文章编号:1001 - 9081(2004)06 - 0114 - 03

基于 Speech SDK 的语音控制应用程序的设计与实现

李禹材[1,2],左友东[2],郑秀清[2],王 玲[2]
(1. 四川师范大学 分析测试中心,四川 成都 610066; 2. 四川师范大学 计算机科学学院,四川 成都 610066)
(lycltb @tom. com)

摘 要:分析了微软 Speech SDK 5.1 里语音应用程序接口(SAPI)的结构和工作原理,提出了语音控制应用程序的设计方法,并以"Z + Z 智能教学平台的语音识别接口"的设计为例,展示了这类系统的主框架和关键技术。

关键词:语音识别;COM;SAPI;语音控制
中图分类号: TP391.42 **文献标识码**:A

Design and Implementation of Voice Controlled Application Based on Speech SDK

LI Yu-cai[1,2], ZUO You-dong[2], ZHENG Xiu-qing[2], WANG Ling[2]
(1. *Analysis and Testing Center, Sichuan Normal University, Chengdu Sichuan* 610066, *China*;
2. *College of Computer Science, Sichuan Normal University, Chengdu Sichuan* 610066, *China*)

Abstract: This paper analyzes and summarizes the architecture and theory of speech application program interface of Microsoft Speech SDK 5.1. A method of designing a voice controlled application is proposed, and its main frame and key technology are illustrated in the procedure of designing speech recognition interface for Z + Z intelligence education platform.

Key words: speech recognition; COM; SAPI; voice controlled application

1 前言

随着计算机技术的快速发展和应用领域的不断扩大,在与计算机的信息交流中,人们越来越需要一种更方便、更自然的方式。而语言是人类交流信息最自然、最有效和最方便的手段。让计算机听懂我们说的话,从而让我们与计算机用语言方便自然地进行交流,一直是近半个世纪以来,语音识别技术追求的目标。

开始于上世纪 50 年代的语音识别技术,在 90 年代从实验室走向实用,特别是 IBM 推出的 ViaVoice 语音系统和微软的 Speech SDK,提供了一个语音识别和合成的二次开发平台,可以识别多种语言,如英文、中文、日文。这些语音开发平台,为我们与计算机用语言进行交流提供了坚实的技术支持,我们可以利用它们在自己开发的软件里嵌入语音识别和合成功能,从而使用户可以用声音来代替原来使用键盘、鼠标完成的操作,比如:文字输入、菜单控制、画图、公式输入、用声音向用户汇报程序执行结果或提示信息等。

在国内,语音识别应用系统的开发开始起步,应用微软的 Speech SDK 开发出一些实用系统,如武汉大学开发的"基于语音识别的火车票查询系统"[1],上海交大的 Call Center 系统[2]。本文根据开发"Z + Z 智能教学平台的语音识别接口"的经验,分析了微软 Speech SDK 5.1 里语音应用程序接口(SAPI)的结构和工作原理,提出了语音控制应用程序设计的方法,并通过"Z + Z 智能教学平台的语音识别接口"的设计和实现,展示了这类系统的主框架和关键技术。对语音控制应用程序的设计具有普遍的参考价值。

文中主要针对用 C++ 建立的 Windows 应用程序。而微软 Speech SDK 5.1 除 C++ 外,还支持 VB、C # 和 Jscript 语言。其它语言系统的实现,与 C++ 系统大同小异,本文也可作为参考。

2 SAPI 5.1 的结构

IBM、微软等几家公司都提供语音识别和合成的二次开发平台,只有微软是免费的,而且经过实验,微软的识别系统在连续语音识别上的识别率虽然不太高,但在命令控制方式下却很高,完全可以满足语音控制应用程序的要求。

微软 Speech SDK 5.1 全面支持中文语音应用程序的开发,SDK 里提供了语音识别和合成引擎相关组件、应用程序层接口、详细的技术资料和帮助文档。它采用 COM 标准开发,底层协议都以 COM 组件的形式完全独立于应用程序层,为应用程序设计人员屏蔽掉复杂的语音技术,充分体现了 COM 的优点,即语音相关的一系列工作由 COM 组件完成:语音识别由识别引擎(Recognition Engine)管理,语音合成由语音合成引擎(Synthesis Engine)负责;程序员只需专注于自己的应用,调用相关的语音应用程序接口(SAPI)来实现语音功能。SAPI 5.1 的结构见图 1。

