KUMARAGURU
COLLEGE OF TECHNOLOGY
Department of Computer Science and Engineering

INTERACTIVE VOICE RESPONSE SYSTEM

PROJECT WORK DONE AT
MENTOR LABS,
BANGALORE.

PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE AWARD OF THE DEGREE OF
M.SC APPLIED SCIENCE (SOFTWARE ENGINEERING)
OF BHARATHIAR UNIVERSITY, COIMBATORE.

SUBMITTED BY

C. BARANI RAJ
REG NO: 0137S0026

Under the guidance of

INTERNAL GUIDE

Mrs. B. Jalaja Jayalakshmi, M.C.A.,
Dept. of Computer Science & Engineering,
Kumaraguru College of Technology,
Coimbatore.

EXTERNAL GUIDE

Mr. S.Sunil Kumar, M.C.A.,
Software Engineer,
Mentor Labs,
Bangalore.

KUMARAGURU
COLLEGE OF TECHNOLOGY
Department of Computer Science and Engineering

(Affiliated to Bharathiyar University)
Coimbatore - 641006
(JUNE-2004 TO OCTOBER –2004)

CERTIFICATE

This is to certify that the project entitled

"INTERACTIVE VOICE RESPONSE SYSTEM"

Done by

C. BHARANIRAJ
REG NO: 0137S0026

Submitted in partial fulfillment of the requirements for the award of the degree of M. Sc (Applied Science) Software Engineering of Bharathiyar University.

Professor and HOD **Internal Guide**

Submitted to University examination held on _____

Internal Examiner **External Examiner**

MENTOR LABS
KNOWLEDGE IMPRINTS

Certificate of Accomplishment

This is to certify that

BHARANI RAJ C (0137S0026)

STUDENT OF KUMARAGURU COLLEGE OF TECHNOLOGY
Coimbatore, Tamil Nadu

Was provided with facilities to do a project work at Mentor Labs, Bangalore as per the following details:

Project Title

"Interactive Voice Response System"

Under the Guidance of

Mr. Sunil Kumar

During the period from 25th June 2004 to 15th September 2004

The student of MSc Software Engineering has completed the project work and during the development period, he has evinced keen interest in the project & Successfully completed the project

Place: Bangalore

Date: 15/09/2004

MENTOR LABS * BANGALORE * KNOWLEDGE IMPRINTS

Signature of Issuing Authority

29/3, 2nd Floor, 7th Main, 4th Block, Jayanagar, Bangalore 560-011.
Phone : 30610549, Mobile : 98453 17595, 31827506
e-mail : training@mentor-labs.com, www.mentor-labs.com

DECLARATION

This is to Certify that this project work entitled **"INTERACTIVE VOICE RESPONSE SYSTEM"** Submitted to Kumaraguru College Of Technology, Coimbatore (affiliated to Bharathiar University) is a record of original work done by **C. Bharaniraj, Reg No.0137s0026** under the guidance of **Mr.S.Sunil Kumar M.C.A.,** Mentor Labs, Bangalore and **Mrs. Jalaja Jayalakshmi, M.C.A.,** Department Of Computer Science and Engineering, Kumaraguru College Of Technology, Coimbatore and that this project formed the basis for the award of any Degree / Associate ship / Fellowship or similar title to any candidate of any university.

Signature Of The Candidate

C Bharani

(C. BHARANI RAJ)

Place: Coimbatore
Date: 24-09-04

24/09/04

Staff in-Charge

ACKNOWLEDGEMENT

I am immensely grateful to **Dr.K.K.Padmanabhan, B.Sc (Engg), M.Tech., Ph.D.,** Principal, Kumaraguru College of Technology for his valuable support to come out with this project.

I really feel delighted in expressing my heartful thanks to **Prof. Dr. S. Thangasamy, BE. (Hon's)., Ph.D.,** Head of the Department of Computer Science and Engineering for his endless encouragement in carrying out this project successfully.

My heartfelt thanks to our Course Coordinator **Mr. K.R. Baskaran, B.E., M.S.,** Assistant Professor, Department of Computer Science and Engineering for his encouragement and guidance that paved me to the completion of this project.

I am indent to express my heartiest thanks to **Mrs. B. Jalaja Jayalakshmi, M.C.A.,** Department of Computer Science and Engineering, my project guide who rendered her valuable guidance and support to do this project work extremely well.

I am greatly indebted to **Mr. S.Sunil Kumar, M.C.A., Mentor Labs**, Bangalore who rendered his valuable guidance for my project.

I am also thankful to all the faculty members of the Department of Computer Science and Engineering, **Kumaraguru College of Technology, Coimbatore** for their valuable guidance, support and encouragement during the course of my project work.

My humble gratitude and thanks to my parents who have supported to complete the project and to my friends for lending me valuable tips, support and cooperation through out my project work.

Content

Abstract

The main objective of this project entitled "Interactive Voice Retrieval System" is automating the result announcement system. In this system a student may not even go to college to see the result. To can know them the result through the telephone line itself, by just dialing to the respective college and entering the Examination number.

Here IVRS serves as a bridge between people and computer databases connected through telephone users with the information they need, from anywhere at any time. These systems are used for result announcement, support stock trade transactions, make travel arrangements, and manage bank accounts.

In IVRS speech recognition solutions in the IVR industry is currently driven by improvements in speech algorithms, natural language processing, vocabulary management, and language modeling.

The current project is aimed at developing the IVRs based system for college in which the user need not go to college to know the marks instead they will type his register number through the interface developed by .Net application which is connected through a telephone line, with a voice enabled modem to the IVRS system. The IVRS system will verify the register number, process other authentication details, search its database and will give the required details using voice enabled modem. The response will be through voice.

1. Introduction

An interactive voice recognition system (IVRS) application is a set of rules and commands. These instructions allow a voice-response engine to lead a telephone caller through a hierarchy of menus, collect voice and data input, and perform other operations on behalf of the caller or the program sponsor. The recordings also provide information such as Result of the student in examination. This can save a lot of time and very informative for the students otherwise they have to spend a lot of time in browsing for their results. This application can also save from the point of view of expenses also.

An IVRS is used to collect and provide information to a caller through an automated process. This allows the caller to be helped with minimal or no interaction with an actual human. In an IVRS, a call is answered automatically when it is received, and a prerecorded message is played over the call. This message usually offers callers choice, such as "Press one for CS department, press two to IS department...." Callers can then press a touch-tone button on their phones, generating a tone that the IVRS can detect. The call is then routed according to the response from the caller.

An IVRS can also collect additional numeric information from the caller. For example, a student may call in and enter register number through phone pad. The IVRS can collect that information, look up information related to register no. and provide that information to the student. Text-to-speech features are also commonly incorporated into IVRS. For example, IVRS may have a feature that will read the caller's result to them. That result may be rendered speech through a text-to-speech engine. The required information is searched in the database and converted into speech accordingly.

When we analyze IVRS the basic tasks break down into the following areas:

- File playback
- Tone detection
- Text to speech

Note: Each of these tasks has been explained in detail in implementation section.

2. Literature Survey on IVRS

Since its inception, Microsoft's Telephony API (TAPI) has provided a robust and steadily evolving interface between the telephone and the computer. TAPI 3.0, the version of Microsoft's Telephony API that ships with Microsoft® Windows® 2000, improves on previous versions of TAPI in several ways. By supporting IP Telephony, combining call and media control, and exposing COM objects, TAPI 3.0 makes IP and traditional telephony applications easy to develop. This article assumes the reader is generally familiar with TAPI 3.0.

TAPI 3.1, the next version of TAPI, will continue to evolve TAPI into an even more compelling development platform, incorporating expanded media processing and control API. This paper looks in depth at the plans for these features and shows how they will build on the foundations of TAPI 3.0.

2.1 Telephony Scenarios

TAPI new advanced media processing and control support will enable simple development of many telephony scenarios and applications as follows

- Interactive voice recognition (IVR) systems
- Voice mail or unified messaging (UM) systems
- Speech-enabled Web applications

IVR Systems:-

An IVR system is used to collect and provide information to a caller through an automated process. This allows the caller to be helped with minimal or no interaction with an actual human.

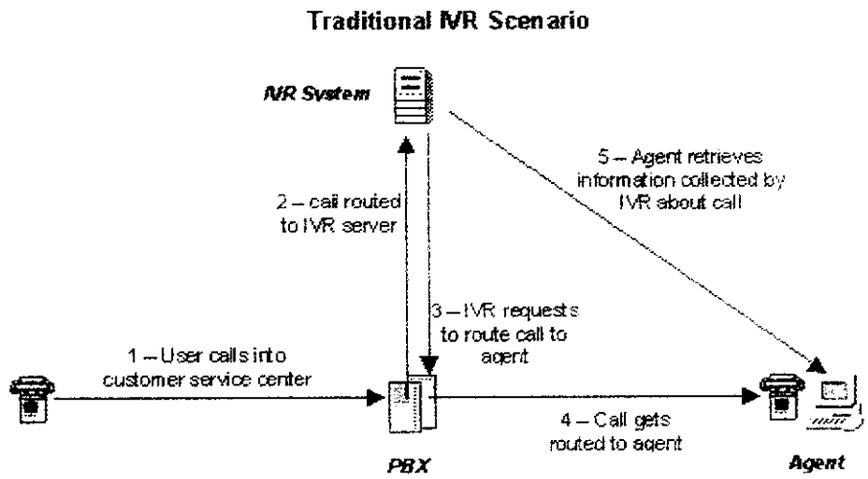
In an IVR system, a call is answered automatically when it is received, and a prerecorded message is played over the call. This message usually offers callers choices, such as "Press one to talk to customer service, press two to talk to accounting...." Callers can then press a touch-tone button on their phones, generating a tone that the IVR system can detect. The call is then routed according to the response from the caller.

An IVR system can also collect additional numeric information from the caller. For example, a customer may call her credit card company's customer service line. Before she is transferred to a customer service agent, the caller enters her

account number in through her phone pad. The IVR system can collect that information, look up information related to her account, and provide that information to the agent's computer when the call arrives at the agent's phone.

IVR systems can also use speech recognition for caller input. The prerecorded message played to the user could be something like "Say 'yes' if you would like to talk to a customer service agent." The IVR system would then use speech recognition to listen for the caller response. This is very similar to the previous example. The difference is the way that the caller provides information to the system. A more advanced speech recognition enabled system may support more extensive and natural grammars, such as being able to interpret the caller asking, "What is my current balance and why did my check bounce?"

Text-to-speech features are also commonly incorporated into IVR systems. For example, a credit card company's IVR system may have a feature that will read the caller's current balance to them. That balance may be rendered speech through a text-to-speech engine.



Above is a representation of a traditional IVR scenario. The PBX and IVR system are both stand-alone hardware devices. The PBX handles routing the call to the request extension, and the IVR system performs the media-processing-related tasks of playing outgoing messages and detecting input from the user. The PBX can transfer the call to the agent's extension, but the agent's desktop computer retrieves the information collected from the IVR.

2.2 Literature Survey on Tools Used

2.2.1 Windows 2000

Microsoft® Windows® Internet Name Service (WINS) has been enhanced for the release of Microsoft Windows 2000 Server. The result is an easier-to-manage and more robust solution for mapping NetBIOS names to IP addresses on Transmission Control Protocol/Internet Protocol (TCP/IP) networks. Windows 2000 includes server enhancements, additional client functions, and an improved management tool. WINS provides a distributed database for registering and querying dynamic computer name-to-IP address mapping in a routed network environment. This support for dynamic registering of NetBIOS computer names means that WINS can be used with Dynamic Host Configuration Protocol (DHCP) services to provide easy configuration and administration of Windows-based TCP/IP networks.

WINS solves the problems inherent in resolving NetBIOS names through IP broadcasts, and frees network administrators from the demands of updating static mapping files, such as LMHOST files. WINS, which is compliant with the NetBIOS Name Server (NBNS) RFCs (1001/1002), also automatically updates the WINS database when dynamic addressing through DHCP results in new IP addresses for computers that move between subnets. Neither the user nor the network administrator needs to make manual accommodations for such name resolutions.

The new implementation of WINS provides a number of features, including:

- **Persistent connections** - This configurable feature allows each WINS server to maintain a persistent connection with one or more replication partners to eliminate the overhead of opening and terminating connections and to increase the speed of replication.

- **Manual tomb stoning** - se of the Manual tomb stoning feature marks a record for deletion so that the tombstone state for the record is replicated across all of WINS , preventing an undeleted copy of the record on a different server database from being re-propagated.

- **Improved management tools** - The WINS Manager is fully integrated with the Microsoft Management Console (MMC), providing a more user-friendly and powerful environment for viewing and managing WINS information.

- **Enhanced filtering and record searching** - these functions help locate records of interest by showing only those that fit a specific criteria. This is particularly useful for analyzing very large WINS databases.

- **Dynamic record deletion and multi-select** - managing the WINS database is made easier with dynamic record deletion and multi-select. Dynamic and static records can be deleted, and the point-and-click interface makes it possible to delete files with non-alphanumeric characters that could not be handled from the command iine.

- **Record verification and version number validation** - two tools are available for quickly checking the consistency between various WINS Services. The tests are done by comparing the IP addresses of a NetBIOS name query returned from different WINS Services or by examining owner address to version-number mapping tables.

