

University of Windsor

Scholarship at UWindor

Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

2006

Using VXML to construct a speech browser for a public-domain SpeechWeb

Li Su

University of Windsor

Follow this and additional works at: <https://scholar.uwindsor.ca/etd>

Recommended Citation

Su, Li, "Using VXML to construct a speech browser for a public-domain SpeechWeb" (2006). *Electronic Theses and Dissertations*. 4485.

<https://scholar.uwindsor.ca/etd/4485>

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

Using VXML to Construct a Speech Browser for a Public-Domain SpeechWeb

By

Li Su

A Thesis

Submitted to the Faculty of Graduate Studies and Research

through the School of Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Science at the University of Windsor

Windsor, Ontario, Canada

2005

© 2005 Li Su



Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 978-0-494-17054-0

Our file Notre référence

ISBN: 978-0-494-17054-0

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

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

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

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

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.


Canada

Abstract

Despite the fact that interpreters for the voice-application markup language VXML have been available for around five years, there is very little evidence of the emergence of a public-domain SpeechWeb. This is in contrast to the huge growth of the conventional web only a few years after the introduction of HTML. One reason for this is that architectures for distributed speech applications are not conducive to public involvement in the creation and deployment of speech applications. In previous research, a new architecture for a public-domain SpeechWeb has been proposed. However, a non-proprietary speech browser is needed for this new architecture. In this thesis, it is shown that through a novel use of VXML, a viable public-domain SpeechWeb browser can be built as a single VXML page. This thesis is proven through the development and implementation of a single VXML page SpeechWeb browser.

[Keywords: SpeechWeb, VXML, public-domain, non-proprietary, speech recognition, distributed system]

Acknowledgements

I would like to thank my advisor Dr. Richard A. Frost for helping me to find this thesis topic, giving me invaluable guidance, encouragement, and generous help.

I would also like to thank Dr. H. K. Kwan, Dr. Jianguo Lu and Dr. Alioune Ngom for reading my thesis report and giving me their valuable comments and suggestions.

And a special thanks to my parents for their love and support.

Table of Contents

Abstract	iii
Acknowledgements	iv
List of Tables	ix
List of Figures	ix
Chapter 1 Introduction	1
1.1 Background	1
1.2 Problems with existing SpeechWeb browsers	1
1.3 Why a Public-Domain SpeechWeb?	2
1.4 Thesis Statement	2
1.5 Why is this not obvious	2
1.6 Why proof of this thesis statement is important	3
1.7 How the thesis will be proven	3
Chapter 2 Architecture for a Public-Domain SpeechWeb	5
2.1 Overview	5
2.2 Existing architectures for distributed speech applications	5
2.3 A New public-domain SpeechWeb architecture	6
Chapter 3 Comparison of the New Architecture with Existing Architectures for Distributed Speech Applications	9
3.1 Overview	9
3.2 Comparison of the new architecture to speech interface to HTML pages	9
3.2.1 Relative advantages of the new architecture	9
3.2.2 Relative disadvantages of the new architecture	9

3.3 Comparison with networks of VXML pages	10
3.3.1 Relative advantages of the new architecture	10
3.3.2 Relative disadvantages of the new architecture	10
3.4 Comparison with the call-center architecture	10
3.4.1 Relative advantages of the new architecture	10
3.4.2 Relative disadvantages of the new architecture	11
3.5 Summary of the relative strengths and weaknesses of the new architecture	11

Chapter 4 A Novel Use of VXML to Create a Speech Browser for the

New SpeechWeb Architecture	13
4.1 Overview	13
4.2 Why VXML?	13
4.3 Our novel use of VXML	14

Chapter 5 Different Approaches Investigated for Designing the

SpeechWeb Browser as a Single VXML Page	15
5.1 Overview	15
5.2 Approach 1: Multiple Forms	15
5.2.1 Approach 1 architecture	16
5.2.2 Approach 1 problem	17
5.3 Approach 2: Rewrite the grammar	18
5.3.1 Approach 2 architecture	18
5.3.2 Approach 2 problem	20
5.4 Approach 3: Sub Dialog	21
5.4.1 Approach 3 problem	21
5.5 Summary and analysis	22

Chapter 6 A Novel SpeechWeb Browser Design	23
6.1 Overview	23
6.2 The final approach: Rewrite the VXML page	23
6.3 The SpeechWeb browser Graphic User Interface	24
6.4 The SpeechWeb browser structure	24
6.5 Examples	27
 Chapter 7 Analysis of the New SpeechWeb Browser	 29
7.1 Overview	29
7.2 Clarity of design	29
7.2.1 Design concept	29
7.2.2 The design goal	31
7.3 Free distribution	31
7.4 Ease of implementation and installation on client devices	31
7.5 Use of common communication protocols	32
7.6 Ease of creation and deployment of speech applications	33
7.7 Capabilities	33
7.7.1 Basic functions	33
7.7.2 Improved speech-recognition accuracy	34
7.7.3 Unable to support dialogue	34
7.8 Efficiency	35
7.8.1 Communication cost	35
7.9 Speed	35
7.9.1 Graphic User Interface	38
7.9.2 Speech interaction	39
7.9.3 Transition to a new application	39
7.9.4 Timing results	40
7.9.5 Analysis of the results	41

Chapter 8 Future Work: Dialogue Support in the SpeechWeb	43
8.1 Spoken dialogue in speech interaction	43
8.2 Unable to support dialogue	44
8.3 How to support dialogue in the SpeechWeb	44
8.3.1 Communication protocols	45
8.3.2 Dialogue support in remote applications	46
 Chapter 9 Conclusions	 47
9.1 What has been achieved?	47
9.2 Contributions	48
9.3 Suggestions for future work	48
 Bibliography	 49
Related Publications by the Author	50
Appendix I: A Survey	51
Appendix II: SpeechWeb Browser Implementation Manual	76
Vita Auctoris	138

List of Tables

Table 1: Sample session recorded in <i>vxml.log</i>	35
Table 2: <i>vxml.log</i> trace code	37
Table 3: SpeechWeb browser speed (in <i>seconds</i>)	40

List of Figures

Figure 1: The LRRP SpeechWeb Architecture	6
Figure 2: Approach 1 Architecture	16
Figure 3: Approach 2 Architecture	19
Figure 4: SpeechWeb Browser Graphic User Interface	24
Figure 5: SpeechWeb Browser Structure	25
Figure 6: SpeechWeb Browser Design	30

Chapter 1 Introduction

1.1 Background

HTML was developed to make the Internet a very rich, diverse and resourceful place through which one can visually access a huge amount of knowledge. In a similar way, technology has developed to such an extent that today one can even interact with computers through voice, and get the response from them through different modes such as voice as well as text. This technological extension of the web, to accept and to synthesize speech, is known as the SpeechWeb [Frost, 2005].

For instance, many companies have started embedding speech recognition into their IVR (Interactive Voice Response) systems, so that customers, by just dialing up a number and speaking to computers, can purchase or sell stock or check flight reservations, etc. and the computers can respond by exploiting pre-recorded forms and synthesized voice.

A public-domain SpeechWeb contains these hyperlinked IVR systems and other speech applications that are developed by individual users, which are accessible through end-user devices using voice commands.

Just like we need an HTML browser to view conventional web pages, it is necessary to have a SpeechWeb browser, deployed on the end-user devices to access SpeechWeb applications distributed over the Internet.

1.2 Problems with existing SpeechWeb browsers

A prototype SpeechWeb browser was developed by Frost and Chitte [1999] using the Java programming language. However, it uses IBM's proprietary APIs for voice recognition and speech synthesis, therefore it cannot be freely distributed.

Speech-recognition technology is still under development, and in order to distribute the browser, we need a non-proprietary SpeechWeb browser which can make use of the latest speech-recognition technology.

Therefore, we decided to use VXML to build the SpeechWeb browser.

1.3 Why a Public-Domain SpeechWeb?

A public-domain SpeechWeb would allow users to verbally navigate through networks of hyperlinked speech applications using cell phones or computers. Imagine in the near future, as [Frost, 2005] describes, while driving through heavy traffic, you could ask for information to help you avoid congestion; while walking you could ask for weather updates; as you walk to the subway station, you could ask your electronic calendar if you have any plans for the evening.

There are endless possibilities for the SpeechWeb. More importantly, it would give blind people much needed access to the huge amount of knowledge on the Internet.

Like the conventional web, the SpeechWeb should be open to public, speech applications should be easily setup on conventional web servers by end-users, and speech browsers should not contain any proprietary components so that they can be distributed freely for use by others.

Because the new public-domain SpeechWeb browser is easy to deploy and distribute, it is possible to have huge public involvement in developing and expanding the SpeechWeb. The new SpeechWeb browser can provide a platform for future research on improving speech recognition, speech processing, language understanding and so on.

1.4 Thesis Statement

A viable public-domain SpeechWeb browser can be built as a single VXML page.

1.5 Why is this not obvious

1. VXML has been around for only a few years. VoiceXML 1.0 was released on May 7th, 2000 [W3C, 2000] and VoiceXML 2.0 was released on October 23rd, 2001 [W3C, 2001]. It is a new technology and its capabilities are not well understood.

2. Like HTML pages, VXML pages are normally used as hyperlinked pages. VXML has never been used to construct a single page which accesses remote distributed non-VXML applications.
3. The combination of VXML with Java interfaces has not tested for this kind of application.
4. Efficiency and speed are not known for the proposed browser architecture.
5. The most recent version of VXML has a known deficiency. Recognition grammars cannot be changed “on-the-fly” without loading a new VXML page. However, the proposed single-VXML-page-architecture requires this “on-the-fly” grammar change, when the user moves from one remote application to another.

The novelty of the proposed approach has been confirmed to some extent by the publication of a paper by the author of this thesis in the premier Artificial Intelligence conference in Canada: A.I. 2005. [Su and Frost, 2005]

1.6 Why proof of this thesis statement is important

Different VXML interpreters are now widely available. Some beta versions of the interpreters can be downloaded for free. Some of the interpreters are deployed on a wide range of different platforms. This is an ideal environment for developing the SpeechWeb.