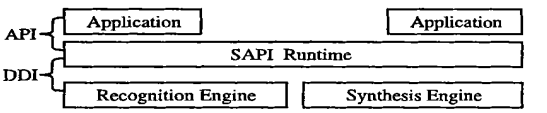

图 1 SAPI 5.1 的体系结构

收稿日期:2003 - 12 - 04;**修订日期**:2004 - 03 - 29

作者简介:李禹材(1966 -),男,四川安县人,讲师,硕士研究生,主要研究方向:计算机软件; 左友东(1975 -),男,江苏海安人,讲师,硕士,主要研究方向:计算机软件、网络.

3　SAPI 5.1 的工作原理

本文只介绍语音识别的工作原理。语音识别的功能由一系列的 COM 接口协调完成,下面先介绍语音识别的主要接口[3]:

·IspRecognizer 接口:用于创建语音识别引擎的实例,在创建时通过参数选择引擎的种类。识别引擎有两种:独占(InProc Recognizer)的引擎和共享(Shared Recognizer)的引擎。独占的引擎对象只能由创建的应用程序使用,而共享的引擎可以供多个应用程序共同使用。

·IspRecoContext 接口:主要用于接受和发送与语音识别消息相关的事件消息,装载和卸载识别语法资源。

·IspRecoGrammar 接口:通过这个接口,应用程序可以载入、激活语法规则,而语法规则里则定义着期望识别的单词、短语和句子。通常有两种语法规则:听写语法(Dictation Grammer)和命令控制语法(Command and Control Grammer)。听写语法用于连续语音识别,可以识别出引擎词典中大量的词汇,例如,可以识别报纸上的一篇文章、你的一段讲话、一个故事等,也就是说,可以用语音代替键盘进行文字输入;命令控制语法主要用于识别用户在语法文件里自定义的一些特定的命令词汇和句子,譬如,菜单命令(打开文件,保存文件,插入等),画图命令(画一条直线,过直线外一点 A 作直线的垂线等)。这些语法规则以 XML 文件的格式编写,通过 IspRecoGrammar 接口载入、激活。

·IspPhrase 接口:用于获取识别的结果,包括识别的文字、识别了哪一条语法规则等。

语音识别的功能由上面的 COM 接口共同完成,而且遵守特定的工作程序。概括地说,语音识别的工作原理遵循 COM 组件的工作原理和一般 Windows 应用程序的工作原理(消息驱动机制),具体如下:

·首先,初始化 COM;

·接着要实例化各个语音接口(以特定的顺序),设置识别语法、识别消息,使识别引擎处于工作状态;

·当有语法规则被识别后,语音接口向应用程序发出语音识别消息;

·在识别消息响应函数里,通过 IspPhrase 接口获取识别的结果;

·应用程序退出时,卸载 COM。

4　语音控制应用程序设计的方法

语音控制应用程序,就是在传统的键盘、鼠标操作方式基础之上,再加上语音控制的功能,比如语音控制命令、用语音命令控制作图等。

在具体设计语音控制应用程序时,要将语音识别的工作程序融入到 Windows 应用程序的运行机制里去,在 Windows 应用程序实现的各个阶段,并行地实现对应的语音识别模块所要求的功能。具体如下:

·设计出完整的 Windows 应用程序,即用键盘、鼠标操作的程序。

·编写语法文件(.XML),它定义了需要识别的命令。

·在 Windows 应用程序的初始化部分,加入这些功能代码:初始化 COM;实例化各个语音接口(以特定的顺序);设置识别语法、识别消息。

·在 Windows 应用程序的消息回调函数里,加入语音识别消息,通过 IspPhrase 接口获取识别的结果。将识别结果映射成相应的控制命令,并执行命令。

5　“Z+Z 智能教学平台的语音识别接口”的设计

“Z+Z 智能教学平台”是一个用于中小学教学的智能化教学平台,既可用于课堂教学,又可用于学生自学。当用于课堂教学时,教师希望该平台能用语音操作,即在老师讲课的同时,计算机就能实时地进行相应的操作,比如:画图,演示例题等。这样会大大地方便教学,提高课堂教学质量。“Z+Z 智能教学平台”已经是一个实用的成熟的平台,我们只需要按照本文提出的方法,将语音识别的代码嵌入原来的应用程序即可。下面就是“Z+Z 智能教学平台的语音识别接口”的主框架和关键技术代码。