- **Export function** - The Export command can be used to place WINS data into a comma-delimited text file that can be imported into Microsoft Excel, reporting tools, scripting applications, and so on, for analysis and reporting.

- **Increased fault tolerance** - Windows 2000 and Windows 98 allow a client to specify more than two WINS servers (up to a maximum of 12 addresses) per interface. The extra WINS addresses are used only if the primary and secondary WINS fail to respond.

- **Dynamic re-registration** - WINS clients can now re-register their NetBIOS name-to- IP address mapping without rebooting the server.

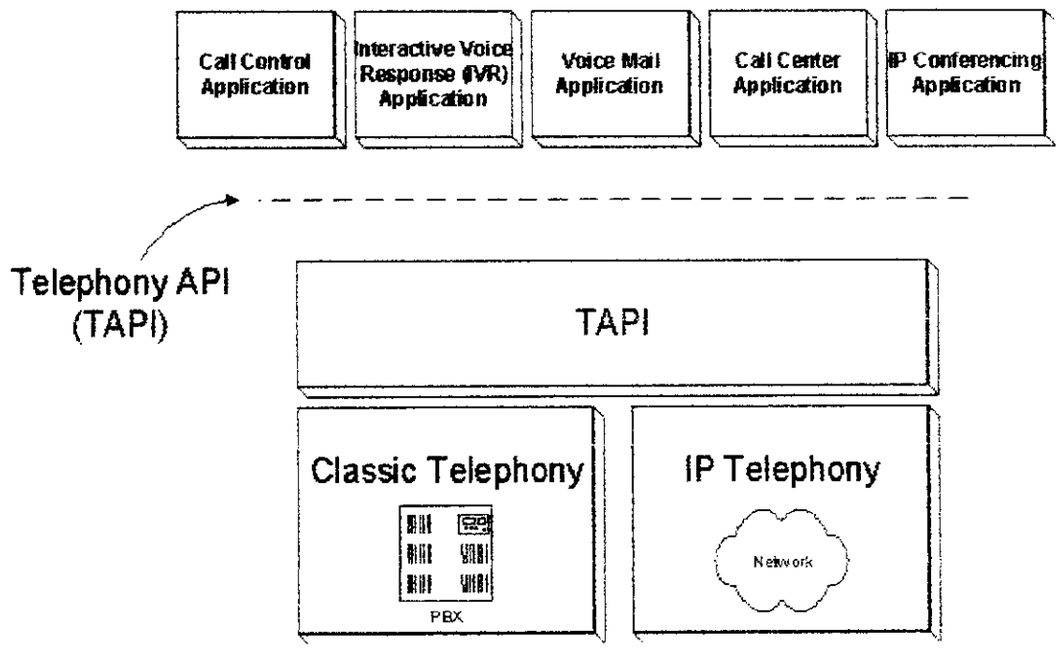
This features make Windows 2000 OS a very powerful operating system.

2.2.2 Introduction to TAPI 3.0

As telephony and call control become more common in the desktop computer, a general telephony interface is needed to enable applications to access all the telephony options available on any computer. The media or data on a call must also be available to applications in a standard manner.

TAPI 3.0 provides simple and generic methods for making connections between two or more computers and accessing any media streams involved in that connection. It abstracts call-control functionality to allow different, and seemingly incompatible, communication protocols to expose a common interface to applications.

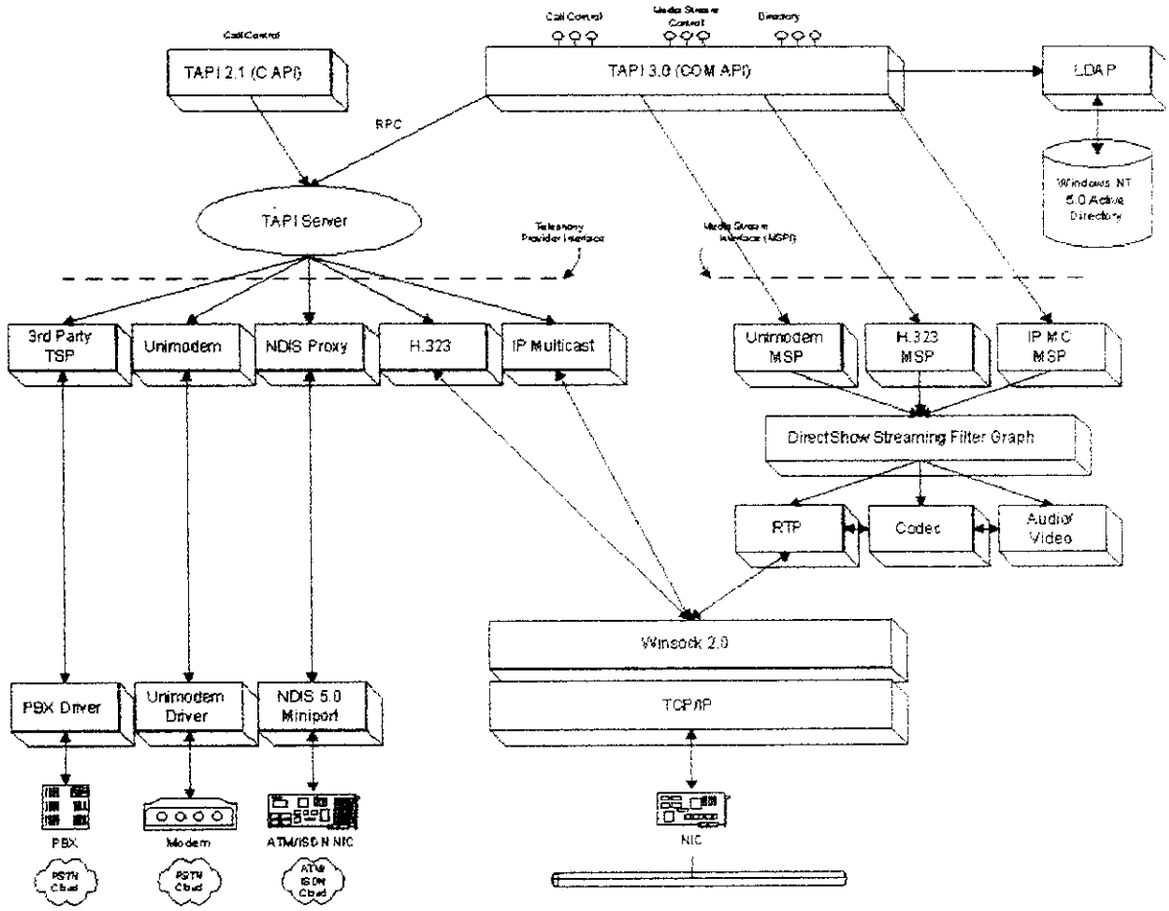
IP telephony is poised for explosive growth, as organizations begin a historic shift from expensive and inflexible circuit-switched public telephone networks to intelligent, flexible, and inexpensive IP networks. Microsoft, in anticipation of this trend, has created a robust computer telephony infrastructure, TAPI. Now in its third major version, TAPI is suitable for quick and easy development of IP telephony applications. See Figure



2.2.3 Inside TAPI 3.0:-

TAPI 3.0 integrates multimedia stream control with legacy telephony. Additionally, it is an evolution of the TAPI 2.1 API to the COM model, allowing TAPI applications to be written in any language, such as C/C++ or Microsoft® Visual Basic®.

Besides supporting classic telephony providers, TAPI 3.0 supports standard H.323 conferencing and IP multicast conferencing. TAPI 3.0 uses the Windows 2000 Active Directory service to simplify deployment within an organization, and it supports quality-of-service (QoS) features to improve conference quality and network manageability. See TAPI architecture in Figure.



There are four major components to TAPI 3.0:

- TAPI 3.0 COM API
- TAPI Server
- Telephony Service Providers
- Media Stream Providers

In contrast to TAPI 2.1, the TAPI 3.0 API is implemented as a suite of COM objects. Moving TAPI to the COM model allows component upgrades of TAPI features. It also allows developers to write TAPI-enabled applications in any language.

- **TAPI Server:-**

  The **TAPI Server** process (TAPISRV.EXE) abstracts the TSPI (TAPI Service Provider Interface) from TAPI 3.0 and TAPI 2.1, allowing TAPI 2.1 Telephony Service Providers to be used with TAPI 3.0, maintaining the internal state of TAPI.

- **Telephony Service Providers:-**

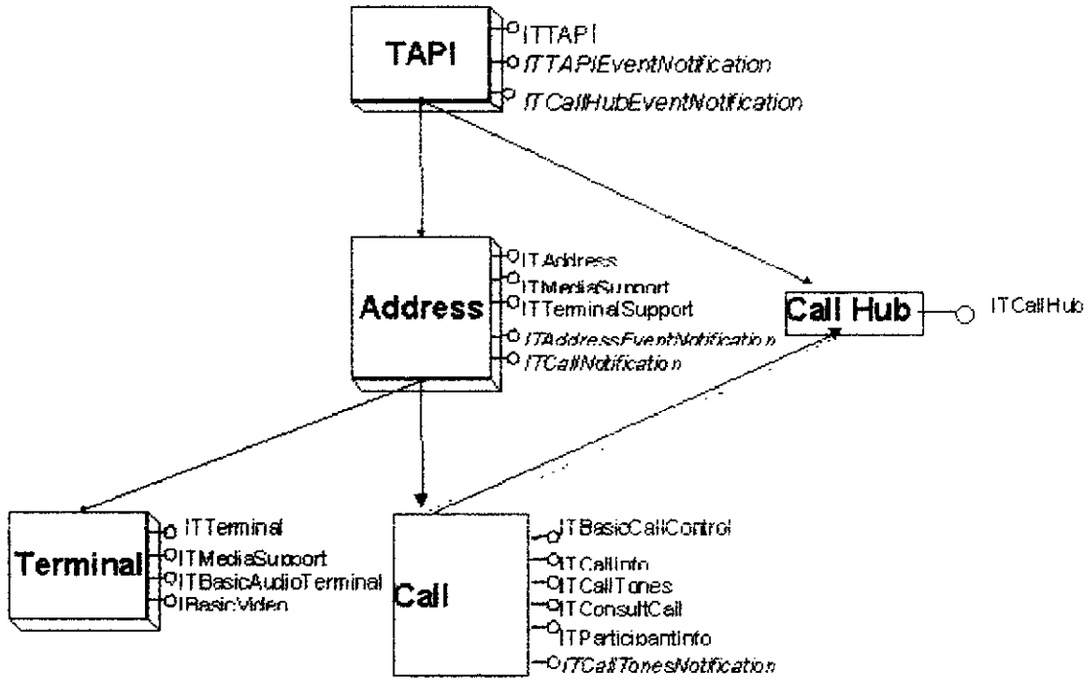
  The **Telephony Service Providers** (TSPs) are responsible for resolving the protocol-independent call model of TAPI into protocol-specific call-control mechanisms. TAPI 3.0 provides backward compatibility with TAPI 2.1 TSPs. Two IP telephony service providers (and their associated MSPs) ship by default with TAPI 3.0: the H.323 TSP and the IP Multicast Conferencing TSP, which are discussed below.

  TAPI 3.0 provides a uniform way to access the media streams in a call, supporting the DirectShowTM API as the primary media-stream handler. TAPI Media Stream Providers (MSPs) implement DirectShow interfaces for a particular TSP and are required for any telephony service that makes use of DirectShow streaming. Generic streams are handled by the application.

- **Call Control Model:-**

  There are five objects in the TAPI 3.0 API, as illustrated in Figure.

  - TAPI
  - Address
  - Terminal
  - Call
  - CallHub



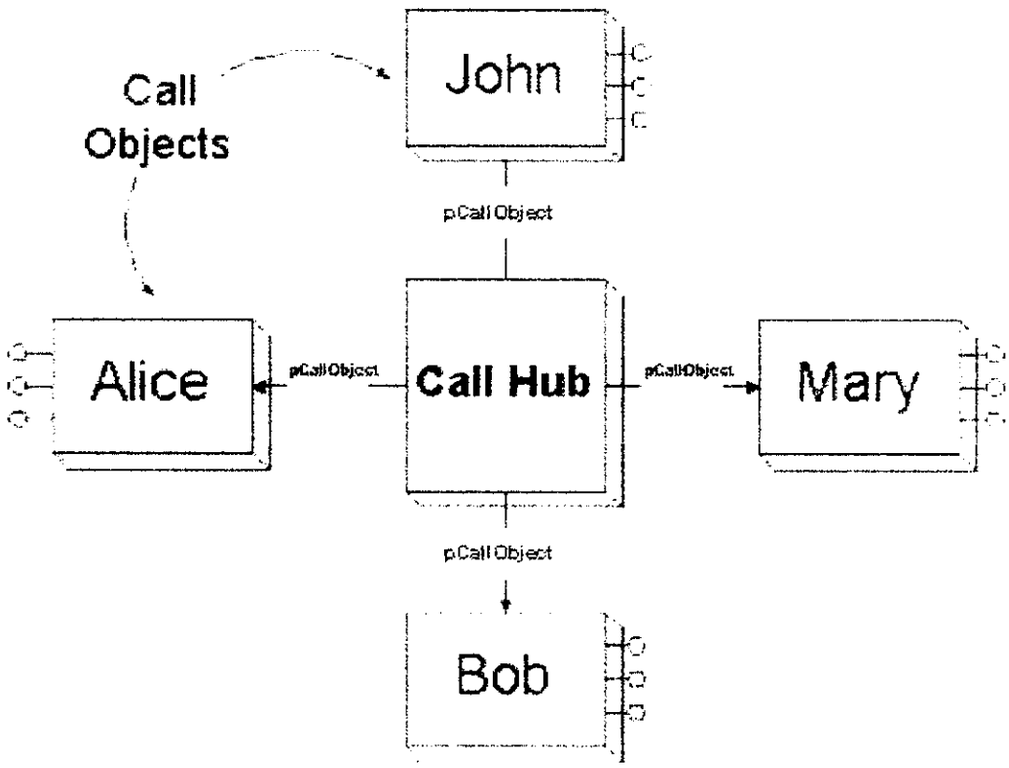
The **TAPI object** is the application's entry point to TAPI 3.0. This object represents all telephony resources to which the local computer has access, allowing an application to enumerate all local and remote addresses.

