ABSTRACT

The range of modern computing is continually being extended as a result of accelerated developments in processing power, the roll out of high speed Internet and inter-networking. Many computer applications today rely heavily on the use of the Internet which in turn relies on existing and new telecommunications infrastructure to deliver services, thereby giving rise to remote computing. Remote computing, however, continues to be heavily dependent on textual input despite the existence and improvement in speech technology.

Remote Voice Computer Command and Control is an application that relies on improvements in speech recognition engines, coupled with existing telecommunications infrastructure to provide the ability to connect to a computer and execute voice commands. This is intended to present an alternative, a backup mechanism to text based remote computing as we transition to full convergence network.
# TABLE OF CONTENTS

ABSTRACT............................................................................................................................... ii

TABLE OF CONTENTS.................................................................................................................. iii

LIST OF FIGURES....................................................................................................................... v

1. INTRODUCTION AND BACKGROUND--------------------------------------------------1
   1.1 Brief History.................................................................................................................. 1
   1.2 Input Devices................................................................................................................ 4
   1.3 Connectivity and Network Access............................................................................ 5
   1.4 Putting it all together................................................................................................. 7

2. REMOTE VOICE COMMAND AND CONTROL-----------------------------------------9
   2.1 The Goal.................................................................................................................... 9
   2.2 Initial System Setup and Checks............................................................................. 10
   2.3 Training the ASR Engine.......................................................................................... 12
   2.4 Using Vista's Speech Recognition options for Training.......................................... 13
   2.5 Testing the Command Set......................................................................................... 13

3. SYSTEM DESIGN-----------------------------------------------------------------------------15
   3.1 Base Development Language Selection...................................................................... 15
   3.2 Automatic Speech Recognition Engine....................................................................... 16
   3.3 Modem Selection........................................................................................................ 19
      3.3.1 The Interface Card.............................................................................................. 20

4. EVALUATION AND RESULTS------------------------------------------------------------25
   4.1 Modem Testing........................................................................................................... 25
   4.2 ASR Testing............................................................................................................... 27
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Milestones in Speech Technology Research</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>Modem Blaster V.92 PCI Voice Modem</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>Logitech Headset with Microphone</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>PC Ports</td>
<td>11</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Possible TSP devices</td>
<td>21</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Selection of Broadxent Modem</td>
<td>22</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Registration of Modem</td>
<td>22</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Incoming call notification</td>
<td>23</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Call answered</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>Registered TSP device</td>
<td>26</td>
</tr>
<tr>
<td>4.2</td>
<td>Incoming call notification</td>
<td>26</td>
</tr>
<tr>
<td>4.3</td>
<td>Call connected</td>
<td>27</td>
</tr>
<tr>
<td>4.4</td>
<td>Remote recognition ‘ping localhost’</td>
<td>29</td>
</tr>
<tr>
<td>A.1</td>
<td>Windows Start menu</td>
<td>35</td>
</tr>
<tr>
<td>A.2</td>
<td>Configure your Speech Recognition experience window</td>
<td>36</td>
</tr>
<tr>
<td>B.1</td>
<td>Control panel tab in Windows Vista</td>
<td>37</td>
</tr>
<tr>
<td>B.2</td>
<td>Phone and modem options in classic view</td>
<td>37</td>
</tr>
<tr>
<td>B.3</td>
<td>Modem tab with Modem selected</td>
<td>38</td>
</tr>
<tr>
<td>B.4</td>
<td>Diagnostics and query tab</td>
<td>38</td>
</tr>
<tr>
<td>B.5</td>
<td>Output of query, notice AT+FCLASS=? 0, 1</td>
<td>39</td>
</tr>
<tr>
<td>C.1</td>
<td>Control panel in Windows Vista</td>
<td>40</td>
</tr>
</tbody>
</table>
Figure C.2 Phone and Modem options-----------------------------------------------41
Figure C.3 Telephony service providers-------------------------------------------41
Figure D.1 Terminal emulation software connection------------------------------43
Figure D.2 Available ports with Com3 selected---------------------------------43
Figure D.3 Port properties-------------------------------------------------------44
Figure D.4 Output of executing AT Command----------------------------------------44
1. INTRODUCTION AND BACKGROUND

1.1 Brief History

The beginnings of Automatic Speech Recognition (ASR) appear to be shrouded in confusion. Some literature dates it back to the late 1930s with Homer Dudley’s proposed system model for speech analysis and synthesis, others date it to the 1940’s, and there are others going as far back as the time of Alexander Graham Bell. What is clear is that the earliest attempts to devise systems for ASR were made in the 1950’s within the laboratories of large research and development firms. Most of the research was funded by the National Security Agency (NSA), National Science Foundation (NSF) and the Defense Advanced Research Projects Agency (DARPA) [Global Security, 2006].

Early technologies used to produce ASR software were very expensive, had limited vocabulary with low accuracy. The associated computer chip would recognize limited vocabulary of about 15 words, but required high computing power for processing. This made the price tag for such systems hefty, i.e. thousands of dollars for software recognizing just fewer than 50,000 words [Juang & Rabiner, 2004].

The business sector capitalized on the relevance and applications of ASR in automation. One such area is “Call Centers” for processing transactions and automating call handling functions i.e., routing in-coming calls. The introduction of ASR in “Call Centers” reduced the need for large numbers of call agents required to handle these functions. As a practical example, AT&T’s Voice Recognition Call Processing (VRCP) service, introduced in 1992, handled about 1.2 billion voice transactions per year back then [Juang & Rabiner, 2004].
The medical field is another area which has seen the relevance of ASR in relation to medical transcription; where ASR is used by medical transcriptionists to listen to dictated recordings by physicians and transcribe them into reports and other administrative materials. ASR is poised to provide a critical link as the industry (health care) moves towards implementing fully computerized medical records, which provides a direct capture of physician’s notes. Electronic documentation will enable easy, fast access and sharing of records to ensure optimal patient outcome [Terheyden, 2005].

Similar transcription application is found in the legal systems, where court reporters use ASR enabled devices to record court proceedings and create verbatim transcripts of such proceedings when written accounts are necessary for correspondence. Just like in medical transcription, direct dictation reduces the time on a given case and makes electronic records readily available for review. Accuracy rate of more than 95% is expected in such applications.

