



广东外语外贸大学毕业论文

课题：基于 RTC Client API 的即时通讯软件的设计与实现

Real-time Communication Applications Based on RTC Client API

信息技术学院 计算机科学与技术系 013 班

学 生： 杨 滔

指导教师： 姜 灵 敏

二〇〇五年五月

摘要

一个典型的商业应用程序应该是基于企业部署的实时性服务，它能确保主管与即使是处在出差中的员工保持实时的联系。RTC（Real-time Communications）Client API（Application Programming Interface）使得你能够设计通过互联网进行 PC-PC、PC-phone 或者 phone-phone 的即时消息会话的应用程序。RTC Client API 支持包含视频音频的 PC-PC 的会话请求，还可以在任何类型的会话中加入应用程序共享和白板技术来增强通讯交流能力。

本设计课题是从实际商业应用角度出发，以计算机网络原理为指导，结合当前计算机软件中的一些常用技术，来进行基于 RTC Client API 的即时通讯软件的设计与实现。该即时通讯软件有视频会议模式、应用程序共享模式、白板模式、即时消息模式等功能。

关键字： RTC Client API；COM（组件对象模型）编程；即时通讯软件。

ABSTRACT

A typical business application might be a corporate-deployed presence service that tracks service agents even traveling in the field. And the RTC Client API enables you to build applications that can make PC-PC, PC-phone, or phone-phone calls or create IM (Instant Messaging) sessions over the Internet. Both voice and video calls can be established on PC-PC calls. Application sharing and whiteboard can be added to enhance the communication capabilities of any type of session.

In the opinion of business application and guided by computer network principal and taking some common computer technologies, this academic project is the design and implementation of the instant messaging software based on RTC Client API. Video Conference Mode, Application Share Mode, Whiteboard Mode and Instant Message Mode are all integrated in this project.

KEY WORDS: RTC Client API; COM (Component Object Model) Programming; Real-time Communication Software.

目录

摘要	I
ABSTRACT	II
目录	III
第一章 绪论	1
1.1 项目开发的背景	1
1.2 项目开发的主要内容	2
第二章 基于 RTC CLIENT API 的即时通讯软件的关键技术及开发方案	3
2.1 COM 编程技术	3
2.1.1 COM 简介	3
2.1.2 COM 的使用	4
2.2 WINDOWS 编程技术	5
2.2.1 Windows 的消息传递机制	5
2.2.2 MFC 技术概述	7
2.3 RTC CLIENT API 介绍	10
2.3.1 RTC 对象模型	10
2.3.2 RTC Client 中的 T.120 协议	11
2.3.3 使用 RTC Client API 来进行通信的一般模型	13
2.4 系统方案设计	13
2.4.1 其他可行方案	13
2.4.2 项目方案介绍	14
第 3 章 基于 RTC CLIENT API 的即时通讯软件的设计与实现	16
3.1 系统流程图	16
3.2 数据流	17
3.3 数据结构设计	19
3.3.1 几个重要数据结构的描述	19
3.3.2 各模块中数据结构的设计	20
3.4 界面设计	21
3.5 对象结构的设计	23
3.6 RTC CLIENT API 实例的初始化与释放	25

3.7 使用 RTC CLIENT API 进行通信类型的选择	27
3.8 对于系统多媒体设备的调整设置	28
3.9 RTC EVENTS 的处理	29
3.9.1 RTC 媒体事件的处理	30
3.9.2 音频强度事件的处理	31
3.9.3 会话状态改变事件的处理	31
3.9.4 即时消息事件的处理	32
3.9.5 RTC 客户事件的处理	33
3.10 RTC SESSION 的建立	34
3.11 视频会议会话中的要点分析	35
3.11.1 视频会议会话中 RTC Client 事件的处理	35
3.11.2 视频会议会话中视频音频的播放	37
3.12 即时消息会话中的要点分析	39
3.12.1 即时消息会话中消息发送与接收的处理	39
3.12.2 即时消息会话中参与者状态的处理	40
第四章 总结	41
参考文献	42
致谢	43

第一章 绪论

1.1 项目开发的背景

如今，作为网络应用之一的即时通讯，越来越受到人们的欢迎和重视，它不仅广泛娱乐，在商业协作特别是在跨地区跨国度的商业活动中，即时通讯也越来越显示出其明显的优势，其低廉的成本及其优异的效率，使企业应用程序的开发都趋向于集成即时通讯模块以加强其功能来提升商业绩效。

现在已有的即时通讯软件和聊天工具不胜枚举，流行的主要有 QQ，ICQ，MSN Message 和各种的网页聊天室等等。这些软件特别是 QQ 升级很快，其功能是越来越强大，将代表着聊天软件的发展方向。然而这些软件的一个共同的缺点是采用了 C/S 的结构模式，如果服务器出现故障，则这个聊天系统将瘫痪。下面简要的对现在流行的聊天软件进行分析。

作为现在最流行的聊天软件之一 QQ，功能已经非常强大，基本上朝着三个大的方向发展，即个人及时通信，企业实时通信和娱乐资讯。它的功能主要有：

- ◇ 文本聊天
- ◇ 视频聊天
- ◇ 图像传输
- ◇ 文件传输
- ◇ 在线游戏
- ◇ 手机通讯
- ◇ 聊天室聊天
- ◇ 登录状态
- ◇ 群聊天

可以看到，现在流行的聊天软件都是向着多维化，人性化，综合化发展的。

网页聊天室作为另一种聊天软件形式，也是最早流行的基于 WEB 的即时通讯模式，网页聊天室也受到用户的普遍欢迎。网页聊天室的优点就是很好的利用了浏览器的功能，用户不需要安装任何新的软件，即可

加入聊天室聊天。但是由于其客户端的简化，其功能必然受到限制。

当前所有流行的即时通讯软件几乎都是基于 C/S 模式的。C/S 模式是一种两层结构的系统：第一层是在客户机系统上结合了表示与业务逻辑；第二层是通过网络结合了数据库服务器。

交互性强是 C/S 模式的一个最显著的优点，但是随着网络规模的日益扩大，应用程序的复杂程度不断提高，逐渐也暴露了一些缺点：开发成本较高；移植困难；用户界面风格不一；使用繁杂，不利于推广；维护复杂，升级麻烦；客户端与后台数据库服务器数据交换频繁，而且数据量大，当大量用户访问时，易造成网络瓶颈；新技术不能轻易应用，因为一个软件平台及开发工具一旦选定，不可能轻易更改。在 Web 和 Intranet 技术还没有走进市场的前几年里，C/S 技术也曾经帮助了世界各地的公司和企业提高了工作效率，然而这些局限性在今天的网络技术环境里，最终将面临像主机终端式网络同样的命运。

针对基于 C/S 模式即时通讯软件的不足，Microsoft 于 2001 年推出其最新版的 Windows XP 时，在其中集成 Windows Messenger，它提供了所有的实时通信功能。为了使得应用程序开发人员能够获得与在 Windows Messenger 中实现同样的 RTC 客户端功能，Windows XP 通过 RTC (Real-time Communications) Client API (Application Programming Interface) 开放了 RTC 通讯功能。这些 API 使应用程序能够创建从 PC-PC, PC-phone 或者 phone-phone 的会话请求。应用程序能够增添在 Internet 或者 Intranet 上创建 RTC Client 会话的功能，增强其协作功能。在 PC 机用户之间，语音和视频呼叫都可以建立。应用程序能获得并显示一系列联系人的现场信息。通过增加应用程序和白板的共享的功能，可以加强双方的协作能力。

1.2 项目开发的主要内容

为了追求服务器的灵活性，采用 RTC Client API 技术开发即时通讯软件，该软件的功能有：视频会议、即时消息、应用程序共享、白板及视频音频调节。项目的目标系统将服务端与客户端合为一体，即在同一应用程序内实现服务端和客户端的功能，因此我们也可以将其称为基于点对点的即时通讯应用。

涉及的技术包括 COM (Component Object Model) 编程、Windows GUI (Comfortable Graphical Interface) 编程、MFC (Microsoft Foundation Classes) 技术及 RTC Client API 的使用等等。

第二章 基于 RTC Client API 的即时通讯软件的关键技术及开发方案

关键技术及开发方案

2.1 COM编程技术

RTC Client 的各种功能都是通过 COM 来实现的，要在程序中使用 Windows XP 所提供的 RTC Client API 就必须通过 COM 来进行调用。下面介绍 COM 的一些重要概念及其使用方法。

2.1.1 COM简介

COM 是一种说明如何建立可动态互变组件的规范，此规范提供了为保证能够互操作，客户和组件应遵循的一些二进制和网络标准。通过这种标准可以在任意两个组件之间进行通信而不用考虑其所处的操作环境是否相同、使用的开发语言是否一致以及是否运行于同一台计算机。COM 具有语言无关性，它可以用任何语言编写，也可以在任何语言平台上被调用。

在 COM 规范下将能够以高度灵活的编程手段来开发、维护应用程序。可以将一个单独的复杂程序划分为多个独立的模块进行开发，它的每一个独立模块都是一个自给自足的组件，可以采取不同的开发语言去设计每一个组件。在运行时将这些组件通过接口组装起来以形成所需的应用程序。构成应用程序的每一个组件都可以在不影响其他组件的前提下被升级。这里所说的组件是指在二进制级别上进行集成和重用而能够被独立生产获得和配置的软件单元。COM 规范所描述的即是如何编写组件，遵循 COM 标准的任何一个组件都是可以被用来组合成应用程序的。至于对组件采取的是何种编程语言则是无关紧要的，可以自由选取。作为一个真正意义上的 COM 组件，应具备如下特征：

- (1) 实现了对开发语言的封装。
- (2) 以二进制形式发布。
- (3) 能够在不妨碍已有用户的情况下被升级。
- (4) 在网络上的位置必须能够被透明的重新分配。