An **Address object** represents the origination or destination point for a call. Address capabilities, such as media and terminal support, can be retrieved from this object. An application can wait for a call on an Address object or can create an outgoing call object from an Address object.

A **Terminal object** represents the sink, or renderer, at the termination or origination point of a connection. The Terminal object can map to hardware used for human interaction, such as a telephone or microphone, but can also be a file or any other device capable of receiving input or creating output.

The **Call object** represents an address's connection between the local address and one or more other addresses (This connection can be made directly or through a CallHub). The Call object can be imagined as a first-party view of a telephone call. All call control is done through the Call object. There is a Call object for each member of a CallHub.

The **CallHub object** represents a set of related calls. A CallHub object cannot be created directly by an application—it is created indirectly when an incoming call is received through TAPI 3.0. Using a CallHub object, a user can enumerate the other participants in a call or conference, and possibly (because of the location independent nature of COM) perform call control on the remote Call objects associated with those users, subject to sufficient permissions. See Figure.



- **Media Streaming Model**

The Windows® operating system provides an extensible framework for efficient control and manipulation of streaming media called DirectShow. DirectShow, through its exposed COM interfaces, provides TAPI 3.0 with unified stream control.

At the heart of DirectShow is a modular system of pluggable components called filters, arranged in a configuration called a filter graph. A component called the filter graph manager oversees the connection of these filters and controls the stream's data flow. Each filter's capabilities are described by a number of special COM interfaces called pins. Each pin instance can consume or produce streaming data, such as digital audio.

While COM objects are usually exposed in user-mode programs, the DirectShow streaming architecture includes an extension to the Windows driver

model that allows the connection of media streams directly at the device-driver level. Figure 6 below shows a simple PSTN-to-IP bridge: A 64 Kbps voice stream from an ISDN line is compressed into a G.723 audio stream and passed to an RTP payload handler to be sent out over the network.

These high-performance streaming extensions to the Windows Driver Model avoid user-to-kernel mode transitions and allow efficient routing of data streams between different hardware components at the device driver level. Each kernel mode filter is mirrored by a corresponding user-mode proxy that facilitates connection setup and can be used to control hardware-specific features.

DirectShow network filters extend the streaming architecture to computers connected on an IP network. The Real-time Transport Protocol (RTP), designed to carry real-time data over connectionless networks, transports TAPI media streams and provides appropriate time-stamp information. TAPI 3.0 includes a kernel-mode RTP network filter.

TAPI 3.0 utilizes this technology to present a unified access method for the media streams in multimedia calls. Applications can route these streams by manipulating corresponding filter graphs; they can also easily connect streams from multiple calls for bridging and conferencing capabilities.

2.2.4 SAPI 5 Overview

The SAPI application programming interface (API) dramatically reduces the code overhead required for an application to use speech recognition and text-to-speech, making speech technology more accessible and robust for a wide range of applications.

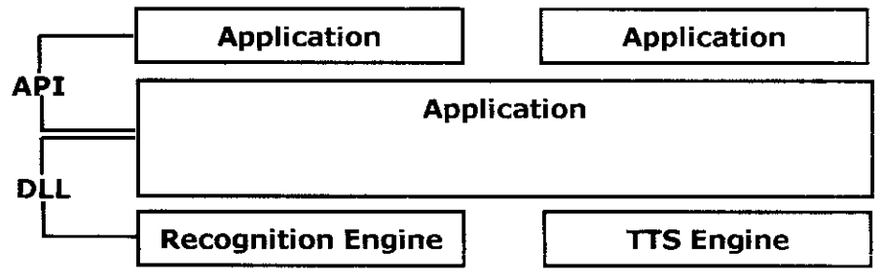
This section covers the following topics:

- API Overview
- API for Text-to-Speech

API Overview:-

The SAPI API provides a high-level interface between an application and speech engines. SAPI implements all the low-level details needed to control and manage the real-time operations of various speech engines.

The two basic types of SAPI engines are text-to-speech (TTS) systems and speech recognizers. TTS systems synthesize text strings and files into spoken audio using synthetic voices. Speech recognizers convert human spoken audio into readable text strings and files.



API for Text-to-Speech

Applications can control text-to-speech (TTS) using the **ISpVoice** Component Object Model (COM) interface. Once an application has created an ISpVoice object, the application only needs to call **ISpVoice::Speak** to generate speech output from some text data. In addition, the IspVoice interface also provides several methods for changing voice and synthesis properties such as speaking rate **ISpVoice::SetRate**, output volume **ISpVoice::SetVolume** and changing the current speaking voice **ISpVoice::SetVoice**

Special SAPI controls can also be inserted along with the input text to change real-time synthesis properties like voice, pitch, word emphasis, speaking rate and volume. This synthesis markup **sapi.xsd,** using standard XML format, is a simple but powerful way to customize the TTS speech, independent of the specific engine or voice currently in use.

The IspVoice::Speak method can operate either synchronously (return only when completely finished speaking) or asynchronously (return immediately and speak as a background process). When speaking asynchronously (SPF_ASYNC), real-time status information such as speaking state and current text location can polled using **ISpVoice::GetStatus**. Also while speaking asynchronously, new text

can be spoken by either immediately interrupting the current output (SPF_PURGEBEFORESPEAK), or by automatically appending the new text to the end of the current output.

In addition to the **ISpVoice** interface, SAPI also provides many utility COM interfaces for the more advanced TTS applications.

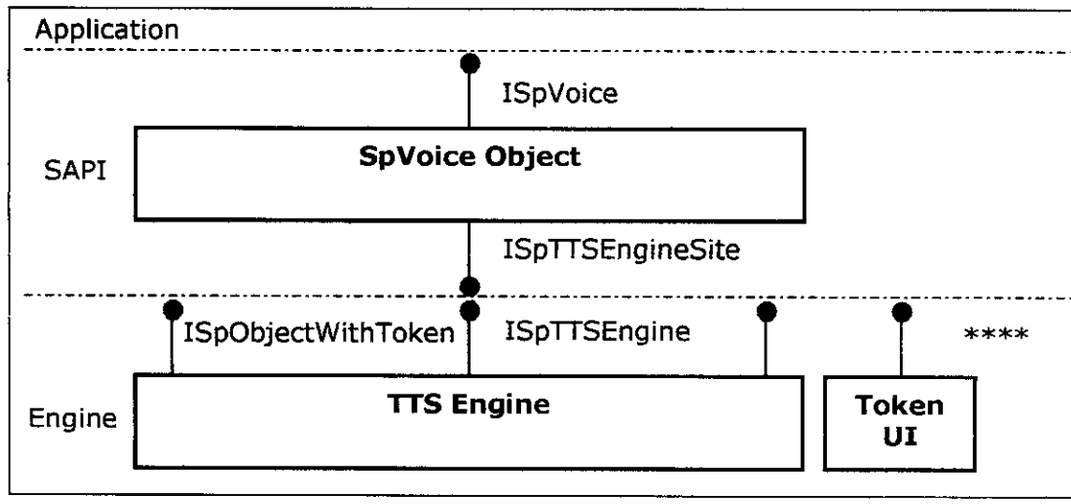
2.2.5 Overview of SAPI 5.0 Architecture

The Microsoft Speech API (SAPI) is a layer of software which sits between applications and speech engines, allowing them to communicate in a standardized way. One of its main goals is enabling application developers to use speech technology in a simple and straightforward way. Another goal is solving some of the more basic complications of developing speech engines, such as audio device manipulation and threading issues, thus allowing engine developers to focus on speech. From an engine vendor's point of view, there are a number of technical advantages to using SAPI 5 over SAPI 4:

- The SAPI 5 DDI has been greatly simplified.
- SAPI 5 can handle all audio format conversion for the TTS engine.
- SAPI 5 parses SAPI 5 XML for the TTS engine. Engine proprietary tags are passed to the engine untouched, allowing the engine to interpret them.
- SAPI 5 performs parameter validation for the engine.
- SAPI 5 has lexicon management features.
- SAPI Objects and Interfaces

There are two main objects of interest to a TTS Engine developer: the SpVoice object (SAPI) and the TTS Engine object (refer to figure). The third object in the figure is a UI component which an engine may or may not implement.

The **SpVoice** object implements two interfaces which we will be concerned with – **ISpVoice**, which is the interface which the application uses to access TTS functionality, and **ISpTTSEngineSite**, which the engine uses to write audio data and queue events. The TTS Engine must implement two interfaces as well – **ISpTTSEngine**, which is the interface through which SAPI will call the engine, and **ISpObjectWithToken**, which is the interface through which SAPI will create and initialize the engine. The UI object, if it exists, must implement **ISpTokenUI**, through which it will be accessed by the SAPI control panel (or, potentially, other applications).



2.2.6 DirectX:

Microsoft® DirectX® is a group of technologies designed by Microsoft to make Microsoft Windows®-based computers an ideal platform for running and displaying applications rich in multimedia elements such as full-color graphics, video, three-dimensional (3-D) animation, interactive music, and surround sound. Built directly into the Windows family of operating systems, DirectX is an integral part of Windows 98 and Windows 2000, as well as Microsoft Internet Explorer. DirectX components may also be automatically installed on your system by advanced multimedia games and applications.

DirectX gives developers a single set of APIs (application programming interfaces) that provides them with improved access to the advanced features of high-performance hardware such as 3-D graphics acceleration chips and sound cards. These APIs control what are called "low-level functions," including two-dimensional (2-D) graphics acceleration; support for input devices such as joysticks, keyboards, and mice; and control of sound mixing and sound output. DirectX also provides developers with tools that help them get the best possible performance from the machines they use. It automatically determines the hardware capabilities of your computer and then sets the application's parameters to match. DirectX also allows you to run multimedia applications that require support for features that a given system doesn't offer by simulating certain hardware devices through a "hardware emulation layer" (called HEL) that provides software-based drivers that act like hardware. For example, a DirectX game that makes use of 3-D imagery can run on a

machine that doesn't have a 3-D acceleration card because DirectX simulates the services of a 3-D acceleration card. Before DirectX, developers creating multimedia applications for Windows machines had to customize their products so that the products would work well on the wide variety of hardware devices and configurations available on Windows machines. Direct X provides something called a "hardware abstraction layer" (HAL for short) that uses software drivers to communicate between game software and computer hardware. As a result, developers can write single versions of their products that utilize DirectX without worrying about the wide range of hardware devices and configurations in existence. DirectX also provides developers with tools that help them get the best possible performance from the machines they use.

DirectSound:-

The audio component of DirectX, DirectSound, uses features such as low-latency mixing playback, accelerated hardware, and 3-D positioning to achieve professional-quality audio performance. With DirectSound, multiple audio signals can be mixed and played at the same time; sound card features are automatically engaged or emulated in the software if not supported by the existing hardware; and custom mixing engines can directly access the sound hardware. Voice management ensures that advanced audio acceleration features found in today's sound cards can be used to the maximum extent possible. DirectX 9.0 provides three new software 3-D audio algorithms, a high-speed Pan3D algorithm, and two different VMAx™ The fastest Pan3D algorithm enables applications to use the DirectSound3D API and be assured that the CPU utilization will be no more than if they had used a regular DirectSound 2-D buffer. Essentially a sound mixing engine, DirectSound places sets of sounds in secondary buffers, combines them and writes them into primary buffers. Only the available processing time limits the number of buffers that DirectSound can mix.

DirectShow:-

DirectShow is a media-streaming architecture for the Microsoft Windows platform that enables the high-quality capture and playback of multimedia streams. The streams can contain video and audio data compressed in a wide variety of formats, including MPEG, audio-video interleaved (AVI), MPEG-1 Layer 3 (MP3), and WAVE files. Capture can be based on either Windows Driver Model (WDM) or

legacy Video for Windows (VFW) devices. DirectShow is integrated with DirectX technologies so that it automatically takes advantage of any video and audio acceleration hardware to deliver the highest possible performance. DirectShow is designed to make it as simple as possible to perform basic playback, format conversion, and capture tasks while at the same time providing access to the underlying stream control architecture for those applications that require custom solutions. You can even create your own DirectShow components to support new formats or custom effects. Examples of the types of applications you can write with DirectShow include: DVD Players, Digital Video Editors, and wave to ASF Converters; MP3 Players; Digital and Analog Video and Audio Capture applications.