The military industrial complex is making use of ASR in high performance fighter jet control systems; weapons release systems and training of air traffic controllers [Wikipedia, 2008]. Presently ASR is being used in portable devices (laptops, phones, etc) in battle fields to help army personnel communicate with the locals in operating theaters around the world. One of such devices is the “Phraselator” (hand held language translation device) capable of translating English phrases into over 30 foreign equivalent phrases [Global security, 2006].

In personal computing, ASR is used mostly in word processing and emailing, to dictate words into a word processor. With increasing processor power in today’s
computing, ASR is being used in database retrieval systems and other applications like gaming and command control.

The media, both electronic and print in general has served to popularize and raise interest among the general population in ASR technologies and automation. This popularization and interest increases with movies depicting androids and supercomputers with natural language capabilities, like HAL in 2001: A Space Odyssey, Star Wars and I-robot just to mention a few [Juang & Rabiner, 2004].

Home-Automation also referred to as smart-homes or domotics has found favor in ASR. Incorporating ASR in the automation process gives the home owner mobility and increased flexibility. Example of this is the voice control of multimedia entertainment equipment, lightening systems, microwaves, washing machines, and home security systems.

The automobile industry has not lagged behind; today we see applications of ASR in modern vehicles such as Global Positioning Systems (GPS) and Audio Systems. These devices are more suitable in inputting direction, requesting assistance, placing phone calls etc. There is more work to be done to increase the accuracy of such systems. As the cost of memory and computers embedded in vehicles decreases, there will be an expected increase in the roll out of devices and application making use of ASR in the industry in general [Hansen, 2001].

Despite the aforementioned advances and demonstrable potentials, speech recognition applications and devices have not seen the expected increase in usage and roll out. This small increase in usage is due to a myriad of problems, among them are the inherent differences in the way people speak (inter-speaker differences), differences in
pronunciation and or local accent, and most importantly meaning of spoken sentences i.e. resolution of ambiguity. As speed and accuracy of speech recognition systems gets better its usage and adoption will become common thing and make ASR the input system of choice. Figure 1.1 illustrates the chronology and advancement in speech recognition technology.

![Figure 1.1 Milestones in Speech Technology Research](image)

1.2 **Input Devices**

The history and evolution of computing and computing devices is marked by many technological breakthroughs and advancements. Among these advancements is the emergence of different input devices mostly combining electronics with mechanics. Speech and voice recognition in computing have not lagged behind, yet the main interface and interaction with computers, for many users continue to be the keyboard, mouse, and touch screen devices.

The advent of wireless devices in computing, mobile computing, and gaming has given users some degree of mobility, yet, users still require the use of a keyboard and
mouse in their interaction with computers. Introduction of speech recognition in computing would add to this found freedom of movement and greatly enhance applications that harness this power [Markowitz 2003].

Speech Recognition (in computing sense) is the process by which a computer or an electronic device identifies spoken words [CSLU 1996]. This input method, allows an individual to communicate (speak) with electromechanical and electronic devices and or control devices via voice or speech activation (command control). If properly implemented the possibilities are immense.

1.3 Connectivity and Network Access

The computer has evolved from being a huge machine occupying large office space into smaller units we see in our offices and greater portion of households around the world. As the number of computers deployed increased so did the need to interconnect them, share, and or exchange resources. This need has fueled the rapid adoption of the Internet leading to a symbiotic relation between computer and the Internet.

Network accessibility is the driving force behind the widespread use of computers in today’s world. The evolution of the Internet revolves around existing telecommunication infrastructure dominated in the early days by the Plain Old Telephone System (POTS) via dial up modem. There continues to be improvement in the area of connectivity, driven and fueled by the quest for speed and mobility as evidenced by the roll out of fiber optics networks, cellular networks, and high speed wireless data networks.
Today, high speed networks are a common in major cities around the world. High speed fiber networks, cable broadband and now wireless broadband are the competing technologies promising high data throughput and mobility. These advances have not been able to eliminate the reliance on dial-up or the POTS infrastructure as a principal means of connections but rather have sought to complement and extend its range. ADSL / DSL uses the existing POTS lines to offer high speed data transmission to home users in inner cities and their immediate surroundings. Dial up continues to be the principal means of connecting to the Internet in many countries, rural areas, and people on low budget as it is offered cheap or free in some areas. Almost every new personal computer (home computing) comes with some variant of dial up modem, network interface card or a COM port for connectivity.

The emergence and roll out of cellular phone networks was thought to signify the demise of POTS and home phones (land lines). The increase in penetration of cellular phones has not seen any marked replacement in the use of POTS infrastructure in households and business establishments. It is rather complimentary. Cellular and wireless networks are yet to catch up in high data throughput comparable to those offered by wired networks. Existing copper networks continue to offer reliable services to businesses and households. Its replacement would require major investment and an evolutionary path that leads to no dead ends, a path which really leads to the future [Carter 2001]. This only means that legacy network (with continuous improvements and adaptations) will be with us for some time to come, although we are seeing a migration towards Voice over IP (VoIP) and Converged Network.
1.4 Putting it all together

Existing speech recognition applications limit mobility due to current design principles and considerations as stated above. Sensitive applications also require human intervention (close monitoring) to ensure correct operation and desired results, again limiting mobility. These and other related issues continue to hinder the full adoption of speech recognition systems. Nevertheless, research and use of speech recognition continue in a myriad of areas.

An example of this innovation is in the area of accessibility. Satoru Iehira, a wheel chair patient and a long standing user of headsets (due to difficulty in holding cellular phones), quickly adopted Bluetooth, the new wireless technology, coupled with improvement in accessibility features in Windows Operating System (now standard in the latest Windows OS, Windows 7), he is able to use the headset and Speech recognition features of the Window OS to control and command applications on the PC [Microsoft 2008].

