\boldsymbol{P})}(ind(\boldsymbol{Q}))=pos_{ind(\boldsymbol{P}-\{R\})}(ind(\boldsymbol{Q}))$，则称 R 为 $\boldsymbol{P}$ 中 $\boldsymbol{Q}$ 不必要的；否则 R 为 $\boldsymbol{P}$ 中 $\boldsymbol{Q}$ 必要的。如果 $\boldsymbol{P}$ 中的每个 R 都为 $\boldsymbol{Q}$ 必要的，则称 $\boldsymbol{P}$ 为 $\boldsymbol{Q}$ 独立的。

设 $\boldsymbol{S}\subseteq\boldsymbol{P}$，$\boldsymbol{S}$ 为 $\boldsymbol{P}$ 的 $\boldsymbol{Q}$ 约简当且仅当 $\boldsymbol{S}$ 是 $\boldsymbol{P}$ 的 $\boldsymbol{Q}$ 独立子族且 $pos_{\boldsymbol{S}}(\boldsymbol{Q})=pos_{\boldsymbol{P}}(\boldsymbol{Q})$。$\boldsymbol{P}$ 的 $\boldsymbol{Q}$ 约简简称为相对约简。$\boldsymbol{P}$ 中所有 $\boldsymbol{Q}$ 必要的原始关系构成的集合称为 $\boldsymbol{P}$ 的 $\boldsymbol{Q}$ 核，简称为相对核，记为 $core_{\boldsymbol{Q}}(\boldsymbol{P})$。同样有 $core_{\boldsymbol{Q}}(\boldsymbol{P})=\cap red_{\boldsymbol{Q}}(\boldsymbol{P})$。下面的讨论中出现的约简和核都是指相对约简和相对核。

约简和核是粗糙集理论中非常重要的概念，在下面的文章中，将结合决策表 4.1 介绍这些比较抽象的概念。

4.2.3 利用区分矩阵（*discernibility matrix*）求核（*core*）算法

求核是进行决策表属性约简的基础，在已知关于粗糙集研究成果中，波兰华沙大学数学家 *Skowron* 提出的区分矩阵为求取属性约简提供了很好的思路。该方法将信息表中所有有关属性区分信息都浓缩进一个矩阵当中，可以通过这个矩阵方便地得到信息表的属性核[46]（即信息表中不可删除的属性）。

定义 4.9 决策系统 $S=(U, C \cup D)$ 中，U 为论域且 $U=\{x_1, x_2, \ldots, x_n\}$，$C$ 为条件属性集，D 是决策属性，$a(x)$ 是记录 x 在属性 a 上的值，区分矩阵 M 可表示为：

$$(m_{ij})=\begin{cases}\{a\in C: a(x_i)\neq a(x_j)\} & D(x_i)\neq D(x_j)\\ 0 & D(x_i)=D(x_j)\quad i,j=1,2...n\\ -1 & a(x_i)=a(x_j)\quad D(x_i)\neq D(x_j)\end{cases} \tag{4.4}$$

决策表 4.1 的区分矩阵如表 4.2 所示，由于区分矩阵是对称矩阵，因此我们仅需考虑矩阵的上三角或下三角元素。

表 4.2 决策表 4.1 的区分矩阵

	U1	*U2*	*U3*	*U4*	*U5*	*U6*	*U7*	*U8*
U1								
U2	0							
U3	*b,c,e*	*a,f,h,*						
U4	0	0	*a,b,e,g*					
U5	*a,b,d,e,f*	*c,d,h*	0	*b,c,d,e,f,g*				
U6	0	0	*e*	0	*a,c,d,e,f*			
U7	*b,c,d*	*a,d,e,f,h*	0	*a,b,d,g*	0	*d*		
U8	*a,c,d,e,f,g*	*b,d,g,h*	0	*d,e,f*	0	*a,b,d,e,f,g*	0	

由区分矩阵的定义我们知道：矩阵中的元素是条件属性集 C 中的属性组合，属性组合数为 1 表明，除该属性外其余条件属性无法将信息表中决策不同的两条记录区分出来，即该属性必须保留，这与决策表中核属性的概念一致。因此，矩阵中所有属性组合数为 1 的元素的属性均为决策表的核属性（显然，核属性可能为空）。令 C_H 是区分矩阵 M 中的核属性集，有 $C_H \subseteq C$。这种求核算法简称算法 ***H***。设决策表

有 m 个属性，n 条记录，算法 H 的时间复杂度是 $O(n^2m)$。

根据上面分析，表 4.2 区分矩阵的核 $C_H=\{d,e\}$。

4.2.4 属性约简算法

粗糙集理论中已有一些属性约简方法[44]。例如可以根据区分矩阵得到决策表的属性约简。考虑到区分矩阵包含了决策表中所有属性区分信息，因此，核属性外的其余有用属性应从属性组合数不为 1 的矩阵元素中分析取得。假设某区分矩阵仅中有两个属性组合没有核元素（多于两个的情况可依此类推），分别表示为 $a_1a_2..a_m$ 和 $b_1b_2...b_n$。根据区分矩阵可知，如果要识别所有决策不同的记录，则 a_i（$i=1,2,...,m$）与 b_j（$j=1,2,...,n$）之中必然至少各需保留一条属性。将属性值设为布尔变量，值为 1 表示包含该条件属性，值为 0 表示不包含该条件属性。构造表达式：

$$P=(a_1\vee a_2\vee ...\vee a_m)\wedge(b_1\vee b_2\vee ...\vee b_n), \qquad (4.5)$$

由以上分析知道 $P=1$。P 为合取范式形式，将其转化为析取范式形式，则 P 中任何合取式的值如果为 1，$P=1$ 都成立[47]。因此该合取式代表的属性组合连同核属性即可将原决策表中的所有决策区分出来。依照上面原理即可得到决策表所有的属性约简。表 4.2 的区分矩阵中，不包含核属性的属性组合为：$\{a,f,h\}$，$P=a\vee f\vee h$，则决策表的所有属性约简为$\{a,d,e\}$或$\{d,e,f\}$或$\{d,e,h\}$。

利用区分矩阵得到决策表的属性约简被称为基本算法，但是其中重要的一步，即将 P 由合取范式形式转化为析取范式形式，要使用布尔表达式的分配律、吸收律等定律，编程实现很复杂，而且此算法的时间复杂度可以证明为 $O(2^{|M|}n^2m)$（$|M|$ 是核的基数），是指数阶的算法，因此，实际应用算法都大多还是从相对约简的定义入手，但参考基本算法得出的结论还是很方便的。

在其他求属性约简的算法中，*Jelonek* 等人提出了逐步扩展型算法[44]应用比较广泛，其基本思想是：

从 $R=core(C)$开始，如果$\gamma_R=\gamma_C$，（γ_R为 R 近似分类质量，与近似分类精度相似，在公式 4.3 中分母以$|U|$代替 f 分类所有 R 上近似之和）则 R 为属性约简；否则，对所有的属性 $a\in C\backslash R$（设 B 为 A 的子集，A/B 表示 A 中除去 B 集合元素的剩余元素集合，下同）计算 $gain=\gamma_{R\cup\{a\}}-\gamma_R$，若 a'是使相应的 $gain$ 达到最大的属性，则

置 $R=R\cup\{a'\}$并判断它是否为约简，若是，则结束；否则继续上述过程。在该算法中，从核 (可以是空集)开始，每扩展一次，都要对所有的属性 $a\in C\backslash R$，计算新的近似分类质量 $\gamma_{R\cup\{a\}}$，计算量是比较大的。因此我们提出 α 算法。

4.2.5 α 算法介绍

考虑到 *Jelonek* 算法中，从核开始每扩展一次，都要计算所有属性的近似分类质量，计算量大。于是考虑利用单个属性的近似分类精度进行扩展，而不是计算所有属性的近似分类质量，并利用区分矩阵进行修正，这样显然能减少计算量。根据此想法得到一个新的属性约简算法 (称为 α 算法)，其基本过程如下：

决策系统 $S=(U, C\cup D)$ 中，U 为论域，C 为条件属性集，D 是决策属性，

步骤 1　对每个条件属性 C_i，$i=1,\cdots,m$，计算 α_{C_i}（公式 4.3）；
令 $Q:=core(C)$（由算法 H 求得，可以为空集），$P:=Q$，$M:=C\backslash P$；
如果 P 满足 $pos_P(D)=pos_C(D)$，P 为约简（定义 4.8），结束，否则步骤 2；

步骤 2　令 $\alpha_{C_g}:=\max(\alpha_{C_i}: C_i\in M)$，$P:=P\cup\{C_g\}$，$M:=M\backslash\{C_g\}$，执行步骤 3；