2.2.7 Introduction to object oriented component technology:

In the early days of personal computing, programmers diligently wrote procedural code to create character-based PC applications, knowing that individual users would run separate copies of the program on their own machines. Programmers rarely needed to be concerned about how a program would communicate with other code running on the local machine, let alone worry about how their program would interact with other programs running on a remote machine. Times change quickly. Today, the computing power available from a typical desktop computer can well exceed the computing power of room-size computers of the early days. It's now commonplace for Internet-enabled computers to browse data from hundreds of thousands of computers interconnected around the globe. Now that computers can readily connect to each other, programmers need the tools and a unified architecture that allows them to write modular programs to communicate in a standard, organized fashion. Object technology provides the power to deliver this functionality. This chapter gives you a broad overview of the object technology foundation Microsoft has made available to you for developing Windows applications with Visual Studio and VB Dot net.

Object technologies need an underlying framework that can allow complex and interrelated programs to work together in harmony. The object architecture that Microsoft created--called the Component Object Model or simply COM--is the foundation underlying OLE and ActiveX technologies and is Microsoft's solution for providing object-to-object communication. With the enhancements of the Distributed Component Object Model (DCOM), this technology even provides communication services across networks to objects physically located on different machines.

2.2.8 Introduction to VB Dot net:

A lot of press has been given to the new language Microsoft has created: C# (pronounced "C-Sharp"). This is a new language, based on C/C++. C#, like VB.NET, inbuilt specifically for the .NET Framework, and much has been written about it. Given all the hype, some people might wonder why they should choose VB.NET over C#. Although both VB.NET and C# projects are created in the Visual Studio.NET environment, VB.NET was created specifically for VB developers and has a number of unique features that make it a great choice for building .NET applications. VB.NET is still the only language in VS.NET that includes background compilation, which means that it can flag errors immediately, while you type. VB.NET is the only .NET language that supports late binding. In the VS.NET IDE, VB.NET provides a dropdown list at the top of the code window with all the objects and events; the IDE does not provide this functionality for any other language. VB.NET is also unique for providing default values for optional parameters, and for having a collection of the controls available to the developer. Don't forget that C#, like its C and C++ brethren, is case sensitive, something that drives most experienced VB developers crazy. In addition# uses different symbols for equality (=) and comparison (==). Finally, let's face it: If you know VB, you are further down the road with VB.NET than you are with C#. Even though much has changed, the basic syntax of VB.NET is similar to VB, so you already know how to declare variables, set up loops, and so on. As you can see, VB.NET has some advantages over the other .NET languages.

Many people have looked at VB.NET and grumbled about the changes. There are significant changes to the language: a new optional error handling structure, namespaces, true inheritance, free threading, and many others. Some see these changes as merely a way that Microsoft can place a check mark next to a certain feature and is able to say, "Yeah, we do that." However, there are good reasons for the changes inVB.NET.

If you took a Visual Basic 1.0 developer and showed him an niter application with VB COM component middle tier, and a SQL Server back end full of stored Procedures, it would look quite alien to him. Yet, over the past few years, the vast majority of developers have been using Visual Basic to create COM components. The needs for reusability and centralization (away to avoid distributing components to the desktop) have driven this move to the niter model.

The move to the Web revealed some problems. Scalability was an issue, but more complex applications had other requirements, such as transactions that spanned multiple components, multiple databases, or both. To address these issues, Microsoft created Microsoft Transaction Services (MTS) and COM+ Component Services. MTS (in Windows NT 4) and Component Services (an updated MTS in Windows 2000) acted as an object-hosting environment, allowing you to gain scalability and distributed transactions with relative ease. However, VB components could not take full advantage of all that Component Services had to offer, such as object pooling, because VB did not support free threading.

Microsoft threw their support behind SOAP, Simple Object Access Protocol, which allows developers to call a component over HTTP using an XML string, with the data returning via HTTP in an XML string. Components sport URLs, making them as easy to access as any other Web item. SOAP has the advantage of having been a cross-industry standard, and not just a Microsoft creation. At this point, you might be tempted to think that SOAP is all you need, and that you can just stick with VB6. Therefore it is important to understand what VB.NET gives you, and why it makes sense for you, and many other developers, to upgrade to .NET. For example, you create components and want them to be callable via SOAP, but how do you let people know that those components exist? .NET includes a discovery mechanism that allows you to find components that are available to you. "Building Web Services with VB.NET." .NET also provides many other features, such as garbage collection for freeing up resources, true inheritance for the first time, debugging that works across languages and against running applications, and the ability to create Windows services and console applications.

The advantages of VB Dot net are as follows:

- **The Common Language Runtime:-**

  Before proceeding, it's important to understand a little bit more about what is meant by ".NET." There are many ".Nets" here. There is VB.NET, which is the new version of Visual Basic. There is Visual Studio.NET, an Integrated Development Environment that hosts VB.NET, C#, and C++.NET. Underlying all this is the. NET Framework and its core execution engine, the Common Language Runtime.

- **Microsoft Intermediate language (MSIL):**

  In the .NET model, you write applications that target the .NET Framework. This gives them automatic access to such benefits as garbage collection (which destroys objects and reclaims memory for you), debugging, security services, inheritance, and more. When you compile the code from any language that supports the .NET Framework, it compiles into something called MSIL, or Microsoft Intermediate Language. This MSIL file is binary, but it is not machine code; instead, it is a format that is platform independent and can be placed on any machine running the .NET Framework.

- **Just in time compiler (JIT):**

  Within the .NET Framework is a compiler called the Just-In-Time, or JIT, and compiler. It compiles the MSIL down to machine code specific to that hardware and operating system. Your code does not stay IL for long, however. It is the PE file, containing the IL, that can be distributed and placed with the CLR running on the .NET Framework on any operating system for which the .NET Framework exists, because the IL is platform independent. When you run the IL, however, it is compiled to native code for that platform. Therefore, you are still running native code; you are not going back to the days of interpreted code at all. The compilation to native code occurs via another tool of the .NET Framework: the Just-In-Time (JIT) compiler. With the code compiled, it can run within the Framework and take advantage of low-level features such as memory management and security. The compiled code is native

  Code for the CPU on which the .NET Framework is running, meaning that you are indeed running native code instead of interpreted code. A JIT compiler will be available for each platform on which the .NET Framework runs, so you should always be getting native code on any platform running the .NET Framework. Remember, today his is just Windows, but this could change in the future.

- **Inheritance:**

  In looking at the fundamental changes, it's important to understand that the number one feature request from Visual Basic developers, for years, has been inheritance. VB has had interface inheritance since VB4, but developers wanted real or implementation inheritance. Why? What are the benefits? The main benefit of inheritance is the ability to create applications more quickly. This is an extension of the promise of component design and reusability. With implementation inheritance, you build a base class and can inherit from it, using it as the basis for new classes. For example, you could create a Vehicle class that provides basic functionality that could be inherited in both a Bicycle class and a Car class. The important point here is that Bicycle and Car inherit the functionality, or the actual code, from the Vehicle class. In VB4, the best you could do was inherits the structure, minus any implementation code. In VB.NET, the functionality in that base class is available to your other classes as is, or you can extend and modify it as necessary.

- **Debugging tools:**

  .NET provides you with integrated debugging tools. If you also had C++ components in the mix, you had to use the C++ debugger on those components. With .NET, there is one debugger. Any language that targets the .NET Framework can be debugged with that single debugger, even if one part of your application is written in VB.NET and calls another part written in C# (pronounced "C-Sharp"), or any other language built to target the .NET Framework.

- **Security and Components:**

  .NET supplies a standard security mechanism, available to all parts of your application. .NET provides a possible solution to DLL Hell, and removes much of the complexity of dealing with COM and the registry. .NET allows you to run components locally, without requiring the calling application to go to the registry to find components

- **Cross-Language Interoperability:**

If you've been building COM components for a while, you know that one of the great promises of COM is that it is language independent. If you build a COM component in C++, you can call it from VB, and vice versa. However, to reach that point, your code had to be compiled to a COM standard. Much of this was hidden from the VB developer, but your component had to implement the Unknown and Dispatch interfaces. Without these interfaces, they would not have been true COM components.

COM is only giving you cross-language interoperability at the binary level, however. This means that you can only take advantage of this interoperability at run time. Now, however, the CLR gives you much better language interoperability. Not only can you inherit classes from one PE written in language A and use them in language B, but debugging now works across components in multiple languages. This way, you can step through the code in a PE written in C# and jump to the base class that was not just the run time given to you by COM Written in VB.NET. This means that your cross-language interoperability is happening at design time and run time. In addition, you can raise an error (now called an exception) in one language and have it handled by a component in another language. This is significant because now developers can write in the language with which they are most comfortable and be assured that others writing in different languages will be able to easily use their components.

- **Assembly's manifestation:**

One of the new structures you will create in VB.NET is the assembly. An assembly is a collection of one or more physical files. The files are most often code, such as the classes you build, but they could also be images, resource files, and other binary files associated with the code. Such assemblies are known as static assemblies because you create them and store them on disk. Dynamic assemblies are created at runtime and are not normally stored to disk (although they can be). An assembly represents the unit of deployment, version control, reuse, and security. If this sounds like the DLLs you have been creating in Visual Basic for the past six years, it is similar. Just as a standard COM DLL has a type library, the assembly has a manifest that contains the metadata for the assembly, such as the classes, types, and references contained in the IL. The assembly often contains one or more classes, just like a COM DLL. In .NET, applications are

built using assemblies; assemblies are not applications in their own rights. Perhaps the most important point of assemblies is this: All runtime applications must be made up of one or more assemblies. The manifest is similar in theory to the type library in COM DLLs. A manifest contains:

- Assembly name
- Version
- Files in the assembly
- Referenced assemblies

- **The Common Type System:**

The Common Type System specifies the types supported by the CLR. The types specified by the CLR include:

- **Classes**—the definition of what will become an object; includes properties, methods, and events
- **Interfaces**—the definition of the functionality a class can implement, but does not contain any implementation code
- **Value Types**—User-defined data types that are passed by value
- **Delegates**—Similar to function pointers in C++, delegates are often used for event handling and callbacks

- **Code Access Security (CAS):**

This security does not control who can access the code; rather, it controls what the code itself can access. This is important because it allows you to build components that can be trusted to varying degrees. With .NET, however, you can actually specify, with the tools in the .NET Framework, what actions your component can and, more importantly, cannot perform. This has the benefit of preventing others from using the code in ways that you did not intend. Perhaps the main benefit of CAS is that you can now trust code that is downloaded from the Internet. Security can be set up so that it becomes impossible for the code to perform any mischievous actions. This would prevent most of the macro viruses that are spread via e-mail today. With .NET, however, you can actually specify, with the tools in the .NET Framework, what actions your component can and, more importantly, cannot perform. This has the benefit of preventing others from using the code in ways that you did not intend. Security can be set up so that it becomes impossible for the code to perform any mischievous actions. This would prevent most of the macro viruses that are spread via e-mail today.

- **Role-Based Security:**

  Role-based security is the same type of security you get when you use MTS or COM+ Component Services. In .NET, the Framework determines the caller, called a principal, and checks the principal's individual and group permissions. Unlike COM/COM+ role-based security, however, .NET cannot make an assumption that the user will have a valid NT user account and token to pass in. Therefore, .NET allows for generic and custom principals, as well as standard Windows principals. You can define new roles for each application if you want.

- **Self-Describing Components:**

  In traditional VB, compiled components created a type library that attempted to define what was in the component as far as classes, interfaces, properties, methods, and events. Communication occurred through a binary interface at the COM level. Unfortunately, one language could expect as parameters data types or structures that are not available to other languages, or that are at least difficult to implement. For example, C++ components often expect pointers or structures to be passed in, and this could be problematic if the calling program is written in Visual Basic. The .NET Framework attempts to solve this by compiling additional data into all assemblies. This additional data is called metadata and allows compiled components to interact seamlessly. Couple this with a common type system so that all runtime-compatible languages share the same types, and you can see that cross-language compatibility is enhanced. The metadata that is stored in the components is binary, and contains all types, members, and references in that file or assembly. The metadata is compiled into the PE file, but when the file is used at runtime, the metadata is moved into memory so that it can be accessed more quickly.

  Having a unified debugger, gaining free threading, and finally having full inheritance are major leaps forward. The runtime also gives you a nice security framework, and you don't have to worry about registering components anymore. Web Services allows you to easily create services that are consumable over standard HTTP connection. As you work with VB.NET, understand that you are also working closely with the .NET Framework. It is hard to separate the two, and the more you understand about the Framework, the better VB.NET developer you will be.

2.2.9 MS Access database:

The term database means different things to different people. For many years, in the world of x Base (dBase, FoxPro, CA-Clipper), database was used to describe a collection of fields and records. In a client/server environment, database refers to all the data, schema, indexes, rules, triggers, and stored procedures associated with a system. In Access terms, a database is a collection of all the tables, queries, forms, reports, macros, and modules that compose a complete system. Access databases are made up of tables, queries, forms, reports, macros, and modules. Each of these objects has its own special function. The Access environment also consists of several miscellaneous objects. These include relationships, toolbars, menus, database properties, and import/export specifications. Together, these objects enable you to create a powerful, user-friendly, integrated application.

Let's take a tour of the objects that make up an Access database Tables are the starting point for your application. Whether your data is stored in the Access format or you are referencing external data using linked tables, all the other objects in your database either directly or indirectly reference your tables. To view all the tables that are part of your database, click on the Table tab of the Database window. If you want to view the data contained within a table, double-click on the name of the table that you want to view (you can also click on the Open button). The data within the table is displayed in a datasheet. Notice that all the fields and records within the table are displayed. Many of the attributes of the datasheet can be modified. Furthermore, you can search for and filter data from within the datasheet.

