Constructing modular speech interface to remote applications.

Sanjay P. Chitte
University of Windsor
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
NOTE TO USERS

Page(s) not included in the original manuscript are unavailable from the author or university. The manuscript was microfilmed as received.

iii, viii

This reproduction is the best copy available.

UMI
Constructing Modular Speech Interface to Remote Applications

By
Sanjay P. Chitte

A Thesis
Submitted to the College of Graduate Studies and Research through the Department of Computer Science in Partial Fulfillment of Requirements for the Degree of Master of Science in Computer Science at the University of Windsor.

Windsor, Ontario, Canada
1999

© 1999 Sanjay P. Chitte
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-52528-7
Abstract

Speech technology is becoming popular within personal and enterprise computing and is being used to develop speech interfaces to various applications.

In this thesis a prototype speech interface that accesses remote application was developed. This interface uses speech recognition as input and speech synthesis as output and navigates through the remote application called sikhos. New concept of deploying and accessing speech-accessible hyperlinked objects over the Internet and downloading the grammar associated with them is introduced. Methods to access sikhos and choice of speech technology were investigated.

The interface was constructed using Java, IBM’s implementation of JSAPI. Different users used the interface and from different locations over the Internet. findings from this investigation are discussed in this thesis.
Acknowledgments

I would like to express my respect, appreciation and thanks to Dr. Richard A. Frost, his guidance and enthusiasm made this thesis a very enjoyable endeavor for me. I am very thankful to Dr. Ono Tjandra and Dr. William Miller for their motivation and constructive comments. I would also like to thank Dr. Peter Tsin for chairing my thesis defense seminar.

I wish to thank Mr. Walid Maaymneh and Mr. Maunzer Batal for their technical help. Also many thanks to the helpful secretaries of the Department of Computer Science Mary, Margaret and Gloria.

Last, but by no means least, I would like to thank my family and the graduate students in the Department of computer science.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>IV</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>V</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>VIII</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 The Problem</td>
<td>1</td>
</tr>
<tr>
<td>1.2 The Thesis Statement</td>
<td>2</td>
</tr>
<tr>
<td>1.3 How the thesis was proven</td>
<td>2</td>
</tr>
<tr>
<td>1.4 A quick overview of the browser that was constructed to prove the thesis</td>
<td>3</td>
</tr>
<tr>
<td>1.5 Organization of the thesis report</td>
<td>4</td>
</tr>
<tr>
<td>2.0 Background I: Overview of speech-recognition technology</td>
<td>6</td>
</tr>
<tr>
<td>2.1 Design guidelines for speech interfaces</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Speech recognition</td>
<td>10</td>
</tr>
<tr>
<td>2.3 Speech Synthesis</td>
<td>12</td>
</tr>
<tr>
<td>3.0 Background II: Overview of network communication protocols</td>
<td>14</td>
</tr>
<tr>
<td>3.1 CGI Protocol</td>
<td>14</td>
</tr>
<tr>
<td>3.2 Common Object Request Broker Architecture (CORBA)</td>
<td>16</td>
</tr>
<tr>
<td>3.3 Java Remote Method Invocation (RMI)</td>
<td>17</td>
</tr>
<tr>
<td>3.4 Telnet</td>
<td>19</td>
</tr>
</tbody>
</table>
4.0 Design Decisions I: Choice of speech recognition technology

4.1 Reasons for using the Java speech API are as follows: ..................22
4.2 Reasons for choosing IBM Via-Voice: ........................................23

5.0 Design Decisions II: Choice of network communication protocol

6.0 Recognition Grammars .........................................................28

7.0 The Architecture of the prototype speech browser .......................35
  7.1 Outline of the Prototype ....................................................35
  7.2 The Recognizer Object .......................................................37
  7.3 The Http Object .............................................................38
  7.4 The Synthesizer Object ......................................................38

8.0 Details of the Browser Program Code .......................................39

9.0 Methods for "Wrapping" Remote Applications ...............................45

10.0 How grammars are used to specialize the prototype speech-browser ........................................................................................................46
  10.1 Downloading grammars from a remote location .............................46
  10.2 Improving recognition by modifying the grammar ......................46

11.0 Investigating the use of the prototype speech browser ...............48

12.0 Analysis of results ..................................................................57
  12.1 Modifications to the Speech Interface ......................................58
13.0 Overview of Related Work ............................................. 60

14.0 Conclusion and Future Work ........................................... 63
  14.1 What has been achieved ............................................. 63
  14.2 Other findings ...................................................... 64
  14.3 Further work identified ............................................ 66
  14.4 Speech applications in the future .................................. 67

References: .............................................................................. 69

Appendix A-Java Speech API ................................................... 72

Appendix B- Program Listing .................................................. 74
  SpeechClient.java ......................................................... 74
  Httppost.java ............................................................. 96
  Httpget.java .............................................................. 98
  MC.java .................................................................. 99

Programs on the server side ................................................. 101
  C program that wraps Miranda code. .................................... 101
  preferences.h ................................................................ 106
  printer.h ..................................................................... 107
  Unix Shell Script .......................................................... 107

Appendix C- Sessions with the interface ............................... 108
  Solar.gram .................................................................. 108
  Monty.gram ................................................................ 111
  Judy.gram .................................................................. 112

Appendix D - List of Acronyms .............................................. 117

VITA AUCTORIS ...................................................................... 118
1 Introduction

Speech technology is becoming popular within personal and enterprise computing and is being used to develop speech interfaces to various applications. Speech interfaces provide a new means for human interaction with computers. They allow hands-free use of computers and access to remote applications over telephones. Speech interfaces provide an effective way for visually-challenged users to access computers.

1.1 The Problem

There is a need for low-cost, user-independent continuous-speech interfaces to remote applications. This need arises for the following reasons:

- To improve existing interfaces and support new means of user interaction with computers.
- To improve accessibility for disabled users.
- For Hands-free computer interface for data input and output.
  For example: an airline inquiry application.
- To have mobile access to remote applications.
- For speaker verification for secure access to the web.
- For speech-controlled robots.
Constructing such speech interfaces to large knowledge bases is difficult. As increase in the scope of knowledge often results in a decrease in speech-recognition accuracy.

1.2 The Thesis Statement

The problem above can be solved by constructing "speech browsers" using object-oriented programming techniques, off-the-shelf speech-recognition technology, and a common network communication protocol. Application specific grammar is downloaded and the recognizer is configured only to recognize application specific language, which in turn improves the speech-recognition accuracy.

1.3 How the thesis was proven

The thesis has been proven to be true by constructing and testing a prototype speech-browser. The speech browser is constructed using the Java programming language, IBM via-voice speech recognition technology, and HTTP/CGI as a communication protocol.
1.4 A quick overview of the browser that was constructed to prove the thesis

The speech browser that was constructed accepts speech input from the user and recognizes it based on an "active grammar". The following figure illustrates the general functionality of the speech browser.

![Diagram of speech browser functionality]

Fig 1.0 General Structure of the speech browser.

An application-specific grammar file is first obtained from a remote location or locally. The user input is obtained via a microphone. It is then formatted into an http Post/Get request and sent to the application. The response obtained from the application is then
synthesized to spoken English. The prototype speech interface that was constructed allows user interaction such as the following:

*User*: hello there

*Computer*: my name is computer what is your name

*User*: my name is Mickey mouse

*Computer*: hello Mickey, how are you?

### 1.5 Organization of the thesis report

Chapter 2 gives an overview of speech-recognition technology, speech recognition, and speech synthesis and design guidelines for speech interfaces. Chapter 3 describes some protocols that can be used to communicate with remote applications. Chapters 4, 5, and 6 discuss design decisions made for selecting the protocol, the speech recognition and synthesis software, and the use of recognition grammars. Chapter 7 describes the architecture of the speech browser followed by an explanation of the browser program code in chapter 8. Chapter 9 describes methods used to wrap remote applications. Chapter 10 describes how grammars can be used to specialize speech browsers. Chapter 11 includes a sample session with a remote application and comments about what users felt while using the browser. Chapter 12 discusses what has been achieved and what modifications can be made. Chapter 13 gives an overview of related work in construction of Speech Interfaces. Chapter 14 concludes the
work with some suggestions of how to make the browser more robust and its use in projects like SpeechNet.
2.0 Background I: Overview of speech-recognition technology

Speech interfaces provide users with more flexibility; an extra dimension of control and a new mode of access for visually-challenged users. Essential components of a speech interface are: a recognizer, an interpreter, and a synthesizer. Speech recognition is a process by which audio input containing speech is converted into text. Speech synthesis is the reverse process of producing synthetic speech from text generated by user application. The Speech API is the interface between the application and the recognizer and the synthesizer.

The following factors influence the design of speech-recognition applications:

- Speaker-dependent or speaker-independent systems.
- Vocabulary of recognizable words.
- Continuous or discrete speech recognition.

Speaker-independent systems are preferred over speaker-dependent systems because use of speaker-dependent interfaces requires training. This training is time consuming and users may lose interest in using...
the system. However the recognition errors of speaker-dependent systems are less compared to speaker-independent systems.

Large languages are more likely to contain ambiguous words as compared to small input language and hence it lowers the recognition accuracy. They also require more time to search the speech-model database. One way to avoid large vocabularies is to constrain the application so that it understands a specific set of words and phrases, For example building a vocabulary of medical terms.

Discrete-speech recognizers recognize words more accurately compared to continuous-speech recognizers because of the fact that in discrete speech the word boundaries are short silences between the words and are easy to find. However discrete recognition places a burden on the user and also reduces the rate at which the information is fed into the system.

2.1 Design guidelines for speech interfaces

Graphical User Interface (GUI) design can not directly be transformed into a Speech User Interface for a speech-only environment because the Speech User Interface has to converse with the application and with the user. As there is no or minimal GUI, it is
the designer's responsibility to start the application with a message. For example:

*Hello, I am solar man I can answer questions regarding the solar system.*

Or the application should be able to respond to general greeting's e.g. "Hello," or "Hi there". Thus one of the major requirements is to establish and maintain a conversational flow. Maintaining a context as much as possible would allow users to speak more naturally, similar to human-human conversation.

Another problem is that since users are talking to the computer, they would like to ask all sorts of questions, however in some cases the recognizer is unable to recognize a question due to a limited input language or the application itself is not designed to answer certain questions. Thus it is very important for the speech interface to inform the user about questions they can ask. For example, if a user asks "solar man" "how is the weather tomorrow", the program will not be able to respond even though the recognizer may have recognized the input. Thus for a speech interface, it is very important to inform the user of its functionality.

The graphical layout or presentation that is normally used in a graphical environment can not be directly translated into a speech
interface. Another important human aspect needed to be considered is that people can speak easily and quickly, but are unable to listen to synthesized speech at the same rate. So the designer has to carefully balance the information it reads out to the users.

Another problem with speech-only interfaces is response time. For example a user asks a question but the system takes a long time to process it. The user may wonder whether his query is heard or not and repeats the same query again. This results in missing the application's response, or causing a recognition error. To avoid this situation there should be feedback to the user.