步骤 3　如果 P 满足 $pos_P(D)=pos_C(D)$，执行步骤 4，否则转步骤 2；

步骤 4　如果 $P=core(P)$（由算法 H 求得），P 为一个约简，结束；否则步骤 5；

步骤 5　任取 $C_j\in P\backslash core(P)$,令 $P:=P\backslash\{C_j\}$，转步骤 4；

下面分析算法的原理和时间复杂度：

α 算法中，P 满足 $P\subseteq C$, $P=core(P)$，则 P 是 C 的 D 独立子族，满足 $pos_P(D)=pos_C(D)$，根据定义 4.8，P 是 C 的属性约简。在步骤 2 中选择近似分类精度高的属性进行扩展，是因为根据近似分类精度的定义（定义 4.6），属性的近似分类精度越高越有可能是属性的约简中的元素，而且我们也希望求得的属性约简中能有尽可能多的近似分类精度高的属性。

设决策表有 m 个属性，n 条记录。求每个属性的近似分类精度 α_{C_i}（公式 4.3）时，对每个属性进行 U/R（$R\subseteq C$）分类的算法时间复杂度是 $O(n^2)$，求 R 上、下近似的算法时间复杂度是 $O(n)$，所以，公式 4.3 的时间复杂度是 $O(n^2)$。则计算 m 个属性的 α_{C_i} 的时间复杂度是 $O(n^2m)$。对 m 个 α_{C_i} 用快速排序算法进行排序的

时间复杂度是 $O(m\log(m))$，利用算法 H 求核的时间复杂度是 $O(n^2m)$，求 $pos_P(D)$（定义 4.8）的时间复杂度是 $O(n^2)$，所以当 α 算法执行到步骤 4 时，算法的时间复杂度是 $O(n^2m)$。进入步骤 4 后，每执行一次步骤 4 都要对逐步缩小的 P 计算 $core(P)$，因此最坏情况下求得属性约简的时间复杂度是 $O(n^2(1+2+...+m))=O(n^2m^2)$。综上所述，$\alpha$ 算法的时间复杂度是 $O(n^2m^2)$。

在 *Jelonek* 算法中，每求一次 γ_R 的时间复杂度也是 $O(n^2m)$。但从核开始每进行一次扩展，对所有属性都要计算 $\gamma_{R\cup\{a\}}$，所以最坏情况下求得属性约简的时间复杂度是 $O(n^2m(1+2+...+m))=O(n^2m^3)$。因此，*Jelonek* 算法的时间复杂度是 $O(n^2m^3)$。

根据上面的分析，就时间复杂度而言，α 算法比 *Jelonek* 算法少了一个数量级，因此从理论上改进了 *Jelonek* 算法。根据上一节的介绍可知，决策表的属性约简并不唯一，而用 α 算法只能得到一个属性约简。但考虑到如果要得到所有的属性约简，问题将变的极为复杂，而且在实际应用中也没必要，并且通过 α 算法的到的属性约简基本上是最简属性约简，而这一点正是我们对 *EPSS* 系统决策表进行属性约简所希望得到的。所以 α 算法是适合于我们目前讨论的实际应用。下面介绍如何根据 α 算法对表 4.1 进行属性约简：

由算法 H 得表 4.1 决策表的核为 $\{d,e\}$，初始状态 $P=\{d,e\}$，对每个条件属性 C_i $i=1,\cdots,m$，计算 α_{C_i}：

由 D 决定的 U 划分为 $U/D=\{X_1,X_2\}=\{\{U1,U2,U4,U6\},\{U3,U5,U7,U8\}\}$；

由 C 决定的 U 划分为 $U/C=\{\{U1\},\{U2\},\{U3\},\{U4\},\{U5\},\{U6\},\{U7\},\{U8\}\}$；

条件属性 h 决定的 U 的划分为：$U/h=\{\{U1,U3,U4,U5,U6,U7,U8\},\{U2\}\}$；

则 $\underline{h}X_1=\{U2\}$，$\underline{h}X_2=\{\phi\}$；

$\overline{h}X_1=\{\{U1,U2,U3,U4,U5,U6,U7,U8\}\}$，$\overline{h}X_2=\{\{U1,U3,U4,U5,U6,U7,U8\}\}$；

则 $\alpha_h=1/15$；根据此方法求得其他属性 α 值都为 0；

所以步骤 2 对 P 进行扩展时，将属性 h 加入，$P=\{d,e,h\}$；

根据定义 4.8 计算 $pos_C(D)=\{U1,U2,U3,U4,U5,U6,U7,U8\}$；

由 P 决定的 U 划分为 $U/P=\{\{U1,U4,U6\},\{U2\},\{U3\},\{U7\},\{U5,U8\}\}$，

$pos_P(D)=\{\{U1,U2,U3,U4,U5,U6,U7,U8\}$，则有 $pos_P(D)=pos_C(D)$；

步骤 4 中求 *core(P)*，将表 4.2 区分矩阵中的 *a,b,c,f,g* 元素去掉得到 *P* 的区分矩阵，可得 *core(P)*= {*d,e,h*}，则 *P* =*core(P)*。至此，算法结束，得到了属性约简{*d,e,h*}。原决策表取属性 *d,e,h*，去掉重复项后得到如表 4.3 的简化决策表。

表 **4.3** 属性约简后的决策表

No.	*L4*（*d*）	*L5*（*e*）	*L8*（*h*）	满意度（*i*）
1	1	1	1	1
2	1	0	0	1
3	1	0	1	0
4	0	1	1	0
5	0	0	1	0

该决策表提供了 5 条基本的决策规则。对这些决策规则还可做进一步简化得到最简决策规则，下面就将讨论决策规则约简的算法。

4.2.6 决策规则约简算法

决策规则的约简是利用决策逻辑分别消去决策算法中每一个决策规则的不必要条件，它不是整体上的简化属性，而是针对每一个决策规则，去掉表达该规则的冗余属性值，以便进一步简化决策规则法。

这里有必要先介绍一下相容的概念[43]：

决策系统 $S=(U, C Y D)$ 中，任何一个蕴涵式 $\varphi \rightarrow \mu$ 称为一个决策规则，φ, μ 分别称为 $\varphi \rightarrow \mu$ 的前件和后件。若 $\varphi \rightarrow \mu$ 在 S 中为真，则称 $\varphi \rightarrow \mu$ 在 S 中是相容的，否则，称为在 S 中是不相容的。以表 4.3 决策表为例，规则 $d_1 e_1 h_1 \rightarrow i_1$ 在 S 中是相容的，而规则 $e_0 h_1 \rightarrow i_1$ 在 S 中是不相容的。与属性约简类似，规则 $\varphi \rightarrow \mu$ 的所有必要属性所成的集合，称为该规则的核，记作 *core*（$\varphi \rightarrow \mu$）。

在实际决策表中，如果有两条记录条件属性完全一样，但决策属性却不相同，那么这两条记录决定的决策规则显然是 S 不相容的，这两条记录对我们决策产生没有帮助，因此，在规则约简前，可以将互不相容的两条规则中在总记录 U 中比例少的那条去掉。这个过程的时间复杂度是 $O(n^2)$。

关于决策规则约简，文献[43]给出了详细的理论分析，这里将其归纳整理为实际

应用的算法，简称算法 ***G***。其步骤如下：

步骤 1　对决策表中每条决策规则的条件属性进行逐列考察，除去该列后，若产生不相容规则，则保留该属性的原属性值；若没产生不相容规则但有重复记录，将该属性值标为＃；若产生了新的规则，将该属性标为？，步骤 2；

步骤 2　删除可能产生的重复记录，并考察每条含有标记？的记录，若仅由未被标记的属性值即可得到与原决策表相容的规则，则将？改为＃，否则修改为原属性值；若某条 记录的所有条件属性均被标记，则将标有？的属性项修改为原属性值，步骤 3；

步骤 3　删除可能产生的重复记录，并删除所有条件属性被标为＃的记录，步骤 4；

步骤 4　如果两条记录仅有一个条件属性值不同，且其中一条记录该属性被标为＃，那么如果属性为＃的那条记录的未标记属性能得到相容规则，则删除另外一条记录，否则删除本记录，结束。

容易证明，最坏情况下步骤 1 的时间复杂度是 $O(n^2m)$，步骤 2 的时间复杂度是 $O(n^2m)$，步骤 4 的时间复杂度 $O(n^3)$，一般情况下，决策表中 $n>m$，所以算法 ***G*** 在最坏情况下时间复杂度是 $O(n^3)$。

以表 4.3 为例，介绍算法 ***G*** 的具体执行过程：

表 4.3 原决策表

No.	*L4*（*d*）	*L5*（*e*）	*L8*（*h*）	*i*
1	1	1	1	1
2	1	0	0	1
3	1	0	1	0
4	0	1	1	0
5	0	0	1	0

表 4.4 步骤 1 结果

No.	*L4*（*d*）	*L5*（*e*）	*L8*（*h*）	*i*
1	1	1	?	1
2	?	?	0	1
3	#	0	1	0
4	0	#	?	0
5	#	#	?	0