As a developer, you will most often want to view the design of the table, which is the blueprint or template for the table. To view the design of a table, click Design with the table selected. The design of the table appears. Here you can view or modify all the field names, data types, and field and table properties. Access provides you with significant power and flexibility in customizing the design of your tables. "What Every Developer Needs to Know About Tables." To properly maintain the integrity of your data and facilitate the process of working with the other objects in the database, you must define relationships among the tables in your database. Together, these objects enable you to create a powerful, user-friendly, integrated application.. Whether your data is stored in the Access format or you are referencing external data using linked tables, all the other objects in your database either directly or indirectly reference your tables.

Processor	**Intel Pentium II**
RAM Capacity	**32 MB**
Hard Disk Capacity	**5 GB**
Keyboard	**Standard 104- keyboard**
Mouse	**2-button mouse**
Modem	**Internal/External 56Kbps Data-Fax-Voice modem**

4.1 Introduction:-

SRS is a document, which outlines the necessity and usage of an existing system. This document describes all the basic requirements for the existing system and also gives out a detailed process flow and also the controlled flow with in the system. By looking in to the SRS document, the project developer and other related personnel would be able to understand about the existing system.

4.1.1 Purpose

The purpose of this SRS is to describe all the external requirements of the system. It also describes the interfaces for the system.

4.1.2 Scope

The scope of this SRS is only to describe the requirements of the system. This is meant for the use by the developer, and will also be the basis for validating the final delivered output. The developer is responsible for asking for clarification, where necessary, and will not make any alterations without the permission of the client.

4.1 3 Definitions

- **Interactive Voice Recognition System (IVRS):**

  An Interactive Voice Recognition System is used to collect and provide information to a caller through an automated process. This allows the caller to be helped with minimal or no interaction with an actual human.

- **Interoperability:**

  Interoperability is defined as mixed language programming. Not only can you inherit classes from one PE written in language A and use them in language B, but debugging now works across components in multiple languages. This way, you can step through the code in a PE written in C# and jump to the base class that was not just the run time given to you by

COM Written in VB.NET. This means that your cross-language interoperability is happening at design time and run time.

- **Interface Inheritance:**

  Inheritance is the capability to define increasingly more specific objects, starting with the existing characteristic definitions of general objects. COM supports interface inheritance, the capability for one interface definition to inherit characteristics from another interface. COM, however, doesn't support implementation inheritance, the capability for one object to inherit the actual implementation code from another object.

4.1.4 Problem statement

The main objective of this project is automating the result announcement system. In this system a student may not even go to college to see his/her result. He/she can get to known his /her result through the telephone line itself, by just dialing to the respective college and entering the register number.

4.1.5 Developer's Responsibilities Overview

The developer is responsible for Developing the system Installing the software on the sender and the receiver system Before installation the developer should ensure that the systems are multicast enabled conducting the user training if needed.

4.2 Feasibility Assessment

The main objective of feasibility assessment is to test operational, technical and economical feasibility in developing the proposed system.

4.2.1 Operational Feasibility

The potential user of the system agreed that the methods being used here is time consuming. It enhances the accessibility of information.

4.2.2 Technical Feasibility

All hardware and the software required for the development of the system were already available. Consequently the technology is not a constraint to system development. Memory is not a constraint in developing the system.

The proposed system is developed using modular techniques.

- Further expansion poses no problem.
- Accuracy, ease of access, security and reliability is ensured.

4.2.3 Economic Feasibility

There is no cost incurred for the hardware and the software used for this project, as they are readily available. The proposed system is user friendly and is expected to cost effective. Training required is minimum. In view of the above facts the project was considered feasible and highly desirable.

4.3 General Description

This section describes the general factors that affect the product and its requirements document. Specific requirements are not mentioned, but a general overview is presented to make the understanding of the specific requirements easier.

4.3.1 Product Perspective

This system or the product is to be developed in such a way so that it supports a single caller. In other words the user gets engaged tone when another user is using IVRS.

4.3.2 Product Functions Overview

Interactive Voice Recognition System is an application project designed for a user to retrieve his/her result of the examination through telephone. The project comprises of three modules File playback module, Tone detection module and Text to speech module.

- File playback module is a necessary component in IVRS—it is used to play an outgoing message over a call. In an IVRS, this message usually contains a series of choices for the caller to choose from: "Press '1' for..." .In IVRS, file playback often involves playing several files sequentially over the call.
- Tone detection module involves Dual tone multifrequency tones (DTMFs) are more commonly known as the tones that are generated by pressing the buttons of a touch-tone phone. They have always been used as input to an IVRS.
- Text-to-speech module involves taking text and rendering it into speech

4.3.3 User Characteristics

The users should have good computer knowledge. The user should be a good listener and respond accordingly to the instructions played back.

4.3.4 General Constraints

This system should run on Windows 2000 platform system. The workstations should also contain VB Dot net frameworks.

4.3.5 Assumptions and Dependencies

Several assumptions have been made in building the system, which are as follows:

- The system runs on Windows 2000 platform.
- The package runs only when user calls.
- The system is connected to modem and telephone.
- The database had been given manually by the institution.
- Only single user can access the information at a time.
- There is no multiple user can access the information at the same time.

4.5 Functional Requirements

When a user calls, a pre-recorded message is played and the user should respond according to the inputs and outputs of which are described below.

Output 1:A welcome message from IVRS is played back The user is given options of selecting his choice of department "press 1 for CSE", "press 2 for ISE", "press 3 for ECE" , "press 4 for TE".

Input 1: The user enters his choice of department. If user press other than the option given to the user such as *,# keys then IVRS system plays the previously listen pre-recorded file once again.

Output 2: The user is asked to enter his semester, which exists from 1-8.The subjects varies for different semesters ...first seem consists of 7 subjects and that of eighth seem just 3 subjects.

Input 2: The user enters his choice of semester. If user press other than the option given to the user such as *,# keys then IVRS system plays the previously listen pre-recorded file once again.

Output 3: The user is asked to enter his register number which is of 5 digits. First two digits indicate the year and the next three indicate his corresponding register no which is assigned according to the alphabetical order of the names in a particular semester.

Input 3: The user enters his register no. If user press enter invalid 5 digits [Ex-09232] register number means IVRS system search the register number in the database if does not exist it

Plays back the error message saying that "The Register you entered doesn't exist"

Output 4: The result of that particular register no is fetched from the database, converted into speech and the result is announced. A thank you message for using IVRS is played back and the user hangs up.

4.6 External Interface Requirements

4.6.1 User Interfaces

The system should prompt the user for all inputs when necessary. A pre-recorded instruction file is provided which gives sufficient details how to use the application.

4.6.2 Hardware Interfaces

A Telephone line connected to the Internal/External Data-Fax-Voice modem

4.6.3 Software Interfaces

The VB .Net Framework is enough to run the application on windows 2000 platform. In this section, all the other software that is needed for this software to run is specified.

- Speech SDK
- Directx 8 or later

4.6.4 Acceptance Criteria

Before accepting the system the developer will have to demonstrate that the system works with one caller. The developer will have to show by suitable test cases that all conditions are satisfied.

5. System Design

System design here serves to bridge the gap between the requirements specified for this system and the final solution for satisfying the requirements. While the requirements specification activity is entirely in the problem domain, system design is the first step to moving from the problem domain towards the solution domain.

The goal of the design process is to produce a model or a representation of a system that can be used later to build that system. The produced model is called the design of the system. The design of a system is essentially a blueprint, or a plan for a solution for the system. Here we consider a system to be a set of components with clearly defined behavior, which interact with each other in a fixed, defined manner, to produce some behavior or services to its environment.

Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm that is used. Once the software requirements have been analyzed and specified, software design is the first of the three technical activities – Design, Code and Test that are required to build and verify software. Without a design we risk building an unstable system, on that will fail when small changes are made, one that is difficult to test, and one whose quality cannot be assessed until late in the process of software engineering.

5.1 Existing System

In earlier days students use to get there result information in their respective colleges once the technology has developed students started getting there results through websites. The main disadvantage in this is server breakdown, delayed result.

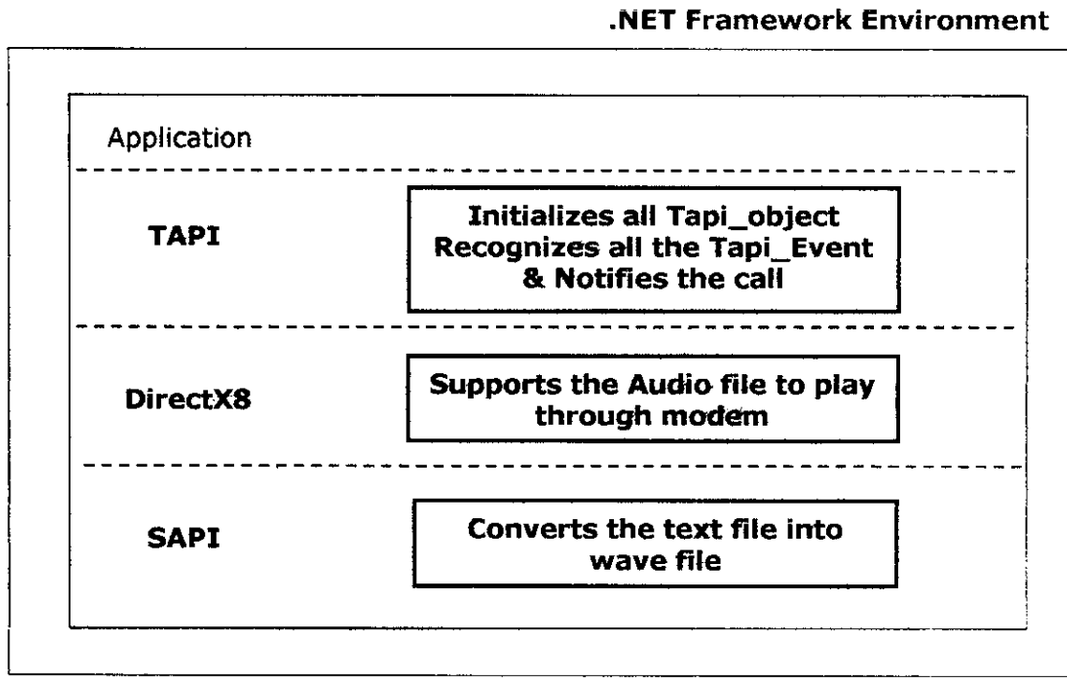
Colleges use to maintain a detail records of students and there result. All these are usually done using manual methods. Since the task involved in manual method, is time consuming and laborious.

5.2 Proposed System

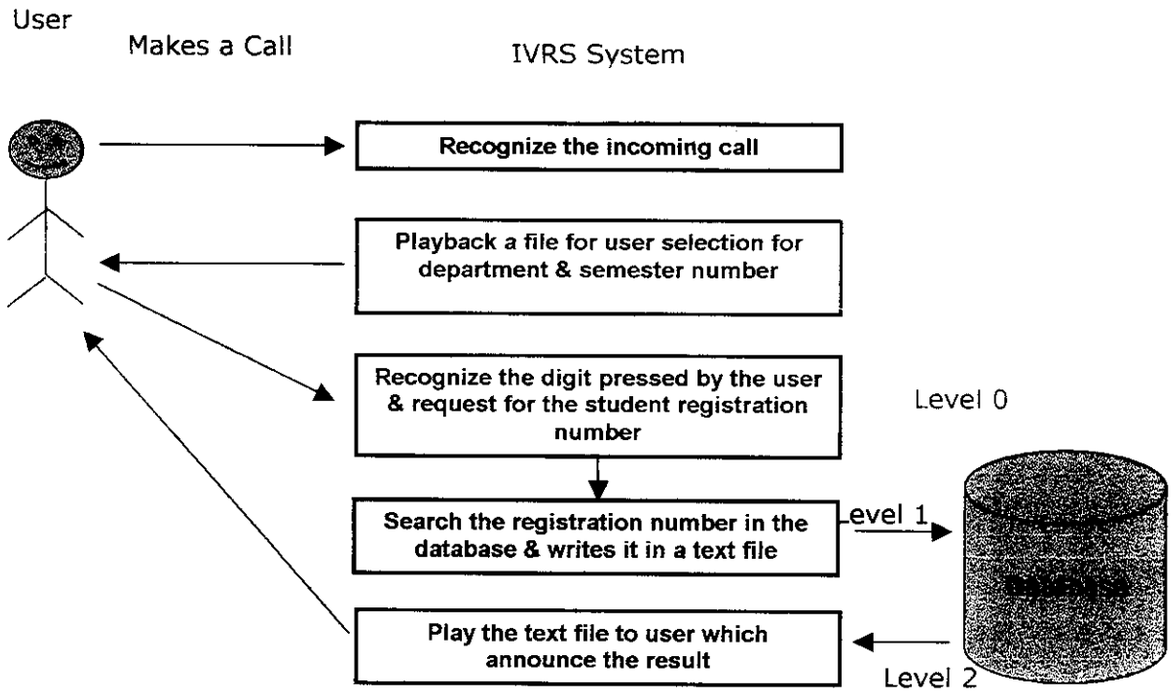
The current project is aimed at developing the IVRs base system for college in which the user need not go to college to know his/her marks instead he/she

will type his register number through the interface developed by .net application which is connected through a telephone line , with a voice enabled modem to the IVRS system . The IVRS system will verify the register number , process other authentication details , search its database and will give the required details using voice enabled modem, The response will be through voice.