VXML is a non-proprietary language. The proposed browser, if it can be built will consist of a single VXML page and three Java Objects. Therefore, it can be distributed freely.

As speech-recognition technology advances, we can also make use of the latest speech technology in VXML interpreters, to improve the recognition-accuracy of the SpeechWeb browser.

1.7 How the thesis will be proven

1. Investigate different possible architectures for the browser as single VXML page.

2. Identify a potential architecture.
3. Implement the architecture.
4. Analyze the architecture with respect to:
 - i. Clarity of design.
 - ii. Ability to install on various platforms.
 - iii. Ease of extension and modification.
 - iv. Efficiency.
 - v. Capability.

Chapter 2 Architecute for a Public-Domain SpeechWeb

2.1 Overview

A Public-Domain SpeechWeb is a collection of hyperlinked speech applications, developed by individuals, that are distributed over the Internet and which are accessible by spoken commands and queries that are input through remote end-user devices [Su and Frost, 2005].

The SpeechWeb works in a similar way to the conventional web. Speech applications that are created by end-users reside on conventional web servers. Users can access SpeechWeb applications using SpeechWeb browsers residing on their computers and cell phones.

2.2 Existing architectures for distributed speech applications

In the past, different speech architectures and technologies have been developed to accommodate verbal commands and provide speech interaction between users and distributed speech applications. The following are three architectures described in Su and Frost [2005] and their disadvantages as a basis for public-domain SpeechWeb architectures. The following is based on Su and Frost (2005):

1. Speech interfaces to conventional HTML web pages. These interfaces run on end-user devices and allow users to scan downloaded web pages and follow hyperlinks through spoken commands [Hemphill and Thrift 1995]. More sophisticated versions process the downloaded web pages and provide spoken summaries and allow some limited form of content querying.
2. The second architecture involves the use of networks of hyperlinked VXML pages. VXML [Lucas 2000] is similar to HTML except that it is used to create hyperlinked speech applications. VXML pages, which are executed on VXML browsers, include commands for prompting user speech input, for invoking recognition

grammars, for outputting synthesized voice, for iteration through blocks of code, for calling local Java scripts, and for hyperlinking to other remote VXML pages that are downloaded and executed in a manner similar to the linking of HTML pages in the conventional web. Speech recognition is carried out by the VXML browser running locally on an end-user device, or at a remote site which is accessed through the telephone.

3. The third architecture is the one used in call centers. End-users communicate with the call center using remote telephones. Speech recognition is carried out at the call center. Often VXML is used to code the call-center application.

2.3 A new public-domain SpeechWeb architecture

The last two speech architectures, described in Section 2.2, can be categorized as follows: the characteristic of the second architecture can be summarized as Local Recognition and Local Processing (LRLP). The third one is Remote Recognition and Remote Processing (RRRP) architecture. Frost et al [2000] has proposed a new public-domain SpeechWeb architecture: a Local Recognition and Remote Processing (LRRP) architecture.

The following diagram shows the structure of the LRRP SpeechWeb architecture:

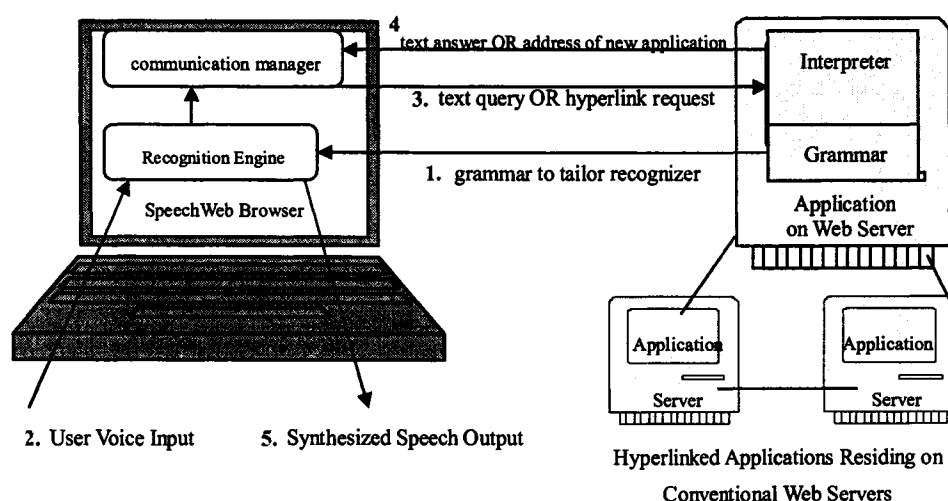


Fig. 1. The LRRP SpeechWeb Architecture [Su and Frost, 2005]

In the LRRP-SpeechWeb architecture, users interact with remote applications using voice through the SpeechWeb browser located on the local end-user device. The SpeechWeb browser contains a speech-recognition engine and a communication manager. Grammar-based speech recognition is carried out on the end-user device. Applications reside on conventional web servers. Each application contains a recognition grammar and an interpreter. The grammar defines the application's input language. When a speech browser first contacts a remote application, the grammar is downloaded and used to tailor the browser for that application. This is necessary to achieve sufficient recognition-accuracy for non-trivial applications [Su and Frost, 2005].

A SpeechWeb session has the following steps:

1. The recognition grammar is first downloaded, from the remote application, to the local speech browser.
2. The user enters voice input. The speech recognizer in the local SpeechWeb browser processes the input. If the input is recognized, the speech recognizer sends the input to the communication manager in text format. Otherwise, the recognizer will prompt the user to re-speak the input.
3. The communication manager sends the text query to the interpreter in the remote server application using a pre-defined Internet communication protocol.
4. The interpreter accepts the input, processes it and returns the answer back to the communication manager using the same Internet protocol (or redirects the browser to a new application, see below).
5. The answer is then output as synthesized voice.

Steps 2 to 5 are repeated as user gives more inputs and gets answers back from remote application.

If the user's input is a request to follow a hyperlink to another application, in step 4, the interpreter recognizes the request and returns the new-application URL address back to the SpeechWeb browser. The browser follows the URL address and contacts

the new application. Then, it downloads the recognition grammar from the new application as described in step 1. Steps 2 to 5 are repeated as many times as it is needed for the user to interact with the remote application through the browser in the local end-user device until he or she asks to transfer to a different application.

Chapter 3 Comparison of the New Architecture with Existing Architectures for Distributed Speech Applications

3.1 Overview

In the previous chapter, Section 2.2, we described three existing speech architectures. How does our new public-domain SpeechWeb architecture compare to them? What are the advantages and disadvantages our new architecture has? It will be explained in this chapter.

3.2 Comparison of the new architecture to speech interface to HTML pages

3.2.1 Relative advantages of the new architecture

- i. HTML pages are designed for visual browsing. Some pages have complex visual design with little text. It is difficult for the speech interface to search for the right information. However, in the new architecture, as long as the user asks the “right” questions, this issue does not exist.
- ii. Users cannot ask query-like questions about HTML-page content. For example, “Which department does Dr. Goldman work in?” However, the new architecture is designed for this kind of question.

3.2.2 Relative disadvantages of the new architecture

- i. In the new architecture, the SpeechWeb browser cannot access regular HTML pages as it is not designed to have an HTML-page scanning mechanism.

3.3 Comparison with networks of VXML pages

3.3.1 *Relative advantages of the new architecture*

- i. In the architecture of networks of VXML pages, applications have to be coded in VXML and they have to be executed locally on end-user machines. This is not appropriate for lightweight end-user devices such as cell phones. However, in the new architecture, applications can be coded in any language and they are executed remotely on web servers. It is possible to use cell phones to access the SpeechWeb.
- ii. In the architecture of networks of VXML pages, sometimes large applications coded in VXML have to be downloaded from remote servers. Users have to wait for the download to be completed in order to execute the applications. This depends on the Internet connection speed. It can delay interactions between client and server. However, the new architecture uses text transmission between the server and client and only a few bytes are transmitted each time during the speech interaction. It can be used over low-bandwidth Internet connections.

3.3.2 *Relative disadvantages of the new architecture*

- i. There are many VXML interpreters available that fully support the architecture of the networks of VXML pages. Because it is the conventional way of using VXML pages to create distributed speech systems. However, we have to develop a non-proprietary speech browser for the new architecture.

3.4 Comparison with the call-center architecture

3.4.1 *Relative advantages of the new architecture*

- i. User voice profiles can be used to improve speech-recognition accuracy. Voice profiles are stored where the speech recognition is carried out. In the new architecture, the voice profiles can be stored locally. However, in the call-center architecture, they have to be stored in every call center. Therefore,

it is too expensive and virtually impossible for a call center to store every user's voice profile.

ii. In the new architecture, applications can be deployed on conventional web servers. It is very easy for users to set up their own SpeechWeb servers. Because everything is processed remotely at the call centers, there is a large computational burden for the call centers. Powerful computers are required. Therefore, many providers cannot afford the hardware and software needed to set up a call center to host their applications.

iii. In the call-center architecture, voice is transmitted over the phone line. The transmission degrades voice quality. Therefore, it will affect the recognition accuracy at the call center. However, in the new architecture, the voice input is recognized locally and translated into text. During text transmission, there is no degradation.

3.4.2 *Relative disadvantages of the new architecture*

i. The new architecture requires end-user devices with computing power which have speech-to-text capabilities and the SpeechWeb browser installed. However, in the call-center architecture, it is as simple as picking up the phone and dialing a number.

3.5 Summary of the relative strengths and weaknesses of the new architecture

Compared to the existing speech architectures described in Section 2.2, the new LRRP SpeechWeb architecture has several strengths, including the following:

1. In grammar-based speech recognition, having a local recognizer improves accuracy in two ways: there is no voice degradation over the communication channel and locally-stored voice profiles can be used.
2. It is more flexible. Applications are deployed on conventional web servers. No special hardware or software is needed. Also, applications can be coded in

any language. No special software is needed.