对于表 4.3，以记录 4 为例，若去掉属性 d，剩余属性构成的规则 $e_1h_1 \rightarrow i_1$ 与第 1 条记录不相容，则保留原属性值；去掉属性 e，$d_0h_1 \rightarrow i_0$ 与记录 5 重复，将该属性值标记为#；去掉属性 h，产成了新的规则 $d_0e_0 \rightarrow i_0$，将属性值标记为？；依此经过步骤 1 得到表 4.4。

对于表 4.4，记录 1 的 h 属性为？，记录 1 未标记属性形成的规则 $d_1e_1 \rightarrow i_1$ 是相容规则，则将？改写为#；依此经过步骤 2、3，得到表 4.5。

对于表 4.5，记录 3 和记录 5 仅 e 属性不同，且记录 5 的 e 属性为#，由于 $h_0 \rightarrow i_0$ 是不相容规则，所以保留记录 3，去掉记录 5；依此完成步骤 4，得最简决策表以及相应的决策规则。

表 **4.5** 最简决策表及相应决策规则

No.	*L4*（*d*）	*L5*（*e*）	*L8*（*h*）	满意度（*i*）	决策规则
1	1	1	#	1	$d_1e_1 \rightarrow i_1$
2	#	#	0	1	$h_0 \rightarrow i_1$
3	#	0	1	0	$e_0h_1 \rightarrow i_0$
4	0	#	#	0	$d_0 \rightarrow i_0$
~~5~~	~~#~~	~~#~~	~~1~~	~~0~~	

通过算法 G 我们得到了 4 条决策规则，我们最终目的从最简决策规则中得出那些 lesson 是该用户群的 *MCL*。我们做如下处理：

步骤 1　从决策规则中取后件值为 1 的规则，因为后件值为 1 表式用户对学到的内容表示满意，我们显然是想知道那些 Lesson 学会了能使用户对学习效果满意。

步骤 2　对于每个后件为 1 规则，计算样本种符合该规则的样本的个数，取在样本中出现次数最多的规则作为最终决策规则。取其中属性值为 1 的元素作为 *MCL*。当两条规则在样本中出现次数一样，把前件中属性为 1 的元素多的规则作为决策规则，这样使得该 Tutorial 的 L 集包含的元素多一些，更能保证反映大多数人的意见。

经过步骤 1，去掉了 3、4 两条规则；由于决策表中有 3 个样本符合规则 1，1 个样本符合规则 2，所以我们取决策规则 $d_1e_1 \rightarrow i_1$ 为最终决策规则，由此决策得出 *L4* 和 *L5* 为该用户群的 *MCL*。于是当该用户群的用户学习这个 Tutorial 时，*L4*、*L5* 将被安排为用户必须学的 Lesson，其他 Lesson 则列出供用户选学。

4.3 知识样本过滤

4.3.1 问题分析

在 4.1 节的讨论中，我们规定进行数据挖掘的知识库应该是可变的，以便能随时反映用户需求的变化，这就需要不断加入新的知识进入知识库。而针对采集到的“新知识”存在着两个疑问：

1> 该知识是否正确，是否与已掌握的知识一致；

2> 该知识是否揭示了新的规律。

如果该知识是由于某种原因的谬误的，显然不应加入到知识库中，否则会破坏原有决策系统的性能；如果该知识实质上是原有知识的一种隐式体现，符合已掌握的知识，则应将其加入知识库中；如果该知识确实是我们以前所不掌握的新知识，这时就应该扩展知识库的结构和内容。由此可以看出，对待获取的原始知识并不能简单地直接加入知识库。

这里我们考虑利用粗糙集理论中知识依赖度的概念，对新知识进行评价，在新知识输入到知识库之前对知识进行过滤，以保证知识的一致性，达到维护和完善知识库的目的。于是图 4.1 的知识发现模型应该修改为图 4.2 的加入知识过滤的知识发现模型：

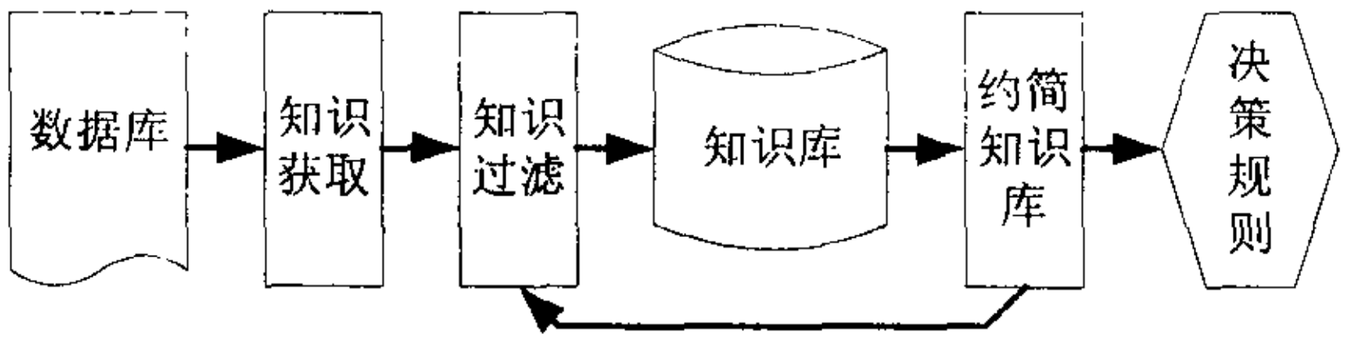

图4.2 加入知识过滤的知识获取模型

4.3.2 知识样本过滤的基本原理及算法[39][45]

定义 **4.10** 决策系统 $S=(U, C\cup D)$ 中，C 是条件属性，D 是决策属性，当

$$k=\gamma_C(D)=\frac{|pos_C(D)|}{|U|} \tag{4.5}$$

时，我们称 D 是 k（$0\le k\le 1$）度依赖于知识 C 的，记做 $C\Rightarrow_k D$。其中$|A|$表示集合 A

的基数。

由定义 4.8 正域 $pos_C(D)$ 的概念和上面依赖度的定义可知，当 $C \Rightarrow kD$ 时，由 D 导出的分类 U/D 的正域覆盖了知识库的 $k\times100\%$ 个元素；另一方面，对象的 $k\times100\%$ 个元素可以通过知识 C 划入分类 U/D 的模块中。

设获取一条新知识，与原信息系统 S 组成一个新的决策系统 $S'=(U', C'\cup D')$，计算决策系统 S'中的决策属性 D'对于条件属性集 C'的依赖度 k'：

$$k'=\gamma_{C'}(D')=\frac{|pos_{C'}(D')|}{|U'|}$$

对于新获取的知识，它只能属于下面两种可能性之一。

可能性 1：新知识与已掌握的知识一致。

这里一致的意义是指粗糙近似，即新知识包含在原知识的上近似集以内。

如果当 $pos_C(D)=U$，这时新知识在原有分类的基础上能划入 U'/D'分类，则

$$k'=\frac{|pos_{C'}(D')|}{|U'|}=\frac{|pos_C(D)|+1}{|U|+1}=1=k$$

当 $pos_C(D)\neq U$，原知识存在边界域，如果新知识不属于边界域，则

$$k'=\frac{|pos_{C'}(D')|}{|U'|}=\frac{|pos_C(D)|+1}{|U|+1}>k$$

如果知识属于原知识的边界域，由于正域是计算知识分类的下近似集，所以新知识同样不能划入 U'/D'分类，则

$$k'=\frac{|pos_{C'}(D')|}{|U'|}=\frac{|pos_C(D)|}{|U|+1}<k$$

一般情况下 U 的基数较大，可见，当新知识于已掌握的知识近似时，依赖度值维持不变或稍有增减。同时下式必然成立：

$$k'\geq\frac{|pos_C(D)|}{|U|+1} \tag{4.6}$$

可能性 2：新知识与已掌握的知识不一致。

这时，新加入的知识破坏了原有的知识的分类能力，则除了新知识外，原有知识中与新知识构成等价类的那些元素，也不能划入 U'/D'分类。此时

$$k'=\frac{|pos_{C'}(D')|}{|U'|}=\frac{|pos_C(D)|-|[x]_{ind(C)}|}{|U|+1}<\frac{|pos_C(D)|}{|U|+1} \tag{4.7}$$