COM 的结构模型见图 2.1。

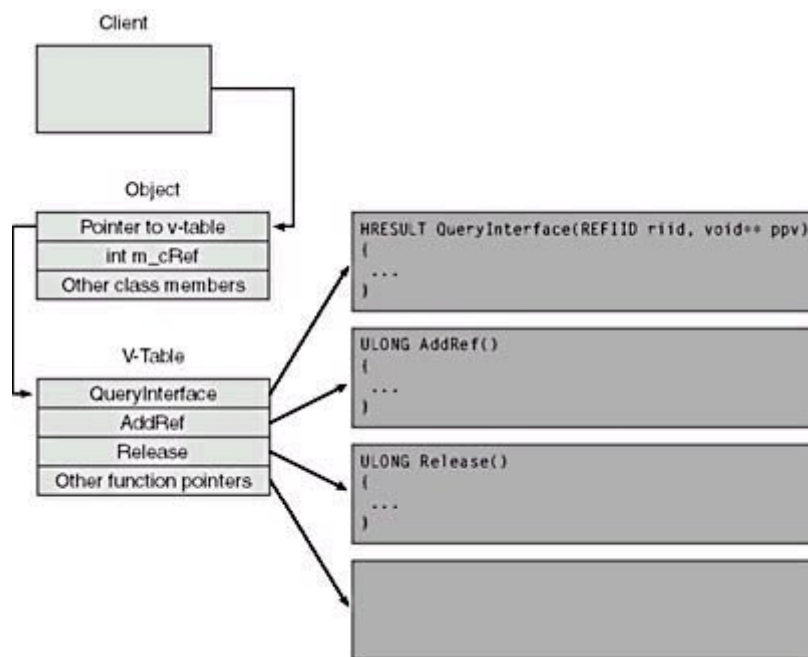


图 2.1 COM 结构的示例图

每一个 COM 必须实现一个名为 IUnknown 的接口。IUnknown 接口只包含三个方法，见表 2-1。

表 2-1 IUnkonw 接口的三个方法

方法名称	描述
QueryInterface	返回一个指向其它接口的指针
AddRef	类引用计数器加 1
Release	类引用计数器减 1

客户程序在与 COM 组件进行交互时，只需知道与哪个 COM 对象进行交互即可而不必关心组件模块的具体名称和位置，即 COM 对象的位置对客户是透明的。客户将通过一个 128 位的全局标识符（globally unique identifier, GUID）完成对象的创建和初始化工作。对于 COM 对象，此全局标识符也被称作 CLSID（class identifier, 类标识符）。

2.1.2 COM 的使用

使用 COM 的一般过程（假设 COM 对象没有实现 IClassFactory 接口）：

- （1）获取目标 COM 的 GUID。
- （2）使用 CoInitialize 函数初始化 COM 库。

(3) 使用 `CoCreateInstance` 函数根据 `GUID` 生成一个特定的 `COM` 实例。生成的 `COM` 实例不必显式调用本身的 `AddRef` 来声明引用。

(4) 调用 `COM` 实例的 `Initialize` 方法对其进行初始化。

(5) 使用 `COM` 实例的 `QueryInterface` 方法去获取当前 `COM` 实例支持的接口。

(6) 对于任何使用过的 `COM` 实例必须调用其本身的 `Release` 方法来释放引用。

2.2 Windows编程技术

在本项目中，`Windows` 编程中的消息传递机制及 `MFC` 技术显得十分重要，整个程序的构建及 `RTC Client` 实例的生成及处理都要使用到这两种重要的技术。

2.2.1 Windows 的消息传递机制

消息是指 `Windows` 发出的一个通知，告诉应用程序某个事情的发生。例如，单击鼠标、改变窗口尺寸、按下键盘上的一个键都会使 `Windows` 发送一个消息给应用程序。消息本身是作为一个记录传递给应用程序的，这个记录中包含了消息的类型以及对此消息的其他描述信息。消息机制则是指 `Windows` 系统将会维护一个或多个消息队列，所有产生的消息都回被放入或是插入队列中。系统会在队列中取出每一条消息，根据消息的接收句柄而将该消息发送给拥有该窗口的程序的消息循环。每一个运行的程序都有自己的消息循环，在循环中得到属于自己的消息并根据接收窗口的句柄调用相应的窗口过程。而在没有消息时消息循环就将控制权交给系统，所以 `Windows` 可以同时进行多个任务。

常用的与消息有关的三个参数见表 2-2：

表 2-2 与 `Windows` 消息有关的三个参数

参数	描述
<code>Message</code>	<code>UINT</code> 。用于区别其他消息的常量值，这些常量可以是 <code>Windows</code> 单元中预定义的常量，也可以是自定义的常量。
<code>wParam</code>	通常是一个与消息有关的常量值，也可能是窗口或控件的句柄。
<code>lParam</code>	通常是一个指向内存中数据的指针。

一般来说，Windows 应用程序都有一个类型为 callback 的窗口处理函数。这种类型的函数由于应用程序开发人员编写，提供给 Windows 调用的对特定的 Windows 消息的处理进行处理。在程序运行过程中，消息由输入装置经由消息循环的抓取，源源不断地传送给窗口并进而送到窗口函数中进行处理。下面给出 Windows 程序的一般模型，见图 2.2。

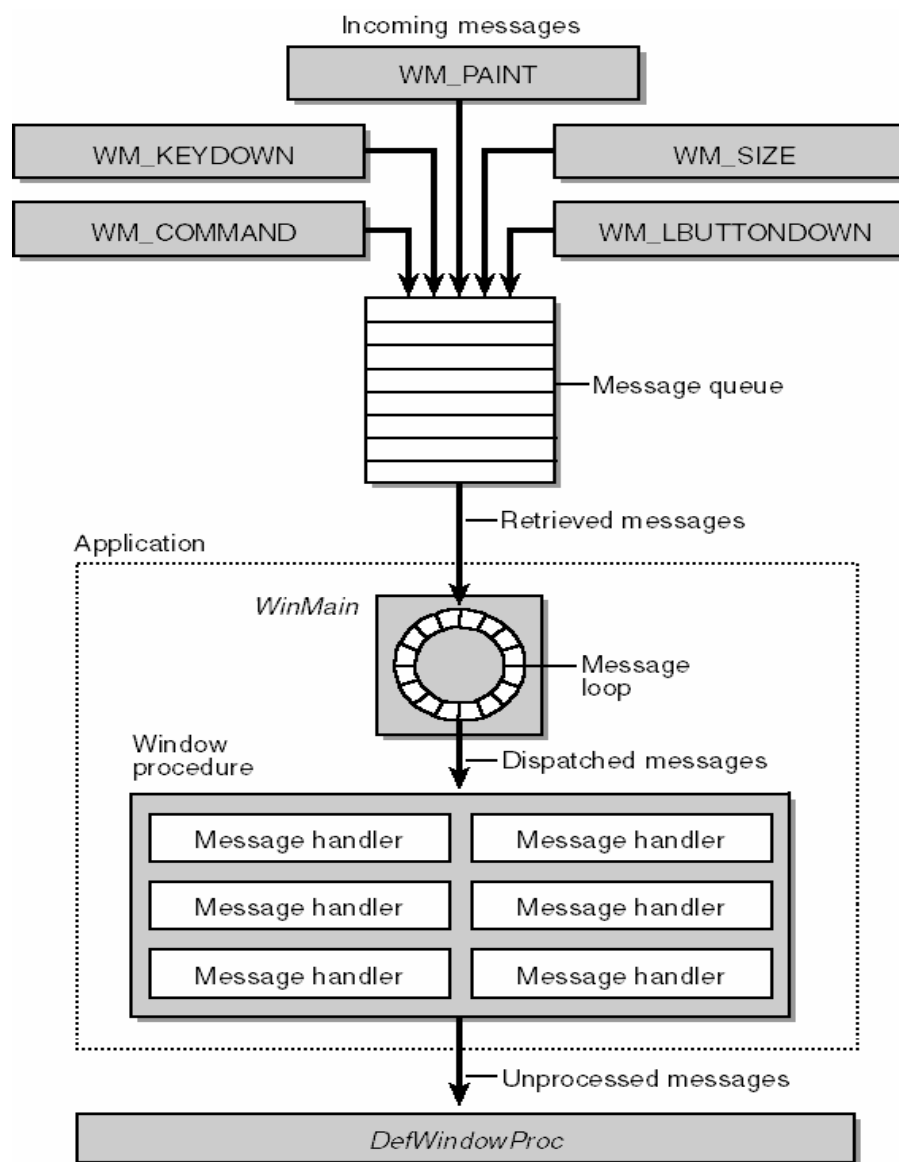


图 2.2 Windows 编程模型

下面给出 Windows 程序的一般结构：

```
int PASCAL WinMain( HINSTANCE hInstance,HINSTANCE hPrevInstance,
                  LPSTR lpszCmdLine,int nCmdShow )
{
    ....
    if ( !InitApplication(hInstance) ) return (FALSE);
```

```
if ( ! InitInstance(hInstance, nCmdShow) ) return (FALSE);
while ( GetMessage( &msg,NULL,0,0) )
{
    TranslateMessage( &msg );
    DispatchMessage( &msg );
}
return msg.wParam;
}

LRESULT CALLBACK MainWndProc( HWND hwndMain, UINT message,
                              WPARAM wParam, LPARAM lParam )
{
    ... ..
    Switch ( message )
    {
        ... ..
    default:
        return( DefWindowProc( hwndMain, message, wParam, lParam ));
    }
    return( 0L );
}
```

2.2.2 MFC 技术概述

MFC (Microsoft Foundation Class Library) 中的各种类结合起来构成一个应用程序框架，它的目的就是让程序员在此基础上建立 Windows 下的应用程序，这是一种相对 SDK 来说更为简单的方法。因为总体上，MFC 框架定义了应用程序的轮廓，并提供了用户接口的标准实现方法，程序员所要做的就是通过预定义的接口把具体应用程序特有的东西填入这个轮廓。Microsoft Visual C++ 提供了相应的工具来完成这个工作：AppWizard 用来生成初步的框架文件（代码和资源等）；资源编辑器用于帮助直观地设计用户接口；ClassWizard 用来协助添加代码到框架文件；程序开发最后的编译，则通过类库实现来应用程序特定的逻辑。

2.2.2.1 MFC 的封装

构成 MFC 框架的是 MFC 类库。MFC 类库是 C++ 类库。这些类或者封装了 Win32 应用程序编程接口，或者封装了应用程序的概念，或者封装了 OLE 特性，或者封装了 ODBC 和 DAO 数据访问的功能，等等。

(1) 对 Win32 应用程序编程接口的封装

用一个 C++ Object 来包装一个 Windows Object。例如：class CWnd 是一个 C++ window object，它把 window 有关的 API 函数封装在 C++ window object 的成员函数内，后者的成员变量 m_hWnd 就是前者的窗口句柄。

(2) 对应用程序概念的封装

使用 SDK 编写 Windows 应用程序时，总要定义窗口过程，登记 Windows Class，创建窗口，等等。MFC 把许多类似的处理封装起来，替程序员完成这些工作。另外，MFC 提出了以文档-视图为中心的编程模式，MFC 类库封装了对它的支持。文档是用户操作的数据对象，视图是数据操作的窗口，用户通过它处理、查看数据。

(3) 对 COM/OLE 特性的封装

OLE 建立在 COM 模型之上，由于支持 OLE 的应用程序必须实现一系列的接口(Interface)，因而相当繁琐。MFC 的 OLE 类封装了 OLE API 大量的复杂工作，这些类提供了实现 OLE 的更高级接口。

(4) 对 ODBC 功能的封装

以少量的能提供与 ODBC 之间更高级接口的 C++ 类，封装了 ODBC API 的大量的复杂的工作，提供了一种数据库编程模式。

2.2.2.2 MFC 的继承

首先，MFC 抽象出众多类的共同特性，设计出一些基类作为实现其他类的基础。这些类中，最重要的类是 CObject 和 CCmdTarget。CObject 是 MFC 的根类，绝大多数 MFC 类是其派生的，包括 CCmdTarget。CObject 实现了一些重要的特性，包括动态类信息、动态创建、对象序列化、对程序调试的支持，等等。所有从 CObject 派生的类都将具备或者可以具备 CObject 所拥有的特性。CCmdTarget 通过封装一些属性和方法，提供了消息处理的架构。MFC 中，任何可以处理消息的类都从 CCmdTarget 派生。