5.3 Architecture of the System

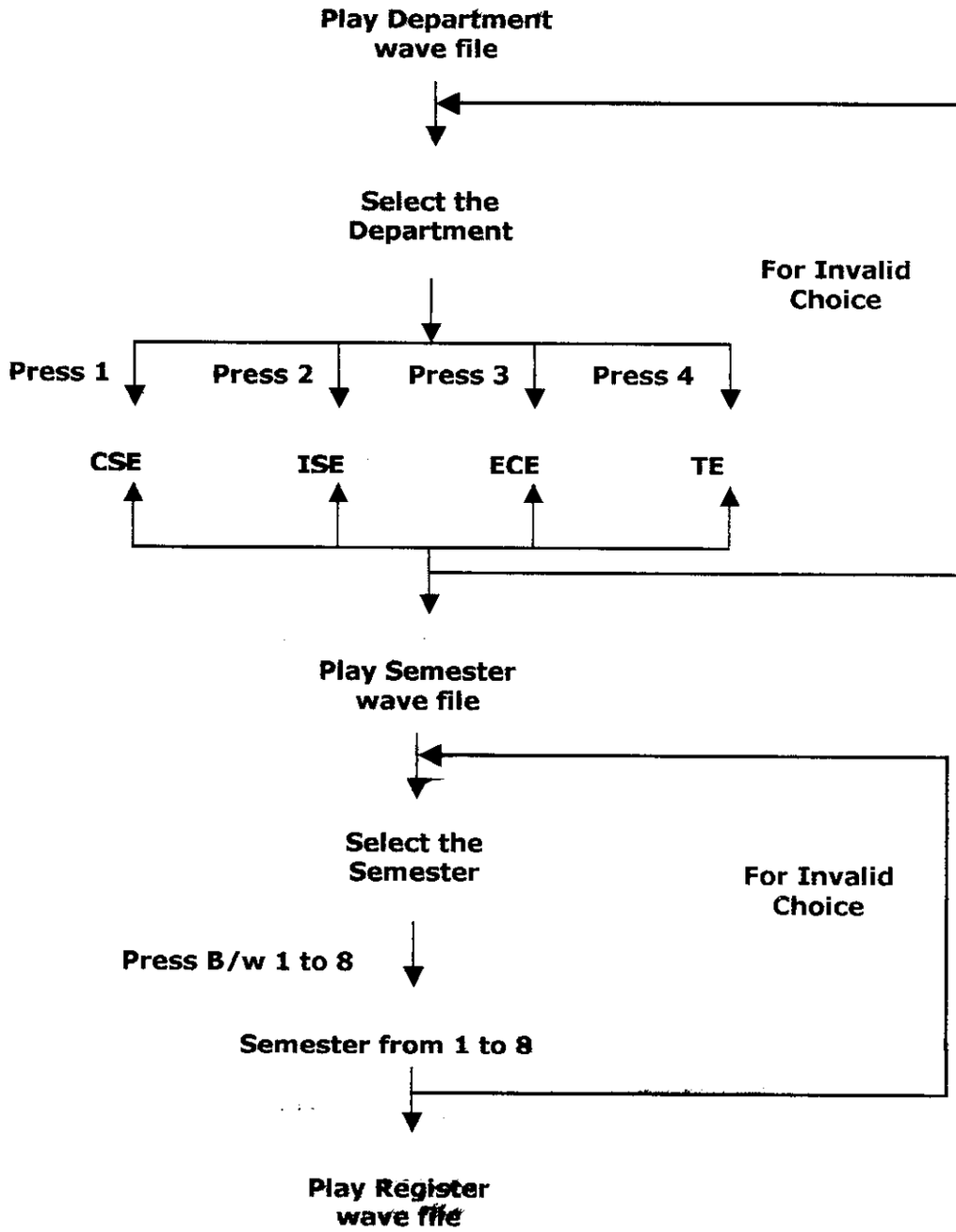


5.4 Data Flow Diagrams



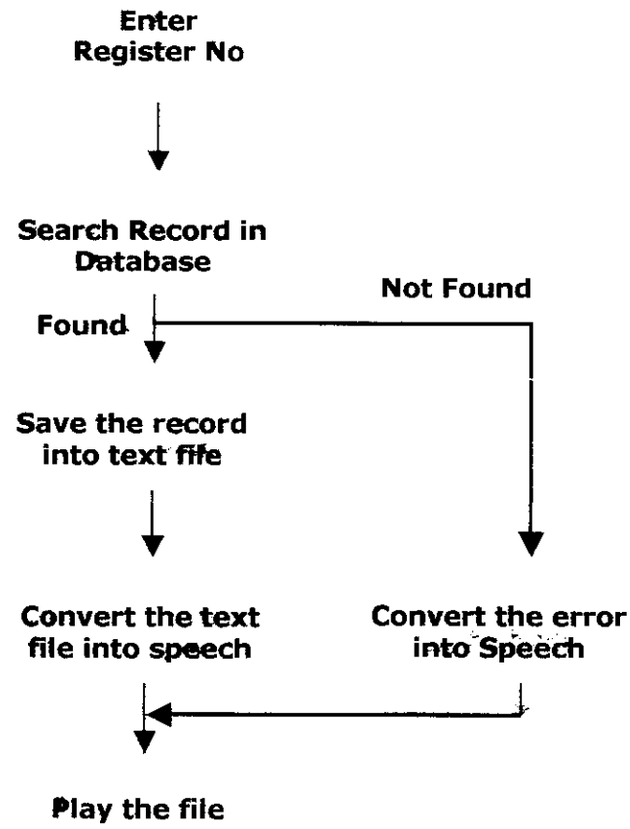
The above diagram shows the flow of information through the system and its process.

5.4.1 Detailed Description of Tone Detection & File Playback DFD:



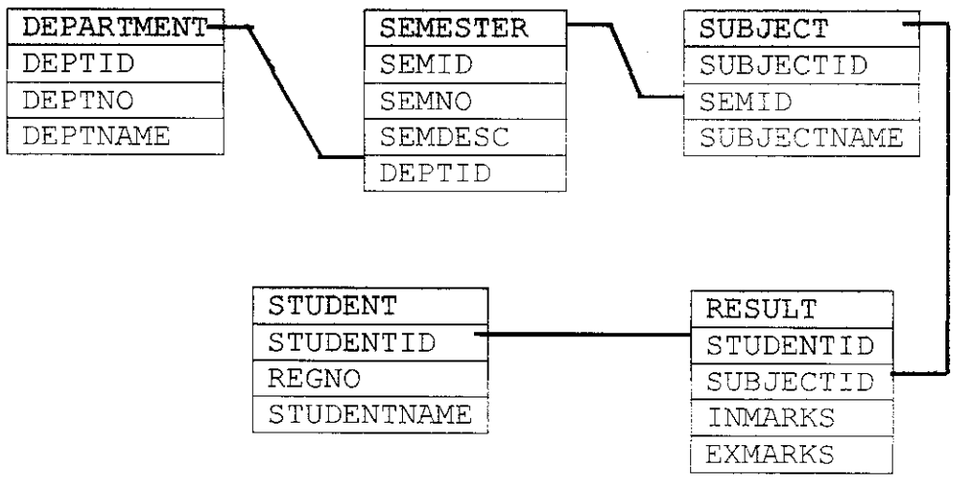
Level 1 Data Flow Diagram of File Playback:

5.4.2 Detailed Description of Text to speech DFD



Level 2 Data Flow Diagram of Text to speech

5.5 ER-Diagram:-



5.6 Design of Table:-

These sections below describe the schema for the tables used by the system.

DEPARTMENT		
DEPTID	DEPTNO	DEPTNAME

SUBJECT		
SUBJECTID	SEMID	SUBJETNAME

STUDENT		
STUDENTID	REGNO	STUDENTNAME

SEMESTER			
SEMID	SEMNO	SEMDESC	DEPTID

RESULT			
STUDENTID	SUBJECTID	INMARKS	EXMARKS

6. System Implementation

Implementation is the stage of the project when the theoretical design is turned into a working system. At this stage the main workload, the upheaval and the major impact on the existing practices shift to user department. Implementation is the phase where the system goes for actual functioning. Hence in this phase one has to be cautions because all the efforts undertaken during the project will be fruitful only if the software is properly implemented according to the plans made.

The project is mainly implemented using Microsoft VB .net on Windows 2000 platform operating system.

6.1 Detailed implementation of Interactive Voice Recognition System

An interactive voice recognition system (IVRS) application is a set of rules and commands. These instructions allow a voice-response engine to lead a telephone caller through a hierarchy of menus, collect voice and data input, and perform other operations on behalf of the caller or the program sponsor. The recording also provides information such as Result of the student in examination. This can save a lot of time and very informative for the students otherwise they have to spend a lot of time in browsing for their results. This application can also save from the point of view of expenses also.

A typical IVRS scenario is given below along with the use of TAPI.

IVRS System	TAPI 3.1 usage
The customer calls a toll-free number for support.	The IVRS system is notified of a call through TAPI.
The call is answered automatically, and the IVRS system plays a message indicating the student's options.	The TAPI file terminal is used to play the message.
The student chooses the option to get her result	The TAPI tone detection terminal is used to listen for the student's selection.
The IVRS system asks the user to input her account number.	The TAPI file playback terminal is used to play the prompt.
The student enters her account number.	The TAPI tone detection terminal is used to listen for the student's department, semester and register number.
The IVRS system looks up the database, and finds the result.	
The IVRS system uses TTS to provide the result to the student.	The TAPI TTS terminal is used to convert the result to speech.
The student hangs up.	

6.2 List of Tapi Object Used in IVRS Project:

- **ITTAPI**

  The ITTAPI interface is the base interface for the TAPI object. The TAPI object is created by goCreateInstance. All other TAPI 3 objects are created by TAPI 3 itself. ITTAPI methods are provided to initialize a TAPI session, enumerate available addresses, register for Call Hub and Call Event notifications, and shut down a TAPI session. The ITTAPI2 interface derives from the ITTAPI interface. It adds additional methods on the TAPI object to support phone devices.

- **IT Address:-**

  he IT Address interface is the base interface for the Address object. Applications use this interface to get information about and use the Address object.

- **ITCallInfo:-**

  The ITCallInfo interface gets and sets a variety of information concerning a Call object. The ITAddress::get_Calls and IEnumCall::Next methods create the ITCallInfo interface.

- **ITLegacyCallMediaControl:-**

  The ITLegacyCallMediaControl interface supports legacy applications that must communicate directly with a device. This interface is exposed on the Call Object and can be created by calling QueryInterface on ITBasicCallControl.

- **ITTerminal:-**

  The ITTerminal interface is the base interface for a Terminal object. This object, and the interface, is available only when an MSP exists. It provides methods that allow an application to obtain information such as terminal class and media supported. The following methods create the ITTerminal interface:

  ITBasicCallControl2::RequestTerminal

  ITTerminalSupport::CreateTerminal

  IEnumTerminal::Next

- **ITMediaSupport**

  The ITMediaSupport interface provides methods that allow an application to discover the media support capabilities for an Address Object that exposes this interface. A pointer to this interface can be obtained by calling QueryInterface using any address interface pointer, such as ITAddress

- **ITAddressCapabilities**

  The ITAddressCapabilities interface is used to obtain information about an address's capabilities. It is on the Address object, and an application can access it by calling QueryInterface on the Address object.

- **ITCollection**

The IT Collection interface allows Automation client applications, such as those written in Visual Basic, to retrieve collection information. C or C++ programs use enumerator interfaces to retrieve the same information. Collection methods return a VARIANT that contains a pointer to an IT Collection interface.

The ITCollection2 interface is an extension of the IT Collection interface. ITCollection2 exposes additional methods for modifying the collection.

- **ITTerminalSupport**

The ITTerminalSupport interface is exposed on an Address object only if an MSP exists. The methods of this interface allow an application to discover available terminals and/or create one, and get pointers to required Terminal objects.

An ITTerminalSupport pointer can be obtained by calling QueryInterface on any Address interface, such as ITAddress. If E_NOINTERFACE is returned, the service provider associated with the address does not support media controls.

The ITTerminalSupport2 interface is derived from the ITTerminalSupport interface. ITTerminalSupport2 supports the retrieval of information about pluggable terminal classes and super classes by C, C++, and scripting applications.

- **ITTAPI::UnregisterNotifications**

The UnregisterNotifications method removes any incoming call notification registrations that have been performed using ITTAPI::RegisterCallNotifications.

```
HRESULT UnregisterNotifications(
  long register
);
```

Parameter	Description
register	[in] The value returned by the RegisterCallNotifications method in the plRegister parameter.