This project uses these readily available devices, voice modems, phones (cellular and landlines) and most importantly a speech engine (standard in most recent Windows OS distributions) to develop a remote command control application. The final objective is to combine the aforementioned devices coupled with increased recognition capabilities of an ASR engine and a basic grammar to provide a voice computer command and control over existing telephone networks in its simplest form. That is, the intended user (systems administrator) would be able to connect using traditional phone or cellular phone to the computer (at this point authentication is not a major consideration although it could be integrated into the application) and issue voice queries (commands)
to the computer. These queries are based on predefined command set (grammar). The result of executing such command if any is then read back to the user through the synthesizer.
2. REMOTE VOICE COMMAND AND CONTROL

2.1 The Goal

The ultimate aim of this project is about taking advantage of ongoing developments and improvements in Automatic Speech Recognition (ASR) commonly referred to as computer speech recognition to provide limited “voice” remote computer control or remote voice command and control via telephone network as opposed to Voice over IP (VoIP). Remote voice command and control in this sense refers to the ability to dial into the computer and issue limited voice commands, mostly checking the state of the host computer or other networked computer. The idea could be extended to other areas of computing with the appropriate adjustments. The output and or result of executing such command are then fed back to the user via the Text To Speech (TTS) capabilities embedded in the Speech Engine. No attempt is made to provide any video feedback as this would require additional gear and increase in computing complexity and or bandwidth limitations.

Since this is a remote voice command control, there is not much user interface other than remembering to start the program before one leaves the location or locale. Alternatively, the system start-up could be altered by a script to start the applications when the computer boots up or is rebooted (not implemented in this project). The assumption is made that Caller ID (service and service provider dependent) is not always available, so the application does not have to screen calls base on this feature. Instead the application would answer any calls made to that particular Telephone Service Provider.
(TSP) via the modem. To prevent unwanted calls, a dedicated phone line is recommended although it is not required.

2.2 Initial System Setup and Checks

In its simplest form the system would comprise 3 parts.

1. A PC with voice compatible Modem
2. An active phone line and device for input and output
3. A speech and voice synthesizer engine for recognition and TTS

Some form of a voice modem is required to stream the audio commands to the engine for recognition. It turns out that modern computers do not come readily with this unit. They tend to come with a soft modem which only supports data and in some cases fax transmissions. Figure 2.1 shows a typical PCI 56K Voice modem which could be used for this purpose. Alternatively an external voice capable modem could be used, if one does not feel comfortable opening up and accessing the computer's internal parts.

![Figure 2.1 Modem Blaster V.92 PCI Voice Modem [Creative Labs]](image)
A microphone and a set of speakers (headset could be a nice substitute) would be handy but not required. Figure 2.2 shows a typical headset very popular in the computing industry for gaming, chatting and Voice over Internet Protocol (VoIP) applications. The headset would be used for training and calibrating the ASR engine of choice. Figure 2.3 shows common port locations on a conventional Desktop PC. Port placement may differ on different models and makes of the Desktop PC.

Figure 2.2 Logitech Headset with Microphone [Google images]

Figure 2.3 PC Ports [WPClipart]
Once all the necessary hardware and drivers have been installed, if not already installed, user would proceed to install or start the applications.

2.3 Training the ASR Engine

Most of the readily available recognizers can start recognition out of the box without prior training. It is highly recommended that the intended user spends every opportunity to train the chosen recognizer with his voice and idiosyncrasies; this will greatly improve the accuracy of the recognizer in question. Although not required, the intended user is encouraged to use the integrated training functionality of the chosen recognizer, if available, as it contains words chosen to help improve recognition.

This project is developed under Windows Vista Operating System which comes readily with Speech Application Program Interface (SAPI) 5.3 embedded engine running under the .NET framework. Earlier versions of Windows Operating Systems are also shipped with SAPI 5.0 or SAPI 5.1. Given the development platform, it follows that SAPI 5.3 will be the engine of choice, although SAPI 5.1 could also be used.

Most computer systems come readily with some form of audio output device. We cannot say the same for an audio input device, i.e., microphone. The intended user is expected to ensure that there is some form of an audio input device attached to the computer in question. Although the aforementioned are not required, without it there would be no prior training of the recognizer which would result in low accuracy and high false recognition.
2.4  Using Vista's Speech Recognition Options for Training

From the project's point of view this would probably be the only interface or user's interaction with the applications. As stated in the previous section, this training would increase the accuracy of the engine and thereby reduce false recognitions.

Windows Vista, provides a recognizer which can be operated as a shared recognizer, the recognizer is shared with other applications or in process recognizer in which case the calling process owns the recognizer. It also comes with a training program which is used to tune the recognition engine. The recognition tuning application is not started by default and would require intervention to get it started, read the provided training text into the chosen audio input [Microsoft 2008]. If done correctly, this will increase the computer's ability to closely recognize ones speech pattern. Appendix A shows how to start the shared recognizer on a Windows Vista Operating System box.

2.5  Testing the Command Set

At this stage the capabilities of the installed modem device should have been verified to either support voice or not. As stated previously in section 2.2 most modems (if any) installed on recent computers are soft modems which tend to support data only. If that is the case, then an additional internal or external voice modem would be required to proceed any further. For the current project development, an additional internal PCI voice modem was installed.

After successfully completing Microsoft embedded training, one can proceed to start the application, and test the command set against the corresponding operation to be performed upon recognition. This would present an opportunity to ensure the system is
operating as intended before putting it into real time applications. The following is a modified sample command to be implemented:

1. Ping - a command line tool used to test whether a particular computer is reachable across the network; it is also used to test the network interface of the host computer [Wikipedia].

2. Netstat - network statistics is a command line tool that displays network connections, routing tables and network interface statistics. Used for finding problems in the network and to determine the amount of traffic on the network as a performance measurement [Wikipedia 2009].

3. Shutdown - used to shutdown, restart or stop process on a host or remote computer from the command line.

4. Restart - as the name suggest, used to restart a computer. This is included as an option in the shutdown command.
3. SYSTEM DESIGN

3.1 Base Development Language Selection

The initial idea was to use Open Source software for the design, but a lengthy search revealed that most of the available free software had limited or no support at all and often discontinued. This difficulty led to a change in decision and a move toward Microsoft software products. Luckily the available Microsoft products in Speech and Telephony are free and readily available as explained in subsequent chapters.