Prompting the user leads to a smooth interaction with a speech-only application. Prompts can be explicit, implicit or incremental. Explicit prompts direct the user to say what input the application is expecting. For an example "say Yes\No to submit this application", thus the interface is expecting "yes" or "no" as the user's response. Implicit prompts are used when the application is able to accept more flexible input based on a conversation with the user. For example if the user says, "fax document to Bill", the application could respond by asking a question "Bill Smith" or "Bill Hunt". Incremental prompts can be used to speed the interaction and provide help for less-experienced users.
2.2 Speech recognition

The steps involved in speech recognition are: grammar processing, signal processing, phoneme recognition, word recognition and result generation. In off-the-shelf speech-recognition technology all these processes are automatic and the developer does not have any control over them. Only the grammar can be created and activated for a recognizer, the grammar defines the words and the patterns in which they may be spoken.

The grammar constrains the recognition process that makes the recognition faster and more accurate as the recognizer only has to recognize the words and sentences defined by the grammar.

In the case of continuous speech the words spoken cannot be identified clearly due to overlaying of wave patterns, as one sound modifies the sound following it. This effects the separation of different sounds and hence increasing errors and inconsistent responses.

A common problem is that the user may say something at one time and it is recognized but is misrecognized later on. This can be frustrating. In response the user may try to change his accent or pitch and talk
which may lead to more recognizing errors. This is a natural tendency among users because they may feel that they are unable to use the application.

Other causes of recognition errors are when the users speaks before the application is ready to listen, an accent, cold, high background noise or the users speaks something that is not covered by the grammar or the dictionary.

There are three kinds of recognition errors: rejection, substitution and insertion. A rejection error occurs when the recognizer has no clue what the user is saying. When a recognizer substitute user's utterance with a different legal utterance it is called substitution error. The recognizer may pick up noise as a legal utterance this is called as insertion error.

Handling errors improves the quality of the speech interface. Rejection errors can be handled by saying "Please repeat I don't understand" or by providing some assistance for example the first message might be say "what?". If another error occurs then say "sorry I don't understand". This technique prevents the user form getting discouraged. Misrecognition and misfires are harder to detect and
harder to handle. Switching off the recognizer when not in use can reduce errors pertaining to background noise.

2.3 Speech Synthesis

Speech synthesis is text-to-speech conversion, the two major parts for producing spoken language are structure analysis, that processes the text to determine the formatting data used in this stage, and text preprocessing which analyses the sentences for special constructs such as dates, time, acronyms etc.

The other steps are text-to-phoneme conversion, prosody analysis and the final stage is waveform production. Humans can detect errors from synthesized speech easily, it is the application developer's responsibility to minimize the errors and improve the quality of spoken text.

The important factors while judging the quality of spoken text is naturalness and understandability. Naturalness means to what degree the synthesizer spoken sounds like human. Understandability is measure of how reliably a listener understands spoken text.

.html] provides different methods for an application developer to improve the quality of spoken text.
3.0 Background II: Overview of network communication protocols

The remote applications that we are trying to access are programs written in an extension of Miranda called W/AGE. The major constraint in developing this application was lack of portability of Miranda and W/AGE. The Miranda interpreter currently supports only the Unix operating system it was necessary to wrap W/AGE programs in a wrapper program written in C.

The various choices of communication protocols available to connect the client with the remote application using the Java SpeechAPI were:

- CGI
- Common Object Request Broker Architecture (CORBA)
- Java Remote Method Invocation (RMI)
- Telnet

3.1 CGI Protocol

CGI applications use the HTTP protocol to communicate between the client, typically a Web browser, and the Web server. HTTP is a stateless protocol, which means that each request-response session is independent and does not contain any history information. Stateless
protocols are straightforward to implement. This simplifies the software required to support it.

HTTP is based on a request/response paradigm. An HTTP transaction consists of the following stages:

- **Connection**: establishing a connection from the client to the server.
- **Request**: client sends a request message to the server.
- **Response**: server sends a response to the client.
- **Close**: server closes the connection.

CGI is the most popular approach for web applications, some of its drawbacks are:

1) As CGI is request/response protocol it requires a new process to be created for every request. The entire request has to go through a web server hence it creates web traffic. As the server side is responsible for generating HTML presentation for the client, this puts additional load on the server.

2) CGI applications are slow, although some of these drawbacks can be overcome to some extent by using server plug-in APIs (NSAPI, ISAPI, Servlets) server side scripting and cookies.
3) CGI protocol is not suitable for having dialog due to the statelessness of the HTTP protocol.

3.2 Common Object Request Broker Architecture (CORBA)

CORBA [http://www.cs.wustl.edu/~schmidt/corba-overview.html] is a set of specifications that define the way software objects work together in a distributed environment. CORBA defines a language and a platform-independent bus called an Object Request Broker (ORB) that allows the objects to interoperate across networks, operating systems and languages. ORB lets the objects transparently make requests to, and receive responses from, other objects located locally or remotely. It also takes care of locating servers, marshalling requests and responses, handling concurrency and handling exception conditions.

CORBA uses an open communication protocol called Internet InterORB protocol (IIOP), this protocol enables communication between ORB's transparently over TCP/IP.

CORBA ORB's bind a wide variety of languages including C++, Java, C, Ada, COBOL, Smalltalk and others thus enables software development in a mixed programming language environment. Programmers no longer have to concern themselves with low-level
programming details such as opening a socket, reading or writing from a socket etc. Also using multiple threads, servers can serve many more clients. Compare to CGI programming there is little client-server overhead as in the case of a HTTP/CGI call where a new instance is created every time the client invokes a method on the server. CORBA-based applications are much faster compare to HTTP/CGI applications. As CORBA supports language independence it is easier to integrate existing applications.

A limitation of CORBA is that it has a steep learning curve, training is required so that application developers get used to CORBA-style programming. Another drawback is lack of test tools for testing CORBA-based applications and message passing over IIOP. One has to be careful in adopting a new technology like CORBA because distribution should be done only when it is necessary and the system requirements are clearly understood.

3.3 Java Remote Method Invocation (RMI)

The Java Remote Method Invocation (RMI) enables the programmer to create Java-to-Java distributed applications, in which the methods of remote Java objects can be invoked from other Java
virtual machines residing on the same host or a different host [http://java.sun.com/docs/books/tutorial/rmi/overview.html].

A Java program can make a call to a remote object once it obtains a reference to that remote object by looking up the remote object in the naming service provided by RMI or by receiving the reference as an argument or a return value. RMI uses Object serialization to marshal and unmarshal parameters and support the object-oriented features of Java.

Java RMI has several advantages. For example, it makes programming simpler because RMI requires no mapping to any common interface-definition languages, and all the code that handles communication is generated directly from the class files. The syntax of Java-Remote-Method invocation is similar to local-method invocation.

RMI removes the burden of memory management from the programmer, because the underlying system provides distributed garbage collection. Java RMI also provides other capabilities such as dynamic distribution of the executable code, this means the code need not be preinstalled on client machines, thus greatly reducing the burden of software distribution and system maintenance.
RMI works reliably across different operating systems wherever a Java-compatible Virtual Machine is available. Thus making Java RMI an obvious choice for developing 100% Java client/server and distributed applications.

A major disadvantage is that it is only useful to build Java-based applications. RMI uses a proprietary communication protocol, however support for IIOP has been agreed upon to be implemented in future releases. Only the naming service is available as compared to several services provided by CORBA.

3.4 Telnet

The Telnet protocol provides a general bi-directional eight-bit byte-oriented communication facility. It allows a standard method for interfacing terminal devices and terminal-oriented processes to each other.

The Telnet protocol is built upon three main ideas the concept of "Network Virtual Terminals"(NVT), the principal of negotiated options and a symmetric view of terminals and processes.
The network virtual terminal (NVT) is an imaginary device from which both ends of the connection, the client and the server, map their real terminal to and from. That is the client operating system must map whatever type of terminal the user is on to the NVT. The server must map the NVT into whatever terminal type the server supports. The NVT is a character device with a keyboard and a video display unit. Data typed by the user on the keyboard is sent to the server, and data received from the server is output to the video display unit.

The principle of negotiated options enables the use of additional services that may be used by many users compared to the minimal services offered by network virtual terminal. The basic strategy for setting up use of options is to have either party or both initiate requests that some option takes effect. The other party may then accept or reject. If the request is accepted the option immediately takes effect. If it is rejected the associated aspect of the connection remains as specified for an NVT.

As a matter of fact, a party may always refuse to enable an option, but it should never refuse to disable some option since all parties must be prepared to support the NVT. The syntax of option negotiation has been set up so that if both parties request an option simultaneously,
each will have each other's request as the positive acknowledgment of its own.

Telnet option negotiation is symmetrical; either end can initiate the negotiation of an option. This can lead to non-terminating loops to prevent such loops the parties should follow the rules as specified in Telnet Protocol specification.

Telnet is a promising protocol to implement this application however the user needs to have a login identification and password in order to access the application on a remote server. Due to security reasons anonymous logins are best avoided. The other disadvantage is that users should be knowledgeable enough to run the executables on that particular operating system they are logged on to through telnet.
4.0 Design Decisions I: Choice of speech recognition technology

The following were the options available for selecting the speech synthesizer and recognizer.

- Using Microsoft Speech API (SAPI)
- PE500 recognizer from SSI
- DECtalk text to speech synthesizer
- IMB ViaVoice implementation of the Java speech API

It was decided to use the IBM ViaVoice implementation of the Java speech API, and the IBM ViaVoice speech-recognition software.

4.1 Reasons for using the Java speech API are as follows:

The Java Speech API is an open standard and is an extension to the Java platform. The API supports speech recognition and speech synthesis. It provides a robust cross-platform and cross-vendor interface to speech synthesis and recognition. It also supports command-and-control and dictation-speech recognition.
4.2 Reasons for choosing IBM Via-Voice:

It is the first speech-processing software that supports the Java Speech API; it is a low cost and accurate commercial product. It is easy to use and install and does not require any specialized expensive hardware. It uses any SoundBlaster-32 or higher-version soundcard installed in a desktop or laptop. However, currently the Via-Voice only supports Windows95\98\NT and LINUX operating-system environments.

The other option was to use the PE500 speech technology from SSI Inc., which has dedicated hardware and software for speech recognition. This system doesn't support speech synthesis. The Dectalk text to speech synthesizer is used along with it. This complicates and increases the cost of, the setup and application development. Also speech applications constructed using the PE500 are not portable since Java language support is not available.

Applications written in Microsoft speech API are not easily portable and it doesn't support the Java Speech API.
5.0 Design Decisions II: Choice of network communication protocol

After looking at the advantages and disadvantages of the various communication protocols, and keeping the application in mind let us take a look at the deployment issues.

In order to use Java Remote Method Invocation or CORBA, the 'C' program, which is the W/AGE wrapper, needs to be wrapped around by a Java Wrapper using Java native methods.

Java RMI and CORBA are the best methods to build many distributed-object applications in our case they are not effective for various reasons. CORBA cannot be used directly because of the lack of an idlToMiranda mapping. A round-about-way to use CORBA is to wrap the C code which access the Miranda code into a Java object using Java Native methods.

A problem with the JAVA RMI approach (for our application) is that in order to reflect the changes in W/AGE program we need to rebuild the link library using the Java native method implementation process. Since one of our objectives is to use this as tool to study language processors we need to modify the W/AGE code, hence rebuilding the
native libraries is required so as to reflect the changes in the W/AGE code.