- **ITTAPI::RegisterCallNotifications**

The RegisterCallNotifications method sets which new call notifications an application will receive. The application must call the method for each address, indicating media type or types it can handle, and specifying the privileges it requests. An application that will make only outgoing calls does not need to call this method. The ITTAPIEventNotification outgoing interface must be registered prior to calling this method. If both owner and monitor privileges are needed for an address, this method should be called only once, with both fMonitor and fOwner set to TRUE.

```
HRESULT RegisterCallNotifications(
  ITAddress *pAddress,
  VARIANT_BOOL fMonitor,
  VARIANT_BOOL fOwner,
  long lMediaTypes,
  long lCallbackInstance,
  long *plRegister
);
```

Parameter	Description
PAddress	[in] Pointer to ITAddress interface.
fMonitor	[in] Boolean value indicating whether the application will monitor calls. VARIANT_TRUE indicates that the application will monitor calls; VARIANT_FALSE that it will not.
fOwner	[in] Boolean value indicating whether the application will own incoming calls. VARIANT_TRUE indicates that the application will own incoming calls; VARIANT_FALSE indicates that it will not.
lMediaTypes	[in] Media types that can be handled by the application.
lCallbackInstance	[in] Callback instance to be used by the TAPI 3 DLL. Can be the gulAdvise value returned by IConnectionPoint::Advise during registration of the ITTAPIEventNotification outgoing interface.
plRegister	[out, retval] On success, the returned value that is used by ITTAPI::UnregisterNotifications.

- **ITCallNotificationEvent**

The ITCallNotificationEvent interface contains methods that retrieve the description of call notification events. When the application's implementation of the ITTAPIEventNotification::Event method indicates a TAPI_EVENT equal to TE_CALLNOTIFICATION, the method's pEvent parameter is an IDispatch pointer for the ITCallNotificationEvent interface. The methods of this interface can be used to retrieve information concerning

the call notification event that has occurred. This outgoing interface is registered with the TAPI object to get all information about calls. An application must call the ITTAPI::RegisterCallNotifications method on the TAPI object before registering this interface.

Note:- You must call the ITTAPI::put_EventFilter method and set an event filter mask that includes the TE_CALLNOTIFICATION event to enable reception of call notification events. If you do not call ITTAPI::put_EventFilter, your application will not receive any events

- **ITBasicCallControl**

The ITBasicCallControl interface is used by the application to connect, answer, and perform basic telephony operations on a call object. The ITBasicCallControl2 interface is an extension of the ITBasicCallControl interface. ITBasicCallControl2 supplies additional methods that allow an application to select a terminal onto a call. The ITAddress::CreateCall method creates the ITBasicCallControl interface.

- **ITCallStateEvent**

The ITCallStateEvent interface contains methods that retrieve the description of call state events. When the application's implementation of the ITTAPIEventNotification::Event method indicates a TAPI_EVENT equal to TE_CALLSTATE, the method's pEvent parameter is an IDispatch pointer for the ITCallStateEvent interface. The methods of this interface can be used to retrieve information concerning the change that has occurred in the call state.

Note:- You must call the ITTAPI::put_EventFilter method and set an event filter mask that includes the TE_CALLSTATE event to enable reception of call state events. If you do not call ITTAPI::put_EventFilter, your application will not receive any events.

- **ITDigitDetectionEvent**

The ITDigitDetectionEvent interface contains methods that retrieve the description of DTMF digit events. When the application's implementation of the ITTAPIEventNotification::Event method indicates a TAPI_EVENT equal to TE_DIGITEVENT, the method's pEvent parameter is an IDispatch pointer for the ITDigitDetectionEvent interface. The methods of this

interface can be used to detect DTMF digits during a call. This interface is implemented by the application and called by the TAPI 3 DLL.

- **ITAddressEvent**

  The ITAddressEvent interface contains methods that retrieve the description of address events. When the application's implementation of the ITTAPIEventNotification::Event method indicates a TAPI_EVENT equal to TE_ADDRESS, the method's pEvent parameter is an IDispatch pointer for the ITAddressEvent interface. The methods of this interface can be used to retrieve information concerning the type of event, which address the event has occurred on, and for which terminal.

- **Tapi_Event**

  The TAPI_EVENT enum is used to notify an application that a change has occurred in the TAPI object. The ITTAPIEventNotification::Event method implementation uses members of this enum to indicate the type of object associated with the IDispatch pointer passed by TAPI.

```
typedef enum TAPI_EVENT
{
  TE_TAPIOBJECT,
  TE_ADDRESS,
  TE_CALLNOTIFICATION,
  TE_CALLSTATE,
  TE_CALLMEDIA,
  TE_CALLHUB,
  TE_CALLINFOCHANGE,
  TE_PRIVATE,
  TE_REQUEST,
  TE_AGENT,
  TE_AGENTSESSION,
  TE_QOSEVENT,
  TE_AGENTHANDLER,
  TE_ACDGROUP,
  TE_QUEUE,
  TE_DIGITEVENT,
```

```
    TE_GENERATEEVENT,
    TE_ASRTERMINAL,     // Windows XP and above
    TE_TTSTERMINAL,     // Windows XP and above
    TE_FILETERMINAL,    // Windows XP and above
    TE_TONETERMINAL,    // Windows XP and above
    TE_PHONEEVENT,      // Windows XP and above
    TE_TONEEVENT,       // Windows XP and above
    TE_GATHERDIGITS     // Windows XP and above
} TAPI_EVENT;
```

Members	Description
TE_TAPIOBJECT	Change is in TAPI object itself.
TE_ADDRESS.	An Address object has changed.
TE_CALLNOTIFICATION	A new communications session has appeared on the address and the TAPI DLL has created a new call object. This could be a result from an incoming session, a session being handed off by another application, or a session being parked on the address. See ITCallNotificationEvent and ITTAPI::RegisterCallNotifications
TE_CALLSTATE	The Call state has changed. See ITCallStateEvent
TE_CALLMEDIA	The media associated with a call has changed. See ITCallMediaEvent.
TE_CALLHUB	A CallHub object has changed.
TE_CALLINFOCHANGE	The call information has changed.
TE_PRIVATE	A provider-specific private object has changed. The precise type of object referenced is implementation dependent for general information and the service provider's documentation for the specific meaning.
TE_REQUESTA	Request object has changed.
TE_AGENT	An Agent object has changed.
TE_AGENTSESSION	An AgentSession object has changed.
TE_QOSEVENT	A QOS event has occurred.
TE_AGENTHANDLER	An AgentHandler object has changed.
TE_ACDGROUP	An ACDGroup object has changed.
TE_QUEUE	A Queue object has changed.
TE_DIGITEVENT	A digit event has occurred.
TE_GENERATEEVENT	A digit generation event has occurred.
TE_TTSTERMINAL	An event has occurred on a TTS terminal. Valid only for Windows XP and above.
TE_FILETERMINAL	An event has occurred on a file terminal. Valid only for Windows XP and above.
TE_TONETERMINAL	An event has occurred on a tone terminal. . Valid only for Windows XP and above.
TE_PHONEEVENT	A Phone object has changed. Valid only for Windows XP and above.
TE_TONEEVENT	A tone event has been fired. Detection of in-band tones will be enabled or disabled. Valid only for Windows XP and above.
TE_GATHERDIGITS	A gather digits event has been fired. Digits will be gathered on the current call.

6.3 List of Function used in IVRS System is as follows

Function Name	Description
Answer ()	This function is used to answer the call and setup for DTMF detection events.
Cmddevice ()	This function is used to verify that the device meets our criteria, such as supports 'TAPIMEDIATYPE_AUDIO
Disconnect ()	This function is used to disconnect a call. Note that we sometimes try to disconnect a call that we haven't answered yet (a call comes in but is dropped before the app has answered. Other than tracking the state of whether or not we actually answered the call we just attempt that last in this sub)
GetDeviceByName(ByRef Card_name As String, ByRef objCollAddresses As TAPI3Lib.ITCollection)	This function is used to get an index to that device from the collection of ITAddress's by passing the value to the parameter Card_name
GetLegacyWaveIDAsDXDeviceGUID(ByRef line As Short, ByRef strWave As String)	This function calls GetDeviceId which returns the legacy media id. We then enumerate the render or capture devices exposed by DirectSound and look for a match. If a match is found we return the DirectSound device GUID string else we return vbNullString
GetResult(ByVal m_RegNum As String)	This Function is used to retrieve the record from access database to announce the result to the user.
gobjTapiWithEvents_Event(ByVal TapiEvent As TAPI3Lib.TAPI_EVENT, ByVal pEvent As Object)	This is used to handle the Event which get fired automatically as soon as the incoming call comes.
InitTAPI()	This function is used to initialize the TAPI object to the application.
PlayFile(ByVal strFile As String)	This function is used to plays a given file name specified by the parameter strFile
RegisterForNotification()	This function is used to register the token generated for application to the system.
TTS(ByVal pResult As String)	This function is used to convert the string into speech format passed by the parameter pResult

6.4 Modules

When we analyze IVRS the basic tasks break down into the following areas:

- File playback
- Tone detection
- Text to speech

6.4.1 File Playback:

File playback is a necessary component in IVRS—it is used to play an outgoing message over a call. In an IVRS, this message usually contains a series of choices for the caller to choose from: "Press '1' for...".In IVRS, file playback often involves playing several files sequentially over the call. For example, in our project they are often used to inform callers of their current result of examination. The result can be generated by TTS, as described below, or can be generated by concatenating several prerecorded files. The application can easily determine which files need to be played to inform the user of their result. The TAPI File Playback terminal will make file playback in a TAPI application easy. The terminal will allow the application to specify a file, set of files, or an IStream COM object as an input. The terminal will automatically handle opening the file, reading the contents, and sending the data into the outgoing media stream.

List of object are used in these modules are as follows:

- **DirectX8.DirectSoundCreate :-**

  Creates a DirectSound8 object.

```
object.DirectSoundCreate( guid As String ) As DirectSound8
```

Parameter	Description
object	Object expression that resolves to a DirectX8 object.
guid	A GUID string that identifies the sound capture device. The value of this parameter must be one of the GUIDs returned by DirectSoundEnum8.GetGuid, vbNullString for the default device, or one of the following values: DSDEVID_DEFAULTPLAYBACK System-wide default audio playback device. DSDEVID_DEFAULTVOICEPLAYBACK Default voice playback device.

6.4.2 Tone Detection:

IVRS have always been based on tone detection. Dual tone multifrequency tones (DTMFs) are more commonly known as the tones that are generated by pressing the buttons of a touch-tone phone. They have always been used as input to an IVRS, although ASR is slowing becoming more common for user input as it becomes more reliable and more accessible to applications.

Accurately detecting DTMFs in a media stream is a very processor-intensive activity, and is almost always handled off the host computer, usually on a digital signal processor (DSP). When detected, the hardware signals an event to the host computer indicating the DTMF that was detected.

In addition to DTMFs, some systems need to detect other well-known tones, such as dial tone or busy tone, as well as generic tones that can be described as a collection of frequencies, volumes, durations, and cadences. Again, tone detection is not normally handled on the host computer, and not all systems can handle detecting tones other then DTMFs. Like ASR, tone detection is event driven—that is, when a tone is detected, the terminal generates an event, and the application processes the event and bases its actions on the event.

6.4.3 Text To Speech:

Text-to-speech (TTS) is conceptually the opposite of ASR, taking text and rendering it into speech. TAPI's TTS terminal will be based on SAPI 5.0 interfaces in the same way that the ASR terminal's are. Since the speech generated by TTS engines does not sound as good as prerecorded speech, TTS is usually in scenarios where a prerecorded message cannot be used. An example of this would be when a user can call into a UM system and have their e-mail read to them. The TTS terminal will have a simple interface with two main methods—loading the string of text to render into speech, and choosing the voice that the speech will be generated in. When a TTS terminal is selected, the application developer simply provides the string to speak, and the terminal handles sending the generated speech out the media stream.

The Method and Object used in these modules are as follows:-

- **SpeechVoiceSpeakFlags**

  The SpeechVoiceSpeakFlags enumeration lists flags that control the **SpVoice.Speak** method.

Elements	Description
SVSFDefault	Specifies that the default settings should be used. The defaults are: To speak the given text string synchronously (override with SVSFlagsAsync), Not to purge pending speak requests (override with SVSFPurgeBeforeSpeak), To parse the text as XML only if the first character is a left-angle-bracket (override with SVSFIsXML or SVSFIsNotXML), Not to persist global XML state changes across speak calls (override with SVSFPersistXML), and Not to expand punctuation characters into words (override with SVSFNLPSpeakPunc).
SVSFlagsAsync	Specifies that the Speak call should be asynchronous. That is, it will return immediately after the speak request is queued.
SVSFPurgeBeforeSpeak	Purges all pending speak requests prior to this speak call.
SVSFIsFilename	The string passed to the Speak method is a file name rather than text. As a result, the string itself is not spoken but rather the file the path that points to is spoken.
SVSFIsXML	The input text will be parsed for XML markup.
SVSFIsNotXML	The input text will not be parsed for XML markup.
SVSFPersistXML	Global state changes in the XML markup will persist across speak calls.
SVSFNLPSpeakPunc	Punctuation characters should be expanded into words (e.g. "This is it." would become "This is it period").
SVSFNLPMask	Flags handled by SAPI (as opposed to the text-to-speech engine) are set in this mask.
SVSFVoiceMask	This mask has every flag bit set.
SVSFUnusedFlags	This mask has every unused bit set.

- **SpeechStreamFileMode Enum**

  The SpeechStreamFileMode enumeration lists the access modes of a file stream.Used by SpFileStream.Open.