Having decided upon Microsoft Products, the problem was reduced to deciding on which of the many programming languages and Integrated Development Environment (IDE) to use. Visual C# would have been the perfect language for this project. Visual C# combines Object Oriented Programming (OOB) style of C, the ease of Visual Basic and present an opportunity to learn a new language. Visual C# comes as part of the Visual Studio 2008 (VS 2008) Integrated Development Environment (IDE) and that would be the programming environment required for this project. Visual Studio 2008 requires the installation of .NET 3.0 or higher.

As it turned out TAPI is not supported from managed code (CLR). According to MSDN documentation the managed wrapper created by VS 2008 .NET is not able to adequately handle the complexity of TAPI 3.x COM interface. This revelation caused a rethink and a shift toward a language that supports native mode in VS 2008, in this case Visual C++ in Native Mode. Visual C++ in Native mode is unmanaged and hence does not use CLR a Microsoft's implementation of the Common Language Interface (CLI) and does not have issues associated with using Managed Code.
3.2 Automatic Speech Recognition Engine

Opting for Microsoft’s Visual Studio Programming environment made Microsoft Speech Application Program Interface (SAPI) the natural choice. Microsoft's SAPI 5.x is available free of charge, with continuing improved capabilities and a large support community [Microsoft 2009]. SAPI 5.x has both recognition and Text To Speech (TTS) components so it is well suited for command and control applications and would be the ASR of choice for this project.

SAPI 5.x present two modes of operation, Shared Recognition and In Process Recognition (InProc Speech Recognition). In shared Recognition mode, the recognizer uses the default audio settings of the host computer and is shared among applications requiring it services. In Process Recognition mode, the recognizer is not shared and requires setting of the audio input.

From the preceding paragraph, it is clear, In Process Recognition mode presents a better option for this project design. This enables the setting of the recognizer’s input to a non standard input, in this case the modem’s wave device. Similarly the audio output for the Synthesizer is set to the modem’s wave device. It is instructive to know that failure to set the audio input will result in the recognizer not working. Sadly enough there are no errors generated for this oddity. Hopefully in future implementation Microsoft will address this issue.

The procedure for creating and setting up an instance of ASR is straightforward and recommended by Microsoft in its documentation:

1. Create a recognition object
2. Create an instance of the of this object
3. Set event handler and event of interest i.e. recognition

4. Set audio input / output if not Shared Recognizer (set to the modem)

5. Load the grammar (used for recognition)

6. Activate grammar for recognition to start listening

The following is a code snippet showing how to create an instance of an InProcess Recognizer (InProc Recognizer) and set event of interest to us (adopted from Microsoft's sample code in its MSDN). Refer to Appendix F for a complete SAPI initialization code.

```c++
HRESULT COperator::InitializeSapi()
{
    HRESULT  hr;
    //create a reco engine for recognizing speech
    //inproc recognizer since we want to manipulate audio input
    hr = m_cpIncomingRecognizer.CoCreateInstance ( CLSID_SpInprocRecognizer );
    if  ( FAILED ( hr )  )
    {
        //process error here
        return hr
    }
    //create recognition context
    hr = m_cpIncomingRecognizer->CreateRecoContext ( &m_cpIncomingRecoCtxt );
    if ( FAILED ( hr ) )
    {
        //process error
        return hr ;
    }
    //set event of interest to us
    Const Ulonglong ullInterest  = SPFEI ( SPEI_PHRASE_START | SPFEI ( SPEI_RECOGNITION ) | SPFEI ( SPEI_RECOGNITION ) );
    hr = m_cpIncomingRecoCtxt->SetInterest ( ullInterst ,
                                            ullInterest );
    if ( FAILED ( hr ) )
    {
        //process error
        return hr;
    }
}
```

On the surface setting the input appears to be straightforward. It turns out in TAPI programming audio devices are only accessible during active calls. A determination of the audio frequencies supported by the modem audio waves is also
required and is done by querying the active interface. TAPI provides an interface to query for supported audio wave format, this information amongst others is used to set the recognizer’s input to allow for smooth audio rendering. Refer to Appendix G for setting the audio input and output adopted from MSDN sample code.

Once the output of the recognizer is successfully set to the modem’s wave device during an active call, then and only then we actually activate the loaded grammar and begin listening for recognition. Again we must add that using the provided dictation grammar would over burden the recognizer by trying to recognize every imaginable word in its dictionary [Microsoft] and cause an increase in false recognitions. Instead, create a smaller and specific targeted grammar which captures the actual commands to be executed. A smaller grammar reduces the available words and or phrase combination for recognition and increases the accuracy of the recognizer [Microsoft 2008]. SAPI provides a program for creating grammars. Since the intended grammar is limited, it is easier to write out this grammar into text file and load it during the initialization process. The grammar is then compiled by SAPI and used at run time. The following is XML code snippet for recognizing “ping localhost” command or rule:

```xml
<GRAMMAR LANGID="409">
  <DEFINE>
    <ID NAME="ID_PING" VAL="100" />
  </DEFINE>
  <RULE NAME="PingRule" ID="ID_PING" TOLEVEL="ACTIVE">
    <PHRASE>ping</PHRASE>
    <PHRASE>
      <LIST>
        <P>localhost</p>
        <p>hostname</p>
      </LIST>
    </PHRASE>
  </RULE>
</GRAMMAR>
```
Similarly, the output of the TTS engine is set to the modem audio device during an active call; so as to direct the synthesized text and audio rendering to the remote user via the modem. The output from successfully executing the command is preprocessed via string and buffer manipulation (formatting) and then fed to the TTS Synthesizer for delivery. Appendix E shows the complete grammar for this project.

3.3 Modem Selection

A modem (modulation demodulation) is a device that modulates analog carrier signals to encode digital information, and also demodulates such carrier signal to decode the transmitted information. The objective is to produce a signal that can be transmitted easily over a given transmission medium and decoded to reproduce the original data [Wikipedia]. The most familiar example is a voice band modem used in personal computers to convert digital information into sounds transmitted over the telephone line on POTS.

Modems are generally classified by the amount of data they can send in a given time, normally measured in bits per second, or "bps". They can also be classified by Baud, the number of distinct symbols transmitted per second; these numbers are directly connected, but not necessarily in a linear fashion.