这里$[x]_{md(C)}$表表示 C 中包含新知识 x 的等价类。可见，当新知识与已掌握的知识矛盾时，依赖度的值明显降低。

综合以上分析并结合实际应用，归纳出知识过滤的基本算法，简称算法 ***F***。算法流程图如下：

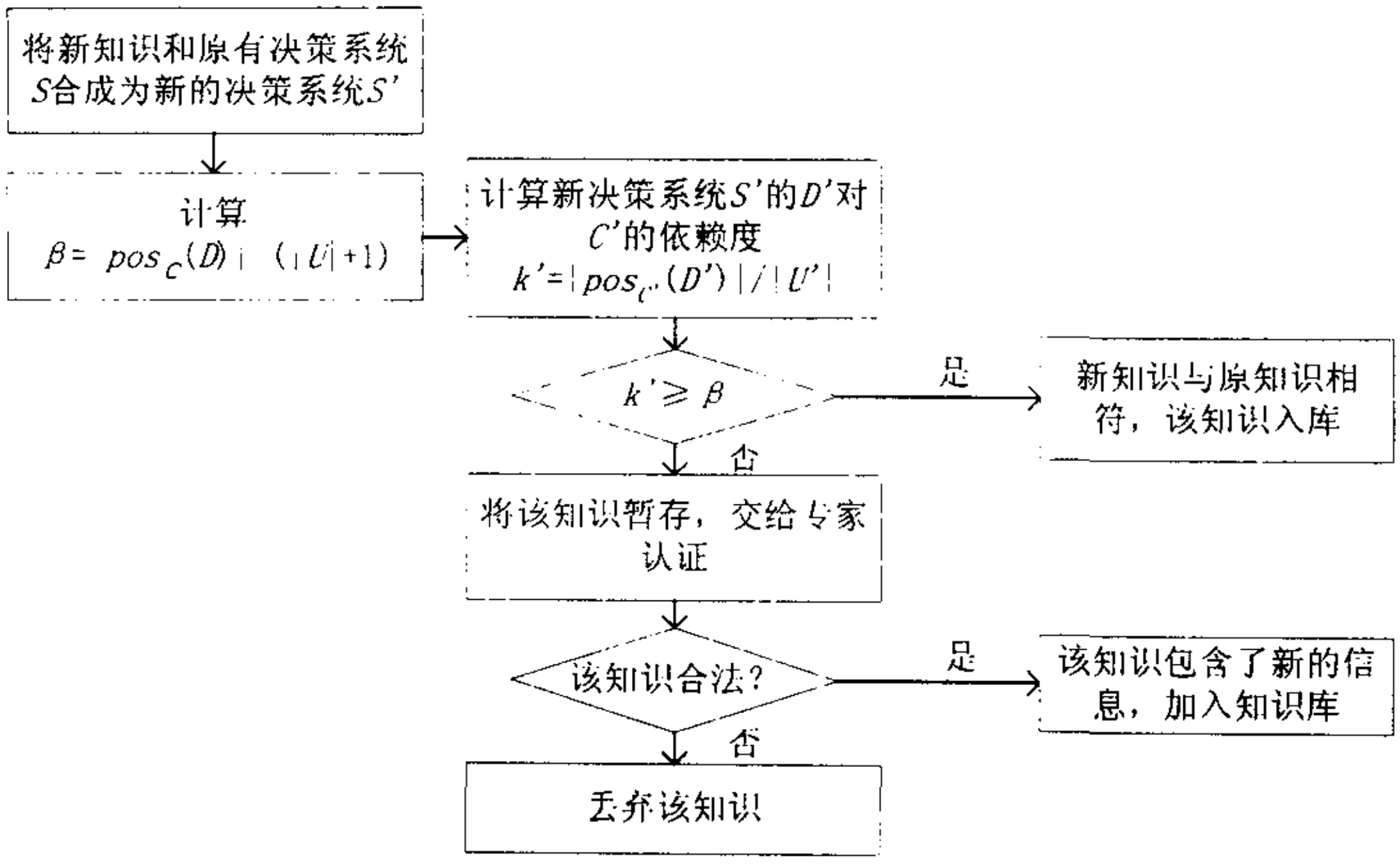

图4.3 算法*F* 流程图

算法中求 $pos_C(D)$的时间复杂度是 $O(n^2)$，则算法 ***F*** 的时间复杂度是 $O(n^2)$。在算法中，考虑到实际应用，新知识就是加入的新的记录，如果新记录与原知识不一致，则有可能该记录是完全无意义的，不过也有可能存在一种我们以前还未了解的属性特征在作用，所以不能盲目丢弃。因此我们将这些记录暂存，交给专家去分析，如果专家认为该知识是合法的，就将该记录加入知识库，否则丢弃。下面结合决策表 4.1 谈谈算法 ***F*** 如何应用。

4.3.3 算法 *F* 应用实例

以表 4.6 基础知识库为例介绍算法 ***F*** 的应用。表 4.6 是在决策表 4.1 的基础上加入一条记录 *U9*，*U9* 的含义是不言自明的，用户所有的课程都没学会，满意度当然选 0。类似这样的记录可以由专家指定，作为基础规则放入知识库中。

表 4.6 基础知识库

条件属性(C)								决策属性(D)
User	*L1(a)*	*L2(b)*	*L3(c)*	*L4(d)*	*L5(e)*	*L6 (f)*	*L7(g)*	*L8(h)*	满意度（*i*）
U1	1	0	1	1	1	0	1	1	1
U2	0	1	0	1	0	1	1	0	1
U3	1	1	0	1	0	0	1	1	0
U4	0	0	0	1	1	0	0	1	1
U5	0	1	1	0	0	1	1	1	0
U6	1	1	0	1	1	0	1	1	1
U7	1	1	0	0	1	0	1	1	0
U8	0	0	0	0	0	1	0	1	0
U9	0	0	0	0	0	0	0	0	0

这时有一条记录 *U10*：

U10	0	0	0	0	0	0	0	0	1

加入知识库，这条记录是无意义的。应用算法 ***F*** 进行分析：

在原知识库决策系统 *S* 中：

由 *D* 决定的 *U* 划分为 $U/D=\{X_1,X_2\}=\{\{U1,U2,U4,U6\},\{U3,U5,U7,U8,U9\}\}$；

由 *C* 决定的 *U* 划分为 $U/C=\{\{U1\},\{U2\},\{U3\},\{U4\},\{U5\},\{U6\},\{U7\},\{U8\},\{U9\}\}$；

根据定义 4.8 计算 $pos_C(D)=\{U1,U2,U3,U4,U5,U6,U7,U8,U9\}$；

加入记录 *U10* 后，形成新的决策系统 *S'*：

计算 $\beta=|pos_C(D)|/(|U|+1)=9/10$；

由 *D'* 决定的 *U'* 划分为 $U'/D'=\{X_1,X_2\}=\{\{U1,U2,U4,U6,U10\},\{U3,U5,U7,U8,U9\}\}$；

由 *C'* 决定的 *U'* 划分为：

$U'/C'=\{\{U1\},\{U2\},\{U3\},\{U4\},\{U5\},\{U6\},\{U7\},\{U8\},\{U9,U10\}\}$；

$\underline{C'}X_1=\{U1,U2,U4,U6\}$，$\underline{C'}X_2=\{U3,U5,U7,U8\}$

$pos_{C'}(D')=\{U1,U2,U3,U4,U5,U6,U7,U8\}$；　$k'=8/10$；

由于 $k'<\beta$，*U10* 在进入知识库前被截获，交给专家认证，最终由于其不合法被丢弃。

上面举了一个典型的例子介绍了知识过滤系统是如何工作的。在实际应用中，我们还要考虑些实际问题：比如初始知识库是从训练样本中提取的，我们至少应该

保证训练样本的记录是有意义的，否则就谈不上什么知识过滤了；还有专家进行认证的方式等问题。由于本章立足于理论方法研究，这些问题就不详细讨论了。

4.4 本章小结

本章对 *EPSS* 系统决策分析问题进行研究，分析如何利用粗糙集理论解决实际问题，进行数据挖掘，并且对决策分析的每一步都提出或归纳总结出具体算法，同时结合实际的例子，介绍了算法的实用性和可操作性。

本章首先分析了 *EPSS* 系统决策分析要解决的实际问题，提出最终目的就是利用决策分析得到的决策规则找到 Tutorial 中不同用户群的 ***L*** 集即 *MCL* 集。综合考虑可操作性等实际问题提出了决策表的设计方案。

粗糙集理论在处理模糊和不确定性知识方面是非常有用的数学工具。为了便于理解本章的各个算法，本章介绍了粗糙集理论的基础知识和基本定义，尤其是上下近似，近似分类精度，属性约简和核等贯穿本章始终的重要概念。

属性约简是粗糙集理论的核心内容，本章在分析了区分矩阵的构成和意义基础上归纳总结出 ***H*** 算法，并得出算法的时间复杂度为 $O(n^2m)$。在讨论了一些典型属性约简算法的缺点和不足的基础上，针对传统 *Jelonet* 属性约简算法效率不高，提出了改进的 α 算法，并分析了 α 算法的正确性，计算了其时间复杂度为 $O(n^2m^2)$。证明比 *Jelonek* 算法的时间复杂度 $O(n^2m^3)$ 少了一个数量级，从理论上改进了 *Jelonek* 算法。

决策规则约简是粗糙集理论的又一重要内容，在得到了决策表的属性约简后，我们还希望得到决策表的值约简，即规则约简。本章在介绍了规则相容性等重要概念后，根据粗糙集中规则约简的理论分析，归纳总结了 ***G*** 算法，得出算法的时间复杂度为 $O(n^3)$。