3. It is more efficient. There is only text transmission between the local end-user device and the remote applications.

Chapter 4 A Novel Use of VXML to Create a Speech Browser for the New SpeechWeb Architecture

4.1 Overview

VXML is also known as VoiceXML. It stands for Voice Extensible Markup Language.

Unlike HTML and traditional web browsers, which rely on keyboard and mouse, and which browse the web visually, VXML relies on a voice browser that enables users to interact with the application by submitting their natural-speaking voice and listening to pre-recorded voice or computer synthesized voice.

VXML has been developed by the *VoiceXML Forum* (an industry organization founded by AT&T, IBM, Lucent and Motorola) and endorsed by W3C. AT&T, IBM, Lucent Technologies, and Motorola created VXML 1.0 in a joint effort to promote the technology. Version 1.0 was released to public on May 7, 2000.

VoiceXML is designed for creating audio dialogs that feature synthesized speech, digitized audio, recognition of spoken and DTMF-key input, recording of spoken input, telephony, and mixed-initiative conversations. Its major goal is to bring the advantages of web-based development and content delivery to interactive-voice response applications [W3C, 2000].

4.2 Why VXML?

VXML is a language developed for creating interactive voice response between users and applications. There are a growing number of VXML interpreters commercially available. Some of them can be downloaded for free in their beta versions. VXML interpreters have built-in speech-processing capabilities, such as speech recognition, which can translate human speech into text, and text-to-speech capability, which can produce synthesized voice.

We realized that the capabilities of VXML can be used in developing our

SpeechWeb. Also, pages created in VXML don't contain any proprietary components. Therefore, they can be distributed freely.

4.3 Our novel use of VXML

The conventional way of using VXML is in the second and third architectures that were described in Section 2.2.

VXML allows users to extend its capability with help of other languages. For example, a user can embed ECMA Script (JavaScript) in VXML code. As well, VXML is able to call specially-coded Java objects.

VXML and VXML interpreters provide a platform for speech recognition and speech synthesis. We don't have to use the application-processing capability of VXML, only the speech-processing capability. Therefore, we realized that we can use **a single VXML page as a SpeechWeb browser.**

From www.ibm.com we downloaded a free beta version of the VXML interpreter and the software developer's kit. It is called *IBM WebSphere Voice Server SDK Version 3.1*. It runs on Java Runtime Environment 1.3.1 and supports VoiceXML 1.0.

We decided to use this free VXML interpreter and VXML version 1.0 to build a prototype of a public-domain SpeechWeb browser. The specification of VoiceXML 1.0 can be found at [W3C, 2000].

Chapter 5 Different Approaches Investigated for Designing the SpeechWeb Browser as a Single VXML Page

5.1 Overview

Our goal is to use a single VXML page to handle all user voice interaction with remote hyperlinked speech applications (the SpeechWeb).

Designing the SpeechWeb Browser involved the use of IBM WebSphere Voice Server SDK 3.1 as the recognition engine and Java objects as the communication manager. The VXML page is implemented in VoiceXML 1.0.

In our single-VXML-page design, when a user interacts with different remote applications, the only difference one can notice is the speech scope. The speech scope of an application is defined by the recognition grammar. The grammar specifies what phrases and sentences the application understands. Since the grammar is downloaded onto the end-user device, on the browser side, the only change between different applications is the grammar. Therefore, in a single VXML page, if we can somehow change the grammar according to different applications, we will be able to achieve our goal.

5.2 Approach 1: Multiple Forms

In VXML, the <form> element is used to create Forms. In each Form, the browser is able to take user input and give output. Each form can use its own grammar and a VXML page can have multiple forms.

Our attempt at designing a single-VXML-page SpeechWeb browser in this approach is to use multiple forms controlled by a main loop and change the grammar in each form dynamically.

5.2.1 Approach 1 architecture

The following diagram is the architecture designed for our first approach.

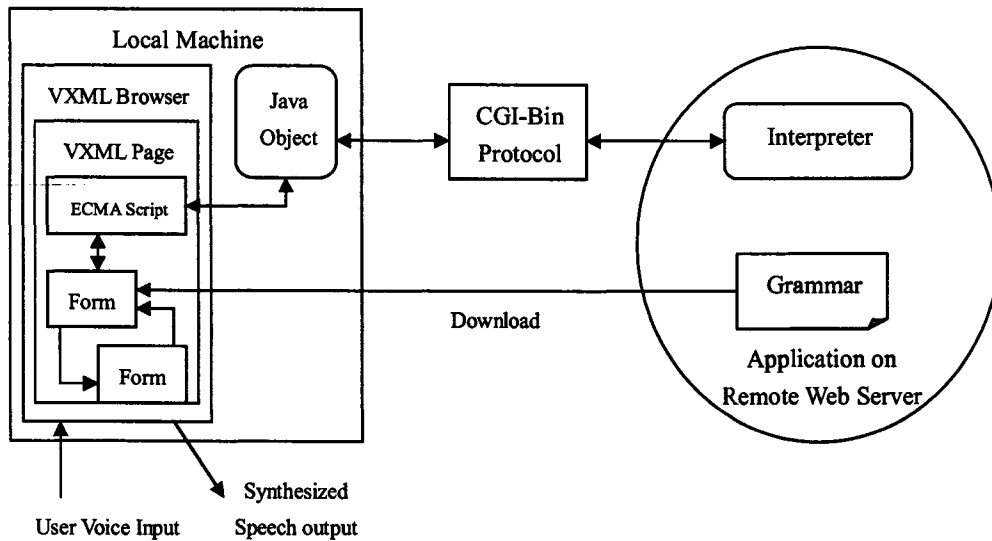


Fig. 2. Approach 1 Architecture

In approach 1, the VXML Brower works as follows:

1. The VXML Browser starts by downloading a recognition grammar from a default remote application and initiates speech by prompting the user.
2. Since VXML is able call specially-coded Java objects, we can make use of this property. Once a user voice input is recognized, it is translated into a string. Then the string is passed to a Java object. In this case, the Java object is the communication manager.
3. Because the grammar file and the application interpreter are located in the same remote application, with the help of the embedded ECMA Script (JavaScript) in the VXML page, the VXML browser is able to extract the application-interpreter address from the grammar address and vice versa. The Java

object follows the extracted application-interpreter address and uses the CGI-BIN protocol to send the input string to the remote interpreter.

4. The interpreter then processes the user input and returns the answer back to the browser as a character string.
5. The VXML browser then outputs the string as synthesized voice.
6. The VXML browser then repeats steps 2 to 5 as the user gives more voice input.

In step 3, if the user input is a request to follow a hyperlink to another application, steps 4 to 6 will change to the following:

- 4a. The application interpreter returns the new application URL address along with a response such as “goodbye” in one string.
- 5a. Using the embedded ECMA Script, the VXML browser then extracts the grammar address from the returned application address. The grammar address is then passed to a new Form as a variable. The main loop moves to the new Form.
- 6a. The new Form follows the new grammar address, downloads the grammar and initiates speech.
- 7a. The VXML browser then repeats steps 2 to 5 as the user interacts with the new remote application.

5.2.2 Approach 1 problem

We identified an unexpected problem with approach 1:

In VXML the <grammar> element defines the recognition grammar. The grammar can be either inline or external. The inline grammar is defined in the current VXML page. It is static. It cannot be changed even if we have a different application. Therefore, we used external grammars stored in a local directory or on individual remote application web servers.

However, for the external grammar, the current version of VoiceXML does not

allow the grammar identifier to have its reference changed (i.e. it cannot be given a new URL to a new remote grammar). The URL address of the grammar has to be specified. If we attempt to pass the grammar URL address in a variable, the VXML browser will try to find the URL address of the local file name after the variable and return exceptions.

Even in the latest version of VoiceXML 2.0, this problem does not seem to be solved.

Therefore, the first approach, of designing the SpeechWeb Browser with a single VXML page, failed.

5.3 Approach 2: Rewrite the grammar

To solve the problem encountered in approach 1, we tried an alternative approach. Our idea was that the VXML browser, instead of downloading the grammar from the remote application server directly, loads the grammar into a local grammar file. Every time the user asks to change the application, we can use another Java object to download the new grammar and write the grammar content to the same local grammar file. Then ask the VXML browser to reload the grammar file.

In this approach, multiple Forms controlled by a main loop are also used in the VXML page, except that the Forms use the same local grammar file while we try to rewrite the grammar file dynamically.

5.3.1 Approach 2 architecture

The following diagram is the architecture designed for approach 2

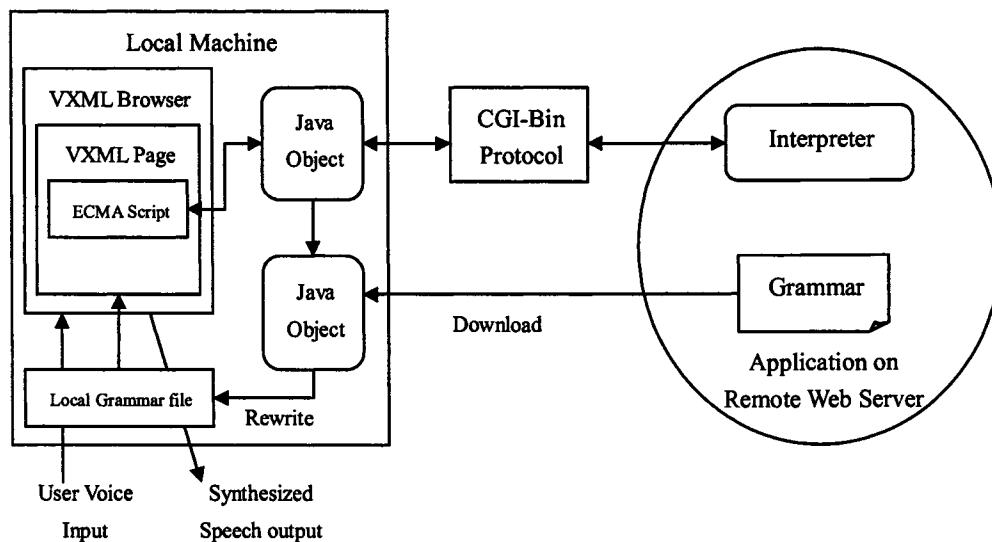


Fig. 3. Approach 2 Architecture

In this approach, we add another Java object besides the communication manager.