Another problem with Java RMI is security restrictions imposed while accessing remote applications. Also the RMI name server has to run continuously as a background process. This requires periodic maintenance and administration.

As Java was the choice to build the speech interface, CORBA applications inherit some of the problems explained above. Some additional issues to be taken care are as follows. CORBA objects are difficult to access beyond a firewall, to access the application through a firewall, http tunneling is required and not all the ORBs support http tunneling. It is difficult to convert the application into an Applet because web browsers do not support the ORBs. For example, Netscape supports only visibroker, HotJava supports Java ORB and so on. So in order to make the application "browser independent", the ORB requires to be bundled with the application, which makes the application a fat client and takes a longer time to download. Also Internet Explorer does not support jar files hence the application needs to be jarred for Netscape and HotJava browsers and in cabinet (cab) files for Microsoft Internet Explorer. To use the naming service and http tunneling, the name server and the http tunneling server
(OSAGENT and Gatekeeper in case of Visibroker) need to be continuously executing as processes so that the application can run smoothly. Exceptions should be caught effectively in the client so that it informs the user regarding the problems at the server end. Also periodic monitoring of the nameservice and the http tunneling service is required and not all the ORBs provide tools to monitor these services.

Considering the problems mentioned above, it was decided not to use Java RMI or CORBA to implement our application. The reason being that we need a simple method with minimal maintenance and monitoring at the server side. One of the choices was the telnet protocol. However the difficulty with the telnet protocol is that the users need to have a login and password so as to access the application and it is not secure to allow users with anonymous login and passwords. A prototype of telnet client was implemented in Java without a speech interface so as to study its use [Chitte, S, 1999].

We found that use of CGI-bin was an effective way to implement with our application. CGI-bin uses the HTTP protocol for communicating with remote applications. In our case this technique allows us to change the Miranda (W/AGE) programs without changing the C-wrapper thus adding flexibility to the application. At the same time
the client is not unduly constrained by security restrictions. Any computer on network can access the application without any additional requirement such as http-tunneling programs as in the case of CORBA. Also most web servers support CGI programming and the server-side applications can be written in Unix shell scripts, C, C++, Java, JavaScript, and Perl etc. Considering the above issues the best choice was CGI and hence the speech interface uses CGI/HTTP as the communication protocol.
6.0 Recognition Grammars

The IBM Speech recognizer is constrained by grammars to achieve recognition accuracy. Java speech Grammar Format (JSGF) defines a platform-independent and vendor-independent way of describing rule grammars. A rule grammar can be described in a text file with a ".gram" extension. Alternatively the rule grammar could be included in the Java source code.

The basic components of a rule grammar are grammar header and grammar body. The grammar header declares the grammar name and a list of imported rules and grammars. The grammar body defines the rules as combination of speakable text or reference to other rules.

The grammar name is either a simple grammar name or full grammar name. For an example "com.uwindsor.solar" this is an example of full grammar name "com.uwindsor" is the package name and solar is the simple grammar name. The basic grammar naming convention follows the naming convention for classes in Java programming Language.

Import declarations follow the grammar header. The import declarations are optional; they allow public rules of another grammar to be referenced locally.
Rules are combination of speakable text and reference to other rules each rule as a unique rulename and are case sensitive. A reference to a rule is represented by the rule's name surrounded by <>. As an example

Public <s> = <linkingvb> <jointermph> [<transvbph> by]
        <jointermph>;
<linkingvb> = is | was | are | were ;

In the example above <linkingvb> is a rule referenced by rule <s> and the tokens "is", "was", "are" and "were" are recognized as linking verbs. The "Public" qualifier indicates that this rule can be referenced within the rule definitions of another grammar by its fully qualified name, or with an import declaration. It can be used as an active rule for recognition which means that the recognizer uses this rule to determine what may be spoken. Also can be referenced by any public or non public rule defined in the same grammar.

A rule can be expanded to a token, a reference rule, reference to any public rule of another grammar which is being imported into current grammar or reference to a public rule of another grammar which is referenced with its fully-qualified rulename. Complex rules are defined by logical combinations of legal expansions using
composition, grouping, unary operators and attaching application specific tags.

Composition consists of rules formed by sequences, alternatives, and weights.
<simple> = ask them to be quiet;

This above rule is composed of a sequence of expansion
<statement> = this <somerule> is <good>; is also a legal sequence, in this case tokens are mixed with references to rules <somerule> and <good>.

<pnoun> = /10/ Earth | /1/ Jupiter | /7/ Mars | /6/Mercury;

Vertical bar characters separate a set of alternative expansions. Weights can be attached to the elements of a set of alternatives, higher the weight the more likely is that entry to be spoken. The above example indicates that earth is 10 times more likely to be spoken than Jupiter, thus providing some flexibility over selecting the elements from a set of alternatives.

Any legal expansion can be grouped using a matching set of parentheses'()'. For an example
<question> = (who | what)<joinverbph> ;
The question can be formed using who or what followed by a join verb. Grouping has higher precedence and so can be used to ensure correct precedence of rules. Square brackets may be placed around any rule definition to indicate that the contents are optional.

There are three unary operators that can be attached to any legal rule; they are Kleene star, the plus operator and tags. For an example

<command> = <polite>* compile;
<polite> = please;

will allow to say "please compile" or "please please compile" or "compile".

<command> = <polite>+ compile;
+ operator imposes the restriction that at least one form of polite is used. Looking at the example above it will allow "please please compile" however just "compile" is not legal.

Tag is an unary operator that can be attached to any legal rule expansion; tags do not affect the recognition of a grammar. If described in the grammar they are attached to the result object returned by the recognizer to the application. Tags provide a
mechanism to attach application-specific information to parts of rule definition. Curly braces '{ }' delimit tags. For an example

```java
public <s> = <linkingvb> <jointermph> [ <transvbph> by ] <jointermph> {sentence};
```

sentence is tag.

```java
void resultSpeech(ResultSet e) {
    // using tags
    FinalRuleResult resultSentence = (FinalRuleResult)(
        e.getSource());
    String tags[] = resultSentence.getTags();
    ResultToken tokens[] = resultSentence.getBestTokens();
    StringBuffer textSpoken = new StringBuffer();
    if (tags[0].equals("sentence")){
        do something ....
    }
    else if (tags[0].equals("echo")){
        do something else....
    }
}
```

The above code snippet shows the use of tags in the grammar definition. Depending on the tag returned by the recognizer a choice is made. This provides flexibility, and simplifies the writing of
applications by simplifying the processing of recognition results. Another use of tags is in internationalizing applications for example

\[
<message> = ( \text{message in English}) \{\text{hello}\};
\]  
\[
<message> = ( \text{message in French}) \{\text{hello}\};
\]  
\[
<message> = ( \text{message in German}) \{\text{hello}\};
\]  

As the tags remain same across all the languages hence simplifying the application software that processes the result obtained from the respective recognizers.

Recursion can be applied to grammar rules thus enabling very complex grammatical formats. Java Speech Grammar Format allows right recursion. In right recursion the rule refers to itself as the last part of its definition. Nested right recursion is also permitted. However left recursion and embedded recursion is not permitted.

The other major type of grammar is a dictation grammar. The naming convention for dictation grammar is similar as that of rule grammar. A dictation grammar typically has a large vocabulary, could be general purpose or specific (e.g. legal, medical), is built-into a recognizer, may support continuous or discrete speech. Applications can change
any part of a rule grammar however it is not the possible in case of a dictation grammar.
7.0 The Architecture of the prototype speech browser

The speech browser was designed based on the following requirements:

- The grammar file should be loaded dynamically from local or remote location. This allows the speech interface to be tailored for different applications.

- The end-user should be allowed to ask questions of the remote executable as per the constraints laid down by the grammar.

- Replies should be spoken back to the user. If the speech interface is to be used with SpeechNet-like applications and if the reply is a speech link then the interface should dynamically load the grammar for the new speech-URL, modify the voice as described in the link and query the new location pointed by the link [Frost 99, Frost and Chitte 99]. *SpeechNet is a network of speech-accessible hyperlinked objects that are deployed on the Internet, called as sihlos. Each sihlos contain an interpreter, a grammar and set of voice properties.*

- The speech interface should be able to recognize continuous user-independent speech.
7.1 Outline of the Prototype

The following figure shows the components of the interface.

![Diagram showing the components of the interface]

Fig 2.0 Components of Speech Browser.

The fig 2.0 shows the major components of the Speech Browser: the Synthesizer object, Recognizer object, Http object, and the Miranda W/AGE program wrapped in 'C'.
The speech interface was built using Java and the 'C' wrapper for Miranda was modified to suit the needs of this application. The recognizer and synthesizer engine of IBM via-voice was used. The speech input used a low-noise microphone and audio output was obtained via the sound card installed in the PC.

7.2 The Recognizer Object

This object initializes the via-voice recognizer engine and recognizes the input as constraint by the Grammar. The grammar-file is a text file with "gram" extension and is based on Java Speech Grammar Format.

The Grammar file is loaded either from the local machine or from the web. This information is obtained during the start up. Also the web address of the remote interpreter program is also obtained at the start-up.

For example if we are talking to the sihlo "solar man" then web address is http://www.cs.uwindsor.ca/user/r/richard/solar.so and the grammar file is at http://www.cs.uwindsor.ca/users/r/richard/solar.gram.
7.3 The Http Object

The Http object is responsible for fetching the grammar files, and the results of the queries. There are two Http objects HttpGet and HttpPost.

The HttpGet object is used to download the grammar file. The HttpGet object executes CGI get request on a shell script, passing the grammar file name as the parameter. The result is stored in a string object and returned to "load grammar" method of the recognizer object.

The HttpPost object is used to query the sihlo. The recognized text is used to construct a HttpPost query on the respective sihlo. The result obtained is stored in a string object. This string is then passed to speak method of the synthesizer. Also this string is appended to the display so that sighted users can read it.

7.4 The Synthesizer Object

The synthesizer object is initialized during the initialization of the interface. The resultant string obtained is passed to the speak method of the synthesizer object. The synthesizer object uses the sound card of the computer to speak out the result.
8.0 Details of the Browser Program Code

The components used in designing the speech client are Synthesizer, Recognizer and the communication protocol. The entire system consists of eleven Java classes. The Speechclient class is the backbone of the system; recognizer, synthesizer and communication objects are created also five inner classes are used. The inner classes basically consist of two action adapters; a speech listener; an audio listener and a data input class. The advantage of this design is that as the classes represent different functionality. If a section of the design needs to be changed, only the modules associated with that functionality need to be modified.

The Http protocol is used for communicating with the remote executable silho. The two methods HttpGet and HttpPost are implemented by the two classes HttpGet and HttpPost respectively. In order to make an Http call a reference to Http object is obtained and the get or post method are invoked. The advantage of creating separate objects is that in the future if the underlying communication needs to be changed, the other code of the speech client remains the same only the protocol objects need to be replaced.
The `DataInputFrame` object is used to obtain information of the remote location, grammar file required. Recognizer object is initiated in this class.

`SpeechClient` has all the necessary methods and object references to support the functionality of the interface. Detailed code along with comments is described in appendix C. As the Java Speech API is new a short description of the methods and objects used in the recognition and synthesis process follows.