Modems can also be classified according to the type of information they carry and the added features they support into Data/Fax, Data/Fax/Voice or Data/Fax/Voice/Speaker phone. Voice in this case means the ability of the modem to support telephone answering machine functions (with appropriate software), it is also able to record and play sound using the wave device on the system [Wikipedia 2009].
3.3.1 The Interface Card

The best option is to use the Internal Modem supplied with most computers. As it turned out and stated in section 2.2, most of these modems or at least the one installed on the PC used during the development phase was a soft modem and does not support voice. Voice capabilities is verified and or confirmed by querying the Modem using the OS modem diagnostics tools. Appendix B shows how to verify modem support for audio in Windows Vista OS environment. Soft Modems only support data and in general are implemented in software, that is, no dedicated hardware implementation except the connecting port (RJ-11). To solve this problem, the choice is to either use an external voice modem or a Peripheral Component Interface (PCI) voice modem. The latter was decided upon in this project.

Creative Labs DSI 3631 V.92 PCI Data/Fax/Voice Modem (PCI Modem) was decided upon. This selection was based on reviews of its wave and voice quality and not on any personal experience. This modem has support for V.92 the latest Protocol that adds features to dial-up connections among them Modem on hold and increase speed.

The thinking was to develop a script that uses Hayes command set (also referred to as AT commands) to communicate with the modem so as to set it to voice mode and pipe this to the recognition engine's input. That is, the script will monitor the status of the Modem for incoming calls. Upon receipt of a call notice the script would initiate a series of event culminating in speech being sent to the recognizer’s input for recognition and conversion to text for further processing. It turns out, Microsoft Telephone Applications Program Interface (MS TAPI 3.x) [Microsoft2008] provides all these services and it is free, contrary to initial thoughts of it being a licensed product. As per suggestion from
the Microsoft Developer Network (MSDN) documentation, below is the sequence of event using any of the available TAPI package.

1. Create a TAPI Object and create an instance of this object
2. Initialize the created object instance
3. Set the event of interest to us i.e. call notification
4. Select and Register an address to listen on
5. Wait for and process call notification event

Appendix H shows the full code segment that implements the bulleted items. The code basically check for all possible TSP capable devices on the host computer and list them. The code then query the interface if it is a unimodem and supports voice. If voice is supported then try registering that address. Figure 3.3.1 show the result of running the code on a PC with a soft modem. Figure 3.3.2 show the result of running the same code on a PC with a PCI broadxent modem. Clearly the voice modem is detected and registered for listening on.

![Figure 3.3.1 Possible TSP devices](image_url)

Figure 3.3.1 Possible TSP devices

21
Figure 3.3.2 Selection of Broadxent Modem

Figure 3.3.3 Registration of Modem
In the event handling section of interest are Call Notification and Call State, which are processed to see if we are getting new calls, or call termination. If it is a new
call, then set the recognizer input to the modem and activate the loaded grammar to be used for recognition and start recognition. Figure 3.3.5 shows a call answered by the application and waiting for recognition events. The application checks the line for call termination. Upon call termination, we just return the used resource and send normal disconnect signal to the application.
4. EVALUATION AND RESULTS

4.1 Modem Testing

The Modem constitutes a critical link; as such no assumption should be made about it. Two important aspects needs to be verified, support for voice and telephony. Testing for voice support was done using the embedded diagnostics query tool under modem properties as opposed to using direct AT Commands. Appendix A shows the result of such a test as performed on the development computer.

As previously stated in section 3.1, a dual carriage approach was used in the development phase; developing the telephony portion and speech recognition separately and integrating these two at the end. Is there a telephony object and does it supports audio? This is answered by using the OS system tools to query the modem or using terminal emulation software and AT Commands to query the modem (Appendix D shows the result). The application enumerates all possible TSP devices on the host computer (Figure 3.3.3 shows the result of running such test on a host computer) and queries for voice support. If it is successful, it gets the address of the device and register this address so we can listen for calls.
Upon successful registration the application waits for incoming calls. Once a call comes in, if the Auto Answer check box is checked then it is answered else the application waits for the user to click the answer button. Once the call is answered, the user is greeted and the applications wait for voice input. Figure 4.2 and Figure 4.3 shows an incoming call answered.
4.2 ASR Testing

After initializing the engine, the synthesizer ability to convert text to audio or speech is tested. This is achieved by feeding arbitrary text strings to the voice object to be read out. It was clear from the rendering that certain networking acronyms, i.e. IP and IPCONFIG were very difficult for the synthesizer. Next, feed the output of executing commands (sample command) to the TTS engine. It became obvious that some level of string manipulation was required to make the output meaningful to the user.

Test for recognition was made easier having done the development under the windows platform. This consisted in using the embedded command control features in Windows Vista OS Platform. Next was to test the engine against the provided sample solitaire grammar and lastly test our actual grammar. Upon satisfying the recognition needs, control and execution are added to extend the application. That is, recognizing and executing commands in the provided grammar and directing the result to the standard output. Once that was achieved, the output was then reconfigured to the TTS and the test was repeated. Except for minor glitches the test was successful.
4.3 Piping the Output of ASR to the Modem

Having successfully tested each individual unit, the command control part of the application is now combined with its complementary telephony part. Connecting these two proved much more difficult than anticipated in the design phase. This difficulty stems from the lack of support for TAPI 3.0 in a managed code. In the end it was decided to use Windows API in Native Code.

Having migrated to Windows API in Native Code programming, it was possible to overcome the limitations of managed code in TAPI programming and successfully connect the TAPI interfaces to the SAPI input and proceed with the testing. At this stage it was just a question of proving the concept and flow control. Make a call, application answers the call, and pass call interface to the engine. Engine recognizes and executes the command. The result of such execution is then read through synthesizer output to the modem and then back to the user. Figure 4.4 shows the application recognizing the ‘PING’ command.
The above procedure was repeated for consistency. Each time the modem device was connected to different phone system including but not limited to PBS Systems. Again the modem was able to handle the call. It would be interesting to see what would happen if we are able to connect to cellular (modem) device in a future development.
5. FUTURE WORK

5.1 Network Convergence

Development in connectivity is geared toward network convergence as in the coexistence of video, telephony, and data communication within a single network infrastructure. Obviously this would require different type of connectors and communication protocols. The application has to be developed to take advantage of such progressions. We are already witnessing the use of SMS, VoIP, Internet, and streaming media all using a single medium. The natural progression is to move the recognition and TTS aspect to the handset (preferably mobile) and send the resulting data (text) to and from the computer.