The initial steps work the same as the steps 1 to 6 described in Section 5.2.1. However, when the user input is a request to change to another application and the new application URL is returned from the current application interpreter, the process is changed:

- 5b. The communication-manager Java object intercepts the result. The communication manager then sends the new application grammar URL to a new Java object. This Java object downloads the new grammar, copies its content to the local grammar file. The main loop moves to the Form which uses the same local grammar file.
- 6b. The new Form reloads the new grammar from the local file and initiates speech.
- 7b. The VXML browser then repeats steps 2 to 5 in Section 5.2.1 as the user interacts with the new remote application.

5.3.2 *Approach 2 problem*

This approach, of designing the SpeechWeb Browser with a single VXML page, also failed.

The local grammar file is used multiple times in the VXML page as they are in different Forms. However, when the VXML browser runs, it goes through the whole VXML page and finds all the grammar files needed for this page. To save time in running the VXML page, the browser fetches the first grammar, which is used right away and caches other grammar files in the memory for future use. When other grammar files are needed, the browser only loads the files from the memory. It does not load the grammar files from either the local hard drive or a remote server. Therefore, there is no effect if we rewrite the grammar using a Java object.

Here is an example of a part of the VXML browser, automatically-generated, log file.

vxml.log

```
00:49:25.202 S: Starting V030227 at Thu Feb 26 00:49:25 EST 2004
00:49:25.202 X: Java: Sun Microsystems Inc. 1.3.1
00:49:25.202 X: requested locale: en_US
00:49:27.109 S: initializing application
00:49:27.109 V: file:D:/VXML/start.vxml (fetch get)
00:49:27.140 F: file:D:/VXML/vxml.gram (re-fetch get)
00:49:27.140 K: re-fetch file
00:49:27.202 K: put 000014 file:D:/VXML/vxml.gram
00:49:27.218 F: file:D:/VXML/vxml.gram (cache)
00:49:27.234 F: file:D:/VXML/vxml.gram (cache)
00:49:27.296 S: running
00:49:27.296 C: Welcome to the speech web browser
00:49:27.327 V: #one
00:49:27.327 A: not listening
00:49:27.327 A: listening
00:49:27.327 .....
.....
```

As you can see in the log file, after initializing the application, but before initializing speech, the VXML browser fetches the “*vxml.gram*” once and caches it twice. It is because, when the browser starts the VXML page *start.vxml*, it loads

vxml.gram as the default grammar. The other cached *vxml.gram* grammars are used in different Forms in the VXML page. However, the browser loads the grammar files right after the browser starts before the Java object has a chance to change them.

5.4 Approach 3: sub dialog

In this approach, we try to use a Sub Dialog in VXML page to solve the problem which occurred in approach 2.

In order to maintain all the variables, we have to use a multi-document application. The first page is the “root” document and other pages are "supporting" documents.

We can start with a default grammar file. When the user wants to interact with a different interpreter, the browser will transit to an intermediate VXML page. The intermediate page will call the Java object to download and rewrite the local grammar file to the desired one, then transit back to the “root” VXML page.

It was hoped that, by putting another grammar in the intermediate page, the browser would drop the grammar file in the root page from the memory. When we transit back to the root page, the browser would load the rewritten local grammar file.

This approach is structurally similar to the first approach. Instead of using multiple forms in one VXML page, we use multiple VXML pages. With the help of Java objects, we can rewrite the local grammar file.

5.4.1 Approach 3 problem

This approach did NOT work out either.

The browser keeps every page and every grammar file, that is ever used, in the memory, so that it does not need to load them every time. We try to force the browser to load a new grammar file in the intermediate page. It won't reload the local grammar file.

We tried to use 10 intermediate VXML pages. Every page contains a grammar file that is different from the others. When we asked the browser to come back to the

root page, it still loaded the page and grammar cached in the memory.

5.5 Summary and analysis

All three approaches described in this chapter failed to work.

The inability to change the content of the grammar identifier dynamically appears to be a shortcoming in all releases of VXML.

Like HTML, VXML is designed to have all of the pages downloaded from the remote server to the local user machine and executed locally (a detailed description is given in Section 2.2). In the original design concept, in the author's opinion, the VXML designers only think of the case where multiple VXML pages each use a single copy of one or more grammars. They did not think of the case where one VXML page can use multiple versions of a grammar. Otherwise, it is like asking the `` tag in HTML to take URL or local hyperlink address dynamically. It is not how HTML or VXML works in the original design.

Approaches 2 and 3 are attempts to solve the grammar problem encountered in approach 1. Their problems are both related to the cache mechanism in VXML.

From further testing, we found out that, by default, the VXML browser has a cache property value set to "fast". That means the VXML browser caches every page and every grammar it ever uses and ignores all changes in those pages and grammars. It is possible to change the page cache property from default value "fast" to "safe". Then the VXML browser will reload a VXML page with cache property "safe" every time it is asked to revisit it. However, this does not change the cache property for the grammar. The browser still caches every grammar that is ever found in a page, even if the grammar will not be used.

We needed to find an alternative solution to solve the grammar problem, once and for all, in order to build the SpeechWeb browser as a single VXML page.

Chapter 6 A Novel SpeechWeb Browser Design

6.1 Overview

From the failed attempts described in chapter 5, we learned that the VXML browser will load and cache any grammar files that appear in a VXML page. However, if there is a link to a new page in a VXML page, the VXML browser will not load the new page unless the page is called. Therefore, if we can write the new page, which contains the URL addresses of the new application grammar and interpreter, at run time, we can solve the problem.

6.2 The final approach: Rewrite the VXML page

In this approach, when a user asks to interact with a new application, and the new application URL address is returned by current application interpreter, instead of rewriting the local grammar file, we can write a new VXML page which contains the new application grammar and interpreter URL addresses. After the new page is written, the VXML browser can transit from the current VXML page to the new one. We don't have to worry about having the same grammar cached in the memory for the new page. The problem is how to write the new VXML pages.

Because the only difference between the current VXML page and the new page is essentially the grammar, when we are rewriting the VXML page, we only need to change the grammar address in the page. Therefore, we can keep a default template page on the local machine. Every time we need to write a new page, we can just copy the template page and add a new URL address for the new grammar.

A new Java object is used in this approach, besides the communication manager. When the remote-application interpreter returns a URL address of a new application, the embedded ECMA Script receives the new application address through the communication manager. The grammar address is extracted by the ECMA Script. The VXML browser then alerts the new Java object and passes the grammar address to the

new Java object. Then the new Java object reads the local VXML template page and rewrites a new page containing the new grammar address. Once the new page is written successfully, the current VXML page is allowed to transit to the new page and the user can start interacting with the new remote application through the VXML page and browser.

6.3 The SpeechWeb browser Graphic User Interface

To make the SpeechWeb browser more user friendly, we decided to make a Graphic User Interface (GUI).

When the SpeechWeb browser starts, it provides a graphic interface to the user. The interface allows the user to input a remote server application URL and start browsing the SpeechWeb by first connecting to that application. The browser can store a remote application URL in a local text file as a default application address so that the SpeechWeb browser can start by connecting to the default application. The user is able to change the default application as needed.

The SpeechWeb Browser Graphic User Interface looks like the following.