The `javax.speech` package defines the classes and interfaces that define the basic functionality of a speech engine. The `javax.speech.synthes` is package and `javax.speech.recognition` adds specific capabilities to the basic functionality provided by `javax.speech` package. Speech synthesizer and Speech recognizer are instances of speech engine.

The basic steps in using a speech engine in a speech application are:

- Identifying the functional requirements such as the language or dictation capability required.
- Creating the engine that meets these requirements.
- Allocating the resources.
- Setting up the engine.
• Using the engine.
• Deallocating the resources of the engine.

Speech User Interfaces required to be designed for various languages and different modes of recognition. To cater for this requirement, Java Speech API provides the Engine Mode descriptor class, which defines the basic engine properties. The RecognizerModeDesc and SynthesizerModeDesc class defines additional specific properties for speech recognizers and synthesizers. Based on specific functional requirements, a speech engine can be selected, created and started.

Referring to the code in appendix C

Create a synthesizer object

SynthesizerModeDesc synthMode = new SynthesizerModeDesc();
synthMode.setLocale(Locale.ENGLISH);

/* create synth for english */
synth = Central.createSynthesizer( synthMode );
synthProp = synth.getSynthesizerProperties();
synthProp.setVoice(voice[0]);
/* make it ready to speak */
synth.allocate();
synth.resume();

Create a recognizer object

/* create a recognizer */
recog = Central.createRecognizer( new EngineModeDesc
(Locale.ENGLISH) );
recog.allocate();

In the above code snippet, synthesizer is created for the English language as specified by setLocale method of the synthMode object. Recognizer is created to recognize the English language.

After the recognizer is created and the necessary resources are allocated, recognition grammars are loaded and enabled. The grammar can be loaded from file on a local machine or from a URL. Next, the grammar is enabled. Once the recognizer is operational an enabled grammar is activated the recognizer is ready for recognition: that is the recognizer compares incoming audio to the active grammars and listens for speech that matches those grammars. The following code shows loading the rule grammar from a file and enabling the grammar.

void readLocalGrammar( Recognizer recog) {
    try{
        Reader reader = new FileReader(grammarFileName);
        rulegram = recog.loadJSGF(reader);
        rulegram.setEnabled(true);
    }
The next important step is catching the events generated by the recognizer and the synthesizer for processing information. Result event is processed to receive recognized text from a recognizer. The following code is a snapshot of Speech Listener class, which extends Result Adapter, the resultAccepted, and resultRejected methods, accept the Result event generated by the recognizer engine and later processed by the appropriate methods in the SpeechClient.

class SpeechListener extends ResultAdapter{
    SpeechClient adaptee;
    SpeechListener( SpeechClient adaptee ){
        this.adaptee = adaptee;
    }
    public void resultAccepted( ResultEvent e){
        adaptee.resultSpeech(e);
    }
    // if the result is rejected say
    public void resultRejected( ResultEvent e){
        adaptee.resultRejectedSpeech(e);
    }// end method result rejected
}//end class speech listener
The synthesizer and the recognizer engines are deallocated before exiting the application. The rest of the code is simple to understand as it is well commented.
9.0 Methods for "Wrapping" Remote Applications

As the communication protocol used is HTTP, the application at the server end should understand the protocol hence it is necessary to wrap the application at the server end in a wrapper code.

Wrapping the application in a 'C' wrapper or Unix shell script is an easy task. The application, which this browser talks to, is a Miranda program that is wrapped in a 'C' wrapper. The 'C' program interprets the HTTP commands, separates the question and passes the question to the Miranda program. The result obtained is sent back to the browser.

However if Java RMI is used as a protocol, the Miranda programs need to be accessed through Java Code using Java native methods. The Java program would call the 'C' program that in turn communicates with the Miranda code using Unix pipes. If applications are designed using CORBA the Miranda programs can be wrapped using any language that is supported by the ORB.
10.0 How grammars are used to specialize the prototype speech-browser

10.1 Downloading grammars from a remote location

As the speech recognition depends on the grammar that it uses, the recognition accuracy is poor for the default dictation grammar and also speaker profile is required for dictation grammar. In order to overcome this problem the speech browser uses a rule grammar which is application specific and can be downloaded from the remote location or made available locally. In case of SpeechNet, the grammar file is downloaded and used with a particular sihlo. The advantage of downloading these files from the remote location is that the user creating the sihlos can create the grammar files associated with each particular sihlo thus reducing the recognizing input language which also increases the recognition accuracy.

10.2 Improving recognition by modifying the grammar

Looking at the "solarman" grammar there are infinite number of questions, which can be asked. It is possible to ask the question does hall orbit? Which is a legal question according to the grammar rules, however as hall is a person, he will never orbit. A solution to this kind of problem is obtained by modifying the grammar rules by to
"represent" semantic constraints. For example the non-terminal \textit{termph} can be divided into different syntactic categories identifying planets, persons, and moons. This technique of dividing the existing rules into qualified syntactic categories reduces the search space of recognition, it reduces the number of possible questions which could be asked as compared to a generalized approach, however it leads to meaningful queries i.e. it improves the semantics [Frost 1995]. A drawback of this approach is that it may reduce the naturalness of spoken queries and the user should be aware the way in which the questions should be asked. Another drawback is that the grammars are bigger and take longer to download and process.
11.0 Investigating the use of the prototype speech browser

The prototype browser has been used to talk with sihlos [Frost et al., 1999]. Sihlos are distributed over SpeechNet, which is a collection of speech-accessible hyperlinked objects. SpeechNet is a network like the World Wide Web for visually challenged users.

Three sihlos are currently accessible through the SpeechNet, they are Solar-man, Monty and Judy. Solar man can answer queries regarding solar system. The following is the grammar loaded by the browser, when solar man is accessed:

```plaintext
grammar solar;
public <s> = <linkingvb> <termph> [ <transvb> by ] <termph>
    {sentence}
    |<quest1> <sent> {sentence ( who| what) <verbph>
    {sentence}
    |(which | how many ) <nouncla> <verbph> {sentence }
    |<simple>  {sentence};
```

```plaintext
<simple> = | ask them to be quiet
        | please introduce yourself
        | hello there
        | hello solar man
```
| goodbye
| goodbye solar man
| fine thanks
| thanks
| thanks solar man
| yes please
| what is your name
| who are you
| where do you live
| what do you know
| how old are you
| who made you
| what is your favorite band
| who is the vice president at the university of windsor
| who is the president at the university of windsor
| who is the executive dean of science at the university of windsor
| who is the dean of science at the university of windsor
| tell me a poem
| know any poems
| tell me a joke
| know any jokes
who is judy

I can I talk to judy

who is monty

I can I talk to monty;

<sent> = <termph> <verbph>;
<termph> = <pnoun> | <detph>;
<verbph> = <transvbph> | <intransvb> ;
<transvbph> = (<transvb> | <linkingvb> <transvb> by ) <termph>;
<detph> = <det> <nouncla> ;
<nouncla> = <adj> <cnoun>|<cnoun> ;
<cnoun> = man | men | person | planet | planets | moon | moons;
<adj> = red ;
<intransvb> = spin | spins | orbit | orbits | orbited | exist ;
<det> = a | an | every | one | two | three | any | all ;
<pnoun> = Earth | Jupiter | Mars | Mercury | Neptune | Pluto |
Saturn | Uranus | Venus | Almathea | Ariel | Callisto |
Charon | Deimos | Dione | Enceladus | Europa |
Ganymede | Hyperion | Iapetus | Io | Janus |
Jupiter Eight | Jupiter Eleven | Jupiter Fourteen | Jupiter Ninth |
Jupiter Seventh | Jupiter Sixth
Pickering;

<transvb> = orbit | orbits | orbited | discover | discovered;

<linkingvb> = is | was | are | were;

<quest1> = did | do | does;

public <bye> = Good bye and go to sleep {bye};

As we see from the above grammar that we can query about solar system and also some questions which are defined by <simple> rule. Following are examples of some queries asked, recognized and the response obtained.

U: hello there
R: hello there
C: good day to you

U: how many planets exist
R: how many planets exist
C: 9

U: *which red moon orbits Jupiter*

R: *which red moon orbits Jupiter*

C: IO

U: *how many planets are orbited by a moon*

R: *how many planets are orbit by a moon*

C: 7

U: *can I talk to monty*

R: *can I talk to monty*

C: *yes, here he is*

After asking the question "can I talk to monty" the response is "yes here he is", then the system switches to the monty sihlo. The solar man sihlo returns a link to monty with the reply. The speech interface then deletes the grammar associated with solar man and loads the grammar for the monty sihlo. So we can ask questions which are appropriate for the monty sihlo. Similarly we can switch to judy or solar man. For grammar and questions asked to monty, judy and solar man refer to appendix B.
As we see from the grammar of solar-man sihlo there are numerous possible questions which one can ask, if the users ask a question which obeys the grammar but the sihlo does not understand, then the following message is replied "Do not know yet will work on it tonight". Which informs the user that the sihlo did not understand the query.

The recognition was accurate except for words that sound phonetically similar, for an example "good bye" and "go back". The recognizer also misrecognized "orbited by" (orbit by) as indicated in one of the above question in the sample session. In one of the sessions the recognizer misrecognized noise generated on a wireless microphone as "clean".

Thus an important recommendation for writing the grammar is to avoid phonetically similar words in the command language. However it does not imply that phonetically similar words should be eliminated from the grammar. Java Grammar Specification Language, which is used for designing the recognition grammar, provides amicable solution to this problem. Weights in form of floating point numbers are attached to the elements of a set of alternatives to indicate the probability that each alternative will be spoken.
For an example
\[
<Somerule> = /10/ \text{ go back} | /1/ \text{ good bye};
\]
Weights are placed before each item in set of alternatives; it is a floating-point number surrounded by forward slashes. In the example above, it is indicated that "go back" is ten times more likely to be spoken than "good bye". Rules to be followed when specifying weights are as follows:

Weight must be specified for every items in a set of alternatives or for none i.e. all or nothing rule. Weights are floating point numbers. Only floating point number and whitespace is allowed within the slashes. Weight can be zero or greater. A zero weight indicates that the item can never be spoken. At least one non-zero positive weight is required. Guessing appropriate weights is difficult. Weights should be obtained by a study of the speech and application requirement.

A few users have experimented with the speech interface. All the users were computer science students. As the users were not familiar with the speech interface a common question was "what to ask".

Designing a speech activated help menu can rectify this. However in our case after explaining which questions could be asked, users were
comfortable with the three silhos. Some of the users suggested having a list of samples questions prompted by the interface.

Users found the system difficult to control at the beginning, as they were not familiar with the questions they could ask, occasionally they got frustrated because the recognizer could not understand them and kept on saying "please repeat I do not understand". However after they became familiar with the system, it was observed that recognition was very accurate.

Users could understand the synthesizer-spoken responses, as the output was very clear and easily understandable. Users could recognize the difference between the voice types, female in the case of judy and male voice in the case of solar-man and monty. A few users suggested improved control on the synthesizer-spoken output for an example a pause command to stop the synthesizer temporarily; abort to abort the response and resume to resume the response. This could be achieved easily but it was not implemented, in the prototype owing to lack of time.

From our experience it can be concluded that the IBM's implementation of Java speech API, and Via-Voice recognition and synthesis engine proves to a effective solution for developing low-cost
speech interfaces, as well as speech enabling existing applications. Java speech API is easy to use and no extra training is required for Java Programmers.
12.0 Analysis of results

The prototype has been successfully implemented using IBM's implementation of Java Speech API that is available from: http://www.alphaworks.ibm.com/formula.