在实际应用中，为了防止无意义的记录进入知识库破坏原有知识决策系统的性能，本章提出了利用粗糙集方法进行知识过滤的模型。在分析了利用知识依赖度来进行知识过滤的原理后，提出了具体的 *F* 算法，并计算了算法的时间复杂度为 $O(n^2)$。

对上文提到的各个算法，本章都以表 4.1 决策表作为基础实例分析算法的执行步骤，以体现算法解决实际问题的能力。本章所作的讨论，其应用对象不仅仅是 *EPSS* 系统，数据挖掘技术在远程教学等各个领域里都有广阔的应用前景。

5 *EPSS* 系统应用实现

本章主要是结合前三章的内容，介绍了 *EPSS* 系统的具体实现。同时还结合实际应用，分析了可能遇到的问题和解决方案。

5.1 系统的运行环境

根据第二章的系统体系结构介绍可知，整个系统分为两个部分：服务器和客户端。本课题的实验环境为：

服务器为 *Intel PentiumIII 550MHz*、*Cache Memory 128K*、*Memory Installed 256M* 的 PC 机，服务器安装的软件为 *Windows 2000 Server Version*+*Microsoft SQL Server 7.0*+*Internet Information Server* (*IIS*)。服务器主要负责提供数据库服务、WWW 服务和组件管理。

客户端为 *AMD Duron 650MHz*、*Cache Memory 128K*、*Memory Installed 128M* 的 PC 机。客户端的操作系统为 *Windows* 98。

应用程序的开发环境为 *Microsoft Visual C++* 6.0（开发所有的 32 位应用程序和 *DLL*）和 *Microsoft Visual C++* 1.5（开发 16 位 *DLL*）。*DCOM* 组件模块使用 *Visual C++* 提供的 *ATL* 模板开发。

5.2 系统实现简介

EPSS 系统的功能、原理和系统各模块的设计在第二章已经做了详细阐述，这里就不再介绍了，下面将结合图例对客户端应用程序和服务器组件管理的实现做简要介绍。

5.2.1 Author Tool

Author 端应用程序（简称 Author Tool）的功能就是作为课件制作者（简称作者）制作和编辑课件的工具。程序运行后，首先登陆服务器进行作者的身份验证，根据该作者的权限，将该作者可以编辑的课件，以章（Tutorial）、节（Lesson）和页（Page）

三级目录树的形式列出。作者可以对这些对象的每个属性进行修改。如果作者要新创建一个 Lesson，那么首先通过 Author Tool 的 FileOpen 对话框选择并运行学习对象（目标应用程序），在确认开始记录后，作者对学习对象进行操作，这里要求作者的每次鼠标点击事件都要点击在对象窗体的文字上，由 AuthorTool 将鼠标点击处的文字截取；作者每进行一次键盘操作，Author Tool 将键盘操作的字符串截取；Author Tool 提供了 Ctrl＋Q 热键，作者在需要的时候按此热键进行拷屏截图。操作完毕后，所有的操作步骤将被列在 Author Tool 的列表框里形成了该 Lesson 的所有 Page。作者对每个 Page 都要进行详细编辑，加入必要的属性信息，对需要特殊说明的地方，加入特殊的 Page，即一个对话框，提示运行结果和当前操作要特别注意的地方。编辑完毕后将整个 Lesson 存入数据库，完成课件的制作过程。图 5.1 为对 Page 进行编辑的操作界面。

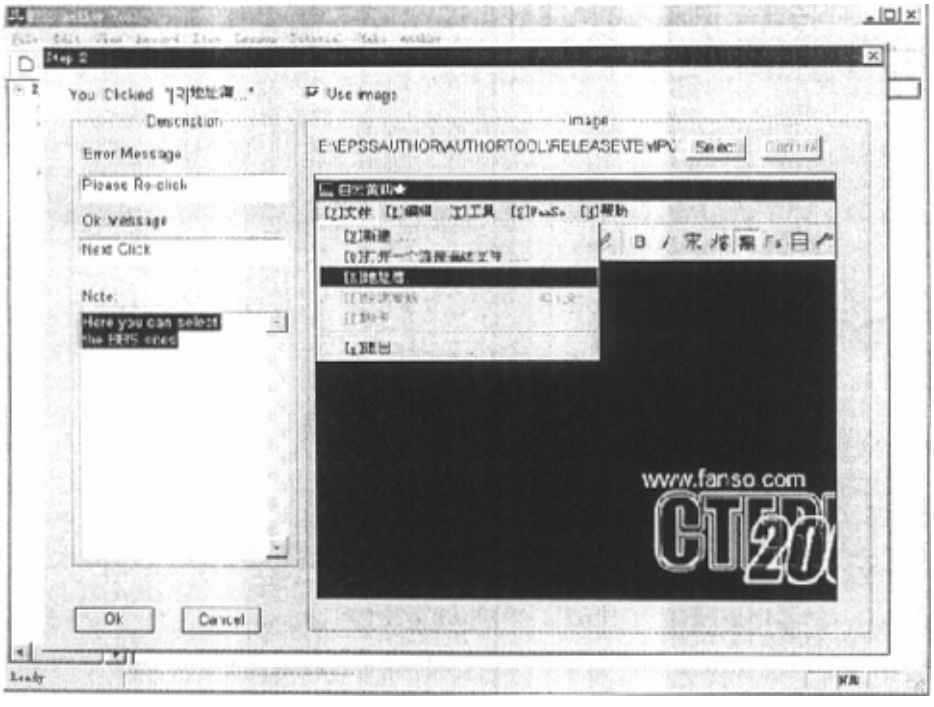

图 5.1 Author Tool 中编辑 Page 的界面

5.2.2 User Tool

User 端应用程序（简称 User Tool）的功能就是指导学习者进行应用软件操作的学习。程序运行后，首先登陆服务器进行学习者身份认证，User Tool 列出服务器上的课件，由用户选择要学习的课件，并且对课件的学习参数进行设定（比如学几次等）。然后 User Tool 根据学习对象的 Module Name（相当于可执行文件名）搜索目标应用程序的可执行文件，如果找到将路径记录下来并执行，否则提示不能进行教学。学习过程开始后，由辅导窗口对话框显示提示信息，学习者根据提示信息对学习对象进行操作，学习者的每次鼠标和键盘操作都会被截取，由 User Tool 进行评价分析，如果正确则进行下一个 Page，同时 User Tool 的浏览器窗口会刷新显示新 Page 的各属性信息，否则停止并提示学习者进行正确的操作。