By placing the ASR into the mobile handset, there would be no requirement for a voice capable modem and any data modem would suffice. The result is that the remote PC would receive text command and execute the command and output text back onto the media. This will fit in well with the progression toward network convergence.

Since some modem devices support video transmission, it would be possible to include video support in these applications provided the host computer and calling party have some video input device. From the application's point of view, this would involve querying the modem for both voice and video support. Obviously a webcam or some form of video device would be required on both ends for rendering. It would be interesting to see the video quality as bandwidth would be a main concern over here.
5.2 Conclusion

Automatic Speech Recognition technology will continue to see an increase in usage as industries continue to roll out applications based on this technology. Current ASR applications limit user’s mobility to within feet of the PC or device to be controlled.

With some modification as espoused in this report, it is possible to combine existing ASR technology with a voice modem to offer voice remote control (command control) of a PC via the telephone network (not the Internet). This is achieved by setting the inputs of the current ASR technology to the voice modem’s active call interface (TAPI or TSAPI support) and hence facilitates the remote voice control. The resulting system will be used as a backup system in the absence or failure of the Internet connectivity.
BIBLIOGRAPHY AND REFERENCES


http://www.microsoft.com/windowsxp/using/setup/expert/moskowitz_02september23.ms

px (visited Jan 21, 2008)


APPENDIX A. START AND TRAIN MS SPEECH ENGINE

1. Start → Control Panel

![Windows Start menu]

Figure A.1 Windows Start menu

2. Double click the Speech Recognition Options Icon
3. Click or Select “Train your computer to better understand you”

![Configure your Speech Recognition experience](image)

Figure A.2 Configure your Speech Recognition experience window

4. This starts the training session
APPENDIX B. QUERY MODEMS FOR VOICE SUPPORT

1. Start → Control Panel

![Control panel tab in Windows Vista](image1)

Figure B.1 Control panel tab in Windows Vista

2. Select phone and modem (you may have to set dialing preferences)

![Phone and Modem options in classic view](image2)

Figure B.2 Phone and Modem options in classic view

3. Click the modems tab and select the appropriate modem
4. Click the property container at the bottom

5. Click diagnostics and then Query
6. Click query tab and wait for output

Figure B.5 Output of query, notice AT+FCLASS=？ 0, 1
APPENDIX C. VERIFY TAPI SUPPORT

Verifying support for TAPI in Windows Vista OS

1. Click start and select control panel
2. Click on Hardware and Sound

![Figure C.1 Control panel in Windows Vista](image)

3. Select phone and modem options
4. Click on the advance tab

It appears that TAPI is supported in all Vista OS Flavors
APPENDIX D. VERIFY VOICE SUPPORT

The selected modem must have support for voice. This would be verified by using the Hayes AT and Extended AT Commands (used to set and communicate with the modems). There exist two sets of commands, the Plus (+) and Hash (#).

The modem capabilities could easily be verified using terminal emulation software to query the modem. Below is a basic step to achieve that in such application:

1. ATZ   this reset the modem to default or factory settings.
2. AT+FCLASS=?   query for modem's supported modes
   output should include 0,1,2,8
   0- Indicating data support
   1, Indicating class of Fax
   2- Indicating class of Fax
   8- Audio support
3. AT+FCLASS=8   set the modem to audio mode if the preceding command showed audio support if not you would get an error.
4. AT+VSM =?   query the modem for supported audio format

The following figures show the output of the above commands and steps:
Figure D.1 Terminal emulation software connection

Figure D.2 Available ports with com3 selected
Figure D.3 Port properties (default settings)