  Definition

```
Enum SpeechStreamFileMode
    SSFMOpenForRead = 0
    [hidden] SSFMOpenReadWrite = 1
    [hidden] SSFMCreate = 2
    SSFMCreateForWrite = 3
End Enum
```

Elements	Description
SSFMOpenForRead	Opens an existing file as read-only.
SSFMOpenReadWrite	[hidden] Opens an existing file as read-write. Not supported for wav files.
SSFMCreate	[hidden] Opens an existing file as read-write. Else, it creates the file then opens it as read-write. Not supported for wav files.
SSFMCreateForWrite	Creates file even if file exists and so destroys or overwrites the existing file.

- **SpVoice**

The SpVoice object brings the text-to-speech (TTS) engine capabilities to applications using SAPI automation. An application can create numerous SpVoice objects, each independent of and capable of interacting with the others. An SpVoice object, usually referred to simply as a voice, is created with default property settings so that it is ready to speak immediately. The SpVoice automation object has the following elements:

Properties	Description
AlertBoundary	Gets and sets the alert boundary, which specifies how a speaking voice pauses itself for alerts.
AllowAudio-OutputFormat-ChangesOnNextSet	Gets and sets the flag that specifies whether the voice is allowed to adjust its audio output format automatically.
AudioOutput	Gets and sets the current audio output object used by the voice.
AudioOutputStream	Gets and sets the current audio stream object used by the voice.
EventInterests	Gets and sets the types of events received by the voice.
Priority	Gets and sets the priority level of the voice.
Rate	Gets and sets the speaking rate of the voice.
Status	Returns the current speaking and event status of the voice in an ISpeechVoiceStatus object.
Synchronous SpeakTimeout	Gets and sets the interval, in milliseconds, after which the voice's synchronous Speak and SpeakStream calls will time out when its output device is unavailable.
Voice	Gets and sets the currently active member of the Voices collection.
Volume	Gets and sets the base volume (loudness) level of the voice.
DisplayUI	Initiates the display of the specified UI.
GetAudioOutputs	Returns a selection of available audio output tokens.
GetVoices	Returns a selection of voices available to the voice.
IsUISupported	Determines if the specified UI is supported.
Pause	Pauses the voice at the nearest alert boundary and closes the output device, allowing it to be used by other voices.

Resume	Causes the voice to resume speaking when paused.
Skip	Causes the voice to skip forward or backward by the specified number of items within the current input text stream.
Speak	Initiates the speaking of a text string, text file or wave file by the voice.
SpeakCompleteEvent	Gets an event handle from the voice that will be signaled when the voice finishes speaking.
SpeakStream	Initiates the speaking of a text stream or sound file by the voice.
WaitUntilDone	Blocks the caller until either the voice has finished speaking or the specified time interval has elapsed.

- `SpFileStream`

The SpFileStream automation object enables data streams to be read and written as files.

SpFileStream objects normally contain audio data, but may also be used for text data.

The Format property and the Read, Write and Seek methods are inherited from the ISpeechBaseStream interface.

Automation Interface Elements

The SpFileStream automation object has the following elements:

Properties	Description
Format Property	Gets and sets the cached wave format of the stream as an SpAudioFormat object.
Close Method	Closes the filestream object.
Open Method	Opens a filestream object for reading or writing.
Read Method	Reads data from an audio stream.
Seek Method	Returns the current read position of the audio stream in bytes.
Write Method	Writes data to the audio stream.

7. Snap Shots

IVRS Application consists of following option

- **Ring to Answer**

  This is used to set the number of rings to lift/answer the incoming user call. By default it is set to 1 ring

- **Wave file location**

  This is used to give the path to store the pre-recorded message wave files in your system.

- **Log files location**

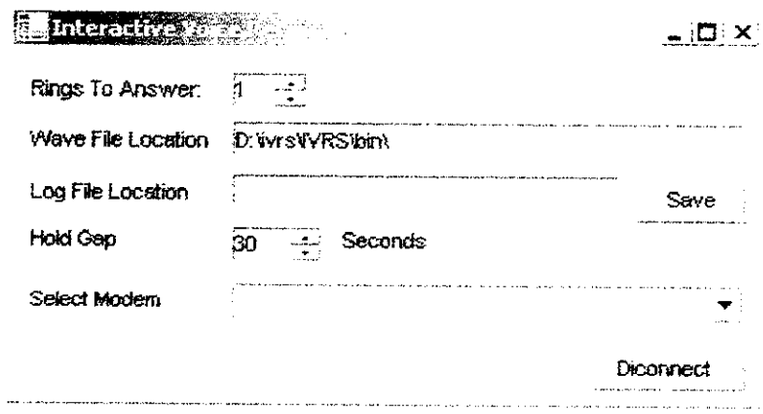
  This is used to give the path to store the logs file which contains all the events triggered by the IVRS application. So for later use we can refer any error occurred in the application.

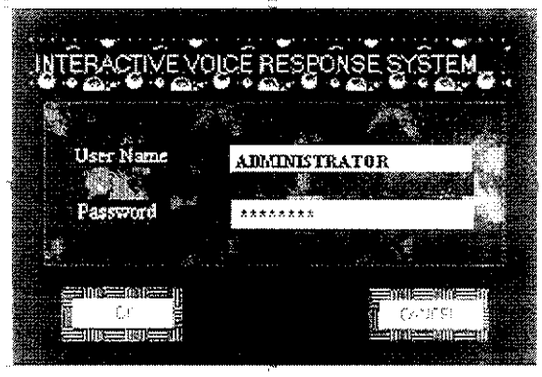
- **Hold gap**

  This is used to set the timer for disconnection of the user call. By default it is set to 30 seconds. If user doesn't respond with in this gap time the application disconnects the call automatically with out user knowledge.

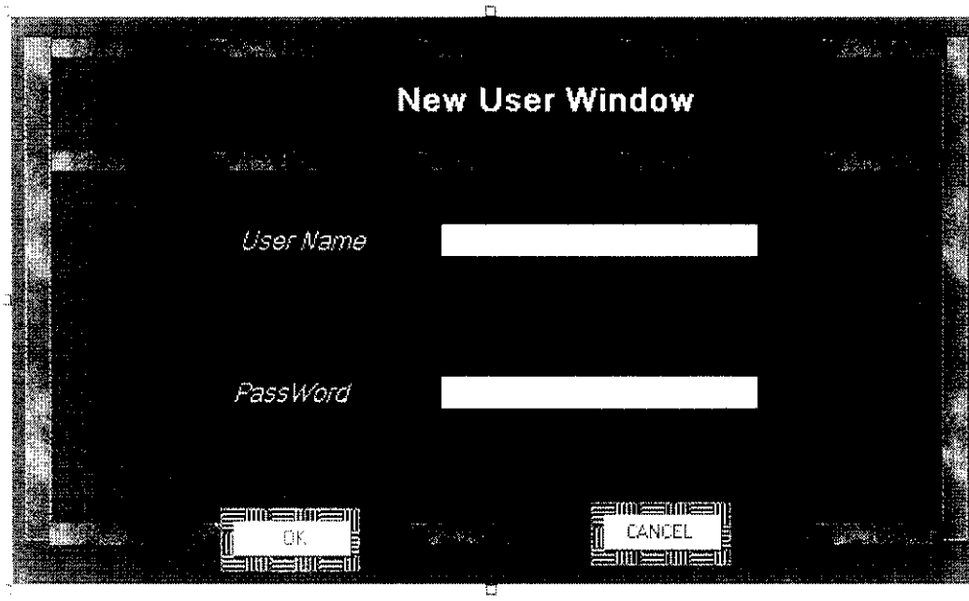
- **Select modem**

  This is used to select the modem present in your system which supports Data, Fax & Voice

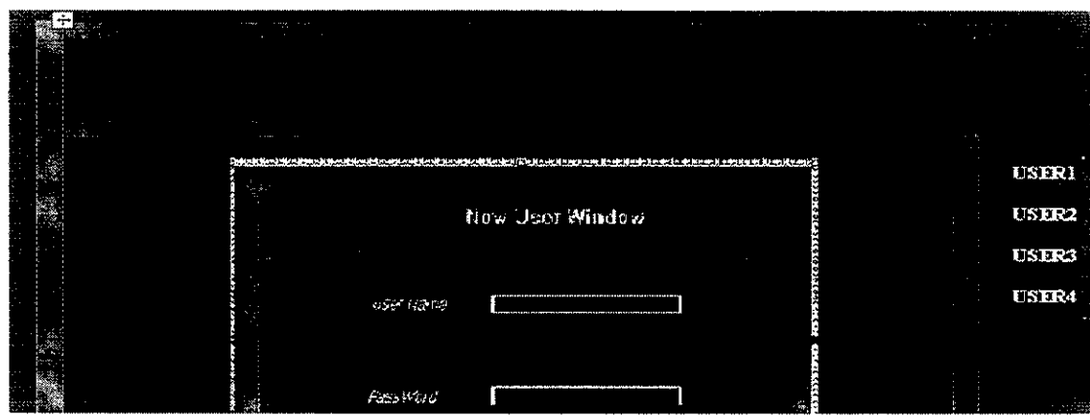




ADMINSTRATOR WINDOW



NEW USER WINDOW



USER LOGIN DETAIL WINDOW

8. Testing

8.1 Importance of testing

Testing is one of the most important stages in software development. It is the stage where it can be proved whether the implementation and the requirements are in fidelity with each other. Testing can also confirm if the software specifications are complete and consistent. One of the main goals of testing is to have a minimum number of test cases that will find a majority of the implementation errors.

8.2 Methodology

There are mainly three kinds of different test that can be performed. These are

- Conformance tests
- Unit testing
- Module Testing
- Integration Testing
- Acceptance Testing

Conformance tests:

They are used to verify whether the tested object is compliant to the standard. Conformance was verified through extensive testing of the implementation. The implementation conforms to the specification as it behaves in the same manner predicted .

Unit testing

Unit testing verifying the smallest units of software design of the module. All validations and conditions are tested in the module level. Control paths are tested to ensure the information properly flows into and out of the program.

In our system, the modules of the system are tested separately. Unit testing has been successfully handled .The data was given to each module in all respects and have been found working properly.

Module Testing

Module testing is the systematic way for constructing the module structure, while conducting the test to recover errors occurred in the components of the same module. A module is a collection of dependent components such as an object class,

an abstract data type or some looser collection of procedures and functions. Each module is tested for the errors and has been corrected.

Integration Testing

Integration testing is a systematic technique for constructing the program structure, while conducting test to recover errors associated with interfacing. Global data structures can present problems in testing when all modules are combined and entire program is tested as a whole.

In this case all the modules were combined and given the test data. The combined modules work successfully with out any side effect on other programs and everything was found working fine.

Acceptance Testing

User acceptance of the system is the key factor for the success of any live data and checks whether software is giving specified outputs. The system under consideration is tested for user acceptance.

8.3 Test Cases:

Test case involves the testing of notification of the incoming calls, recognition of digits and state of the call.

8.3.1 Test Case for File Playback

Input	Output
User is given the option of choosing the department:"press 1 for CSE","press 2 for ISE", "press 3 for ECE" and "press 4 for TE".	If the user chooses only the given option then it will play the semester wave file to select the semester otherwise department file will be played again.
User is given option of entering the semester 1-8.	If user presses his choice of semester i.e. 1-8 then it will play the register wave file to enter the 5 digit register number otherwise semester file will be played again.
User is given the option of entering his register no which is of 5 digits.	If user enter valid register no then result will be announced by text-to-speech module otherwise an error message will be played

Note: If user doesn't response for 30 seconds then the line will be automatically disconnected and waits for the next caller.

8.3.2 Test Case for text to speech:

Input	Output
The user enters his department, semester and register no	The result of that particular department, semester and register no is found then it is fetched from the database & it is stored in text format. The result that is in text format is converted into speech by using SAPI. The result is then announced to the user as the marks obtained in each subject and the total got otherwise an error message is stored in the text format as "Your register number does not exist" and played to the user.

9. Conclusion

The main aim of this project was to analyze, design and implement application package "Interactive Voice Retrieval System" that enables user to retrieve results of examination through telephone. This system has been successfully implemented and the project aims have been met.

The final state of the project is as follows:

- All the modules File playback, Tone detection and Text to Speech have been implemented and are fully functional.
- A complete testing procedure has been conducted, which revealed some errors. Those errors have been corrected.
- All the compulsory requirements have been completed successfully.
- The design and implementation of this application was a very challenging project, as there was no experience in the field of TAPI and VB .NET Programming. This project has been a great learning experience. The experience and skills acquired during this project will prove to be very useful in my future career.

10. Future Enhancements

Although the compulsory requirements of this project were implemented, there is still some more functionality that can be added. These could have been done if there was more time available.

Some extra features that can be added are as follows:

- Unified messaging and Speech enabled web applications could has been added
- Tone generation could have been used to indicate a particular event was finished. When a particular event has been triggered a tone is generated.
- Playback recording could have been used to record details of students that could be used for a later reference.

11. BIBLIOGRAPHY

Books

- **Roger S.Pressman,**"Software Engineering-A Practioners Approach", McGraw Hill International Editions, Fouth Edition, 1990.
- **Elias M.Award,**"System Analysis and Design", Galgotia Publications, Second Edition, 1995.
- **Lee,**"Introducing System Analysis and Design",Galgotia Book Source,1980.
- **Fred Barewell, Richard Blair, Richard Case.**"Professional VB.NET ", Wrox Publication, second Edition, 2004

Web sites

www.programmersheaven.com

www.askme.com

www.microsoft.com

www.apexsc.com