针对每种不同的对象，MFC 都设计了一组类对这些对象进行封装，每一组类都有一个基类，从基类派生出众多更具体的类。这些对象包括以下种类：窗口对象，基类是 CWnd；应用程序对象，基类是 CWinThread；文档对象，基类是 CDocument，等等。

应用程序开发人员可结合自己的实际，从适当的 MFC 类中派生出自己的类，实现特定的功能，达到自己的编程目的。

2.2.2.3 MFC 的虚拟函数和动态约束

MFC 以“C++”为基础，自然支持虚拟函数和动态约束。但是作为一个编程框架，有一个问题必须解决：如果仅仅通过虚拟函数来支持动态约束，必然导致虚拟函数表过于臃肿，消耗内存，效率低下。例如，CWnd 封装 Windows 窗口对象时，每一条 Windows 消息对应一个成员函数，这些成员函数为派生类所继承。如果这些函数都设计成虚拟函数，由于数量太多，实现起来不现实。于是，MFC 建立了消息映射机制，以一种富有效率、便于使用的手段解决消息处理函数的动态约束问题。

这样，通过虚拟函数和消息映射，MFC 类提供了丰富的编程接口。程序员继承基类的同时，把自己实现的虚拟函数和消息处理函数嵌入 MFC 的编程框架。MFC 编程框架将在适当的时候、适当的地方来调用程序的代码。

2.2.2.4 MFC 的宏观框架体系

MFC 实现了对应用程序概念的封装，把类、类的继承、动态约束、类的关系和相互作用等封装起来。这样封装的结果对程序员来说，是一套开发模板。针对不同的应用和目的，开发人员采用不同的模板。例如，SDI 应用程序的模板，MDI 应用程序的模板，规则 DLL 应用程序的模板，扩展 DLL 应用程序的模板，OLE/ACTIVEX 应用程序的模板，等等。

为了支持对应用程序概念的封装，MFC 内部必须作大量的工作。例如，为了实现消息映射机制，MFC 编程框架必须要保证首先得到消息，然后按既定的方法进行处理。又如，为了实现对 DLL 编程的支持和多线程编程的支持，MFC 内部使用了特别的处理方法，使用模块状态、线程状态等来管理一些重要信息。

总之，MFC 封装了 Win32 API，OLE API，ODBC API 等底层函数的功能，并提供更高层的接口，简化了 Windows 编程。同时，MFC 支持对

底层 API 的直接调用。

MFC 提供了一个 Windows 应用程序开发模式，对程序的控制主要是由 MFC 框架完成的，而且 MFC 也完成了大部分的功能，预定义或实现了许多事件和消息处理，等等。框架或者由其本身处理事件，或者调用程序员的代码来处理应用程序特定的事件，并不依赖程序员的代码。

MFC 是 C++ 类库，程序员是通过使用、继承和扩展适当的类来实现特定的目的。

2.3 RTC Client API 介绍

RTC Client API 是 Windows XP 的一个新增 API。该 API 旨在帮助快速开发实时通讯应用程序（如即时消息和视频电话会议软件）。根据 Microsoft 对 Windows XP 的介绍，丰富的通信特性已经被整合并得到增强以便在基础结构中提供为实时通信（RTC）软件提供支持。这些特性被 Microsoft Windows Messenger 用来显示实时语音和视频、即时消息及其他协同信息。更重要的是，RTC Client API 使得任何应用程序都能够轻松地使用其丰富的通信基础结构。

2.3.1 RTC 对象模型

RTC 对象模型见图 2.3。

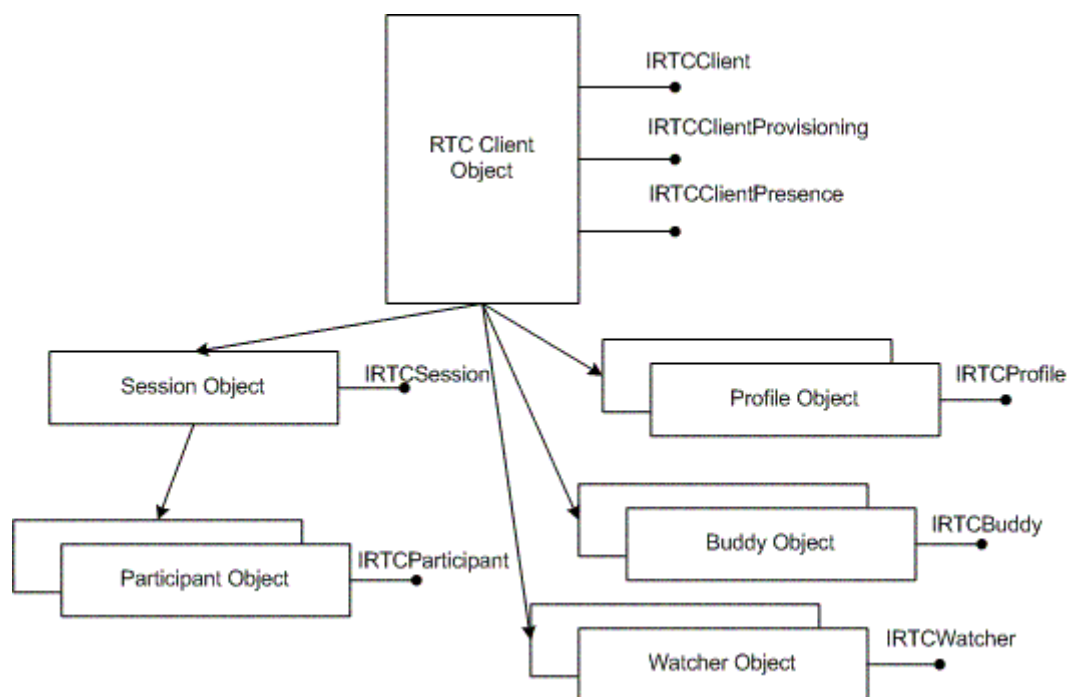


图 2.3 RTC 对象模型

RTC 的基本编码模型是 COM。在 RTC 中用于通信的对象分别有 Client 对象、Session 对象和 Participant 对象。

Client 对象：Client 对象实现了 IRTCClient 接口，并且为会话建立了允许的类型和参数，比如首选设备、媒体类型和比特率以及其它的媒体属性，比如音量和回波抵消。这个接口也被用于创建会话对象。

Session 对象：Session 对象通过 IRTCClient::CreateSession 方法创建，被用于管理一个会话。这个对象实现了 IRTCSession 接口，这一接口用于初始化、回复或者终止一个会议，增加参加者和其它的会议管理。这个对象支持 IM 和其它的会话类型。

Participant 对象：Participant 对象由 IRTCSession::AddParticipant 方法创建，包含了与会议参加者有关的所有方法。它包括了参加者的姓名和当前状态。IRTCParticipant 接口由该对象实现。

出席信息由 Buddy 和 Watcher 对象管理。这些对象提供接口，用于管理获取与联系人和出席成员有关的信息。

2.3.2 RTC Client 中的 T.120 协议

RTC Client 中的视频会议是使用 ITU（International Telecommunication Union）T.120 标准来实现的，它是由一组通信和应用程序协议组成。T.120 协议被设计来用于实现多点数据会议和实时通信，包括多层协议，该协议很大程度上提高了多媒体、和多媒体数字信号编解码器的控制能力。取决于 T.120 实现类型，最终的产品能够完成连接、传输、接收数据和合作过程，这是通过使用相兼容数据会议特征来实现的，如共享程序、白板功能和文件传输等等。

T.120 的主要功能包括，在不依靠任何平台的前提下，创建和维持会议；管理多重参与者和多重程序；通过网络连接，精确而安全地发送和接收数据。

2.3.2.1 T.120 协议组

T.120 协议组包括以下协议：

(1) T.121 为 T.120 资源管理提供了一个模板，用来指导开发者建立应用程序协议。T.121 是一种强制性的标准化应用程序协议，被高度推荐服务于非标准应用程序协议。此模板确保了一致性并降低了在不同协议的实现过程中的不可预见的交互作用的可能性。

(2) T.122 定义了开发者可使用的多点服务。结合 T.125, T.120 形成了 MCS (Multipoint Communication Services) —它是 T.120 会议的多点引擎。MCS 依靠 T.123 发送数据, 是用于解决任何多点应用程序设计需求的有力工具。此外 MCS 是对复杂组织的精辟梗概, 学习如何有效使用 MCS 是成功发展实时应用程序的关键。

(3) T.123 为以下各部分提供了传输配置文件:

公用电话交换网络 PSTN;
综合交换数字网络 ISDN;
电路交换数字网络 CSDN;
包交换数字网络 PSDN;
Novell Netware IPX;
TCP/IP。

应用软件期望下层传输能够为其协议数据单元提供可靠传送并分割和排序数据。

(4) T.124 详细阐述了通用会议控制 (Generic Conference Control , GCC)。GCC 为建立和管理多点会议提供了全面的设备设置。在 GCC 下, 我们可以首先看到特定电子会议的有关特征。

(5) T.125 描述了多点通信服务协议 MCS, 它定义了: 单一协议的数据传输和从一个 MCS 供应商到另一个对等 MCS 供应商的控制信息的相关程序; MCS 协议数据单元的结构和编码方式, 其主要用于数据传输和控制信息。

(6) T.126 定义了一种用于在两个或更多应用程序间观察并注释传输图像的协议。此能力经常在文档会议或共享白色书写板中得到表现。

(7) T.127 为应用程序在会议多终点间传输文件提供了一种方法。文件可以传送给会议中所有的参与者或者传送给特定的会议子网。多文件传送操作可以同时发生在任意指定的会议中, 并且开发者为文件传送定义了优先权。最后 T.127 还为数据传送前的文件压缩提供了可选项。

2.3.2.2 T.120 的协议结构

T.120 体系结构中对层与层之间定义的协议和服务采用多分层方法。每层都假设其下面的所有层都存在。较低的层 (T.122、T.123、T.124 和 T.125) 规定一个独立应用程序机制, 为可以使用这些设施的任何应用程序提供多点数据通信服务。较高的层 (T.126 和 T.127) 规定了支持特定应用程序的协议, 如共享白色写字板和二进制文件传输。在同一个会议

中，这些“标准化应用程序”与“非标准化应用程序”可以共存。

2.3.3 使用 RTC Client API 来进行通信的一般模型

(1) 增强的客户端应用决定客户端通信平台的能力。

(2) 应用程序在通信期间使用首选的视音频设备。

(3) 应用程序发起通信会话。

(4) 在 RTC 层协调数据捕获、压缩和传输，这使得应用程序不用负责这一任务。使用哪一种音视频的编码解码器由通信双方的连接质量决定。

(5) 参与会话的应用程序接受、解压并重放被传输的数据。

通过使用 RTC Client API，在 Windows XP 或者 Windows 2000 下开发通信工具已变得相当简单。现有的即时通讯应用程序可通过添加 RTC 丰富的通信特征而获益。使用 RTC API 进行开发的程序也可以从一个统一的通信协议中获益，提高与其它即时通讯应用程序互相合作的能力。