图 5.2 为 User Tool 中显示 Page 的属性信息的界面。

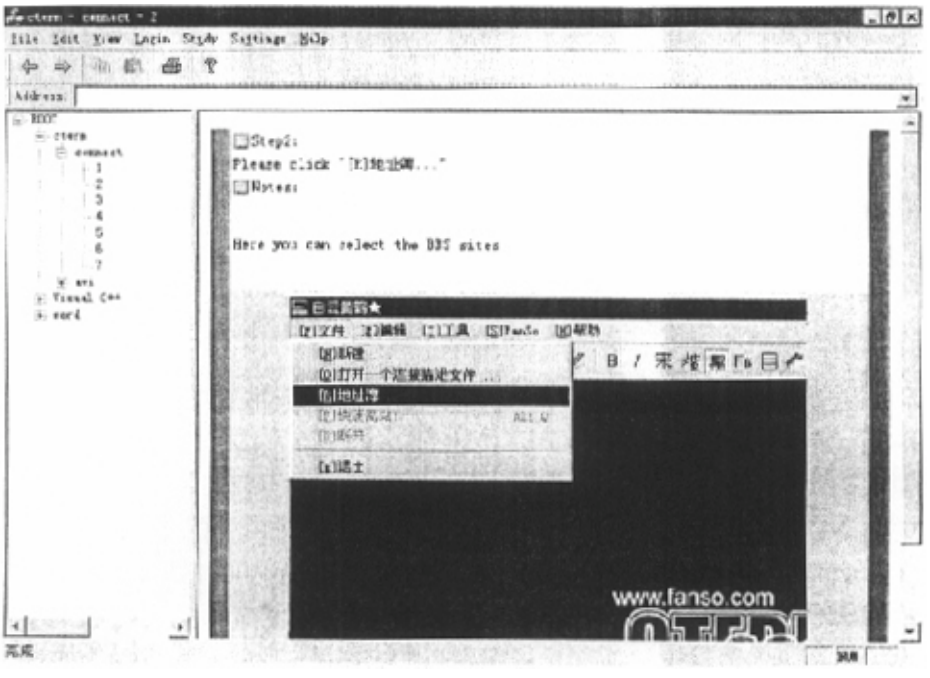

图 5.2 User Tool 中显示 Page 信息的界面

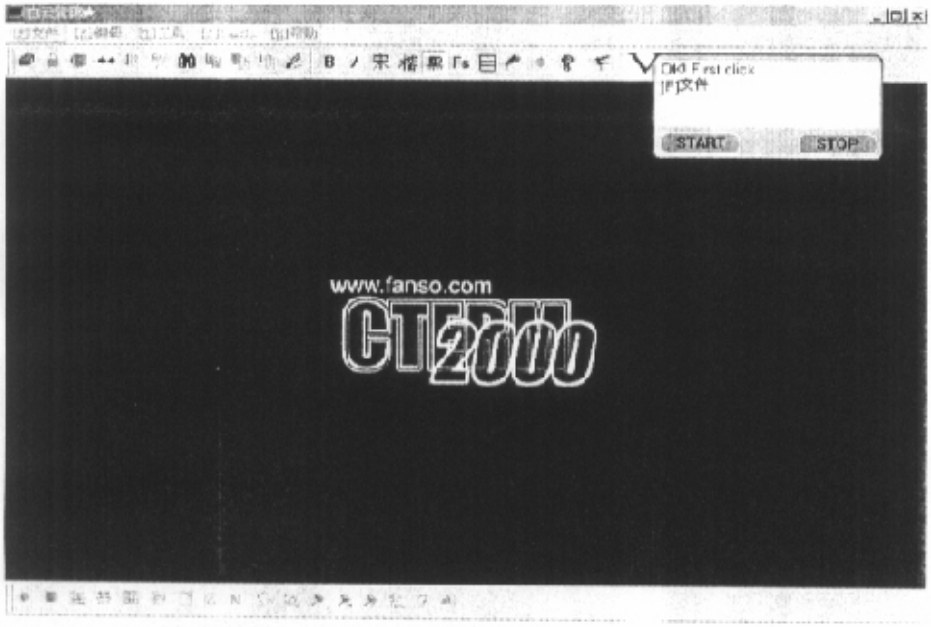

图 5.3 学习实例步骤图例 1

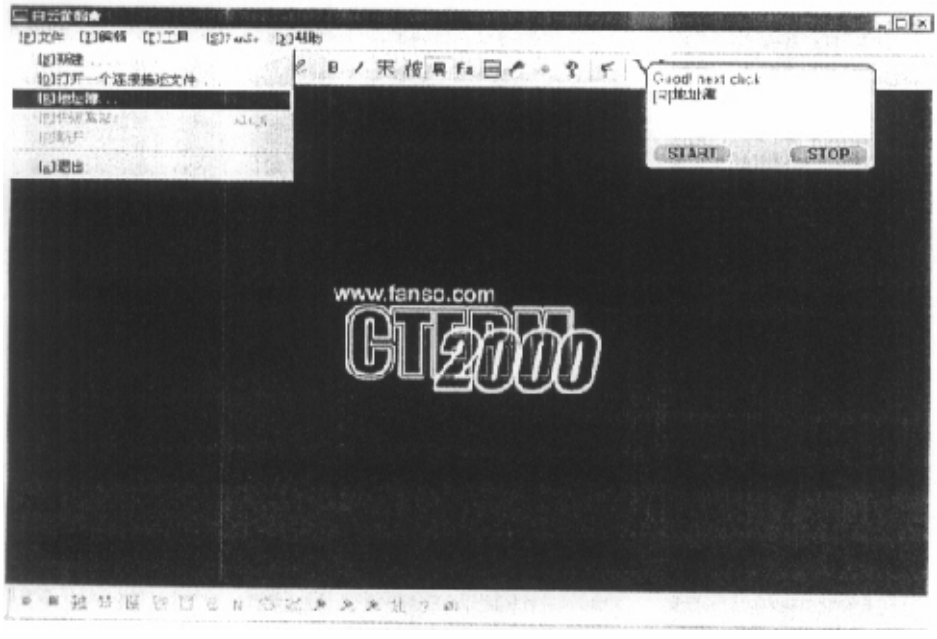

图 5.4 学习实例步骤图例 2

图 5.3 和图 5.4 为一个学习实例的两个步骤。该学习实例的目的是教用户如何使用 CTERM 登陆 BBS，图 5.3 显示提示学习者点击应用程序的“File”菜单，图 5.4 是在点击“File”菜单正确的情况下，提示学习者点击“地址薄”子菜单，进行下一步操作。

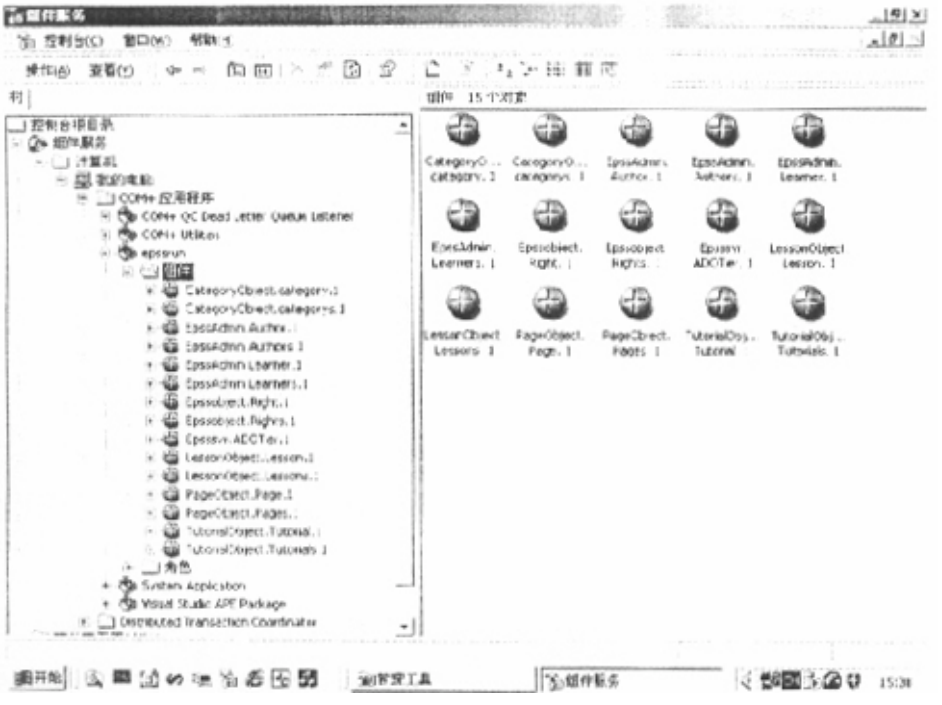

图 5.5 *EPSS* 系统组件管理

5.2.3 组件管理

编制好第二章所介绍的各 *DCOM* 组件模块的 *DLL* 文件后，我们使用 *Windows 2000 Server* 的 *MTS* 组件管理工具对组件进行配置管理。如图 5.5 所示。

5.3 相关问题分析

在 *EPSS* 系统实现的过程中，除了实现软件教学的基本功能外，下列的相关问题需要特别指出的：

1) 在 *EPSS* 系统中，无论是在 User 端应用程序还是 Author 端应用程序，我们都要用到屏幕取词技术来截获用户的操作。但和普通的屏幕取词技术不同的是，这里只有当用户对学习对象的操作进行鼠标操作时，我们才能进行屏幕取词，否则就有可能发生虽然用户不是操作学习对象，但仍然判断操作正确的情况。为了实现只针对学习对象进行屏幕取词，我们使用进程 *Process ID* 作为判断依据。在 *Windows* 95/98 操作系统中，当目标应用程序（即学习对象）运行后，我们可以使用 *Windows API* 函数 *CreateToolhelp32Snapsho* 枚举目前系统运行的所有进程，然后根据学习对象的可执行文件名得到学习对象的 *Process ID*。当用户开始学习后，*EPSS* 客户端应用程序将使用 *HOOK* 截获用户鼠标点击事件所发出的消息，从 *WM_LBUTTONDOWN* 消息中可以得到发出鼠标消息的窗口的句柄 *HWnd*，然后使用 *Windows API* 函数 *GetWindowThreadProcessId* 得到该窗口所在进程的 *Process ID*。如果发消息进程的 *Process ID* 和学习对象的 *Process ID* 相同，我们就进行屏幕取词，否则不进行。
2) 在 *EPSS* 系统中，我们将提供学习测试功能。开始测试后，由学习者对学习对象进行操作，这时 User 应用程序的辅导窗口将不给予指导信息，如果学习者在任何两步操作之间有 2 次鼠标点击错误或 10 秒钟之内没有动作（对键盘操作的处理情况也是一样的），则视为“犯规”并判定测试失败。如果用户在没有提示且没有“犯规”的情况下能完成全部操作，则判定测试通过。
3) *EPSS* 系统中的教学课件之间，可能存在某种相关性，例如有 ***A*** 和 ***B*** 两个课件，只有在学会了 ***A*** 课件的情况下，才能学习 ***B*** 课件，我们称 ***A*** 为 ***B*** 的一个基础课件。在制作课件时，我们必需考虑到这种相关性，因此在制作课件 ***B*** 时，我们允许课件制作者指定该课件的基础课件集。当用户学习 ***B*** 课件时，系统会给予提示，说明必须要先学会该课件的基础课件集才能继续进行，这样可以避免学习者在学习过程中出现无从下手的情况。