Figure D.4 Output of executing AT commands
APPENDIX E. SAMPLE GRAMMAR

<!--Language ID US English-->  
<GRAMMAR LANGID="409">
  <DEFINE>
    <ID NAME="RuleId_Ping" VAL="1" />  
    <ID NAME="RuleId_IPv4" VAL="2" />
    <ID NAME="RuleId_Numbers" VAL="3" />
    <ID NAME="RuleId_Ipcon" VAL="4" />
  </DEFINE>
  <RULE ID="RuleId_Ping" TOLEVEL="ACTIVE">
    <OPT>Please</OPT>
    <P>Ping</P>
    <P>
      <L>
        <P>localhost</P>
        <!--<P>feleta</P>-->
        <!--<DICTATION MAX="INF" /-->  
        <RULEREF NAME="Rule_IPv4"/>
      </L>
    </P>
  </RULE>
  <RULE Name="RuleName_netstat" TOLEVEL="ACTIVE">
    <OPT>Please</OPT>
    <P PROPNAME="netstat" VALSTR="netstat">netstat</P>
    <OPT>
      <Phrase>-</Phrase>
      <LIST PROPNAME="nestat_l">
        <Phrase VALSTR="-a">a</Phrase>
        <Phrase VALSTR="-b">b</Phrase>
        <Phrase VALSTR="-e">e</Phrase>
        <Phrase VALSTR="-f">f</Phrase>
        <Phrase VALSTR="-n">n</Phrase>
        <Phrase VALSTR="-o">o</Phrase>
        <Phrase VALSTR="-p">p</Phrase>
        <RULEREF NAME="Protocol"/>
        <Phrase>
        <Phrase VALSTR="-r">r</Phrase>
        <Phrase VALSTR="-s">s</Phrase>
      </LIST>
    </OPT>
  </RULE>
  <RULE NAME="RuleName_shutdown" TOLEVEL="ACTIVE">
    <Phrase>shutdown</Phrase>
    <RULEREF NAME="RULE_dash"/>
    <Phrase>
    <LIST>
      <Phrase>+s</Phrase>
      <Phrase>+r</Phrase>
      <Phrase>+g</Phrase>
      <Phrase>+a</Phrase>
      <Phrase>+p</Phrase>
    </LIST>
  </RULE>
</GRAMMAR>
<Phrase>+h</Phrase>  
<Phrase>+e</Phrase>  
</LIST>  
</Phrase>  
<OPT><RULEREF NAME="RULE_dash"/> f</OPT>  
<OPT>  
<RULEREF NAME="RULE_dash"/> m <DICTATION MAX="1"/>  
</OPT>  
</RULE>  
<RULE NAME="Rule последние" TOLEVEL="ACTIVE">  
<Phrase>I p config</Phrase>  
<OPT>  
<List>  
<Phrase>all</Phrase>  
<Phrase>renew</Phrase>  
<Phrase>release</Phrase>  
<Phrase>allcompartments</Phrase>  
</List>  
</OPT>  
</RULE>  
<!--Rules are use internally they are not accessible from outside-->  
<RULE NAME="Protocol">  
<Phrase>  
<List>  
<Phrase>tcp</Phrase>  
<Phrase>udp</Phrase>  
</List>  
</Phrase>  
</RULE>  
<RULE NAME="RULE_dash">  
<Phrase>  
<List>  
<Phrase>-</Phrase>  
<Phrase>-</Phrase>  
</List>  
</Phrase>  
</RULE>  
<RULE NAME="Rule IPv4">  
<RULEREF REFID="RuleId IPv4"/>  
<OPT> <RULEREF REFID="RuleId IPv4"/></OPT>  
<OPT> <RULEREF REFID="RuleId IPv4"/></OPT>  
<Phrase>.</Phrase>  
<OPT> <RULEREF REFID="RuleId IPv4"/></OPT>  
<OPT> <RULEREF REFID="RuleId IPv4"/></OPT>  
<OPT> <RULEREF REFID="RuleId IPv4"/></OPT>  
<OPT> <RULEREF REFID="RuleId IPv4"/></OPT>  
</RULE>
<RULE NAME="Numbers" ID="RuleId_IPv4">
  <LIST PROPNAME="TYPE_NUM">
    <Phrase VAL="1">1</Phrase>
    <Phrase VAL="2">2</Phrase>
    <Phrase VAL="3">3</Phrase>
    <Phrase VAL="4">4</Phrase>
    <Phrase VAL="5">5</Phrase>
    <Phrase VAL="6">6</Phrase>
    <Phrase VAL="7">7</Phrase>
    <Phrase VAL="8">8</Phrase>
    <Phrase VAL="9">9</Phrase>
    <Phrase VAL="0">0</Phrase>
  </LIST>
</RULE>
</GRAMMAR>
APPENDIX F. SAMPLE COMMANDS

When Prompted please speak any of the following commands

1.  IPConfig
2.  IPConfig release
3.  IPConfig Renew
4.  IPConfig All
5.  IPConfig Display dns (not activated)
6.  IPConfig Register dns (not activated)
7.  Netstat
8.  Netstat E
9.  Netstat R
10. Netstat udp
11. Netstat tcp
12. Ping localhost
13. Ping <hostname>
14. Ping <ip address>
15. Shutdown <hostname>
16. Restart <hostname>
APPENDIX G. SAPI INITIALIZATION

HRESULT COperator::InitializeSapi()
{
    // Create a voice for speaking on this machine (testing purposes)
    HRESULT hr = m_cpLocalVoice.CoCreateInstance(CLSID_SpVoice);
    if (FAILED(hr))
    {
        DoMessage(L"Could not create a TTS voice on the local machine");
        return hr;
    }

    // Create a reco engine for recognizing speech
    // inproc recognizer since we will be setting the input
    hr = m_cpIncomingRecognizer.CoCreateInstance(CLSID_SpInprocRecognizer);
    if (FAILED(hr))
    {
        DoMessage(L"CoCreateInstance on inproc reco engine failed");
        return hr;
    }

    // Create a reco context for this engine (you can have different context)
    hr = m_cpIncomingRecognizer->CreateRecoContext(&m_cpIncomingRecoCtxt);
    if (FAILED(hr))
    {
        DoMessage(L"Could not create recognition context");
        return hr;
    }

    // Set event interest only in PHRASE_START, RECOGNITION, FALSE_RECOGNITION
    // we could set more here if we need it...
    const ULONGLONG ullInterest = SPFEI(SPEI_PHRASE_START) | SPFEI(SPEI_RECOGNITION) | SPFEI(SPEI_FALSE_RECOGNITION);
    hr = m_cpIncomingRecoCtxt->SetInterest(ullInterest, ullInterest);
    if (FAILED(hr))
    {
        DoMessage(L"Could not set interest in SAPI events");
        return hr;
    }

    // we may want to retain recognized audio for additional processing
    hr = m_cpIncomingRecoCtxt->SetAudioOptions(SPAO_RETAIN_AUDIO, NULL, NULL);
    if (FAILED(hr))
    {
        DoMessage(L"Could not set audio options to retain recognized audio");
        return hr;
    }

    // Create a grammar and load it
    hr = m_cpIncomingRecoCtxt->CreateGrammar(0, &m_cpDictGrammar);
    if (FAILED(hr))
    {
        DoMessage(L"Could not create dictation grammar");
    }
}
return hr;

// Load the grammar file here.
hr = m_cpDictGrammar->LoadCmdFromFile(CT2W(L"C:\recon.xml"), SPLO_STATIC);
if (FAILED(hr))
{
    DoMessage(L"Could not load dictation");
    return hr;
}

// Create a voice for talking on the phone.
hr = m_cpIncomingRecoCtxt->GetVoice( &m_cpOutgoingVoice );
if (FAILED(hr))
{
    DoMessage(L"Could not create a TTS voice for speaking over the phone");
    return hr;
}

return S_OK;
} /* end InitializeSapi */
APPENDIX H. SETTING AUDIO INPUT FOR A CALL