2.4 系统方案设计

2.4.1 其他可行方案

(1) 基于 Winsock 的即时通讯软件

该方案采用 Winsock 作为底层网络通信技术来实现广播消息，即时消息和文件传输。该方案要求在 TCP/IP 协议的基础上运行。由于 Winsock 对广播传输，建立连接传输，数据报传输的完善支持，该方案是可行的。

(2) 基于 DirectX 语音技术的即时通讯软件

该方案采用 DirectSound 和 DirectShow 技术来实现语音视频的采集、回放。DirectSound 语音技术和 DirectShow 视频技术作为 DirectX 的重要组件，已经非常的成熟，可以在 API 层面上进行视频音频的采集和处理，再将其进行压缩传输。因此该方案也是可行的。

(3) 基于网页形式的即时通讯软件

该方案利用现在成熟的网页技术来实现即时通讯应用的开发，不再需要对底层网络通信进行开发。因此工作重心转到语音聊天的开发和服

务器端的开发。现在基于 WEB 的开发非常成熟，因此该方案是可行的。

(4) 基于 NETBIOS 的即时通讯软件

虽然由于 NETBIOS 不能跨网段传输，但是它对于跨平台、跨协议有很好的支持，而且以网络名字为识别地址，对程序编写要求不高，同样能支持即时通讯软件的各种功能，该方案也是可行的。

2.4.2 项目方案介绍

本项目统以 RTC Client API 的基础，简化编程难度，设计一个即具有 C/S 模式的优点，又能避免 C/S 模式固有的缺陷的即时通讯软件，使其具有强大的交互性的同时，实现服务端的灵活创建，能随意建立任意两点间的通讯。同时由于 RTC Client API 与 Windows 系统的整合度非常高，因此本项目能很容易的作为一个子模块嵌入到大型商业软件中。

(1) 开发环境

项目的开发工具为 Microsoft Visual Studio 6.0，使用 MFC 作为应用程序的框架，并安装了 Microsoft Windows Real-Time Communications Software Development Kit (SDK) version 1.3 来进行协助开发。网络通讯采用 Microsoft 的 RTC Client API 来实现，RTC Client API 同时提供对多媒体应用例如视频音频调节、多媒体数据的传输以及即时消息的发送和接收的支持。

(2) 功能简述

本项目开发的通讯软件有视频会议、应用程序共享、白板、即时消息等功能，还能启动调节向导对多媒体设备的参数进行设置。

视频会议：

视频会议建立后，可以显示对方视频以及预览本地视频，同时可以对扬声器和麦克风的音量进行调整。建立了视频会话的连接后，可以使用应用程序共享和白板的功能。

即时消息：

只限于文本的即时交流方式，连接建立后，可以得到对方关于此次会话的状态。

(3) 系统环境要求

软件环境：

Microsoft® Windows Server™ 2003 系统或者 Windows® XP 系统
(RTC Client API version 1.0)

网络环境：

安装有网卡并接入局域网

多媒体环境：

安装有声卡、扬声器、麦克风、摄像头等多媒体设备

第 3 章 基于 RTC Client API 的即时通讯软件的设计与实现

3.1 系统流程图

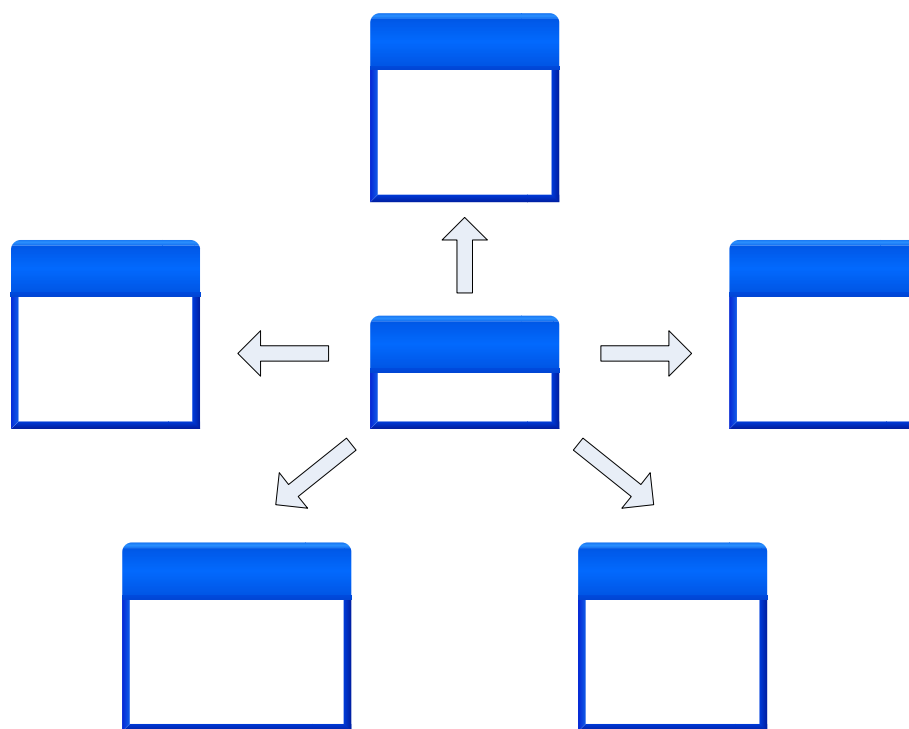


图 3.1 系统流程简图

在程序的主界面中，可以进行视频会议的请求、即时消息交流方式的发起以及运行多媒体设备的设置向导。

(1) 系统采用对等式结构，即任意节点均可作为服务器，又可作为客户端。同一网络上可运行多个服务器，任意两个节点都可以建立连接进行通讯。

(2) 当系统处于程序主界面时，系统已经处于对通讯请求的监听状态，可以接受来自任意客户的会话请求。

(3) 系统初始化后，也可以作为客户端向目标服务器发起会话请求。会话请求的发起需要知道目标服务器的 IP 地址才能建立有效连接。

(4) 系统作为客户端向目标服务器发起视频会议的会话请求时，目标服务器会提示收到请求，并提示用户是否接受当前的会话请求。如果用户接收此次会话请求，则服务器将与客户端连接，否则服务器会向客

户返回拒绝会话请求的信号。见图 3.2。

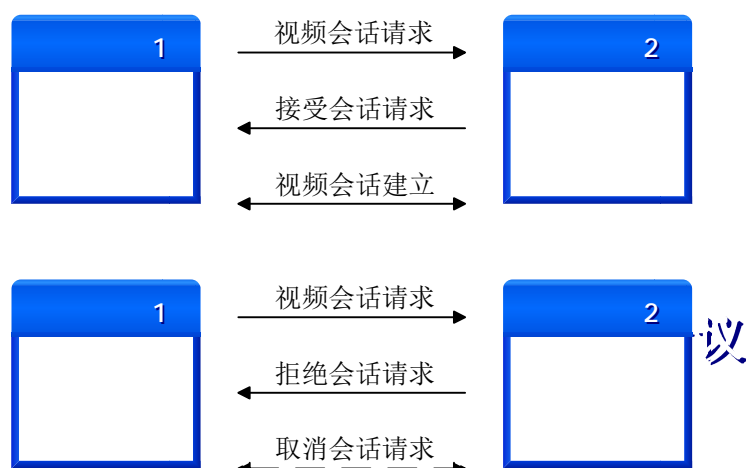


图 3.2 视频会议会话请求示意图

(5) 当系统处于视频会议的会话中时，可以选择进行应用程序共享或者白板或者两者兼用的交流协作方式。

(6) 系统作为客户端向目标服务器发起即时消息的会话请求时，目标服务器并不会提示收到请求，只有在服务器也同时向客户发起同样的连接后，才能正常进行通讯，这一方式类似于网络协议 UDP 的传输方式。见图 3.3。