以上都是针对 EPSS 系统在实际应用中所发现的问题而提出的解决方案，经过编程实现后，在实际应用中取得了较好的效果。

5.4 系统的不足与改进

本课题根据 *EPSS* 系统的设计要求，对系统设计方案和实现手段进行了全面的分析和研究，完成了系统各模块的编制工作，取得比较满意的效果。但 *EPSS* 系统作

为一个远程软件教学系统，在功能和实现上还存在一些缺憾和需要改进的地方，这主要体现在以下几个方面：

1) 截获用户操作的方法有局限性。在本系统中，我们主要使用屏幕取词方法截获用户的鼠标点击操作，但是如果用户点击了工具栏的图标，或操作对象的窗口没有文字，我们就无能为力了。这样课件制作者在制作课件时就受到了一定的限制，如果学习对象某个功能的操作步骤中涉及了过多的点击图标的操作，对这样功能就不能制作成课件供学习者学习了，这是 *EPSS* 系统最大的缺憾。对于这一问题，其可能解决方法是：如果鼠标点击在图标上，设法得到图标的图像信息，将其作为判断操作是否正确的依据。其实现难度主要在于如何根据鼠标点击位置坐标得到图标的边界顶点坐标。一旦得到边界顶点坐标，我们就可以通过拷屏得到图标的象素信息。对图标进行截取是 *EPSS* 系统需要进一步研究的主要工作。
2) 学习对象的当前状态对学习过程可能会造成一定的影响。根据第二章系统设计所介绍的内容，我们知道在 *EPSS* 系统中，使用屏幕取词技术来截获用户的操作已经大大减小了学习对象的运行状态对学习过程的影响，比如对学习对象运行时各窗体的坐标位置就没有了限制。但不能排除这样的情况：由于某些意想不到的原因，使得学习过程中某一步操作无法进行下去，例如，假设当前操作是点击菜单项 ***C***，但菜单项 ***C*** 却意外的被设为 *Disabled*，这样就无法进行下一步操作了。对于这种情况，我们目前还没有很好的解决办法，因为如果学习对象没有提供接口，那么通过 User 端应用程序来设定其状态，使学习对象内部各控件对象的状态和制作课件时完全一样，这几乎是不可能的。这个问题也将是 *EPSS* 系统今后研究工作中的一个难点。
3) 在现有数据库的基础上建立数据仓库。数据仓库就是面向主题的、集成的、不可更新的(稳定性)、随时间不断变化（不同时间）的数据集合，用以支持经营管理中的决策制定过程。与数据仓库密切相关的知识发现（*KDD*）和数据挖掘（*Data mining*）在现代远程教育系统中有着非常广阔的应用前景。上一章介绍的决策分析和知识过滤系统其实现的基础就是数据库以数据仓库的形式进行组织。在 *LTSC* 的 P1848.*系列草案中，有专门针对数据仓库指定的数据和元数据的一系列规范，*EPSS* 系统可以以这些规范为标准创建数据仓库，适应现代远程教育发展的需要。
4) 决策分析的实现。本文的第四章从理论上介绍了 *EPSS* 系统决策分析和知识样本过滤的过程和各实现算法，由于条件限制，并没有编程实现。在今后的研究工作

中，可以根据这些理论和算法开发实际的应用模块。

5) 人机交互的改进。人机交互是远程教育系统中非常重要的部分，它直接影响到系统的应用和推广。在 *EPSS* 系统中，人机交互主要使用自制的辅导对话框（如图 5.3、5.4 所示），我们可以考虑使用在 *Microsoft Office* 系列应用软件中广泛使用的 *Microsoft Agent* 替代它。*Microsoft Agent* 除了在界面上更美观外，还提供了语音输出的功能。在最新的版本中，微软提供了 *Microsoft Agent* 的编程接口，是我们能将其应用到自己开发的系统中。

以上分析了 *EPSS* 系统存在的主要不足和需要改进的地方，明确了 *EPSS* 系统的进一步研究工作的方向。

结束语

随着多媒体技术、计算机技术、特别是计算机通信和网络技术的不断发展，现代远程教育已经成为人们关注的热点。相对于传统的远程教学模式，现代远程教育具有不受时间和空间的限制、教学资源丰富、教学手段灵活、受教育面广等优点，这使其必将成为未来终身教育的主导形式。

远程软件教学系统是现代远程教育研究领域中的一个重要的课题。本文对 *EPSS*（*Electronic Performance Support System*）远程软件教学系统及其相关的决策分析问题在实现方法、实现技术和理论分析等方面做了详尽的论述。先将本文的主要工作总结如下：

1) 在对现有的远程软件教学系统进行了广泛的研究和分析的基础上，针对传统远程软件教学方式重论述、轻实践的缺点和软件教学强调用户操作的特点，本文提出了一个全新的远程软件教学模式：由用户实际操作应用软件，教学系统对用户的每步操作进行跟踪指导。这种模式在很大程度上提高了远程教学的效率和效果。
2) 本文提出了基于分布式组件技术(*DCOM*)的 *EPSS* 系统框架模型，同时在对 *IEEE P1484.1* 标准草案（*LTSA*）进行深入研究的基础上，提出了基于该标准草案的客户端应用程序体系结构模型。在分析了各种可能的技术实现方案的优缺点的基础上，提出了使用屏幕取词技术来截获用户对目标应用程序的操作的技术方案。
3) 屏幕取词技术，特别是 *Windows API* 函数拦截技术中很多都是涉及 *Windows* 操作系统核心的技术，并且有些是未公开系统核心技术，本文在对该领域内的技术文章进行广泛深入的研究并做了大量实验的基础上，对在 *Windows* 操作系统中实现 *API* 函数拦截的各种方法和技术实现手段进行了全面的总结。
4) 本文分析在 *EPSS* 系统中进行知识发现和决策分析的目的和意义，在此基础上提出了将粗糙集（*Rough Set*）理论应用于 *EPSS* 系统中进行知识发现、决策规则提取和知识样本过滤的解决方案，同时根据粗糙集理论给出了相关具体算法，并结合具体例子介绍了算法的实用性和可操作性。
5) 属性约简是粗糙集理论的核心内容，针对传统 *Jelonet* 属性约简算法效率不高，本文提出了改进的 α 算法，并分析了 α 算法的正确性，计算了其时间复杂度为 $O(n^2m^2)$。证明比 *Jelonek* 算法的时间复杂度 $O(n^2m^3)$ 少了一个数量级，从理论上改进了 *Jelonek* 算法。

本课题根据 *EPSS* 系统的设计要求，对系统设计方案和实现手段进行了全面的分

析和研究，完成了系统各模块的编制工作，取得比较满意的效果。但 *EPSS* 系统作为一个远程软件教学系统，在功能和实现上还存在一些缺憾和需要改进的地方，在上述研究工作的基础上进一步的研究工作主要有：

1) 改进截获用户操作的方法。本系统使用屏幕取词方法截获用户的鼠标点击操作，但是操作对象限于有文字属性的控件。为了进一步提高系统的实用性，必须要解决对应用软件中图标等没有文字属性的控件的操作进行截取这一问题，其技术实现有相当的难度。
2) 在现有数据库的基础上建立数据仓库。第四章介绍的决策分析和知识过滤系统其实现的基础就是数据库以数据仓库的形式进行组织。在 *LTSC* 的 P1848.*系列草案中，有专门针对数据仓库指定的数据和元数据的一系列规范，*EPSS* 系统可以以这些规范为标准创建数据仓库，适应现代远程教育发展的需要。
3) 决策分析的实现。本文的第四章从理论上介绍了 *EPSS* 系统决策分析和知识样本过滤的过程和各实现算法，由于条件限制，并没有编程实现。在今后的研究工作中，可以根据这些理论和算法开发实际的应用模块。

本文对远程软件教学系统研究领域中的各方面问题做了有益的探索，但现代远程教育研究是一个庞大的系统工程，它涉及到核心技术研究、网络平台、软硬件环境、教学资源、教学管理以及立法和政策支持等多方面的因素，因此，后续的研究工作是充满挑战的。国家教育部于 2002 年 2 月 6 日，正式印发了《现代远程教育技术标准体系和 11 项试用标准 V1.0 版》，为我国远程教育系统的研究工作走上规范化道路打下了坚实的基础，这必将推动我国远程教育迈向新的台阶。