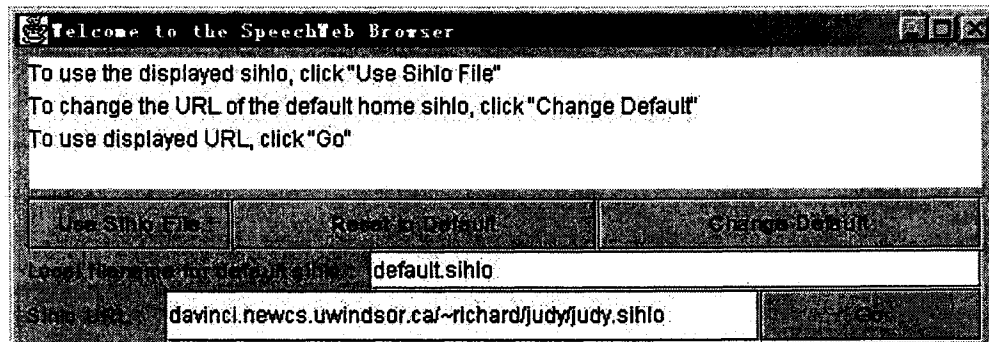


Fig. 4. SpeechWeb Browser Graphic User Interface

6.4 The SpeechWeb browser structure

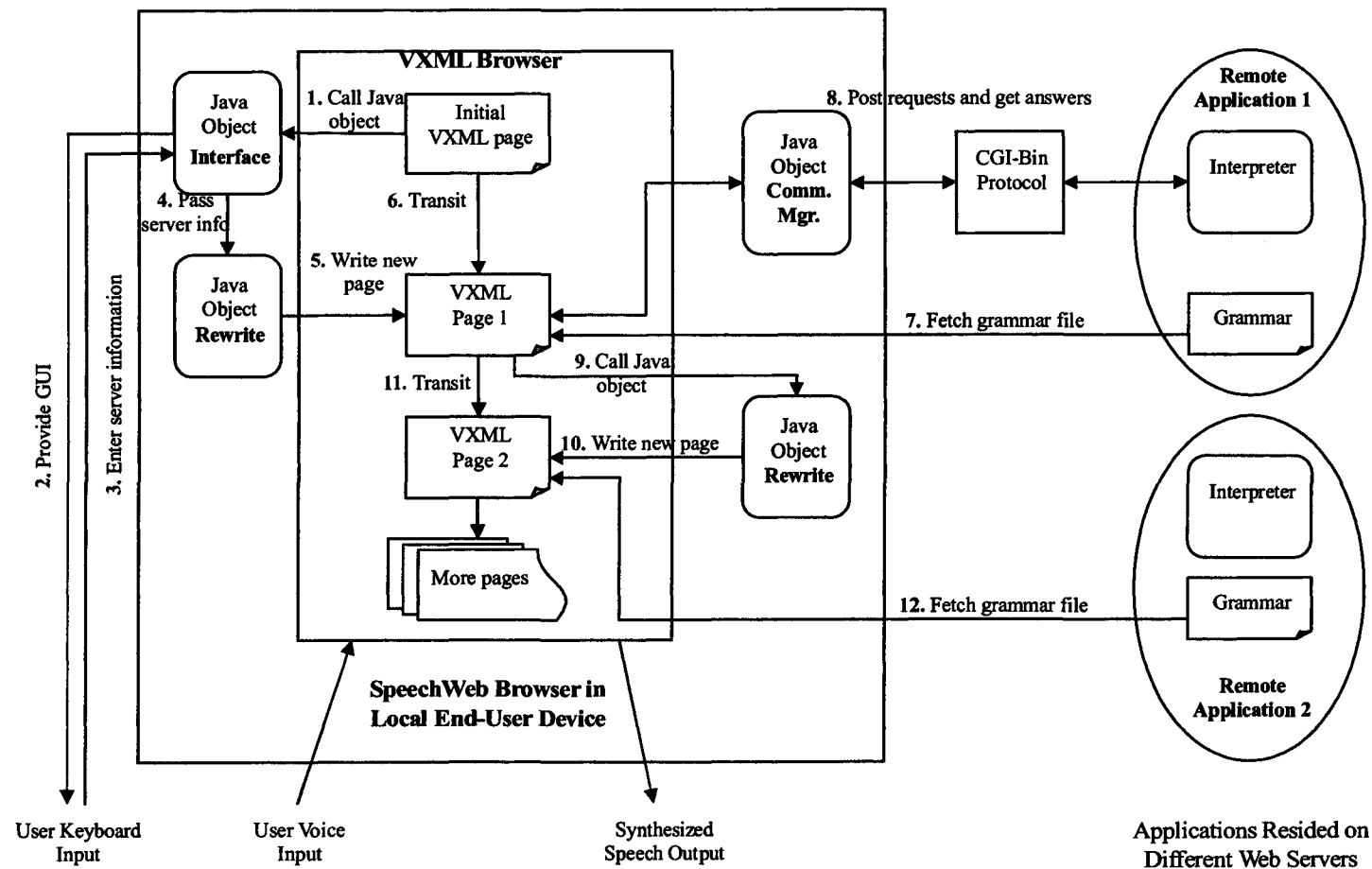


Fig. 5. SpeechWeb Browser Structure

Figure 5 shows the detailed structure of the SpeechWeb browser.

The SpeechWeb Browser consists of a VXML browser and three Java objects. It works in the following way:

1. The SpeechWeb Browser starts by running an initial VXML page in the VXML browser.
2. This page calls a Java object which provides a graphic user interface to the user. The interface is shown in Figure 4 in Section 6.3.
3. The user enters or chooses the correct remote application URL and clicks “Go” button.
4. As soon as the “Go” button is clicked, the remote application information is gathered by the Java object and calls another Java object Rewrite then passes the application information to it.
5. Based on a VXML template page, the Java object Rewrite uses the given application information and writes a new VXML page which contains the application grammar and interpreter URL addresses. In Figure 5, this page is called VXML page 1.
6. After the new VXML page is created, the initial page transits to page 1.
7. The VXML Browser starts the page by downloading the recognition grammar from the remote application and initiates speech.
8. The user interacts with the remote application by using voice. If a user voice input is recognized, it is translated into text format and passed to the Java object communication manager along with the application interpreter URL address. The communication manager follows the address and posts the text input to the application interpreter using the CGI-BIN protocol. The remote interpreter receives the input, processes it, and returns the answer. Then the communication manager gets the answer back using the same protocol. The VXML browser outputs the answer as synthesized voice using a text-to-speech engine. The SpeechWeb browser repeats this step as the user gives more input and gets answers back.
9. If the user input is a request to follow a hyperlink to another application. The

application interpreter returns the URL address of the new application in text format back to the VXML browser. The browser then calls the Java object Rewrite and passes the new application URL address information to it.

10. The Java object Rewrite writes a new VXML page using the given application address. In figure 5, the page is called VXML page 2.
11. Page 1 then transits to page 2.
12. The grammar is then downloaded from the new application server, and the user interacts with the new application.

The process repeats step 8 for more user input. It repeats steps 9 to 11 for new application requests while creating more VXML pages.

6.5 Examples

Here is an example session with a small SpeechWeb using the new SpeechWeb browser:

The user starts the SpeechWeb browser.

The browser provides a graphic user interface shown in Figure 4.

After a correct application address is entered as shown and the user clicks the “Go” button, the SpeechWeb browser connects with the application server and initiates speech:

Application: Hi, my name is Judy. What can I do for you?

User: What do you know?

Application: I know about poems.

User: Tell me a poem.

Application: ... (Application returns a poem)

....

User: Can I talk to Solarman?

The user is requesting a new application. The current application, “Judy”, returns the new application’s address. Then the SpeechWeb browser follows the hyperlink to

the new application “Solarman”.

New application: Hi, I am Solarman. I know about the planets and their moons.

User: Which moons orbit Mars?

New application: Phobos and Deimos.

....

The “Judy” application is written such that if a user gives a request to interact with a new application, he or she should ask “Can I talk to (the new application name)”. If the current application has the new application address, it will return it. Otherwise, the interpreter will not understand the sentence.

Chapter 7 Analysis of the New SpeechWeb Browser

7.1 Overview

In this chapter, we analyze the new SpeechWeb browser with respect to several properties which are related to its use in a Public-Domain SpeechWeb:

1. Clarity of Design.
2. Ease of distribution.
3. Ease of implementation and installation.
4. Use of common communication protocols.
5. Ease of creation and deployment of speech applications.
6. Capabilities.
7. Efficiency and speed.

7.2 Clarity of design

The architecture of the public-domain SpeechWeb discussed in Section 2.3 is different from conventional speech architectures. The idea of LRRP SpeechWebs [Frost et al. 2004] is to separate speech processing and application processing. The SpeechWeb browser is responsible for speech processing, whereas remote servers handle the application processing.

7.2.1 *Design concept*

The following diagram shows the main components in the SpeechWeb browser design.

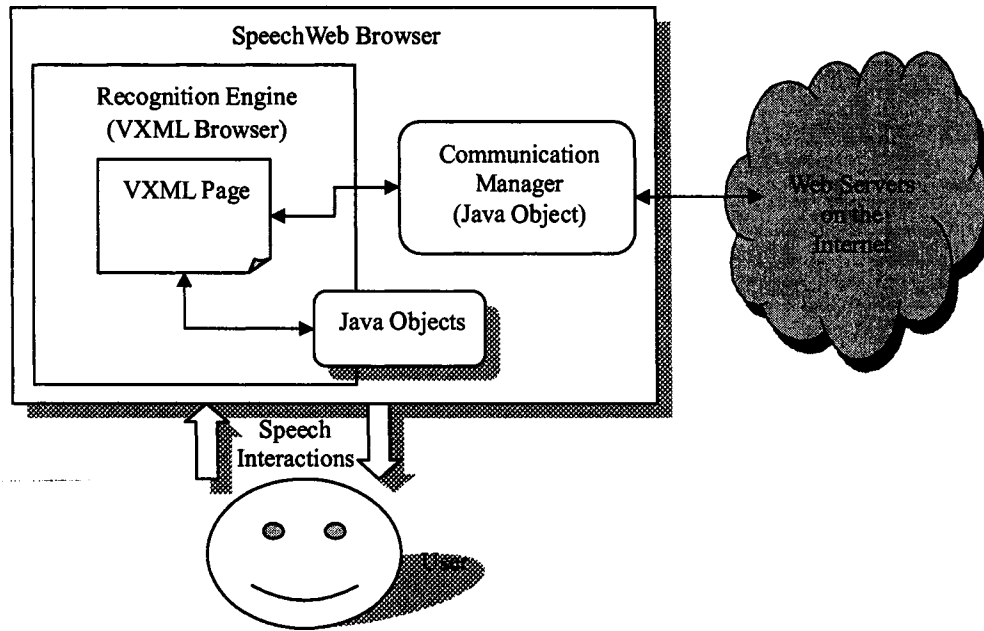


Fig. 6. SpeechWeb Browser Design

We have built a SpeechWeb browser that is capable of speech processing and communicating with remote servers using standard internet protocols. This divides browsing into two parts. One is the recognition engine and the other is the communication manager.

After our initial investigation, for the reasons described in Section 3.2, we chose to use VXML to implement the recognition engine. We only use the speech-processing capability of VXML, not the application-processing capability. Because this is an unusual usage of VXML, we encountered several problems in designing the browser. In order to create dynamic VXML pages which can adapt to different application servers, we had to overcome the deficiency of VXML that the grammar address identifier is not a variable. Despite the failed attempts in designing the browser, finally (because VXML is capable of having embedded ECMA Script and calling specially-coded Java objects) we were able to create VXML pages dynamically. We use this property of VXML and designed the communication manager as another Java object.

7.2.2 The design goal

Our goal was to prove that a speech browser can be constructed as a single VXML page. Although, it seems that the designed SpeechWeb browser uses many pages, for example, the initial VXML page for the graphic user interface and multiple generated VXML pages during runtime, however, essentially, it only uses one VXML page, the template page. A Java object called “Rewrite” reads the template page, changes some variables, according to each application server, and rewrites a new page which is adapted to the new application. This is a novel use of VXML. In this design, the browser uses a small loop within each VXML page to control the interaction between the user and application server. And the browser uses a main loop to control the cycle of VXML pages when user wants to transfer to a different application.

During the design, because we had a clear goal in mind, although we encountered many problems, we were able to solve them by trying different approaches.

7.3 Free distribution

As mentioned in Section 1.2, the prototype version of the SpeechWeb browser constructed by Frost and Chitte [1999] used an integrated IBM speech API for speech recognition and synthesis. Therefore, it cannot be distributed.