The prototype browser application is able to recognize speaker-independent continuous speech, however it misinterprets some words that have close phonetic resemblance. For example "go back" and "good bye". The application developer has to take care while designing the rule grammar by careful rephrasing.

Several users have used the interface using their normal rate of speech, and the recognizer was able to recognize their speech, thus the application is an example of continuous speech and speaker independent recognition.

Some users initially found the application difficult to control owing to the fact that they were not familiar with the questions to be asked and the expected response.

This application uses the built-in sound card of the system; the synthesizer-spoken output was very clear and easily understandable. It did not appear to be robotic or choppy. In general the end-user was
able to grasp the functionality of the user interface within 5-10 minutes of use.

12.1 Modifications to the Speech Interface

From the experiment, we have noted possible future modifications:

- The application can be easily modified to use multiple grammar files at the same time, which would be useful to support additional functionality.

- The application was designed for visually-challenged users, however it could be modified for sighted users by adding a rich set of graphical user interface which can be used using speech to provide hands free operation.

- The recognizer could be disabled while the synthesizer is active so as to avoid any unwanted "feedback" recognition. However having the recognizer active during synthesis did not cause any misfires but deactivating the recognizer would be required in a loud environment.

- The user should have control to cancel, pause or resume speech synthesis.
The user should have a control over the spoken rate some users may feel that the spoken rate is fast or slow.
13.0 Overview of Related Work

Other relevant work and on-going projects in the field of speech-recognition technology are described below. However their use and purpose of design is different from our design. Also not much work other than that reported here is being done in speech-access to remote applications deployed on the Internet.

[Nicole Yankelovich et al. 1995] SpeechActs is a research prototype developed by the Speech Applications Group in the period 1993-1997 at Sun Labs (SunMicro Systems labs) it is used as testbed for developing spoken natural language applications. It enables software developers to build conversational speech applications. This system is used to study various issues in constructing speech user interfaces. Use of telephone provides access to remote applications.

[Savitha Srinivasan. 1999] did work on an object-oriented framework to design speech-recognition applications. This framework was developed as part of the Medspeak/Radiology™ product with John Vergo at IBM T.J. Watson Research Center. IBM MedSpeak/Radiology™ is a speech recognition solution that enables radiologists to create their own diagnostic text reports by simply speaking aloud. MedSpeak/Radiology to dramatically reduce the time
and cost of radiology report generation. This is a commercial product
designed to only for radiologist.

BrookesTalk is a web browser for visually-challenged users impaired
which aims to support users searching the Web for information. It
allows the users to scan Web pages and sites, one of the advantages is
that blind user, and visually impaired and sighted users can use the
application [http://www.brookes.ac.uk/schools/cms/research/speech/
btalk.htm].

Simply Web 98 is a speech enabled accessible web browser that allows
easy navigation of complex pages by visually-challenged users. It uses
Internet Explorer as its engine. It includes a software-based speech
synthesizer and can be used in standalone mode as a "Talking Web
Browser" [http://econointl.com/sw/].

[Josh Bers et al. 1997] Voicelog prototype system that combines pen
and speech input from the on-line user in a web-browser. This system
allows one to obtain vehicle diagrams and order specific parts using
these diagrams. This is an example of multimodal interface to
applications running on web server. It features a novel client-server
approach to speech recognition, modular reusable components and a
simple Java-based interface.
[Arons B 1991] Hyperspeech: Navigating in Speech-Only Hypermedia. "This work the user is allowed to navigate by voice through a database of recorded speech without any visual clues. The ideas being developed can be applied to create a generalized form of interaction with unstructured speech data. Applications for such a technology include the use of recorded speech, rather than text, as a brainstorming tool or personal memory aid. A hyperspeech system would allow a user to create, organize, sort, and filter "audio notes" under circumstances where a traditional graphical interface would not be practical".

[Haddad T., 1997] A web browser developed by Mr. Haddad at the Department of Computer Science, at The University of Windsor. This browser accepts user-independent continuous-speech commands and responds back to the user using a text-to-speech synthesizer. This non-visual web browser uses PE-500 speech recognizer from SSI technology and a Dectalk text-to-speech synthesizer. The grammar associated for its use is packaged within the program code. The navigation method is based on document content rather than its layout. Other techniques such as use of auditory clues and spatial layout in non-visual web browsing are discussed.
14.0 Conclusion and Future Work

14.1 What has been achieved

1. The new idea's speech accessible links and downloading the grammar associated with the sihio's seems to be effective in terms of recognition accuracy and traversing the sihlos. However it is the programmers responsibility to provide links to other sihlos in there code.

2. In this work a speech interface was constructed using Java, which is a popular object-oriented language, and speech recognition and synthesis software used is low cost software readily available. However it was beyond the scope of this thesis work to conduct a detail study of conversational dialog between Humans and Machine. A number of shortcomings were identified and modifications were proposed for constructing speech interfaces.

3. A speech browser was constructed and used with remote applications. This demonstrates that a low-cost, user-independent continuous speech interface to remote applications can be constructed thus satisfying the needs identified in the problem statement.
14.2 Other findings

- The speech browser was constructed using the IBM's implementation of the Java Speech API. This implementation is available for download from the IBM's website [http://alphaworks.ibm.com/formula/speech].

- Graphical interfaces provide good feedback to users however in case of speech-only interfaces the user does not know what is happening whether his request is taken care of or ignored. This often confuses users. To ensure a feedback to the user auditory clues can be used.

- Recognition accuracy depends on the size of the language used. It decreases as the size of the language increases. A way to overcome this problem is to have your own application specific language. As in case of the SpeechNet project the sihlos are application specific for an example "solar-man" can answer only questions about solar system, "judy" can only recite poems. As the application grammar is associated with each sihlo that is downloaded every time the browser points to that particular sihlo, the recognizer's language is restricted only with the grammar defined by the application, thus improving the recognition accuracy.
• Like graphical user interface speech interfaces should also provide a facility to undo, this can be achieved by designing reusable dialog components similar to those used in graphical user interface. For example yes/no dialog box. These components will be used to prompt to the user and obtain user's confirmation before any critical action is performed.

• Speech technology as compared with keyboard and mouse is considered to be immature. Due to misrecognitions it is assumed that the technology is immature. However understanding and avoiding errors using keyboard and mouse is easy as compare to speech interface. For an example if the user accidentally clicks the right mouse button but was suppose to click left mouse button, this can be justified by saying that my finger just hit the neighboring button but source of error in speech recognition is much harder to diagnose. One more reason for speech interfaces not being popular is due to the fact that most operating system software is not designed to accept speech input.
14.3 Further work identified

From the design point of view, the speech browser could be easily modified to recognize other languages supported by the IBM via-voice speech engine. Also a sihlo could be built to browse regular html pages. Also this browser can be extended to provide a dialog between human and computers.

One of the related projects is "SpeechNet", which is a collection of speech-accessible hyperlinked objects; the browser is used as a speech interface to these objects, hence this browser would be helpful in investigating different methods for construction and deployment of sihlos.

An investigation of what rules that should guide the designer of a sihlo to incorporate speech-activated hyperlinks to other sihlos can be studied. The example session illustrated the use of very simple speech prompts: the input "Can I talk to Solar man?" caused the sihlo to send the data to the speech browser causing it to change to the solar man sihlo, download solar man's grammar. Also rules to inform the user's about existence of other sihlo's can be studied. In the example session the user was alerted to solar man's existence through the earlier response "Older than you think but younger than my friend solar man".
A detailed investigation of different aspects of human-computer interaction can be studied for an example of how users react to different situations such as misrecognition, different synthesized voices, rate of spoken text and provide feedback to the designer so that robust reusable conversational interface can be constructed.

A detailed investigation on the use of speech interfaces in computer assisted learning for users with physical disabilities or repetitive stress injury can be studied.

In this implementation when the user switches to a different sihlo using the speech link, the recognizer is reconfigured for the grammar associated with the new sihlo. The recognizer now recognizes queries for the new sihlo. However the recognizer can be configured for multiple grammars and then the query recognized is send to appropriate sihlo.

14.4 Speech applications in the future

In a few years from now there will be tremendous growth in speech interfaces to various applications. There will be a need to make the existing applications "speech enabled". As of today, few speech
browsers exist which are used to browse html documents on the Internet. Speech interfaces for E-commerce, Airline reservations, Airline schedules would be more useful and will be used to replace existing automated answering systems.

As the speech interface is constructed using Java, it will be easier to integrate this browser with home appliances for example with Television set and control all the features with voice.

Computer based training and multimedia software will also benefit from speech interfaces. Automated answering systems can use speech recognition and application specific grammars so that the users can speak into rather than entering information from keypads.

Open standards are been set for speaker identification and verification. Speaker identification and verification systems could be used for secured transactions using computers. Operating systems in future will also support speech input and output.
References:


[Chitte, S, 1999] Telnet Protocol and Implementation of Telnet client using Java. 60-590 Directed Study at Department of computer science at The University of Windsor.


Appendix A-Java Speech API

In the early nineties IBM, and Dragon systems announced that they were working for API for speech technology. Later Speech Recognition committee was formed by WordPerfect and Novell. The SRAPI [http://www.srapi.com] committee is involved not only in setting standards for speech recognition but also speech synthesis, command technology, speaker verification and identification.

As the choice of using speech application is based on several factors such as development platform, development environment, whether you will use telephony. Java Speech API is the most versatile API as it supports Java theme "write once run anywhere" also it can be integrated with telephony.

Java speech API allows concurrent use of speech recognizers and synthesizers of different languages provided they support the JSAPI implementation. Currently only IBM has implemented Java Speech API for Via-Voice speech recognition and synthesizer engine. Java Speech API enables speech input and output for PCs, workstations, computer telephony systems and small personal devices. It provides cross platform cross vendor portability.
Sun has also released Java Speech Markup Language, which uses a standard markup style similar to HTML for effective use of synthesizers. In order to ensure cross-vendor cross platform control of speech recognition Sun and its partners developed Java Speech Grammar format. JSGF defines how to write and interpret grammars.