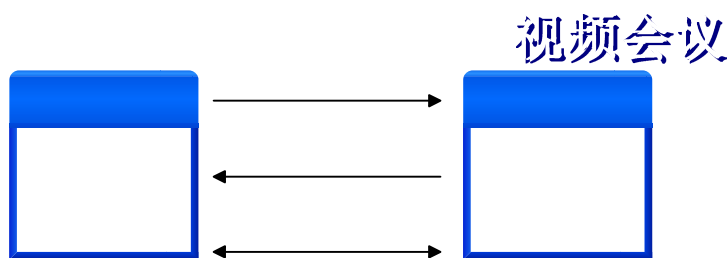


图 3.3 即时消息会话请求示意图

3.2 数据流

(1) 全局数据流程图

下图描绘了全局的数据流程，其中涉及到用户、计算机、输入输出设备和传输媒体这四个实体。

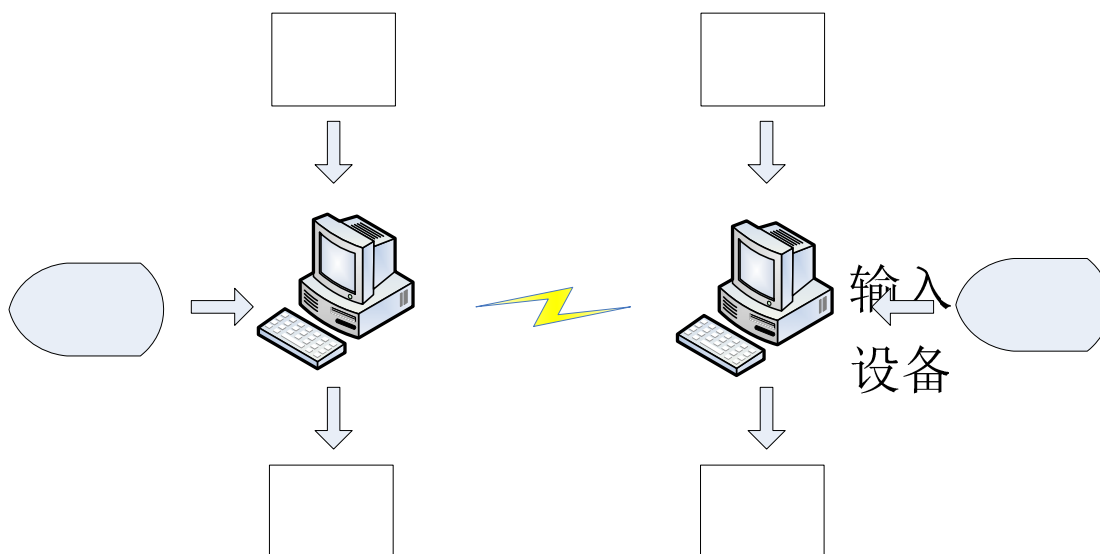


图 3.3 全局数据流程图

(2) 各模块数据流程图

下图描绘了视频会议模块的数据流程，其中涉及到用户、计算机、输入输出设备和传输媒体之间的数据流动。

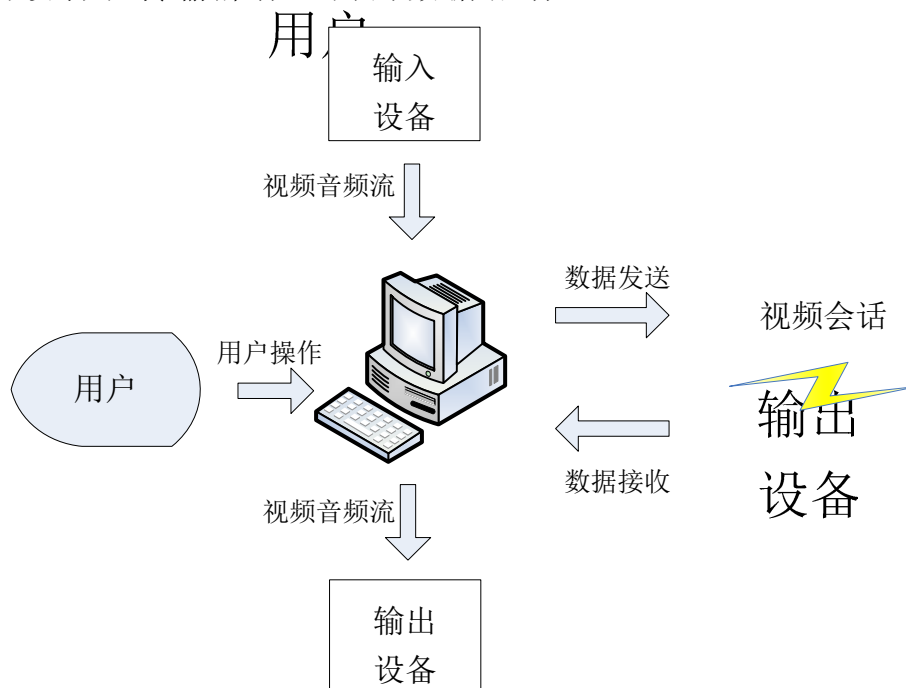


图 3.5 视频会议模块的数据流程图

下图描绘了即时消息模块的数据流程，其中涉及到用户、计算机、输入输出设备和传输媒体之间的数据流动。

通信

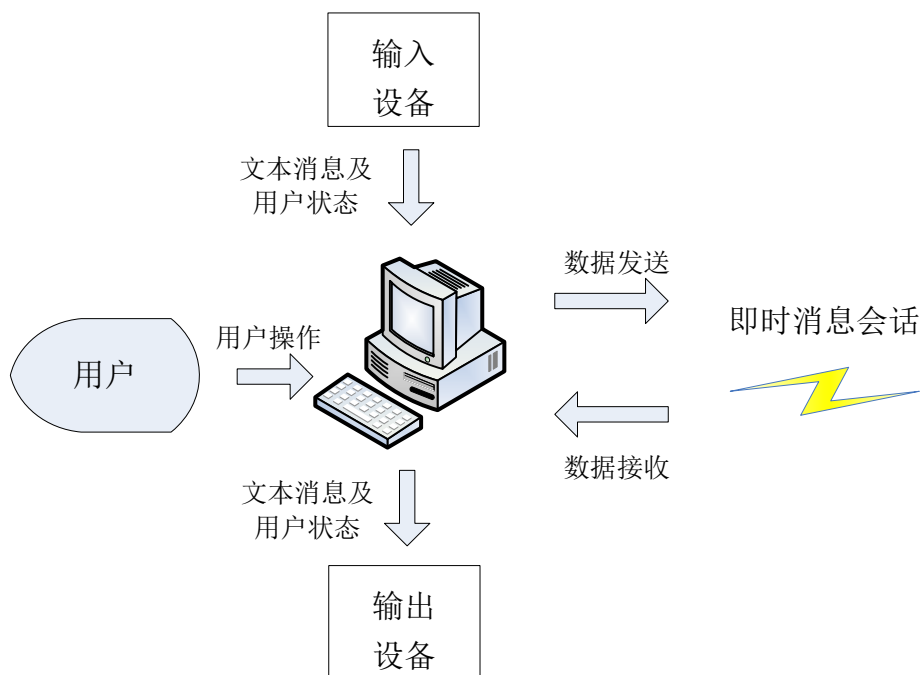


图 3.6 即时消息模块的数据流程图

3.3 数据结构设计

3.3.1 几个重要数据结构的描述

在进行各个模块的对像结构进行设计之前，先对一些全局的或者在各个模块在传送使用的公共变量进行定义与描述。

名称：目标 IP 地址
 别名：g_bstrAddr
 描述：用于建立通讯会话所必须使用的目标 IP 地址

名称：RTC Client 实例
 别名：m_pClient, m_pParentClient
 描述：RTC Client API 的主要接口，用于设置会话状态和会话的创建
 定义：程序初始化过程中使用 COM 调用生成

名称： RTC Events 实例
 别名： m_pEvents
 描述： 用来表示 RTC 通讯中发生各种事件
 定义： 根据 RTC Events 实例和当前窗口句柄生

名称： RTC Events 的消息代号
 别名： WM_RTC_EVENT
 描述： 在 Windows 消息队列中用来表示 RTC 事件

名称： RTC Session 实例
 别名： m_pSession, m_pParentSession
 描述： 使用 RTC Client API 进行通讯的主要手段

名称： RTC 会话状态
 别名： m_rtcState
 描述： 用来表示 RTC Client API 的会话请求的状态

3.3.2 各模块中数据结构的设计

(1) 主对话框 (CAVDConfDlg)

这个对象很重要，几乎所有功能都是由此对像框进行实现，是程序的核心对象。该对像的属性有：

表 3-1 CAVDConfDlg 的属性

IRTCClient * m_pClient	指向 RTC Client 实例的指针
CRTCEvents * m_pEvents	指向 RTC Events 实例的指针

IRTCParticipant * m_Participant	指向 RTC Participant 实例的指针
CAVDlg m_AVDlg	实现视频会议功能的实例
CMessageDlg m_cMessageDlg	实现即时消息的实例

其中 CAVDlg 和 CMessageDlg 又有自己的结构，见以下的定义。

(2) 视频会议模块 (CAVDlg)

这个对象实现了视频会议的功能，它被嵌套到主对话框对象使用。该对象的属性有：

表 3-2 CAVDlg 的属性

IRTCClient * m_pParentClient	指向主对话框对象中的 RTC Client 实例
IRTCSession * m_pSession	RTC 通讯创建的会话指针
RTC_SESSION_STATE m_rtcState	RTC 会话状态

(3) 即时消息模块 (CMessageDlg)

这个对象实现了即时消息通讯的功能，它与 CAVDlg 一样，都是被嵌套到主对话框对象中使用。该对象的属性有：

表 3-2 CMessageDlg 的属性

IRTCClient * m_pParentClient	指向主对话框对象中的 RTC Client 实例
IRTCSession * m_pParentSession	RTC 通讯创建的会话指针
BOOL fIsTyping	会话中用户的操作状态

3.4 界面设计

(1) 系统主界面：



图 3.7 系统主界面

系统的所有功能都可以在此界面执行。在进行会话连接前，可以运行调节向导来对多媒体设备进行设置。在视频会议正常建立之后，就可以使用程序共享及白板来进行协助交流。文本交流方式是只使用文本消息进行交流的方式。

(2) 视频会议界面：



图 3.8 视频会议主界面

在视频会议会话中，本地预览视频以画中画的形式显示在对方的视频画面中。在会话同时还可以拉动扬声器和麦克风下面的滑动条对音量大小进行调整，或者按下静音按钮使得指定设备处于静音状态。按下断开按钮结束此次视频会话。

(3) 会话请求界面：

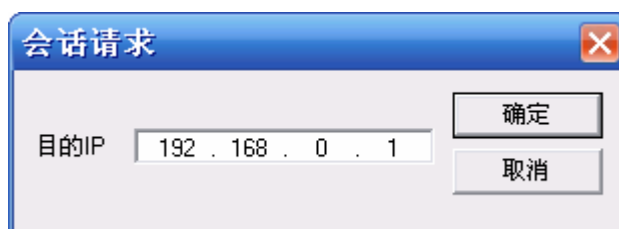


图 3.9 会话请求界面

会话建立前用来接受用户输入目的 IP 地址。

(4) 处理收到会话请求的界面：

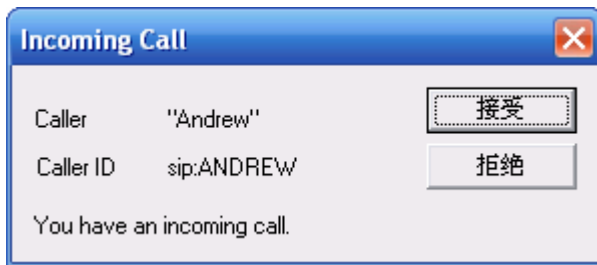


图 3.10 收到会话请求的界面

接收到会话请求时，提示用户的界面，用户可以选择接受或者拒绝此次会话请求。

(5) 即时消息界面：

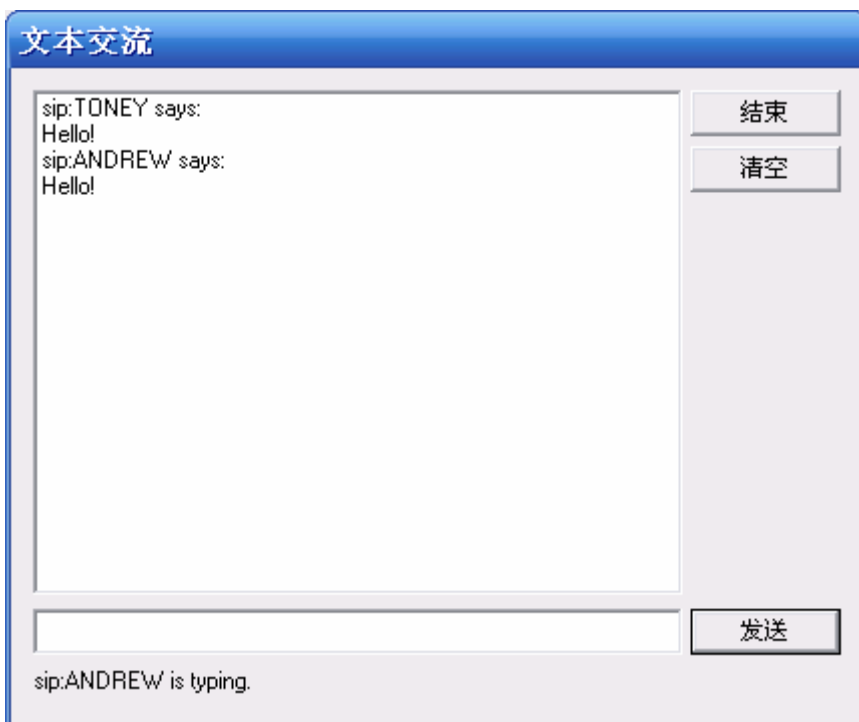


图 3.11 即时消息界面

在进行即时消息交流时，按下清空按钮可以对当前显示的消息进行清空处理，在发送按钮的左边的编程框里输入想要发送的信息，按下发送按钮就会把消息发送出去，同时把消息在本地窗口进行显示。界面的左下角的静态文本可以显示对方用户的状态。