The new SpeechWeb browser described in Chapter 6 was developed using VXML, Java and ECMA Script. They are all non-proprietary languages, and the SpeechWeb browser itself did not have any proprietary components. Therefore, the new SpeechWeb browser can be distributed freely.

7.4 Ease of implementation and installation on client devices

The original version of the SpeechWeb browser constructed by Frost and Chitte [1999] was a prototype which was implemented using a pure Java programming language and integrated an IBM speech API. It consisted of more than ten thousand lines of code. The code was complicated and consisted of several hundred lines associated with the end-user interface, low-level integration of the recognition engine API,

communication with the remote applications, and all of the associated exception-handling [Su and Frost, 2005]. Therefore, the original browser was hard to maintain, change or extend. In comparison, the new browser consists of only a few hundred lines of code.

Thanks to VXML, the VXML browser contains the recognition engine and takes care of exception-handling. The new version of the SpeechWeb browser, described in this thesis, consists of only one VXML page, three Java objects with a total of only a few hundred lines of code. Compared to the original prototype version, it is much smaller. It is also much easier to maintain and change.

The SpeechWeb browser installation package is only 55KB in size. It only takes seconds to extract the files to the same directory in the end-user device and the browser is ready to run (assuming that the VXML interpreter and JRE are already installed).

Our new SpeechWeb browser has been successfully implemented and tested on a variety of PCs. It can be implemented on pocket PCs as well. And soon it will likely to be implemented on cell phones.

However, because it uses Java objects, it needs the Java Runtime Environment to be installed on the client devices. This could be a limitation for some devices which do not have JRE built-in.

7.5 Use of common communication protocols

In the new SpeechWeb browser, Java objects are used to implement the communication manager and the graphic user interface. Java is a powerful language. Although, in order to communicate with VXML page, we have to use specially-coded methods to pass and receive data, there is no limit on what the browser can do. We can add more features and function to help the SpeechWeb browser perform better. The wide variety support of Internet protocols of Java can give us different choices when we want to change communication protocols or extend the capability of the communication manager to support different web servers.

The SpeechWeb browser currently uses the CGI-BIN protocol to communicate with remote applications.

7.6 Ease of creation and deployment of speech applications

Because the current version of the SpeechWeb browser uses the CGI-BIN protocol, the remote applications can be deployed on a conventional web server as long as it supports the CGI-BIN protocol. Because the browser uses text transmission, as long as the remote application is able to take standard input and produce standard output, the application can be coded in any language.

This gives end-users lots of flexibility to develop and deploy their own speech applications.

An example of a simple application interpreter implemented using UNIX Shell Script can be found in Appendix II, Chapter 4.

7.7 Capabilities

7.7.1 *Basic functions*

In the SpeechWeb, the responsibility of the SpeechWeb browser is to provide a graphic user interface for initial application-server information, connect to applications in the remote web server, use speech-recognition capability to translate user voice input into text, transfer the text input to the remote application, get the answer back, and output it as synthesized voice.

The graphic user interface only appears in the initial state when the user wants to start browsing the SpeechWeb by connecting to the first remote application. Users can use the default application URL address or customize the default to a new address, then connect to it.

After connecting to the remote application, depending on the defined recognition grammar of the application, there is no limit what the user can ask. As long as the user asks the “right” questions, the browser can recognize the input. Users can always say “help” to get some hints on how to ask the right questions for the application. After a

user requests to transfer to a new application, a new VXML page is created to adapt to the new application and the current VXML page is kept in the local machine temporarily. Later on, just like using an HTML browser, the user can revisit previous applications by saying “go back” to go back to the previous VXML page. On the other hand, users can say “go forward” if there is a page available. There can be as many as ten pages cached in the local machine.

7.7.2 *Improved speech-recognition accuracy*

The new LRRP SpeechWeb [Frost et al. 2004] architecture can be shown to improve the speech-recognition accuracy compared to existing speech architectures. For example, by using local speech recognitions, there is no degradation in text transmission.

By using a single VXML page running in a VXML browser as the recognition engine, the SpeechWeb browser is capable of further improving the speech recognition accuracy.

As mentioned before, VXML is a new technology. It is still under development. We can make use of the latest technologies when they become available. For example, in the future, the browser will be able to store user voice profiles locally and allow user voice training.

7.7.3 *Unable to support dialogue*

The SpeechWeb browser has been successfully tested in a small SpeechWeb. It can do everything we expected. However, it does not support dialogue. This means neither the remote application nor the SpeechWeb browser can remember what the user said. This is because currently we are using the CGI-BIN protocol to communicate between the SpeechWeb browser and the remote applications. Every time the SpeechWeb browser sends a standard input in text format to the remote application using the CGI-BIN protocol, it causes the remote application to execute and get the standard output back as text. However, as soon as the execution is terminated, the connection is lost. We are currently investigating various techniques to be integrated in the

SpeechWeb browser to overcome this shortcoming.

This issue is discussed further in the next chapter.

7.8 Efficiency

One of the design concerns of the SpeechWeb is efficiency. Through the design of the SpeechWeb browser, we want to make the SpeechWeb more efficient.

7.8.1 Communication cost

We described some existing speech architectures in Section 2.2. Instead of using voice transmission like calling call centers or downloading large VXML pages from a conventional VXML server, the communication between the server and client uses text. Although, when every time the SpeechWeb browser contacts a new application, it downloads a tailored grammar used for speech recognition, the grammar file is relatively much smaller than the combination of VXML page and grammar file in the conventional use. As well, there is no degradation in text transmission compared to voice transmission in the call-center architecture.

7.9 Speed

The speed of the SpeechWeb browser has been determined by experiments.

The exact time needed for each step in the SpeechWeb browser to run is recorded in a VXML log file. A log file of a sample session with a remote application looks like the following.

Table 1: Sample session recorded in *vxml.log*

Line	vxml.log
01	15:16:07.609 S: Starting V020904 at Mon Oct 24 15:16:07 EDT 2005
02	15:16:07.609 X: Java: Sun Microsystems Inc. 1.3.1
03	15:16:07.609 X: requested locale: en_US
04	15:16:09.687 S: initializing application
05	15:16:09.703 V: file:C:/VXML/Test/start.vxml (fetch get)

06	15:16:09.734 S: running
07	15:16:09.734 C: Welcome to the speech web browser
08	15:16:10.984 V: file:C:/VXML/Test/t0.vxml (fetch get)
09	15:16:11.000 F: http://luna.cs.uwindsor.ca/~richard/judy/vxmljudy.gram (re-fetch get)
10	15:16:11.031 K: file not modified
11	15:16:11.140 C: you are talking to judy now
12	15:16:11.140 V: #one
13	15:16:11.140 A: not listening
14	15:16:11.171 A: listening
15	15:16:17.359 A: Too quiet (0.2)
16	15:16:17.671 A: Audio level (0.4)
17	15:16:17.968 A: Too quiet (0.2)
18	15:16:19.453 H: hello judy
19	15:16:19.578 C: hi, how are you?
20	15:16:19.578 V: #one
21	15:16:21.703 A: Too quiet (0.2)
22	15:16:22.250 A: Too quiet (0.2)
23	15:16:23.875 H: fine thanks
24	15:16:24.000 C: Good, so am I. In fact I feel great.
25	15:16:24.000 V: #one
26	15:16:28.937 A: Audio level (0.3)
27	15:16:29.359 A: Audio level (0.3)
28	15:16:29.562 A: Audio level (0.5)
29	15:16:29.875 A: Audio level (0.3)
30	15:16:30.078 A: Audio level (0.4)
31	15:16:30.296 A: Too quiet (0.2)
32	15:16:30.515 A: Too quiet (0.1)
33	15:16:31.750 H: can I talk to monty
34	15:16:31.859 C: yes. here he is
35	15:16:31.875 V: #transit
36	15:16:31.921 V: file:C:/VXML/Test/t1.vxml (fetch get)
37	15:16:31.953 F: http://luna.cs.uwindsor.ca/~richard/monty/vxmlmonty.gram (re-fetch get)
38	15:16:32.015 K: file not modified
39	15:16:32.046 C: you are talking to monty now
40	15:16:32.046 C: Hi, I am Monty. I know a joke
41	15:16:32.046 V: #one
42	15:16:40.968 A: Audio level (0.6)
43	15:16:41.281 A: Audio level (0.6)
44	15:16:41.703 A: Audio level (0.3)
45	15:16:41.906 A: Audio level (0.5)
46	15:16:43.296 H: where do you live
47	15:16:43.406 C: I hang out in one of Frosties computers.
48	15:16:43.406 V: #one
49	15:16:48.265 A: Audio level (0.5)

50	15:16:48.578 A: Audio level (0.5)
51	15:16:48.890 A: Audio level (0.4)
52	15:16:49.000 A: Audio level (0.5)
53	15:16:49.203 A: Audio level (0.3)
54	15:16:50.828 H: tell me a joke
55	15:16:50.953 C: Did you here about the two professors? They were walking through the forest, when they saw some tracks. Moose tracks, yelled one of them. Deer tracks, yelled the other. Moose tracks. Deer tracks. They argued for an hour. Then the train hit them.
56	15:16:50.953 V: #one
57	15:16:58.609 A: Too quiet (0.2)
58	15:16:58.921 A: Audio level (0.6)
59	15:17:00.265 H: goodbye
60	15:17:00.375 V: #goodbye
61	15:17:00.375 C: goodbye
62	15:17:03.765 W: processing problem words...
63	15:17:03.765 W: 0 problem words found
64	15:17:03.765 K: clean cache 1
65	15:17:03.781 K: clean cache 2
66	15:17:03.796 K: clean cache 3
67	15:17:03.796 K: lock retries 0
68	15:17:03.796 S: exit (0)

-----In the VXML log file, each entry/line is in a format of *timestamp code: message*, where the timestamp is in 24-hour format and displayed as *hh:mm:ss.sss*.

The code uses different characters to denote different action taken. Table 2 provides the detailed code descriptions in the log file.