Java Speech API is a collection of objects consisting of three packages:

javaspeech
javaspeech.synthesis
javaspeech.recognition

javaspeech is the root package that defines the features of a generic speech engine. The speech.synthesis and speech.recognition define features of synthesizers and recognizers. Details of Java Speech API are available at http://java.sun.com/products/java-media/speech.
Appendix B- Program Listing

*SpeechClient.java*

```java
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.util.*;
import javax.speech.*;
import javax.speech.recognition.*;
import javax.speech.synthesis.*;
import java.util.Locale;
import HTTPget;
import HTTPpost;
import LocInfo;

public class SpeechClient extends Frame {

    /** class variables */
    Synthesizer synth;
    Recognizer recog;
    RuleGrammar rulegram;
    SynthesizerProperties synthProp;
    Voice[] voice = new Voice[2];
    public String webAddress;
    public String cgiboxFileChair;
    public String grammarFileName;
    public String lastTextSpoken;
    public String delimiterValue = "=";
```
public boolean flagLocal = false;
public boolean flagFemaleVoice = false;
public boolean flagMaleVoice = false;

/** History object Loc Info Objects are stored */
Vector history = new Vector();

/** GUI variables */
GridBagLayout gridBagLayout = new GridBagLayout();
GridBagConstraints gbCon = new GridBagConstraints();
GridBagConstraints gbConBtnPanel = new GridBagConstraints();
BorderLayout layoutDisplay = new BorderLayout();

/** Layout for button Panel */

GridBagLayout layoutButtonPanel = new GridBagLayout();

/** Text components */

TextField textFieldQuery = new TextField();
TextArea textAreaDisplay = new TextArea("", 5, 70,
TextArea.SCROLLBARS_VERTICAL_ONLY);
TextField textFieldStatus = new TextField(15);

/** Label for the frame */
Label title = new Label("Speech Interface");

/** Panels used */
Panel panelStatus = new Panel();
Panel panelButton = new Panel();
Panel panelDisplay = new Panel();
/** Buttons used */

Button buttonConnect = new Button();
Button buttonStop = new Button();
Button buttonPause = new Button();
Button buttonResume = new Button();

/** constructor for speech client */

public SpeechClient(){
    try{
        /** initialise GUI */
        initGUI();
        /** create synthesizer */
        createSynth();
        }catch(Exception e){
        e.printStackTrace();
        System.err.println( "Error in the constructor" );
        }
    }

/** method initialises the GUI */
private void initGUI() throws Exception{

/** add the layout */
this.setSize(new Dimension(550,450));
this.setBackground(new Color(190, 190, 190));
this.setForeground(new Color(0, 0, 0));
this.setLayout(gridBagLayout);

/** add layout for the panels */
panelDisplay.setLayout(layoutDisplay);
panelButton.setLayout(layoutButtonPanel);
textFieldQuery.addActionListener(new
SpeechClient_textFieldQuery_actionAdapter(this));

buttonConnect.setLabel( "connect" );
buttonConnect.addActionListener(new
SpeechClient_buttonConnect_actionAdapter(this));

buttonStop.setLabel( "Stop Speech" );
buttonStop.addActionListener(new
SpeechClient_buttonStop_actionAdapter(this));

buttonPause.setLabel( "Pause Speech" );
buttonPause.addActionListener(new
SpeechClient_buttonPause_actionAdapter(this));

buttonResume.setLabel( "Resume Speech" );
buttonResume.addActionListener(new
SpeechClient_buttonResume_actionAdapter(this));

/** constraints for display panel */
gbCon.fill = GridBagConstraints.HORIZONTAL;
gbCon.insets = new Insets(2,2,2,2);
gbCon.gridx = 0; gbCon.gridy = 0; gbCon.gridwidth=0;
gbCon.gridheight=1;
gbCon.weightx=gbCon.weighty=0.0;
this.add( title, gbCon);

gbCon.fill = GridBagConstraints.BOTH;
gbCon.gridx = 0; gbCon.gridy = 1; gbCon.gridwidth=2;
gbCon.gridheight=4;
gbCon.weightx=gbCon.weighty=1.0;
this.add( panelDisplay , gbCon );

gbCon.fill = GridBagConstraints.VERTICAL;
gbCon.gridx = 3; gbCon.gridy = 1; gbCon.gridwidth=2;
gbCon.gridheight=4;
gbCon.weightx=gbCon.weighty=0.0;
this.add( panelButton );

gbCon.fill = GridBagConstraints.HORIZONTAL;
gbCon.gridx = 0; gbCon.gridy = 6; gbCon.gridwidth=0;
gbCon.gridheight=1;
gbCon.weightx=gbCon.weighty=0.0;
this.add( panelStatus, gbCon );

panelStatus.add(textFieldStatus);

/ ** buttonpanel  */
gbConBtnPanel.fill = GridBagConstraints.BOTH;
gbConBtnPanel.insets = new Insets(1,1,1,1);
gbConBtnPanel.gridx=0; gbConBtnPanel.gridy=2;
gbConBtnPanel.gridwidth=2;gbConBtnPanel.gridheight=1;
gbConBtnPanel.weightx=gbConBtnPanel.weighty= 0.0;
panelButton.add( buttonConnect , gbConBtnPanel );

gbConBtnPanel.gridx=0; gbConBtnPanel.gridy=3;
gbConBtnPanel.gridwidth=2;gbConBtnPanel.gridheight=1;
gbConBtnPanel.weightx=gbConBtnPanel.weighty= 0.0;
panelButton.add( buttonStop , gbConBtnPanel );

78
gbConBtnPanel.gridx=0; gbConBtnPanel.gridy=4;
gbConBtnPanel.gridwidth=2;gbConBtnPanel.gridheight=1;
gbConBtnPanel.weightx=gbConBtnPanel.weighty= 0.0;
panelButton.add(buttonPause , gbConBtnPanel);

gbConBtnPanel.gridx=0; gbConBtnPanel.gridy=5;
gbConBtnPanel.gridwidth=2;gbConBtnPanel.gridheight=1;
gbConBtnPanel.weightx=gbConBtnPanel.weighty= 0.0;
panelButton.add(buttonResume , gbConBtnPanel);

/** add the text area and the text field */
panelDisplay.add(textAreaDisplay);
panelDisplay.add(textFieldQuery,"South");

}

/** create synthesizer method */
void createSynth(){
 try{
 /** create voice object for male and female voice types */
 voice[0] = new Voice();
 voice[0].setGender(Voice.GENDER_MALE);
 voice[0].setAge(Voice.AGE_MIDDLE_ADULT);
 SynthesizerModeDesc synthMode = new
 SynthesizerModeDesc();
 synthMode.setLocale(Locale.ENGLISH);
 /** create synth for english */
 synth = Central.createSynthesizer( synthMode );
 synthProp = synth.getSynthesizerProperties();
 synthProp.setVoice(voice[0]);
 /** make it ready to speak */

79
 synth.allocate();
synth.resume();
} catch (Exception e) {
    e.printStackTrace();
}

/**
call this method when ever a grammar file is loaded */
void startString() {
    String question = "question=" + "startstring".trim();
    System.out.println(question);
    HTTPpost postRequest = new
    HTTPpost(webAddress, cgibinFileName, question);
    String result = postRequest.getResults();
    System.out.println("\n");
    textAreaDisplay.appendText(result);
    textAreaDisplay.appendText("\n");
    speak(result);
}

/**
method to read local grammar */
void readLocalGrammar( Recognizer recog) {
    try {
        Reader reader = new FileReader(grammarFileName);
        rulegram = recog.loadJSGF(reader);
        rulegram.setEnabled(true);

        /** create info object and add to history */
        LocInfo loc = new LocInfo (webAddress,
cgiFile, grammarFile

history.addElement(loc);
}
catch(Exception readlocal)
{
    readlocal.printStackTrace();
    System.err.println("Exception in read local grammar");
}

/** method to read remote grammar */
void readRemoteGrammar(Recognizer recog)
{
    try{
        HTTPget get = new HTTPget("www.cs.uwindsor.ca", 80);
        get.submit("/cgi-bin/richard/h.sh", grammarFile);
        StringReader strReaderGrammar = new StringReader(get.result());
        rulegram = recog.loadJSGF(strReaderGrammar);
        System.out.println(get.result());
        rulegram.setEnabled(true);
        recog.commitChanges();

        /** create info object and add to history */
        LocInfo loc = new LocInfo(webAddress,
                                  cgiFile, grammarFile);
        history.addElement(loc);
    }catch(Exception gram){
        System.err.println("Grammar file exception unable to get from URL");
        gram.printStackTrace();
    }
}
/** method to initialise recogniser */
void initRecogniser(){
    try{

        /** create a recognizer */
        recog = Central.createRecognizer(
            new EngineModeDesc(Locale.ENGLISH));
        recog.allocate();

        /** load the default grammer */
        if (flagLocal){
            readLocalGrammar(recog);
        }else{
            readRemoteGrammar(recog);
        }
        recog.addResultListener( new SpeechListener( this));
        RecognizerAudioListener audioListn =
        new AudioRecoListencr(this);
        recog.getAudioManager().addAudioListener( audioListn );
        recog.commitChanges();
        recog.requestFocus();
        recog.resume();
    }catch(Exception initRecog){
        initRecog.printStackTrace();
    }
}

/** generic method to get the response */
void answerQuestion ( ResultToken tokens[], StringBuffer textSpoken ){

for ( int i = 0; i < tokens.length; i ++ ){
    textSpoken.append( tokens[i].getSpokenText() + " ");
    System.out.print(tokens[i].getSpokenText()+" ");
}

textFieldQuery.setText(""");
textFieldQuery.setText(textSpoken.toString());
lastTextSpoken = textSpoken.toString();
String question = "question=" + 
textFieldQuery.getText().trim();
System.out.print(question);
HTTPpost postRequest = new 
HTTPpost(webAddress, cgibinFileName, question);
String result = postRequest.getResults();
System.out.println( result );
textAreaDisplay.appendText(result);
textAreaDisplay.appendText("\n");
speak(result);
}

public String valueString( String token , String delimiter ){
    StringTokenizer tokensNameValue = new StringTokenizer( 
        token, delimiter);
    String Name = tokensNameValue.nextToken();
    String Value = tokensNameValue.nextToken();
    return Value;
}

public String checkGrammarChange( String result ){

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
recog.deleteRuleGrammar( delGrammar[j]);
recog.commitChanges();
}
readRemoteGrammar(recog);
recog.commitChanges();
} catch( Exception delGram){
    System.err.println( "Exception deleting Grammar");
    delGram.printStackTrace();
}
if (vType.toString().equalsIgnoreCase("female")){
    flagFemaleVoice = true;
    flagMaleVoice = false;
} else {
    flagFemaleVoice = false;
    flagMaleVoice = true;
}
return result1.toString();
}
return result;
}

void textFieldQuery_actionPerformed(ActionEvent e){
    String question = "question=" + textFieldQuery.getText().trim();
    HTTPpost postRequest = new
    HTTPpost(webAddress, cgbinFileName, question);
    String result = postRequest.getResults();
    System.out.println(result);

    /** append the text Area */
textAreaDisplay.appendText(result);
speak(result);
}

void buttonConnect_actionPerformed(ActionEvent e){
    DataInputFrame getInfo = new DataInputFrame(this);
    Dimension getInfoSize = getInfo.getPreferredSize();
    Dimension frmSize = getSize();
    Point loc = getLocation();
    getInfo.setLocation((frmSize.width - getInfoSize.width)/2 + loc.x,
                        (frmSize.height - getInfoSize.height)/2 + loc.y);
    getInfo.show();
    buttonConnect.setEnabled(false);
}

void buttonStop_actionPerformed(ActionEvent e){
    System.out.println("Inside button stop action performed");
}

void buttonPause_actionPerformed(ActionEvent e){
    System.out.println("Inside button Pause action performed");
    synth.pause();
}

void buttonResume_actionPerformed(ActionEvent e){
    System.out.println("Inside button Resume action performed");
    try{
        synth.resume();
    }catch (Exception exRes){
        exRes.printStackTrace();
    }
}
void resultSpeech(ResultSet e) {
    FinalRuleResult resultSentence = (FinalRuleResult)(e.getSource());
    String tags[] = resultSentence.getTags();
    ResultToken tokens[] = resultSentence.getBestTokens();
    StringBuffer textSpoken = new StringBuffer();
    if (tags[0].equals("sentence")) {
        for (int i = 0; i < tokens.length; i++) {
            textSpoken.append(tokens[i].getSpokenText() + " ");
            System.out.print(tokens[i].getSpokenText() + "");
        }
    }
    textFieldQuery.setText(" ");
    textFieldQuery.setText(textSpoken.toString());
    lastTextSpoken = textSpoken.toString();
    /** this is for echo command */
    String question = "question=" +
            textFieldQuery.getText().trim();
    HTTPpost postRequest = new
            HTTPpost(webAddress,cgibinFileName.question);
    String result = postRequest.getResults();
    System.out.println ( result );
    /** pass the result to string processing check for any
     * change in grammar file
     */
    String result1 = checkGrammarChange( result );
    textFieldDisplay.appendText(result1);
    textFieldDisplay.appendText("\n");
}
speak(result1);