3.5 对象结构的设计

(1) 系统的核心对话框对象 CAVDConfDlg 的结构

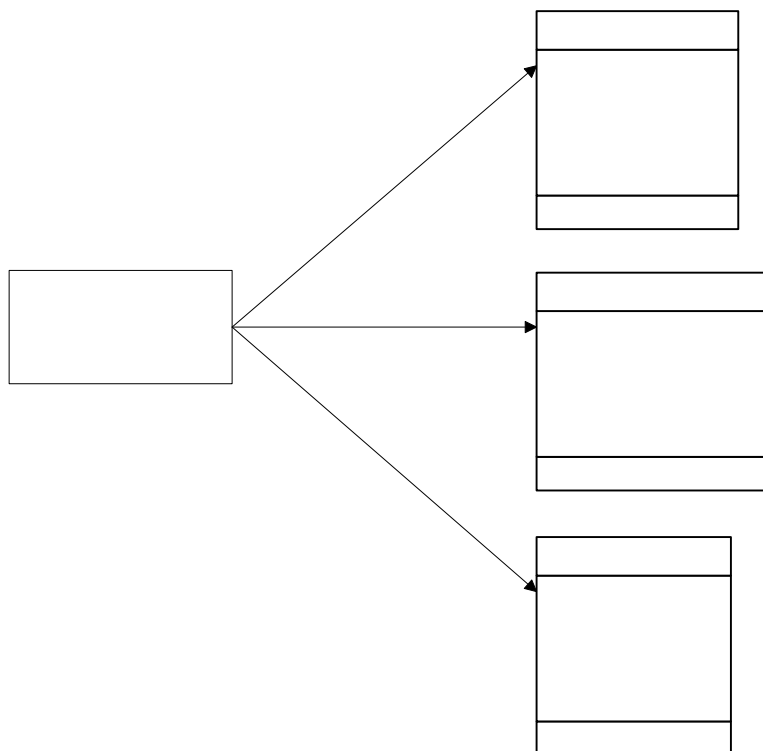


图 3.12 主对话框的结构图

系统的核心对话框对象 **CAVDConfDlg** 包含有一些公共属性，以供各子模块或者功能调用时作为上下文环境使用，当然它也定义和提供了对各模块调用的接口，除此之外，它还必须实现对 **RTC Client API** 的各种事件的处理。

(2) 视频会议模块对象 **CAVDlg** 的结构

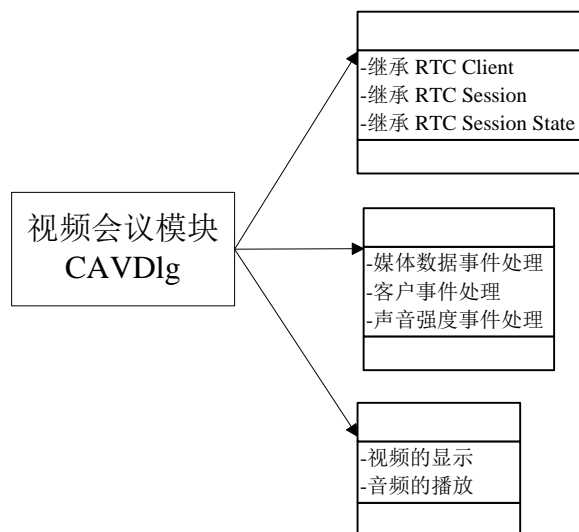


图 3.13 视频会议模块的结构图

视频会议模块对象 **CAVDlg** 首先提供继承父对象中的必要的上下文环境的属性的方法，接着它也要对处于会话过程中发生的与本身相关

的事件的处理，当然，对于视频的显示及音频的播放更是它本身必不可少的功能。

(3) 即时消息对象 CMessageDlg 的结构

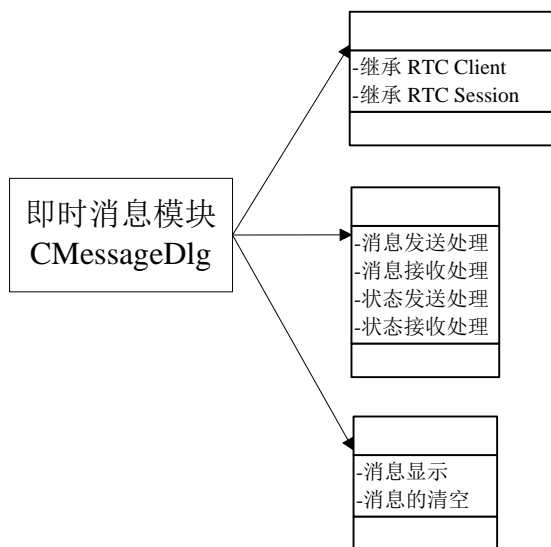


图 3.14 即时消息模块的结构图

即时消息模块的对象 CMessageDlg 和视频会议模块对象 CAVDlg 一样，都是由主对话框直接进行调用的，因此它也需要从父对象中继承一些必要的属性，它本身的主要功能，包括消息的发送和接收、状态的发送与接收以及本地消息的处理也是必须现实的。

3.6 RTC Client API 实例的初始化与释放

在应用程序能够与其它参与者进行连接之前，它必须能够处理在会话期间实时通信的事件。在 PC-PC 的通信中，应用捕获即时消息、音量强度、媒体、客户端消息和会话状态改变等事件。在这里只是实现 RTC Client API 实例的初始化与释放。

```

// RTC Client API 实例的初始化
HRESULT CAVDConfDlg::InitRTCClient()
{
    ... //省略了一些不必要的代码
    //在当前线程中初始化 COM 库
    hr = CoInitialize(NULL);
    //初始化 RTC COM 对象
    hr = CoCreateInstance(CLSID_RTCClient, NULL,

```

```
        CLSCTX_INPROC_SERVER, IID_IRTCClient,
        (LPVOID *) &m_pClient);
//初始化客户端接口
hr = m_pClient->Initialize();
//设置选择的媒体类型
//使用 RTCMT_ALL 参数初始化视频、音频和 T.120 层
m_pClient->SetPreferredMediaTypes(RTCMT_ALL, VARIANT_TRUE);
//设置事件过滤器来指定要监听的 RTC 事件
long lEventMask = RTCEF_SESSION_STATE_CHANGE |
    RTCEF_MESSAGING |
    RTCEF_MEDIA |
    RTCEF_INTENSITY |
    RTCEF_CLIENT;
//初始化本程序需要的回调函数的事件过滤器
hr = m_pClient->put_EventFilter(lEventMask);
//创建 RTC Client Events 对象
m_pEvents = new CRTCEvents;
//初始化事件处理器
hr = m_pEvents->Advise(m_pClient, m_hWnd);
//设置监听 RTC 客户端模式
//RTCLM_BOTH 参数打开标注的 SIP 端口 5060，同时打开一个动态的端口
hr = m_pClient->put_ListenForIncomingSessions(RTCLM_BOTH);
//初始化视频会议模块的状态为 IDLE
if ( m_AVDlg ) m_AVDlg.SetState(RTCSS_IDLE);
return S_OK;
}

// RTC Client API 实例的释放
void CAVDConfDlg::OnOK()
{
    //停止 T.120 程序
    m_pClient->StopT120Applets();
    //准备关闭 RTC Client 实例
    m_pClient->PrepareForShutdown();
    //释放事件处理器与 RTC Client 实例的关联
```

```

m_pEvents->Unadvise(m_pClient);
//关闭 RTC Client 实例同时释放指针
m_pClient->Shutdown();
SAFE_RELEASE(m_pClient);
Cdialog::OnOK();
}

```

3.7 使用 RTC Client API 进行通信类型的选择

在对 RTC Client API 实例进行初始化的同时，里面有一行代码是进行通信类型选择的：

```

m_pClient->SetPreferredMediaTypes(RTCMT_ALL,VARIANT_TRUE)
;

```

在缺省设置的情况下，是能使用所有的通信类型的，如果通信会的参与者都能共享应用程序、传递即时消息、使用音频视频进行会话，那么这些性能都能够自动的可用。如果一个参与者不支持某种特定的通信类型，那么对于所有的会话参与者来说，这种通信类型是不可用的。

会话参与者的平台性能和可用带宽决定了使用何种音频与视频编码解码器。

(1) 视频

RTC Client 在 1/4 GIF 图象格式（176 × 144）分辨率下支持 H.261 和 H.263 编码解码器。这些可变比特编码解码器发送输界于 6~125 Kbit/s 的视频数据。使用 IRTCClient 接口方法 put_MaxBitRate 和 put_TemporalSpatialTradeOff 都会影响到目标的视频转换空间时间分辨率。

(2) 音频

RTC Client 支持许多音频编码解码器。音频编码解码器由连接的两端共同决定。下表列出了所支持的音频编码解码器。

表 3-3 RTC Client 所支持的音频编码解码器的具体参数

编码解码器	采样频率 (kHz)	采样率 (Kbit/s)	帧尺寸 (ms)
G.711	8	64	20
G.722.1	16	24	20
G.723	8	6.4	30, 60, 90
GSM	8	13	20

DVI4	8	32	20
SIREN	16	16	20, 40

在 RTC Client API version 1.2 以后的版本，包括其本身，已经开始支持 G.729 编码解码器，不过需要另外安装其编码解码器。

3.8 对于系统多媒体设备的调整设置

在选择了通信类型和相关设备后，就可以调整这些通信设备了。RTC Client API 提供了一个向导，以便更好的调节摄像头和麦克风。使用 RTC Client API 的 InvokeTuningWizard 方法来调节它们的设置。