致 谢

值此论文完成之际，谨向在我攻读硕士研究生的三年中，曾经关心、帮助、支持和鼓励过我的老师、亲人、朋友和同学致以最真挚的谢意！

首先我要感谢我的导师周曼丽教授，本课题的研究工作从始至终都是在周老师的精心指导与悉心安排下进行的。在我攻读硕士学位的三年时间里，周老师给我提供了良好的学习和科研环境，给了我许多在科研项目中锻炼自己的机会，使我的才能和特长得到了充分的发挥，同时周老师渊博的学识、严谨的治学态度和不断进取的精神给我留下了深刻的印象，使我真正认识到一名科学工作者应具备的精神风貌。谨在此向周老师表示由衷的感谢和崇高的敬意！

感谢许毅平、魏蛟龙两位老师在科研工作和日常生活中给予我的帮助和指导。特别是许毅平老师，他直接指导了本课题的整个研究工作，并提出很多宝贵的意见，在此表示衷心的感谢。

在课题组的蒋文彬、周宁两位博士师兄和梁超、胡萌、周亚军、周文昭等同学的共同努力下，课题取得了阶段性成果。十分怀念大家一起共同奋斗的日日夜夜。在此对他们的努力和支持表示深深的谢意！

96 级提高班的同学们，尤其是何颖、罗彬、郭耀辉等同学经常与我交流学术问题和人生看法，给了我很多启迪，我们相处的十分愉快，在此也要向他们表示感谢！

最后，我要深深地感谢我的父母和家人，没有他们对我在精神上和生活上的支持和鼓励，我不可能取得现在的成绩。

感谢所有关心和关注我的人，愿本文能够回报他们为我付出的一切，谢谢！

郭 飞

2002 年 4 月 于华工园

参考文献

[1] 张劲，朱建平，王强．现代远程教育的含义、发展及实施．杭州师范学院学报，1999，5：117~118

[2] 郭丽云．论我国现代远程教育的发展方向．教育理论与实践，1999，3：23~25

[3] 韦钰．现代远程教育．中国电大教育，1998，10：4

[4] 何克抗．基于计算机网络的教育网络与21世纪的教育革新．中国电子报，1999，5

[5] Randy Garrison. Theoretical Challenges for Distance Education in the 21st Century: A Shift from Structural to Transactional Issues. International Review of Research in Open and Distance Learning, 2000, 1: 11~13

[6] Keegan Desmond. Foundations of Distance Education(2nd ed.). London: Routledge, 1990, 1

[7] 现代远程教育标准化委员会．现代远程教育技术标准体系和 11 项试用标准[DOC]．中国教育和科研计算机网：http://www.edu.cn/html/keyanfz/yuanchengjiaoyu.shtml，2002，2

[8] 潘爱民．COM 原理与应用（第一版）．北京：清华大学出版社，1999．427~433

[9] [美]Randy Abernethy．COM/DCOM 技术内幕（第--版）．汪浩，郭钰，黄正宇等译．北京：电子工业出版社，2000．309~323

[10] 王平，覃理矜．基于 OLE DB 的 ADO 数据访问技术．重庆邮电学院学报，2001，3：65~68

[11] 谢榕．数据挖掘与决策支持系统．计算机系统应用，1999，8：9~11

[12] 楼伟进，孔繁胜．数据仓库与知识发现．计算机工程与应用，2000，10：111~113

[13] Sarabjob Anand, Bryan Scotney. Designing a Kernel for Data Mining. IEEE Expert Intelligent Systems and Their Applications, 1997, 12: 65~74

[14] 胡侃，夏绍玮．基于大型数据库的数据采集研究综述．软件学报，1998，9：23~26

[15] Microsoft Corporation. Microsoft COM White Papers[HTML]. Microsoft Corporation: http://www.microsoft.com/com/wpaper/default.asp#COMPapers, 2002, 1

[16] Microsoft Corporation. Microsoft DCOM White Papers[HTML]. Microsoft

Corporation： http://www.microsoft.com/com/wpaper/default.asp#DCOMPapers，2002，1

[17]Microsoft Corporation. Microsoft MTS White Papers[HTML]. Microsoft Corporation: http://www.microsoft.com/com/wpaper/default.asp#MTSPapers, 2002, 1

[18]John Tyler, Brant Cheikes, Carlos Amaro, etc. IEEE P1484.1 Draft Standard for Learning Technology —Learning Technology Systems Architecture（LTSA）［PDF］. IEEE Learning Technology Standards Committee （LTSC）: http://ltsc.ieee.org/doc/index.html, April 6, 2001

[19]Jack Hyde, Frank Farance, Bill McDonald, etc. IEEE P1484.11 Draft Standard for Computer-Managed Instruction（CMI）[PDF]. IEEE Learning Technology Standards Committee （LTSC）: http://ltsc.ieee.org/doc/index.html, June 5, 1998

[20]Wayne Hodgins, Carlos Amano, Thor Anderson, etc. IEEE P1484.12 Draft Standard for Learning Object Metadata（LOM）[PDF]. IEEE Learning Technology Standards Committee （LTSC）: http://ltsc.ieee.org/doc/index.html, May 3, 2001

[21]张元良，孙世强. 浅谈 Win32 系统钩子机制. 计算机应用. 2001，21：87~88

[22][美] Ted Faison. Borland C++ 3.1 编程指南（第一版）. 蒋维杜，吴志美，张新宇等译. 北京：清华大学出版社，1993

[23][美] James McCord. Microsoft Window 3.1 程序员参考手册（第一版）. 王旭，张军，孙燕等译. 北京：清华大学出版，1993：297~593

[24][美] Jeffrey Richter. Windows 核心编程（第一版）. 王建华等译. 北京：机械工业出版社，2000：300~324

[25]杨虹. 80386 保护模式下的编程. 电脑开发与应用. 2001，14：12~14

[26]李彦昌. 保护模式教程[HTML]. 保护方式下的 80386 及其编程：http://lijun_1.myetang.com/asm/protect1.html，2001，5

[27]夏庆德，朱虹. Windows 95/98 下屏幕抓词的原理和实现方法. 计算机应用，1999，19：63~65

[28][美]Matt Pietrek. Windows 系统编程奥秘（第一版）. 侯俊杰译. 台湾：棋标出版有限公司，1997. 106~594

[29]蔡志平，殷建平，祝恩. 在 Windows 中执行 Ring0 特权级代码的几种方法. 计算机应用，2001，21：97~98

[30]李湘江. Win9X 下的 Vxd 技术与应用. 计算机系统应用，2001，8：21~29

[31]杨传军 王琰. 内存映射文件原理与使用方法. 中国计算机用户，1997，5：45~46

[32]Yariv Kaplan . API Spying Techniques for Windows 9x, NT and 2000[HTML] . Internals Online Resource for System Programmers : http://www.internals.com /articles/apispy /apispy.htm，June 9，2000

[33]马 飞 涛 . 屏 幕 抓 字 教 程 [HTML] . 飞 涛 工 作 室 : http://iflower.myrice.com/pmzhz.htm，2000，3

[34]Aaron Klotz . Direct Thunking[HTML] . Aaron Klotz at Waterloo : http://www.student.math.uwaterloo.ca /~asklotz/thunk.html，July 13，2001

[35]谢志鹏，陈锻生. 用 VC++设计 Win32 全局钩子. 计算机应用，2001，21：85~87

[36]Z. Pawlak. Rough Sets. International Journal of Computer and Information Sciences，1982，11：341~356

[37]Z. Pawlak. Rough Classification. International Journal on Man-Machine Studies，1984，11：469~483

[38]Z. Pawlak，R. Slowinski. Decision Analysis using Rough Sets. International Transactions on Operational Research，1994，1：107~114

[39]B. Walczak，D. Massart. Rough Sets Theory. Chemometrics and Intelligent Laboratory Systems，1999，1：2~16

[40]Chien-Chung Chan. A rough set approach to attribute generalization in data mining. Journal of Information Sciences，1998：169~176

[41]Wang Jue，Miao Duoqian. Analysis on Attribute Reduction Strategies of Rough Set. Journal of Computer Science & Technology，1998，13：189~193

[42]R. Slowinski，J. Stefanowski. Rough Classification in Incomplete Information Systems. Mathematical and Computing Moddelling，1989，10：1347~1357

[43]张文修，吴伟杰，梁吉业等. 粗糙集理论与方法（第一版）. 北京：科学出版社，2001. 1~115

[44]胡可云，陆玉昌，石纯一. 粗糙集理论及其应用进展. 清华大学学报（自然科学版），2001，48：64~68

[45]叶青，杨家本，柴跃廷. 基于粗集理论的知识处理方法在专家系统中的应用. 信息与控制，2001，30：193~198

[46]常犁云，王国胤，吴渝. 一种基于 Rough Set 理论的属性约简及规则提取方法. 软

件学报，1999，10：1206～1211

[47]洪帆．离散数学基础（第二版）．武汉：华中理工大学出版社，1994．309~345

附录 1 攻读硕士学位期间发表论文目录

[1] 郭 飞，周曼丽. Windows API 拦截技术综述. 计算机工程与应用，已录用。