/** change the voice if the flag is set true */
if (flagFemaleVoice){
    synthProp = synth.getSynthesizerProperties();
    try{
        voice[1] = new Voice();
        voice[1].setGender(Voice.GENDER_FEMALE);
        voice[1].setAge(Voice.AGE_DONT_CARE);
        synthProp.setVoice(voice[1]);
    }catch (Exception femVoice){
        System.err.println ("Set Female voice failed ");
    }
}
else if(flagMaleVoice){
    try{
        Voice maleVoice = new Voice();
        maleVoice.setGender(Voice.GENDER_MALE);
        maleVoice.setAge(Voice.AGE_MIDDLE_ADULT);
        synthProp.setVoice(maleVoice);
    }catch(Exception femVoice){
        System.err.println ("Set Female voice failed ");
    }
}
else if (tags[0].equals("smallquestion")){
    answerQuestion( tokens, textSpoken );
}else if (tags[0].equals("back")){
    LocInfo lastInfo = (LocInfo) history.lastElement();

    /** set the web address and cgibin variable */
    webAddress = lastInfo.getWebAddress();
    cgibinFileName = lastInfo.getcgibinFile();
}

88
grammarFileName = lastInfo.getGram();
System.out.println (webAddress + cgibinFileName +
grammarFileName);
} else if (tags[0].equals("abort")){
try {
    synth.deallocate();
    synth.allocate();
} catch (Exception dealloc) {
    dealloc.printStackTrace();
}
}
} else if (tags[0].equals("pause")){
synth.pause();
}
} else if (tags[0].equals("resume")){
try{
    synth.resume();
} catch (Exception resume){
    resume.printStackTrace();
}
}
} else if (tags[0].equals("bye")){
    String s = " ";
    for (int i = 0; i< tags.length ; i ++ ){
        textSpoken.append( tags[i] + " ");
    s += textSpoken.toString();

    /** set output to text area */
    textAreaDisplay.appendText(s);

    /** set output to the speaker */
    speak(s);
    try{
        synth.waitEngineState(Synthesizer.QUEUE_EMPTY);

89
void speak ( String s ) {
    if ( synth != null ) {
        try {
            recog.pause();
            synth.speak( s, null);
            recog.resume();
        } catch ( Exception speakExep ) {
            speakExep.printStackTrace();
        }
    } else
        System.out.println(s);
}

void resultRejectedSpeech(RESULT e) {
    String rejResult = " Please repeat I do not understand ";
    speak(rejResult);
}

void audioDisplay ( RecognizerAudioEvent e ) {
    textFieldStatus.setText(" ");
    textFieldStatus.setText( "Volume:" + e.getAudioLevel());
}
*/

class SpeechClient_textFieldQuery_actionAdapter implements java.awt.event.ActionListener{
    SpeechClient adaptee;
    SpeechClient_textFieldQuery_actionAdapter(SpeechClient adaptee){
        this.adaptee = adaptee;
    }
    public void actionPerformed(ActionEvent e) {
        adaptee.textFieldQuery_actionPerformed(e);
    }
}

class SpeechListener extends ResultAdapter{
    SpeechClient adaptee;
    SpeechListener( SpeechClient adaptee ){   
        this.adaptee = adaptee;
    }
    public void resultAccepted( ResultEvent e){
        adaptee.resultSpeech(e);
    }
    public void resultRejected( ResultEvent e){
        adaptee.resultRejectedSpeech(e);
    }
}

}//end class speech listener

class AudioRecoListener extends RecognizerAudioAdapter{

91
SpeechClient adaptee;
AudioRecoListener (SpeechClient adaptee){
    this.adaptee = adaptee;
}
public void audioLevel( RecognizerAudioEvent e ){
    adaptee.audioDisplay ( e );
}

} // end class audio listener

class SpeechClient_buttonConnect_actionAdapter implements java.awt.event.ActionListener{
    SpeechClient adaptee;
    SpeechClient_buttonConnect_actionAdapter(SpeechClient adaptee){
        this.adaptee = adaptee;
    }
    public void actionPerformed(ActionEvent e) {
        adaptee.buttonConnect_actionPerformed(e);
    }
}

class SpeechClient_buttonStop_actionAdapter implements java.awt.event.ActionListener{
    SpeechClient adaptee;
    SpeechClient_buttonStop_actionAdapter(SpeechClient adaptee) {
        this.adaptee = adaptee;
    }
    public void actionPerformed(ActionEvent e) {
        adaptee.buttonStop_actionPerformed(e);
    }
}
class SpeechClient_buttonPause_actionAdapter implements java.awt.event.ActionListener{
    SpeechClient adaptee;
    SpeechClient_buttonPause_actionAdapter(SpeechClient adaptee) {
        this.adaptee = adaptee;
    }
    public void actionPerformed(ActionEvent e) {
        adaptee.buttonPause_actionPerformed(e);
    }
}

class SpeechClient_buttonResume_actionAdapter implements java.awt.event.ActionListener{
    SpeechClient adaptee;
    SpeechClient_buttonResume_actionAdapter(SpeechClient adaptee) {
        this.adaptee = adaptee;
    }
    public void actionPerformed(ActionEvent e) {
        adaptee.buttonResume_actionPerformed(e);
    }
}

class DataInputFrame extends Frame implements java.awt.event.ActionListener{
    SpeechClient adaptee;
    JPanel displayPanel = new JPanel();
Button okButton = new Button();
Button cancelButton = new Button();
Choice locationChooser = new Choice();
Label labelWebAddress = new Label("WebAddress");
Label labelCgi_bin = new Label("ExecutableFile");
Label labelGrammarFile = new Label("GrammarFile");
Label labelFileLocation = new Label("GrammarFileLocation");
TextField txtWebAddress = new TextField(25);
TextField txtCgi_bin = new TextField(25);
TextField txtGrammarFile = new TextField(25);
GridLayout gridlayoutDIF = new GridLayout();

/** constructor */
DataInputFrame (SpeechClient adaptee) {
try{
  this.adaptee = adaptee;
  init();
}catch (Exception e){
  c.printStackTrace();
}
pack();
}
private void init(){
  this.setTitle("Information of Remote Executable ");

  gridlayoutDIF.setRows(6);
  gridlayoutDIF.setColumns(2);
  displayPanel.setLayout(gridlayoutDIF);

  okButton.getLabel("OK");
  okButton.addActionListener(this);
cancelButton.setLabel("Cancel");
cancelButton.addActionListener(this);
this.add(displayPanel, null);
displayPanel.add(labelWebAddress);
displayPanel.add(txtWebAddress);
txtWebAddress.setText("www.cs.uwindsor.ca");

displayPanel.add(labelCgi_bin);
displayPanel.add(txtCgi_bin);
txtCgi_bin.setText("cgi-bin/richard/solar.so");

displayPanel.add(labelGrammarFile);
displayPanel.add(txtGrammarFile);
txtGrammarFile.setText("solar");

displayPanel.add(labelFileLocation);

locationChooser.add("Local");
locationChooser.add("Remote");
displayPanel.add(locationChooser);

displayPanel.add(okButton);
displayPanel.add(cancelButton);
pack();
}

public void actionPerformed(ActionEvent e){
    if (e.getSource() == okButton){

        /** process the information obtained */
        adaptee.webAddress = txtWebAddress.getText().trim();

    }
adaptee.cgiFileName = txtCgi_bin.getText().trim();
adaptee.grammarFileName =
    txtGrammarFile.getText().trim()+".gram";

if (
locationChooser.getSelectedItem().equalsIgnoreCase("Local")){
    adaptee.flagLocal = true ;
}
adaptee.initRecogniser();
setVisible(false);
dispose();
}
if ( e.getSource() == cancelButton ){

    /** process the information obtained */
    setVisible(false);
dispose();
}
}

Httppost.java

import java.net.URL;
import java.net.URLConnection;
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.io.IOException;
public class HTTPpost {

private URL urlToGo;
private String urlHostName;
private String urlFileName;
private String query;
private String receiveData;
public StringBuffer storageBuf = new StringBuffer();
public HTTPpost(String hostName, String fileName, String query ) {
    try {
        urlToGo = new URL("http", hostName, fileName);
        URLConnection uc = urlToGo.openConnection();
        uc.setDoOutput(true);
        uc.setDoInput(true);
        uc.setAllowUserInteraction(false);
        uc.setRequestProperty("Content-type","text/plain");
        uc.setRequestProperty("Content-length",
                query.length()+"\n");

        DataOutputStream dos = new DataOutputStream(uc.getOutputStream());
        System.out.println("INSIDE QUERY HTTP POST " + query);
        dos.writeBytes( query );
        dos.close();
        // read the response
        DataInputStream dis = new
                    DataInputStream(uc.getInputStream());
        String nextline;
        while ( (nextline = dis.readLine())!= null){
private String contentType = "Text/plain";
private String receiveData = " ";
private String urlName;
private int port;
public StringBuffer storageBuf = new StringBuffer();

public HTTPget(String urlBase, int urlPort){
    urlName = urlBase;
    port = urlPort;
    if (port == -1) port = 80;
public void submit(String scriptName, String queryData) {
    receiveData = "";

    /** create a get query */
    String urlLine = scriptName + "?";
    try {
        String url = urlName + urlLine;
        URL u = new URL("http://" + url + queryData);
        DataInputStream line = new DataInputStream(u.openStream());
        while ( ( receiveData = line.readLine() ) != null ) {
            storageBuf.append(receiveData);
        }
        line.close();
    } catch (Exception et) {
        et.printStackTrace();
    }
}

public String result() {
    return (storageBuf.toString());
}


MC.java

import java.awt.*;
import SpeechClient;
public class MC {

    boolean packFrame = false;

    public MC() {
        SpeechClient frame = new SpeechClient();
        if (packFrame)
            frame.pack();
        else
            frame.validate();
        /** center the window */
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height)
            frameSize.height = screenSize.height;
        if (frameSize.width > screenSize.width)
            frameSize.width = screenSize.width;
        frame.setLocation((screenSize.width - frameSize.width)/2, (screenSize.height - frameSize.height)/2);
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        MC mC = new MC();
    }
}
Programs on the server side

C program that wraps Miranda code.