HRESULT COperator::SetAudioInForCall(ITLegacyCallMediaControl *pLegacyCallMediaControl)
{
    if (NULL == m_pCall)
    {
        return E_UNEXPECTED;
    }
    if (NULL == pLegacyCallMediaControl)
    {
        return E_INVALIDARG;
    }
    // Get the device ID
    UINT *puDeviceID;
    BSTR bstrWavIn = ::SysAllocString(L"wave/in");
    if (!bstrWavIn)
    {
        return E_OUTOFMEMORY;
    }
    DWORD dwSize = sizeof(puDeviceID);
    HRESULT hr = pLegacyCallMediaControl->GetID(bstrWavIn, &dwSize,
                                              (BYTE**)&puDeviceID);
    ::SysFreeString(bstrWavIn);
    // Find out what, if any, formats are supported
    GUID guidWave = SPDFID_WaveFormatEx;
    WAVEFORMATEX *pWaveFormatEx = NULL;
    if (SUCCEEDED(hr))
    {
        // Loop through all of the SAPI audio formats and
        // query the wave/out device about each one.
        // register the first available format
        SPSTREAMFORMAT enumFmtId;
        MMRESULT mmr = MMSYSERR_ALLOCATED;
        DWORD dw;
        for (dw = 0; (MMSYSERR_NOERROR != mmr) && (dw <
                       SPSF_NUM_FORMATS); dw++)
        {
            if (pWaveFormatEx && (MMSYSERR_NOERROR != mmr))
            {
                ::CoTaskMemFree(pWaveFormatEx);
                pWaveFormatEx = NULL;
            }
            // Get the next format from SAPI and convert it into a
            WAVEFORMATEX
            enumFmtId = (SPSTREAMFORMAT)(SPSF_8kHz8BitMono + dw);
            HRESULT hrConvert = SpConvertStreamFormatEnum(
                                    enumFmtId, &guidWave, &pWaveFormatEx);
            if (SUCCEEDED(hrConvert))
            {
            }
        }
    }
}
if ( puDeviceID != NULL )
{
    // This call to waveOutOpen() does not actually open the device;
    // it just queries the device whether it supports the given
    // format
    mmr = ::waveInOpen(NULL, *puDeviceID, pWaveFormatEx, 0, 0,
    WAVE_FORMAT_QUERY);
}
else
{
    return E_UNEXPECTED;
}

// at this point no supported formats
if ( SPSF_NUM_FORMATS == dw )
{
    return SPERR_DEVICE_NOT_SUPPORTED;
}

// Cocreate a SAPI audio in object
if ( SUCCEEDED(hr) )
{
    hr = m_cpMMSysAudioIn.CoCreateInstance(CLSID_SpMMAudioIn);
}

// Give the audio in object the device ID
if ( SUCCEEDED(hr) )
{
    hr = m_cpMMSysAudioIn->SetDeviceId(*puDeviceID);
}
// Use the format that we found works
if ( SUCCEEDED(hr) )
{
    _ASSERTE(pWaveFormatEx);
    hr = m_cpMMSysAudioIn->SetFormat(guidWave, pWaveFormatEx);
}
// We are now done with the wave format pointer
if ( pWaveFormatEx )
{
    ::CoTaskMemFree(pWaveFormatEx);
}
// Set this as input to the reco context
if ( SUCCEEDED(hr) )
{
    hr = m_cpIncomingRecognizer->SetInput(m_cpMMSysAudioIn,
    FALSE);
}
    return hr;
} /*SetAudioInForCall */
HRESULT COperator::InitializeTapi()
{
    // Cocreate the TAPI object
    HRESULT hr = CoCreateInstance(CLSID_TAPI, NULL, CLSCTX_INPROC_SERVER, IID_ITTAPI,(LPVOID *)&m_pTapi);
    if ( FAILED(hr) )
    {
        DoMessage(L"CoCreateInstance on TAPI failed");
        return hr;
    }
    // call ITTAPI::Initialize(). this must be called before
    // any other tapi functions are called.
    hr = m_pTapi->Initialize();
    if ( FAILED(hr) )
    {
        DoMessage(L"TAPI failed to initialize");
        m_pTapi->Release();
        m_pTapi = NULL;
        return hr;
    }
    // Create event notification object and register it
    m_pTAPIEventNotification = new CTAPIEventNotification;
    if  ( NULL == m_pTAPIEventNotification )
    {
        return E_OUTOFMEMORY;
    }
    hr = m_pTAPIEventNotification->Initialize();
    if ( SUCCEEDED( hr ) )
    {
        hr = RegisterTapiEventInterface();
    }
    if ( SUCCEEDED( hr ) )
    {
        hr = m_pTapi->put_EventFilter(TE_CALLNOTIFICATION | TE_CALLSTATE);
    }
    if ( FAILED( hr ) )
    {
        DoMessage( L"Could not set up TAPI event notifications" );
        return hr;
    }
    // Find all address objects that we will use to listen for calls on
    hr = ListenOnAddresses();
    if ( FAILED(hr) )
    {
        DoMessage(L"Could not find any addresses to listen on");
        m_pTapi->Release();
        m_pTapi = NULL;
        return hr;
    }
    return S_OK;
}
HRESULT COperator::ListenOnAddresses()
{
    // enumerate the addresses
    IEnumAddress * pEnumAddress;
    HRESULT hr = m_pTapi->EnumerateAddresses( &pEnumAddress );
    if ( FAILED(hr) )
    {
        return hr;
    }
    TAddress * pAddress;
    bool fAddressExists = false;
    while ( true )
    {
        // get the next address
        hr = pEnumAddress->Next( 1, &pAddress, NULL );
        if (S_OK != hr)
        {
            // Done dealing with all the addresses
            break;
        }
        /*BSTR bstrName;
        hr = pAddress->get_AddressName(&bstrName);
        DoMessage(bstrName);*/
        // Does the address support audio?
        if ( AddressSupportsMediaType(pAddress, TAPIMEDIATYPE_AUDIO ) )
        {
            // If it does then we'll listen.
            HRESULT hrListen = ListenOnThisAddress( pAddress );
            if ( S_OK == hrListen )
            {
                fAddressExists = true;
                m_pAddress = pAddress;
                DoMessage(L"I am in the faddressExistLoop");
            }
        }
    }
    pAddress->Release();
    pEnumAddress->Release();
    if ( !fAddressExists )
    {
        DoMessage( L"Could not find any addresses to listen on");
    }
    else
    {
        return fAddressExists ? S_OK : S_FALSE;
    }
} /* COperator::ListenOnAddress */