·编写语法文件(lyc.XML),定义需要识别的命令。

·在主程序的 WinMain 函数中,主消息循环开始之前,初始化 COM。

```
HRESULT hr = CoInitialize(NULL);
```

·在主程序的 InitInstance 函数中,实例化各个语音接口(以特定的顺序),设置识别语法、识别消息,使识别引擎处于工作状态。

```
//建立语音识别需要的接口对象
CComPtr < ISpAudio >                cpAudio;
CComPtr < ISpRecoGrammar >          g_cpCmdGrammar;
CComPtr < ISpRecoContext >          g_cpRecoCtxt;
CComPtr < ISpRecognizer >           g_cpEngine;
//创建一个共享语音识别引擎
g_cpEngine.CoCreateInstance(CLSID_SpSharedRecognizer);
//创建命令识别上下文
g_cpEngine -> CreateRecoContext(&g_cpRecoCtxt);
//设置语音事件通知消息 ——WM_RECOEVENT
g_cpRecoCtxt -> SetNotifyWindowMessage(hWnd,
    WM_RECOEVENT, 0, 0);
//告诉识别引擎,我们只感兴趣命令识别事件
g_cpRecoCtxt -> SetInterest(SPFEI(SPEI_RECOGNITION),
    SPFEI(SPEI_RECOGNITION));
//从资源文件装载语法规则
g_cpRecoCtxt -> CreateGrammar(GRAMMARID1,
    &g_cpCmdGrammar);
g_cpCmdGrammar -> LoadCmdFromResource(NULL,
    MAKEINTRESOURCEW(IDR_CMD_CFG), L"SRGRAMMAR",
    MAKELANGID(LANG_NEUTRAL, SUBLANG_NEUTRAL), SPLO_
    DYNAMIC);
//激活语法规则
g_cpCmdGrammar -> SetRuleState(NULL, NULL, SPRS_ACTIVE);
```

·识别引擎开始工作,如果有正确的命令输入,则按照语法规则被识别,语音接口向应用程序发出语音识别通知消息——WM_RECOEVENT;

·在主程序的 WndProc 函数里,加入一条消息处理。

```
//有命令被识别的消息
```

```
case WM-RECOEVENT:
//调识别结果处理函数
ProcessRecoEvent(hWnd);
```

·在 ProcessRecoEvent(hWnd)里,通过 IspPhrase 接口获取识别的结果。

```
//通过循环提取,将识别的各个二级语法名和相应的关键词放入
//一个结构数组中
//提取识别的语法规则名
pulIds[iCnt].pwszRule = pRule -> pszName;
//  提取识别的短语
pLycPhrase -> GetText(ulFirstElement, ulCountOfElements,
    FALSE, &(pulIds[iCnt].pwstrCoMemText), NULL);
```

·将结果数组转化为文本形式的控制命令,并执行。

·应用程序退出时,卸载 COM。可以在主程序的 WinMain 函数中,启动主消息循环的代码之后,加入如下代码:

```
//卸载 COM
CoUninitialize();
```

6 结束语

随着语音识别和合成技术的不断提高,语音技术的应用会越来越普遍。目前,语音识别技术中控制和命令(Command and Control)工作方式的识别率非常高,能够达到实际使用的要求,这为用语音代替键盘、鼠标来控制计算机提供了技术保障,相信这种语音控制应用程序会很快发展起来。本文提出的方法并非设计这类系统的唯一方法,实际上有多种途径、多种选择来实现它。例如:可以构造一个类,用于声明要用到的语音接口,构造一个初始化语音识别的成员函数和一个提取识别结果的成员函数,然后在主程序中实例化这个类,调用相应的成员函数完成语音识别功能。限于篇幅,本文只陈述了相对简单明了的一种,用于阐明语音接口的工作原理和工作程序,并示意了如何将语音接口融入到应用程序中去,方便其他用户开发语音应用程序。本文从实践中总结、提出的语音控制应用程序的设计方法,对于开发设计这类系统,具有很高的参考价值。

参考文献

[1] 吴萍,胡瑞敏,等. 火车票查询系统中语音识别的研究及实现[J]. 计算机工程与应用,2003,39(33):227～229.