下面是调节摄像头和麦克风的界面。

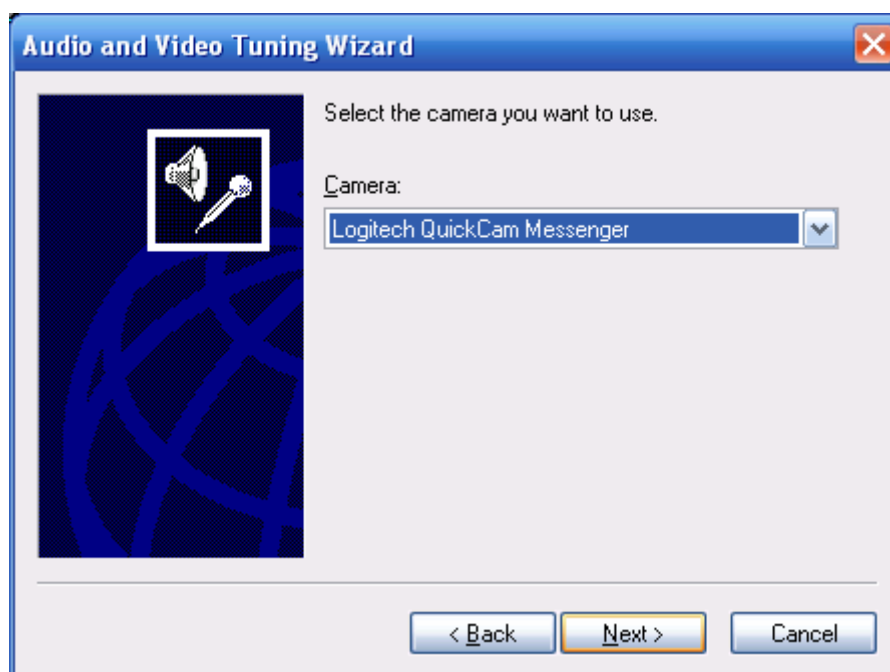


图 3.15 摄像头调节向导



图 3.16 麦克风调节向导

3.9 RTC Events 的处理

一旦事件处理者在 `IRTCEventNotification` 接收器中进行了注册，接收和处理 RTC 事件就变得相当的容易了。当例子程序接收到 RTC 事件时，应用程序的事件处理者就对应用程序的消息处理者发一个消息。`OnRTCEvent()` 处理应用程序接收到的所有事件。根据会话类型的不同，可能会有 RTC 媒体事件、音频强度事件、即时消息事件、会话状态改变事件和客户事件。

```

HRESULT  CAVDConfDlg::OnRTCEvent(UINT  message,  WPARAM  wParam,LPARAM
lParam)
{
    ... ..
    //根据 RTC_EVENT 的类型，查阅合适的时间接口并且调用一个 helper 方法来
    处理事件
    switch ( wParam )
    {
    case RTCE_SESSION_STATE_CHANGE:
        {
            IRTCSessionStateChangeEvent * pEvent = NULL;

```

```
//得到和当前会话相关的事件处理器
hr = pDisp->QueryInterface(IID_IRTCSessionStateChangeEvent,
    (void **)&pEvent);
if ( SUCCEEDED(hr) )
{
    OnRTCSessionStateChangeEvent(pEvent);
    SAFE_RELEASE(pEvent);
}
}
break;
case RTCE_MESSAGING:
    ....
    break;
case RTCE_MEDIA:
    ....
    break;
case RTCE_INTENSITY:
    ....
    break;
case RTCE_CLIENT:
    ....
    break;
}
....
}
```

3.9.1 RTC 媒体事件的处理

处理媒体事件需要取得媒体的类型，然后取得事件类型和原因，然后把消息发送到会话窗口。应用程序可以使用 `RTC Client` 实例的 `get_MediaType` 方法接收用于音频、视频、T.120 和实时转送协议（`RTP`）事件的消息。下面代码说明了如何检索媒体事件并且把它传送到媒体对话框。

```
Void CAVDConfDlg::OnRTCMediaEvent(IRTCMediaEvent * pEvent)
```

```
{
    ... ..
    hr = pEvent->get_MediaType(&lMediaType);
    hr = pEvent->get_EventType(&enType);
    hr = pEvent->get_EventReason(&enReason);
    if ( (m_AVDlg) && (m_AVDlg.GetState() != RTCSS_IDLE) )
        //把媒体的状态传递到会话窗口中
        m_AVDlg.DeliverMedia(lMediaType, enType, enReason);
}
```

3.9.2 音频强度事件的处理

强度事件是当扬声器或者麦克风设备的强度等级改变的时候发生的事件。应用程序可以使用 `RTC Client` 实例的 `get_Direction` 方法获得改变的音频设备。一旦设备被确定，应用程序可以取得当前的设备属性和对改变的内容进行处理。应用程序使用一个进度条来显示音量大小的改变。

```
Void CAVDConfDlg::OnRTCIntensityEvent(IRTCIntensityEvent *pEvent)
{
    ... ..
    hr = pEvent->get_Direction(&enDevice);
    hr = pEvent->get_Level(&lLevel);
    hr = pEvent->get_Min(&lMin);
    hr = pEvent->get_Max(&lMax);
    ... ..
    if ( m_AVDlg.GetState() != RTCSS_IDLE )
        //把强度事件发送到会话窗口中
        m_AVDlg.DeliverIntensity(enDevice, lLevel);
}
```

3.9.3 会话状态改变事件的处理

会话状态遵循和其它即时通讯事件一样的过程。会话状态改变可能包括设置一个新的视频论文会话请求或者向客户端通知一条即时消息的

到来。当应用程序收到一个会话请求时，通过一阵铃声通知客户，请求被接受应答时，然后就可以开始进行会话了。

```
Void CAVDConfDlg::OnRTCSessionStateChangeEvent
(IRTCSessionStateChangeEvent * pEvent)
{
    ... ..
    switch ( enState )
    {
    case RTCSS_INCOMING:
        //当一个会话请求到来的时候，这个事件处理就会被调用
        RTC_SESSION_TYPE enType;
        hr = pSession->get_Type(&enType);
        if ( enType == RTCST_IM )//即时信息请求的处理
            m_pClient->PlayRing(RTCRT_MESSAGE, VARIANT_TRUE);
        else //视频会议会话请求的处理
        {
            ... ..
            //响铃
            m_pClient->PlayRing(RTCRT_PHONE, VARIANT_TRUE);
            //接受会话
            hr = pSession->Answer();
            //把当前会话加入到应用程序中
            hr = AddSession(pSession, enType);
        }
        ... ..
    }
}
```

3.9.4 即时消息事件的处理

即时文本消息通过 `IRTCMessagingEvent` 接口在参与者之间传送。当一个消息事件发生时，这个应用程序必须获得会话和事件类型并且取得会话相关的参与者，以便消息可以被发送到相应的参与者。事件处理程序也可以处理会话参与者的状态的任何改变。

```
HRESULT CAVDConfDlg::OnRTCMessagingEvent(IRTCMessagingEvent * pEvent)
{
    ... ..
    hr = pEvent->get_Session(&pSession);
    hr = pEvent->get_EventType(&enType);
    hr = pEvent->get_Participant(&pParticipant);
    if ( enType == RTCMSET_MESSAGE )
    {
        hr = pEvent->get_MessageHeader(&bstrContentType);
        hr = pEvent->get_Message(&bstrMessage);
        //把消息传到会话窗口中
        if ( m_cMessageDlg )
            m_cMessageDlg.DeliverMessage(pParticipant, bstrContentType,
            bstrMessage);
    }

    //获得消息参与者会话状态改变
    if ( enType == RTCMSET_STATUS )
    {
        hr = pEvent->get_UserStatus(&enStatus);
        m_cMessageDlg.DeliverUserStatus(pParticipant, enStatus);
    }
    return S_OK;
}
```

3.9.5 RTC 客户事件的处理

客户事件类型可能包括关闭连接或者闲置状态事件。对于关闭连接事件的处理，在销毁窗口前必须把 **RTC Client** 实例给释放，或者程序运行会出现错误。对于闲置状态事件处理就非常简单啦，只需把其状态传送到必要的窗口中去就行了。

```
Void CAVDConfDlg::OnRTCClientEvent(IRTCCClientEvent * pEvent)
{
    ... ..
```

```

RTC_CLIENT_EVENT_TYPE enEventType;
hr = pEvent->get_EventType(&enEventType);
if ( enEventType == RTCSET_ASYNC_CLEANUP_DONE )
{
    m_pClient->Shutdown();
    SAFE_RELEASE(m_pClient);
    //RTC 客户已经完成了关闭前的准备，现在销毁窗口
    DestroyWindow();
}
else
if ( m_AVDlg.GetState() != RTCSS_IDLE )
    //把客户状态传送到会话窗口
    m_AVDlg.DeliverClient(enEventType);
}

```

3.10 RTC Session 的建立

在对会话状态改变事件的处理中，有涉及到把当前的会话加入到应用程序中，这里描述一个会话的创建以及把它加入到应用程序中的实现方法。会话是创建是在每次生发起一个请求时，由请求的用户使用 **RTC Client** 实例的 **CreateSession** 方法生成，而接受请求的一端，则是使用 **RTC Client** 的 **IRTCSessionStateChangeEvent** 的 **get_Session** 方法来获得会话实例的。

```

HRESULT CAVDConfDlg::MakeCall(RTC_SESSION_TYPE enType, BSTR bstrURI)
{
    ... ..
    //创建和客户端的会话
    IRTCSession * pSession = NULL;
    hr = m_pClient->CreateSession(enType, NULL, NULL, 0, &pSession);
    //创建并添加一个参与者到会话中
    hr = pSession->AddParticipant(bstrURI, NULL, &m_Participant);
    //把当前会话加入到应用程序中
    hr = AddSession(pSession, enType);
    ... ..
}

```

```
}

HRESULT CAVDConfDlg::AddSession(IRTCSession * pSession, RTC_SESSION_TYPE
enType)
{
    //根据会话类型的不同来决定进行不同的处理
    BOOL fAVSession = ( enType != RTCST_IM );
    if ( fAVSession ) //对于视频会议会话的处理
    {
        BOOL ret = m_AVDlg.Create(IDD_AVDLG, this);
        m_AVDlg.ShowWindow(SW_SHOW);
        m_AVDlg.SetParentClient(m_pClient);
        m_AVDlg.SetSession(pSession);
    }
    else//对于即时消息会话的处理
    {
        BOOL ret = m_cMessageDlg.Create(IDD_MESSAGEDLG, this);
        m_cMessageDlg.ShowWindow(SW_SHOW);
        m_cMessageDlg.SetParentClient(m_pClient);
        m_cMessageDlg.SetSession(pSession);
    }
    return S_OK;
}
```

3.11 视频会议会话中的要点分析

3.11.1 视频会议会话中 RTC Client 事件的处理

在视频会议会话中要处理的事件有 RTC 媒体事件、RTC Client 事件和音频强度事件。以下为现实方法。

```
HRESULT CAVDlg::DeliverMedia(long lMediaType, RTC_MEDIA_EVENT_TYPE
enType, RTC_MEDIA_EVENT_REASON enReason)
{
```

```
... ..
switch ( enType )
{
case RTCMET_STOPPED://当没有收到任何媒体流里，把视频会议的窗口销毁
    ::ShowWindow(m_hRecvVideoParent, SW_HIDE);
    ::SetParent(m_hRecvVideoParent, NULL);
    ::ShowWindow(m_hPrevVideoParent, SW_HIDE);
    ::SetParent(m_hPrevVideoParent, NULL);
    SetState(RTCSS_IDLE);
    DestroyWindow();
    return S_OK;
    ... ..
}
hr = ShowVideo(enVideo, fShow);
... ..
}
```

```
HRESULT CAVDlg::DeliverClient(RTC_CLIENT_EVENT_TYPE enEventType)
{
    switch ( enEventType )
    {
    case RTCCET_VOLUME_CHANGE:
        ShowAudio();
        break;
    case RTCCET_DEVICE_CHANGE:
        ShowAudio();
        ShowVideo(RTCVD_PREVIEW, m_fShowPrev);//显示预览窗口
        ShowVideo(RTCVD_RECEIVE, m_fShowRecv);//显示视频窗口
        break;
    }
    return S_OK;
}
```

```
HRESULT CAVDlg::DeliverIntensity(RTC_AUDIO_DEVICE enDevice, long lLevel)
{
```

```
switch ( enDevice )
{
case RTCAD_SPEAKER:
    m_cSpeakerLevel.SetPos(lLevel); //设置音量显示条
    break;
case RTCAD_MICROPHONE:
    m_cMicLevel.SetPos(lLevel); //设置音量显示条
    break;
}
return S_OK;
}
```