#include <errno.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include "preferences.h"
#include "printer.h"

char *makeword(char *line, char stop);
char *fmakeword(FILE *f, char stop, int *len);
void unescape_url(char *url);
char x2c(char *what);
void plustospace(char *str);

char *makeword(char *line, char stop) {
  int x = 0, y;
  char *word = (char *) malloc(sizeof(char) * (strlen(line) + 1));

  for(x=0;((line[x]) && (line[x] != stop));x++)
    word[x] = line[x];

  word[x] = '\0';
  if(line[x]) ++x;
  y=0;
}
while(line[y++] = line[x++]);
return word;
}

void unescape_url(char *url) {
    register int x,y;
    for(x=0,y=0;url[y];++x,++y) {
        if((url[x] = url[y]) == '%'') {
            url[x] = x2c(&url[y+1]);
            y+=2;
        }
    }
    url[x] = '\0';
}

char x2c(char *what) {
    register char digit;
    digit = (what[0] >= 'A'? ((what[0] & 0xdf) - 'A') + 10 : (what[0] - '0'));
    digit *= 16;
    return(digit);
}

void plustospace(char *str) {
    register int x;

    for(x=0;str[x];x++)
        if(str[x] == '+') str[x] = ' ';

102
/* Convert upper-to-lower and remove all unnecessary characters such as tabs, LF, CR, return modified string as a result */

char *process_question(char *string) {
    char *question;
    int i, ind;
    question = (char *) malloc(strlen(string)+1)*sizeof(char));
    /* skip leading spaces */
    for (i = 0; *(string + i) != '\0') &&
        ((*(string + i) == ' ') || (*(string + i) == '\t') ||
        (*(string + i) == 10) || (*(string + i) == '\n') ||
        (*(string + i) == 13)); i++);
    /* remove new_lines, tabs, convert upper-to-lower */
    for (ind = 0, i = i; *(string + i) != '\0'; i++, ind++) {
        if (((*(string + i) == '\t') || (*(string + i) == 10)) ||
            (*(string + i) == '\n') || (*(string + i) == 13))
            *(question + ind) = ' ';
        else
            if (((*(string + i) <= 90) && (*(string + i) >= 65))
                *(question + ind) = *(string + i) + 32;
            else
                *(question + ind) = *(string + i);
    }
    *(question + ind) = '\0';
    return question;
}
main()
{
    int fifo[2], proc;
    char *argv[4], quest[300], *name, *ques, *questmira, answer[5000],
    tmp[500], tmp1[500], tmp2[500], tmp3[500];

    fgets(quest, 300, stdin);
    plastospace(quest);
    unescape_url(quest);
    name = makeword(quest,'=')
    questmira = process_question(quest);

    if (strcmp (questmira, ""))
    {
        argv[0] = malloc(10*sizeof(char));
        strcpy(argv[0], "moons");
        argv[1] = malloc(strlen(questmira)*sizeof(char));
        strcpy(argv[1], questmira);
        argv[2] = "/quit";
        argv[3] = NULL;

        /* env settings needed to start Miranda interpreter */
        
sprintf(tmp1, "HOME=%s", MIRA_PREFIX);
        putenv(tmp1);
        sprintf(tmp2, "MIRALIB=%s", MIRA_LIB);
        putenv(tmp2);
        sprintf(tmp3, "PATH=%s:%s", MIRA_BIN, MIRA_PREFIX);
        putenv(tmp3);

        printf("Content-type: text/plain\n\n");
    
    
    printf("Content-type: text/plain\n\n");
}
pipe(fifo);

if ((proc = fork()) == -1){
    printf("error: Can't fork");
    exit(errno);
}

if (proc == 0) /* child */{
    close(1);
    dup(fifo[1]);
    sprintf (tmp, "%s/moons", MIRA_PREFIX);
    execvp(tmp, argv);
    printf("error: Can't exec");
    exit(errno);
}

/* Parent */
close(0);
dup(fifo[0]);
fgets(answer, 5000, stdin);
} else{
    printf(\n bad question\n );
}

print_answer(answer);

Unix pipe is created to communicate with the Miranda program, The question is passed to the Miranda code and the answer is printed on stdout.
preferences.h

/* Change these settings if necessary */

/* url prefix of cgi-programs */
#define EXEC_PREFIX "/cgi-bin/richard"

/* url prefix of HTML documents */
#define DOC_PREFIX "/users/r/richard/miranda"

/* Miranda home directory -- this directory should contain .mirarc file as well as all Miranda programs. The file .mirarc should contain the following (or similar) line:

hdve 100000 24000 2020 vi +!

The main purpose of this line is to increase the heap to 100000 (default on janus is 20000) */
#define MIRA_PREFIX "/opt/local/WWW/Servers/NCSA/cgi-bin/richard/mira_data"

/* directory where Miranda is installed */
#define MIRA_BIN "/opt/local/mira/bin"

/* directory where Miranda's stdlib is located */
#define MIRA_LIB "/opt/local/mira/lib"
printer.h

/* print.h */

void print_answer(char *ans) {
    printf( ans );
}

Unix Shell Script

The Unix shell script used to access the grammar file is as follows

#!/bin/csh -f

#
echo "Content-Type: text/plain"
#echo ""
#echo "<b>REQUEST_METHOD</b> = "REQUEST_METHOD"; <br>"
if ( $REQUEST_METHOD == "GET" ) then
    cat $QUERY_STRING
else if ( $REQUEST_METHOD == "POST" ) then
    echo "<b>CONTENT_LENGTH</b> = "CONTENT_LENGTH"; <br>"
setenv v '/bin/cat'
    echo $v
        /opt/local/mira/bin/mira   /home/cs/misc/www/cgi-bin/chittel/wage_demo/
    mira_data/ moons.m <<zzz
    (interpret "$v")
        /q
    zzz
endif
Appendix C- Sessions with the interface

The speech interface was used to communicate with three different silos over the network. The following are the grammar files associated with Solar-man, Monty and Judy.

Solar.gram

grammar solar;

public <s> = <linkingvb> <termph> [ <transvb> by ] <termph>

{sentence} |

<quest> <sent> {sentence} |

( who| what) <verbph>{sentence} |

(which | how many ) <nouncla> <verbph> {sentence} |

<simple> {sentence} ;

<simple> = | ask them to be quiet

| please introduce yourself

| hello there

| hello solar man

| goodbye

| goodbye solar man

| fine thanks

| thanks
| thanks solar man
| yes please
| what is your name
| who are you
| where do you live
| what do you know
| how old are you
| who made you
| what is your favorite band
| who is the vice president at the university of windsor
| who is the president at the university of windsor
| who is the executive dean of science at the university of windsor
| who is the dean of science at the university of windsor
| tell me a poem
| know any poems
| tell me a joke
| know any jokes
| who is judy
| can I talk to judy
who is monty

can I talk to monty;

<sent> = <termph> <verbph>;
<termph> = <pnoun> | <detph>;
<verbph> = <transvbph> | <intransvb>;
<transvbph> = ( <transvb> | <linkingvb> <transvb> by ) <termph>;
<detph> = <det> <nouncla>;
<nouncla> = <adj> <cnoun>|<cnoun>;
<cnoun> = man | men | person | planet | planets | moon | moons;
<adj> = red;
<intransvb> = spin | spins |orbit | orbits | orbited | exist;
<det> = a | an | every | one | two | three | any | all;
<pnoun> = Earth | Jupiter | Mars | Mercury | Neptune | Pluto | Saturn
| Uranus | Venus | Almathea | Ariel | Callisto | Charon | Deimos | Dione | Enceladus | Europa | Ganymede | Hyperion | Iapetus | Io | Janus | Jupitereighth
| Jupitereleventh | Jupiterfourteenth | Jupiterninth | Jupiterseventh | Jupitersixth | Jupitertenth | Jupiterthirteenth | Jupitertwelfth | Luna | Mimas | Miranda
| Nereid | Oberon | Phobos | Phoebe | Rhea | Saturnfirst | Tethys | Titan | Titania | Triton | Umbriel | Bernard | Bond
| Cassini | Dollfus | Fountain | Galileo | Hall | Herschel |

110
<transv> = orbit | orbits | orbited | discover | discovered ;

<linkingv> = is | was | are | were ;

<quest> = did | do | does ;

public <bye> = Good bye and go to sleep {bye} ;

Monty.gram

grammar monty ;

public <s> = <simple> {sentence} ;

public <back> = last speaker please {back} ;

<simple> = | hello

| hello there

| hello monty

| goodbye

| goodbye monty

| fine thanks

| thanks

| thanks monty

| yes please

| what is your name

| who are you
where do you live
what do you know
how old are you
who made you
what is your favorite band
who is the vice president at the university of windsor
who is the president at the university of windsor
who is the executive dean of science at the university of windsor
who is the dean of science at the university of windsor
tell me a poem
know any poems
tell me a joke
know any jokes
who is judy
can I talk to judy
who is solar man
can I talk to solar man;

Judy.gram

grammar poems;
public <s> = <simple> {sentence};
public <back> = last speaker please {back};

<simple> = | hello
        | hello there
        | hello judy
        | goodbye
        | goodbye judy
        | fine thanks
        | thanks
        | thanks judy
        | yes please
        | what is your name
        | who are you
        | where do you live
        | what do you know
        | how old are you
        | who made you
        | what is your favorite band
        | who is the vice president at the university of windsor
        | who is the president at the university of windsor
        | who is the executive dean of science at the university of windsor
        | who is the dean of science at the university of windsor
        | tell me a poem
As we see from the grammar files our sihlos understand a specific set of questions. The following is a sample session with our sihlos.

**Sample session with solar man**

**U:** *hello there*

**C:** *good day to you*

**U:** *how many planets exist*

**C:** *9*

**U:** *which red moon orbits Jupiter*

**C:** *IO*
U: how many planets are orbited by a moon
C: 7

U: which moons are discovered by Hall
C: phobos and deimos

U: which planet is orbited by deimos
C: Mars

U: can I talk to monty
C: yes, here he is

After this the browser receives a speech in the form of a string
"SIHLOURL=www.cs.uwindsor.ca; CGI=cgi-bin/richard; Application=monty;
VOICETYPE=male:" This string is tokenized and the necessary
information is extracted now the application is pointing to monty.

Sample session with monty

U: what is your name
C: my name is monty

U: Hello, how are you
C: Fine thanks
U: *know any poems*

C: *No, but my friend judy does*

U: *Can I talk to judy*

C: *yes, I shall get her for you*

Now the application switches to judy, one can ask similar questions to judy, while talking to judy the voice changes to that of a woman. We can move from one sihlo to other very similar to browsing the web.
Appendix D - List of Acronyms

API     Application Programmer's Interface.
CGI     Common Gateway Interface.
CORBA   Common Object Request Broker Architecture.
GUI     Graphical User Interface.
HTTP    HyperText Transfer Protocol.
IBM     International Business Machines.
IP      Internet Protocol
IIOP    Internet Inter-ORB Protocol.
JAR     Java Archive.
JSAPI   Java Speech Application Programming Interface.
OMG     Object Management Group.
RFC     Request For Proposal.
RMI     Remote Method Invocation.
SUI     Speech User Interface.
TCP     Transport Control Protocol
VITA AUCTORIS

NAME: Sanjay P. Chitte

PLACE OF BIRTH: Thane, India

YEAR OF BIRTH: 1966

EDUCATION: K.H.S High School
1970-1982

University of Bombay, India
1982-1986 Bachelor's in Physics

University of Bombay, India

University of Windsor, Canada