[2] 朱杰,张申生. 基于 COM 技术的语音应用程序的设计和实现[J]. 计算机工程,2001,27(11):143～145.

[3] Microsoft Speech SDK 5.1 Help[EB/OL] http://www.microsoft.com.

(上接第 113 页)

首先测试该渲染节点和控制节点之间的平均反应时间 $t_{delay} = t_{read} + r_{net} + t_{write}$,该反应时间由数据包的读出、写入以及网络延时三部分构成。控制节点发出一个数据包,渲染节点将该数据包立即弹回,送出和收到之间的时间差为 $T_{recv} - T_{send}$,由上面的定义不难看出 $T_{recv} - T_{send} = 2t_{delay}$,所以多次测试的结果表示为:$\overline{t_{delay}} = \frac{1}{2N}\sum_{i=1}^{N}(T_{recv}i - T_{send}i)$。为了避免由于网络的抖动一起的偶然误差的存在,需要统计平均反应时间的方差 $D(t_{delay})$。当 $D(t_{delay}) > h$ 时,认为发生了偶然误差,需要重新进行测试。这里 h 是常数,可以根据实际的网络环境和所需要达到的精度进行调节。

在获得了平均网络反应能力 $\overline{t_{delay}}$ 后,渲染节点向控制节点发送一个带有自己时钟 T_r 的数据包,控制节点收到数据包后立即提取自己的时钟 T_c,这样就可以获得控制节点和渲染节点之间的时钟偏差 $t_{offset} = T_c - T_r - \overline{t_{delay}}$。多次测试的结果为:

$$\overline{t_{offset}} = \frac{1}{N}\sum_{i=1}^{N}(T_ci - T_ri - \overline{t_{delay}})$$

同样也需要计算 $D(t_{offset})$ 的值:

$$D(t_{offset}) = \sum_{i=0}^{N}\left[(T_ci - T_ri - \overline{t_{delay}}) - \overline{t_{offset}}\right]^2$$

并在 $D(t_{offset}) > h$ 时认为发生了偶然误差,重新测试。

该方法的测试结果表明校时后控制节点和渲染节点之间的时钟误差仅为 2 毫秒。这样的误差在帧率低于 100 时人眼是无法感觉到这种误差的存在的。

另外就是换页同步,就是后备显示缓冲区翻到前台缓冲区的同步。因为在重叠区的物体运动,会因为两个渲染节点的 FPS 不一样产生错位。所以,采用强制的由控制节点统一向各个渲染节点发送翻页指令进行尽可能同步地刷新。

5 结论和进一步研究

试验和应用证明,对于绝大多数展示类的市场要求,本系统都能达到很好的效果,即使投影仪之间存在色差和亮度差,由于算法的柔和过渡,仍然可以达到不错的效果。并且由于网络负载很轻,所以对场景进行的互动操作响应的实时性十分好。

但是,STF-XTD 系统还有很多地方有待改进。为了保证系统架构简约和以高性价比推向市场,我们降低了系统的分布式能力,采用了视棱锥切分的方式进行任务分配,并不能让场景元素完全平均分配给各个渲染节点,会产生负载不平衡的问题,所以在分布式设计和系统架构复杂性、造价方面必须取得平衡点;另外,当场景灰度级趋向于极值(纯黑纯白)或者场景极为单调时,Alpha 曲线的效果会明显下降,现正在测试对场景复杂度和亮度等参数的动态跟踪,以求能高效地实时调整 Alpha 曲线,使拼接技术的普适性进一步提高。

参考文献

[1] Yang J, Shi J, Jin Z, *et al*. Design and Implementation of A Large-scale Hybrid Distributed Graphics System[A]. ACM International Conference Proceeding Series: EGPGV'02[C], September 2002. 39～49.

[2] Brown MS, Seales WB. A Practical and Flexible Tiled Display System [A]. PG'02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications[C], October 2002.

[3] 崔炳荣. 分布式虚拟战场环境研究[J]. 计算机应用,2003,23(6):31～34.

[4] 孙红梅. 分布式虚拟场景实时绘制技术的研究与实现[D]. 北京:中国科学院计算技术研究所,2001.