3.11.2 视频会议会话中视频音频的播放

视频会议会话中音频的播放，是在会话正常建立的情况下由系统自动进行播放，应用程序只需对音量进行控制即可，而视频的播放则比较麻烦。在会话建立后，先使用 RTC Client 实例的 `get_IvideoWindow` 方法取得播放视频的 `IvideoWindow` 的句柄，然后创建适合的区域来作为 `IvideoWindow` 显示的载体。

```
HRESULT CAVDlg::ShowAudio()
{
    ... ..
    //取得当前的媒体类型
    hr = m_pParentClient->get_MediaCapabilities( &lMediaCaps );
    //扬声器的音频流的处理
    if ( lMediaCaps & RTCMT_AUDIO_RECEIVE )
    {
        //先判断该设备是否为静音状态
        hr = m_pParentClient->get_AudioMuted(RTCAD_SPEAKER, &fMute);
        //设置静音按钮的状态
        m_cSpeakerMute.SetCheck(fMute?BST_CHECKED:BST_UNCHECKED);
        //取得该设备的音量
        hr = m_pParentClient->get_Volume(RTCAD_SPEAKER, &lVolume);
        //设置控制该设备的音量的滑动按钮的位置
    }
}
```

```
        m_cSpeakerSlider.SetPos(lVolume);
    }
    //麦克风的音频流的处理
    if ( lMediaCaps & RTCMT_AUDIO_SEND )
    {
        hr = m_pParentClient->get_AudioMuted(RTCAD_MICROPHONE, &fMute);
        m_cMicMute.SetCheck(fMute?BST_CHECKED:BST_UNCHECKED);
        hr = m_pParentClient->get_Volume(RTCAD_MICROPHONE, &lVolume);
        m_cMicSlider.SetPos(lVolume);
    }
    ... ..
}

HRESULT CAVDlg::ShowVideo(RTC_VIDEO_DEVICE enDevice, BOOL fShow)
{
    ... ..
    //取得视频窗口的句柄
    hr = m_pParentClient->get_IvideoWindow(enDevice, &pVid);
    //对预览窗口的处理
    if ( enDevice == RTCVD_PREVIEW )
    {
        //设置视频窗口的显示区域
        hRegion = CreateRectRgn(0, 0, m_lRecvWidth, m_lRecvHeight);
        //设置该窗口为可绘图区
        ::SetWindowRgn(m_hRecvVideoParent, hRegion, TRUE);
    }
    else
        //对接收到的视频的窗口的处理
        ::ShowWindow(m_hRecvVideoParent, fShow ? SW_SHOW:SW_HIDE);
    ... ..
    //设置视频容处于可见状态
    pVid->put_Visible(true);
    ... ..
}
```

3.12 即时消息会话中的要点分析

3.12.1 即时消息会话中消息发送与接收的处理

由于 RTC Client API 中，消息是使用 UNICODE 来处理的，这给我们在处理中文字符的时候带来了很大的方便，但是也得注意，在对 UNICODE 格式的数据进行处理时，要使用 UNICODE 版本的函数。

```
Void CmessageDlg::OnSendtext()
{
    ... ..
    //取得输入的文本消息
    GetDlgItemTextW(m_hWnd, IDC_TEXT, szBuf, TEXT_MAX );
    bstrMessage = SysAllocString( szBuf );
    //把消息显示到本地会话窗口
    DoDisplayMessage();
    //发送消息
    m_pParentSession->SendMessage(NULL, bstrMessage, 0);
    ... ..
}

HRESULT CmessageDlg::DeliverMessage(IRTCParticipant * pParticipant,
                                     BSTR bsrtContentType, BSTR bstrMessage)
{
    ... ..
    hr = pParticipant->get_UserURI(&bstrURI);
    //把消息转换为 UNICODE 格式
    wcstombs(szBuf, bstrURI, TEXT_MAX);
    wsprintf(szBuf2, "%s says:", szBuf);
    //显示参与者名称
    SendDlgItemMessageW(IDC_MESSAGE, LB_INSERTSTRING,
                        (WPARAM)-1, (LPARAM)szBuf2);
    wcstombs(szBuf, bstrMessage, TEXT_MAX);
    //显示消息
```

```

SendDlgItemMessageW(m_hWnd, IDC_MESSAGE, LB_INSERTSTRING,
                    (WPARAM)-1, (LPARAM)bstrMessage);
... ..
}

```

3.12.2 即时消息会话中参与者状态的处理

即时消息会话中，参与者的状态只有两种：RTCMUS_IDLE 和 RTCMUS_TYPING。当参与者在输入消息的时候，应用程序向另一方的参与者发送状态事件，并将其状态表示出来。

```

Void CmessageDlg::OnChangeText()
{
    m_pParentSession->SendMessageStatus(RTCMUS_TYPING,0);
}

HRESULT CmessageDlg::DeliverUserStatus(IRTCPparticipant * pParticipant,
                                       RTC_MESSAGING_USER_STATUS enStatus)
{
    ... ..
    if ( enStatus == RTCMUS_TYPING )
    {
        wsprintf(szTyping, "%s is typing.", szBuf);
        SetDlgItemText( IDC_STATUSTEXT, szTyping );
    }
    if ( enStatus == RTCMUS_IDLE )
    {
        char szTyping[TEXT_MAX] = "The session is idle.";
        SetDlgItemText( IDC_STATUSTEXT, szTyping );
    }
    ... ..
}

```

第四章 总结

在这次的开发过程中，笔者第一次体验到了一个完整的软件工程的过程，从需求分析到总体设计，到详细设计，再到最后的测试与维护工作。通过这次开发，笔者不仅对以前的软件只是有了一个全面的认识，而且还学到了很多以前没有学到过的知识，具体说来，有以下几个方面

（1）在开发人员是个人的情况下，或者开发人员对项目的总体情况不能全盘掌握的情况下，宜采用原型的软件开发方法。尽早的开发出软件的模型，使得开发人员 and 用户能尽快看到软件的概貌，然后对软件模型进行评估，提出新的目标和方案，作为下一阶段的目标。以此不断的螺旋上升，最终达到用户的要求，完成软件的开发。

（2）对于具有一定规模的软件，相对结构化开发方法，宜采用面向对象的软件开发方法。采用面向对象的软件开发方法，符合人的思维习惯，开发人员能够比较容易的掌握整个系统的情况，易设计，易修改。

（3）对于该聊天工具来说，任何一端都可以作为服务器，也可以是客户端，也就是说服务器是动态创建的，不是静止的。但对于一个已创建好的服务器来说，他又是一个典型的 C/S 结构，具有很大的灵活性。

（4）RTC Client API 的功能非常强大。除了能实现与 MSN Messenger 几乎同样的功能之外，它还能和 Microsoft® Office Live Communications Server 2005 结合，开发可缩放的应用程序，当然，它的更大的优势在于还能开发除了 PC-PC 之外的 PC-phone、phone-phone 之间会话的应用程序。它还可以应用于 Windows CE 中用于开发 IP 电话。

（5）从 Windows NT 开始，Microsoft 就已经从底层开始对 UNICODE 进行支持，并为所有的 API 函数提供了 UNICODE 版本，并提供了 ASCII 与 UNICODE 相互转换的函数。在应用程序开发的过程中，如果要处理 UNICODE 的数据，调用 UNICODE 版的 API 函数即可。

参考文献

- [1] 侯俊杰. 深入浅出 MFC 第 2 版. 华中科技大学出版, 2001 年 1 月
- [2] Jeff Prosise. Programming Windows with MFC Second Edition (影印版). 北京大学出版社, 2000 年 9 月
- [3] 求是科技. Visual C++ 6.0 程序设计与开发技术大全. 人民邮电出版社, 2004 年 9 月
- [4] 鲁晓东、李育龙、杨健. JSP 软件工程案例精解. 电子工业出版社, 2005 年 1 月
- [5] Brian W. Kernighan、Dennis M. Ritchie. The C Programming Language Second Edition (影印版). 清华大学出版社, 2000 年 9 月
- [6] William J. Collins. Data Structures and the Standard Template Library (英文版). 机械工业出版社, 2003 年 3 月
- [7] 吴炜煜. 面向对象分析设计与编程 OOA/OOD/OOP. 清华大学出版社, 2000 年 4 月
- [8] Charles Petzold. Programming Windows Fifth Edition. Microsoft Press, 1998 年 10 月
- [9] 谢希仁. 计算机网络 第 4 版. 电子工业出版社, 2003 年 6 月
- [10] 汤子瀛、哲凤屏、汤小丹. 计算机操作系统 (修订版). 西安电子科技大学出版社, 2001 年 8 月
- [11] 张海藩. 软件工程导论 (第四版). 清华大学出版社, 2003 年 12 月
- [12] Microsoft Corporation. MSDN Library Visual Studio 6.0 release
- [13] Duane Burton. Integrating Rich Client Communications with the Microsoft Real-Time Communications API. 2002 年 6 月,
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwxp/html/rtc_api_final.asp
- [14] Microsoft Corporation. 用于网络地址转换的 Windows XP 实时通信客户端支持. 2004 年 2 月,
<http://support.microsoft.com/default.aspx?scid=kb;zh-cn;316397>
- [15] Tom Fout. Media Support in the Microsoft Windows Real-Time Communications Client. 2001 年 11 月,
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwxp/html/mediainrtcclient.asp?frame=true>
- [16] Microsoft Corporation. Real-Time Communications (RTC) Client SDK. 2004 年 10 月,
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/rtclnt/rtc/real_time_communications_rtc_client_start_page.asp

致谢

对于一个大学生来说，毕业设计和论文的重要意义已经不仅仅是其字面上所能体现的。在进行毕业设计和论文写作过程中，虽然都是由我个人独立完成的，但是指导老师姜灵敏老师对我的帮助之重要不言而喻，在决定毕业课题时，郑琪老师和黄红桃老师也给了我很多中肯的意见，张晶老师也对我的论文写作给予很多精心的指点。在系统的开发与调试中，杨泓伦同学和张凯同学给了我极大的支持和帮助。同时，我也希望在此表示对我的所有的任课老师的感激之情，特别是王玉山老师、蒋吉频老师、严辉老师和张晶老师，正是由于他们不懈的教导，才构成了这个毕业设计和论文的技术知识基础。也正是因为有了他们的帮助与支持，我才能不断地取得进步，从而能取得今天的成绩。

再次感谢以上提到的所有老师和同学，以及在我的学习、生活中关心我、帮助我的所有人。