Table 2: *vxml.log* trace code [IBM WebSphere, 2002, table 20, pp.67]

Code	Description
A:	Logged when the VoiceXML browser detects audio input, but the speech recognition engine does not return a recognized phrase; this may be due to breath or background noise. The message column contains audio level messages.
C:	Logged when the computer plays a prompt. The message column displays the prompt text.
E:	Logged when the VoiceXML browser throws an event. The message column indicates the type of event.
F:	Logged when the VoiceXML browser fetches a resource such as a grammar file, an audio file, or a script. The message column contains the URI of the file, and whether it was fetched from the server or was in the cache.

H:	Logged when the user responds using voice input. The message column displays the word or phrase that was recognized by the speech recognition engine.
K:	Provides information about cache activity.
S:	Indicates system information, including the version of the VoiceXML browser and the system time. This information is provided only in the vxml.log file; it is not written to the Java console.
V:	<p>Logged when the VoiceXML browser fetches a .vxml file. The message column contains the URI of the file, and whether it was fetched from the server or was in the cache.</p> <p>Note:</p> <p>When transferring control within a document (using a URI that begins with #), the document is not reloaded; in this case, the message column contains the form or menu id.</p>
W:	Provides a list of "problem words" for which the recognizer or synthesizer may not have had reliable pronunciations. You can use the lexicon tool and the <ibmlexicon> tag to generate pronunciations. An audio clip that cannot be loaded (for example, there is a network error when fetching it or it does not exist) and as per the VoiceXML 1.0 specification, the content of the <audio> tag is being substituted.
X:	Specifies the version of the JVM. This information is provided only in the vxml.log file; it is not written to the Java console.
?:	Logged when the speech recognition engine determines that the user said something, but the confidence level is not high enough to justify using the results. In response, the VoiceXML browser throws a nomatch event. The message column contains the word or phrase that was recognized.

The action of each entry is recorded precisely in milliseconds. We can use the timestamp, identify the entries using code character and calculate the exact time used for each entry to run. Therefore, we can find out the speed of the SpeechWeb browser when various actions are taken.

The following is an analysis of time of execution for various tasks involved in speech browsing.

7.9.1 Graphic User Interface

The GUI is only loaded once at the beginning of a user session. It is loaded in the initial VXML page, *start.vxml*. Using the interface involves several steps.

The *start.vxml* first calls a Java object. This Java object reads a local file *default.sihlo* which contains default remote application information. It then creates a

graphic interface shown in Section 6.3 which displays the application URL and waits for user action. When the user clicks the go button, the Java object gathers the remote application address and passes it to another Java object to write a new VXML page called *t0.vxml* based on a template VXML page. Then *start.vxml* transit to *t0.vxml* and the SpeechWeb browser initiates speech and waits for user voice input.

The whole process from displaying the interface to awaiting user voice input is from line 05 to line 14 in the sample log file in Table 1. Use the timestamp in line 14 subtract the time in line 05, the answer indicates the time used for the Graphic User Interface to finish its task in this particular execution. (Actual timings are given in sub-section 7.9.4.)

7.9.2 *Speech interaction*

In the VXML log file, we can identify the recognized user voice input and browser synthesized speech output using codes “H” and “C”. When “H” is followed by “C”, this is normally when the user gives a voice input and the browser returns a speech output.

This process involves the following steps. The VXML browser first recognizes the voice input and translates it into text. Then the text input is transmitted to the remote application through the communication manager using the CGI-BIN protocol. The interpreter in the remote application processes the input and returns a string which contains the sever response back to the communication manager and the browser outputs the text as synthesized voice.

For example, in Table 1, line 18 and 19, line 23 and 24 are the two speech interactions between the user and a remote application. We can calculate the speed of the SpeechWeb browser when user interacts with a remote application by doing subtraction of the two timestamps of the two entries in each speech interaction. (Actual timings are given in sub-section 7.9.4.)

7.9.3 *Transition to a new application*

When the user asks to transit to a new application, the answer returned from the

current remote application is a string that contains both a speech response and a new application URL. The browser then passes the URL to a Java object. This Java object follows the URL and retrieves a header file in the remote server which contains the new application information. The Java object uses the information and writes a new VXML page. Finally, the browser transits to the new page and the user can start interacting with the new application.

In Table 1, an application transition is documented from line 34 to line 41. Using the timestamps, the speed of each transition can be calculated.

7.9.4 *Timing results*

The SpeechWeb browser runs on a PC with Windows XP SP2 operating system on a hardware configuration of AMD 1.733GHz CPU, 512MB 333MHz RAM and equipped with a high-speed cable Internet connection with upload speed up to 640Kbps and download speed up to 5Mbps [Cogeco Cable, 2005].

The applications reside on the University of Windsor Computer Science LUNA server. All applications are coded in Miranda, combined with UNIX Shell Script to support the CGI-BIN protocol.

The speed of the GUI, speech interactions and application transitions was calculated by running the SpeechWeb browser 50 times each day on 3 different days (in case the Internet speed changes).

The experiment results are in the following table.

Table 3: SpeechWeb browser speed (in seconds)

	SpeechWeb GUI	Speech Interactions	Application Transitions
Average	8.4694	0.1203	0.1952
Median	6.868	0.109	0.125
Minimum	1.188	0.062	0.078
Maximum	16.766	0.797	1.547

7.9.5 Analysis of the results

For the speech-interaction experiments, simple questions are used as user input. These questions are not query-like questions. The answers are straightforward. Remote applications can find the answers without doing additional processing, for example, “What is your name?” This kind of question minimizes the remote application processing time. Therefore, they are better reflections of the SpeechWeb browser processing speed. In the experiments, the input and output are simple sentences no longer than a few hundred characters. Although the length of the input and output affects the transmission time over the Internet, normally a voice input is a sentence less than 100 bytes and each output is less than 1000 bytes. Even if the length of each input and output varies, because the size of the data is relatively very small, there should not be significant differences in transmission time over the Internet.

The application transition involves downloading the recognition grammar. Depending on the size of the grammar, the downloading time may vary. However, a complicate grammar normally should not exceed 100 KB, because as the grammar size increases, the recognition accuracy decreases. Even if the grammar is 100KB, for a high-speed Internet user, the download speed should still be tolerable, as it should only take a couple seconds. In the experiments, the recognition grammars are no bigger than 10 KB.

As you can see in Table 3, the average time and median time for speech interactions and application transitions in the experiments are all under 200 milliseconds. Therefore, it is virtually real-time for speech interactions and very fast for application transitions.

However, the browser GUI speed is much slower than the interaction and transition speed. That’s because the VXML browser used in the experiments runs on a Java Virtual Machine, it is already slow when starting up in a Windows operating system. It makes the speed even slower to create a graphic interface. Fortunately, the interface is only used when starting the browser. Once a remote application is connected, the interface is not used any more.

One thing should be pointed out. The minimum time and maximum time are

several times shorter/longer than the average and median time. This is not surprising because in Windows operating systems, there are always some applications running and occupying the resources such as the CPU and RAM. However, the results that are close to the minimum and maximum time only happened a few times. Compared to the large number of the results sampled in the experiments, they did not have much effect on the outcome of the average and median time.

Overall, the SpeechWeb browser runs very fast in an average-configured PC equipped with a home high-speed Internet connection. Although the speed of the browser GUI is relatively slow, with a faster computer, the speed may be more tolerable.

Chapter 8 Future Work: Dialogue Support in the SpeechWeb

8.1 Spoken dialogue in speech interaction

Spoken dialogue in a speech interaction can be defined as a conversation. In a conversation between humans, one may ask a question like “How did you know him?” In order for the other person to answer the question, “him” has to be specified in previous sentences and it is remembered by the person answering the question during the conversation.

In a human-machine spoken dialogue interaction, it is essential for the machine to have a mechanism that is able to “remember”. The machine has to be able to remember which user it is talking to and information already received during previous interactions from the user in order to carry out a good conversation.

For example, a simple task such as filling in user address information requires dialogue support.

System: *What is the street number and name?*

User: *401 Sunset Avenue.*

System: *Which city is it?*

User: *Windsor.*

System: *Which province is it?*

User: *Ontario.*

System: *What is the postal code?*

User: *N9B 3P4.*

System: *Your address is: 401 Sunset Avenue, Windsor, Ontario, Postal code: N9B 3P4. Is it correct?*

User: *Yes.*

In the above example, because the address information is filled by the user step by step, the system has to remember the information already filled.

8.2 Unable to support dialogue

In a telephone-based spoken dialogue system, such as a call center, once a user is connected to a call center, the communication channel is always open until the user hangs up the phone. However, in a web-based spoken dialogue system, such communication channel does not exist for text transmissions. There can be many simultaneous connections from other users to the application server. Unless there is a mechanism in the server which can identify each user, the server is not able to know which user it is communicating with.

As described in Section 7.6.3, the SpeechWeb browser uses the CGI-BIN protocol to communicate with the remote application. In this protocol, once a user input is processed and returned, the connection is closed and the application server loses track of the user. The application server cannot identify different users. Therefore, the new SpeechWeb browser does not support dialogue.

8.3 How to support dialogue in the SpeechWeb

According to Glass [1999], a spoken-dialogue system requires an automatic speech recognizer for speech recognition to perform speech-to-text conversion, some form of dialogue manager or dialogue management system to control the interaction with the user, and a mechanism for conveying information to the user (e.g., text and/or speech generation). More complex systems will also incorporate modules for natural language understanding and a mechanism for handling local discourse phenomena, in addition to the dialogue manager.

In the SpeechWeb, users can have speech interactions with remote servers, because the SpeechWeb browser provides the automatic speech recognition and speech synthesis, and the remote applications are capable of processing user input and generating output. However, the SpeechWeb is not a complete spoken-dialogue system, because the speech applications in the SpeechWeb do not have dialogue managers plus the shortcomings of the CGI-BIN protocol described in the last section.

Therefore, in order to support dialogue in the SpeechWeb, two things have to be

done: change or upgrade the communication protocol used between the server and client, and implement dialogue managers in remote applications.

8.3.1 *Communication protocols*

The current communication protocol used by the SpeechWeb browser is the CGI-BIN protocol. It is possible to adapt the CGI-BIN protocol to support dialogue. However, that requires the application server to have a mechanism which can identify and keep track of each individual user. It also requires the browser to have a mechanism which can provide the server their own identity. It could be a complicated task to integrate these mechanisms in both the server and browser.

Fortunately, there are other protocols available which support the identification and tracking of different users. For example, in an HTML-based online forum, one way to solve this problem is that after a user is logged in, the server creates and sends a Cookie to the user. Each Cookie has a unique number that can be used to identify the user. Every time the user communicates with the server, the cookie is sent along with other information for the server to keep track of the user. This kind of protocol is already supported by Hypertext Preprocessor (PHP), Active Server Pages (ASP), Java Server Pages (JSP), Java Servlets, etc.

In order to support dialogue, instead of using the CGI-BIN protocol, Java Servlets would be a good choice for the SpeechWeb browser and application servers to adopt. However, because Java Servlets are originally designed for HTML-based applications, some modifications need to be done on both the client and server side.

On the client side, in the SpeechWeb browser, a possible way to use the Java Servlets is to create an HTML browser-like environment, so that the SpeechWeb browser is able to act like an HTML browser. For example, the browser can store Cookies, create session and post messages like the `<form>` tag in HTML.

On the server side, instead of sending HTML code, the Servlets can send only the information which the SpeechWeb browser needs.

Adopting the right protocol to support dialogue is not an easy task, especially for the SpeechWeb browser, because it is not the conventional way of using the protocols.

Once the right protocol is adopted, SpeechWeb applications residing on remote servers still need to have a dialogue capability.

8.3.2 *Dialogue support in remote applications*

The development of a spoken dialogue system is a complex process. It involves designing and integrating different dialogue components and technologies. It would be a formidable task to build and integrate a dialogue system from scratch [McTear, 2002].

In the SpeechWeb, one of the most important tasks for supporting dialogue with a remote application is to integrate a dialogue manager.

Appendix I is a comprehensive survey on spoken-dialogue technologies and design strategies used to improve speech-recognition accuracy. In the survey, different approaches used to design a spoken dialogue manager were investigated, such as a finite-state-based approach, an object-orientation-based approach, and so on. Some toolkits such as CSLU (Center for Spoken Language Understanding) Toolkit [Sutton, et al. 1998] for developing spoken-dialogue systems are discussed. However, there isn't a best approach for designing a dialogue manager, but a most suitable one for individual applications.

"Spoken Dialogue Technology by Michael F. McTear" is a good book for someone wanting to build a remote application supporting spoken dialogue. In the book, McTear described working spoken dialogue systems built using the CSLU Toolkit and the IBM WebSphere Voice Server supporting VXML. The sections on VXML, in particular, supply the reader with hands-on experience using the current industry standard [Portele, 2004].

Chapter 9 Conclusions

9.1 What has been achieved?

The Thesis Statement was that:

A viable public-domain SpeechWeb browser can be built as a single VXML page.

The thesis has been proven by:

1. Investigating the new distributed speech markup language VoiceXML.
2. Investigating different potential structures for the speech browser as a single VXML page.
3. Identifying a potential design.
4. Successfully implementing and testing the SpeechWeb browser on a PC.
5. Analyzing the new SpeechWeb browser with respect to its “viability”, showing that:
 - a) The SpeechWeb browser can be implemented using non-proprietary languages, and can be distributed freely.
 - b) The browser can be installed on client devices easily. The browser is also easy to maintain and change because of its small size.
 - c) The browser uses a common Internet protocol which is supported by most web servers.
 - d) It is easy to deploy a SpeechWeb application, because users can use any programming language to code the applications and deploy them on conventional web servers.
 - e) It is very efficient to use text transmission between server and client.
 - f) The SpeechWeb browser has a very fast speed. It has virtually real-time speech-interaction speed when tested on a home PC using a regular high-speed Internet connection.

9.2 Contributions

The successful implementation of the new SpeechWeb browser enables the implementation of the new architecture for a public-domain SpeechWeb proposed by Frost [2005].

The SpeechWeb browser provides a platform for research on natural-language speech processing and language understanding, and its free distribution makes it possible for large public involvement.

The new SpeechWeb browser can be downloaded from:

<http://cs.uwindsor.ca/~speechweb/>

9.3 Suggestions for future work

As discussed in Chapter 8, the SpeechWeb browser could be extended to support spoken dialogue so that users can have more human-like conversations with remote applications. Also, it is likely that the single-VXML-page SpeechWeb browser can be implemented on cell phones using language such as J2ME.

Bibliography

- [1] Cogeco Cable, High Speed Internet - Packages and Pricing - Cogeco Cable - Ontario - Canada, <http://www.cogeco.ca/en/high-speed-internet-o.html>, 2005.
- [2] Frost, Richard A. A Call for a Public-Domain SpeechWeb, Communication of the ACM, Volume 48, Issue 11 (2005) 45-49.
- [3] Frost, R. A., N. Abdullah, K. Bhatia, S. Chitte, F Hanna, M. Roy, Y. Shi and L. Su, LRRP SpeechWebs, IEEE CNSR Conference (2004) 91-98.
- [4] Frost, R. A. and Chitte, S. A New Approach for Providing Natural-Language Speech Access to Large Knowledge Bases. In Proceedings of the Conference of the Pacific Association for Computational Linguistics (University of Waterloo, 1999), 82-90.
- [5] Glass, J. R. Challenges for Spoken Dialogue Systems, In Proceedings of the 1999 IEEE ASRU Workshop, 1999.
- [6] Hemphill, C.T. and Thrift, P. R. Surfing the Web by Voice. Proceedings of the third ACM International Multimedia Conference (San Francisco 1995) 215 – 222.
- [7] IBM WebSphere Voice Server for Windows 2000 and AIX Software Developers Kit, VoiceXML Programmer's Guide Version 3.1, 2002.
- [8] Lucas, B. VoiceXML for Web-based Distributed Conversational Applications. Communication of the ACM 43 (9) (2000) 53-57.
- [9] McTear, M. F. Spoken Dialogue Technology: Enabling the Conversational User Interface, ACM Computing Surveys, Volume 34, Issue 1, 90 – 169, 2002.
- [10] Portele, T., Review of "Spoken Dialogue Technology by Michael F. McTear". ACM Volume 2, Issue 9, 2004.
- [11] Shi, Y., "Investigation of Recognition-Grammar Design" Master's Thesis, School of Computer Science, University of Windsor, ON, Canada, 2003.
- [12] Su, L. and Frost, R., A Novel Use of VXML to Construct a Speech Browser for a Public-Domain SpeechWeb. In Proceedings of the 18th Conference of the Canadian Society for Computational Studies of Intelligence (Victoria, B.C., 2005), 401-405.

[13]W3C, Voice eXtensible Markup Language (VoiceXML™) version 1.0 W3C Note 05 May 2000, <http://www.w3.org/TR/voicexml/>, 2000.

[14]W3C, Voice Extensible Markup Language (VoiceXML) Version 2.0 W3C Working Draft 23 October 2001, <http://www.w3.org/TR/2001/WD-voicexml20-20011023/>, 2001.

Related Publications by the Author

Su, L. and Frost, R. A., A Novel Use of VXML to Construct a Speech Browser for a Public-Domain SpeechWeb. In Proceedings of the 18th Conference of the Canadian Society for Computational Studies of Intelligence (Victoria, B.C., 2005), 401-405.

Appendix I

A Survey on: Improving Recognition Accuracy through Use of Dialogues

Abstract

This survey provides an overview of different spoken-dialogue design strategies and techniques that are used to improve speech-recognition accuracy. In the last decade or so, speech technology has made major advances. Spoken-dialogue systems have been developed along with the speech technology. However, there are many different approaches used to design and implement spoken dialogues. Through dialogue design, it is possible to improve the dialogue management, language understanding, etc. and ultimately, improve speech-recognition accuracy. In this survey, we will present some of the major dialogue-design strategies and toolkits used to implement spoken-dialogue systems.

1 Introduction

Spoken-dialogue systems allow users to interact with computer-based applications such as databases and expert systems by using natural spoken language [McTear, 2002]. We see lots of examples of spoken-dialogue systems in science fiction movies where people can talk to computers just like having conversations with other people. Computers understand perfectly each sentence and they sound like real people speaking. For the computer, the ability to understand human voice is based on the technology of speech recognition; the ability to output human-like voice is based on speech synthesis and the ability of having conversation is based on language understanding, dialogue modeling, design and management. However, those scenes in movies are far from reality. In reality, there are still lots of limitations of having a real

human-like conversation. For example, without limited domain or speech training, the speech recognition accuracy may not be high enough to reach human tolerance, the synthesized voice does not sound sufficiently human, and so on. Our goal is to make the computer be able to listen, understand and speak as human-like as possible. One way to get us closer to that goal is through dialogue design.

2 Spoken-Dialogue Systems

Research on the human-machine conversation problems have been around for several decades. However, it has been observed by McTear [2002] that only in the past decade or so, there has been development of a large number of spoken-dialogue systems. That is because there have been major advances in speech recognition and speech-processing technologies.

Spoken-dialogue systems provide an interface between users and computer-based applications that allow users to interact with the applications using speech. The architecture of different spoken-dialogue systems may vary significantly. But at a minimum, according to Glass [1999], “A spoken dialogue system requires an automatic speech recognizer (ASP) for speech recognition to perform speech to text conversion, some form of dialogue manager (controller) to control the interaction with the user, and a mechanism for conveying information to the user (e.g., text and/or speech generation). More complex systems will generally incorporate modules for automatic speech recognizer, natural language understanding (NLU), language generation, speech synthesis, and a mechanism for handling local discourse phenomena, in addition to the dialogue manager.”

The following diagram is the spoken-dialogue system presented in Glass [1999] used for MIT systems, which is similar to the structures of many other dialogue systems.

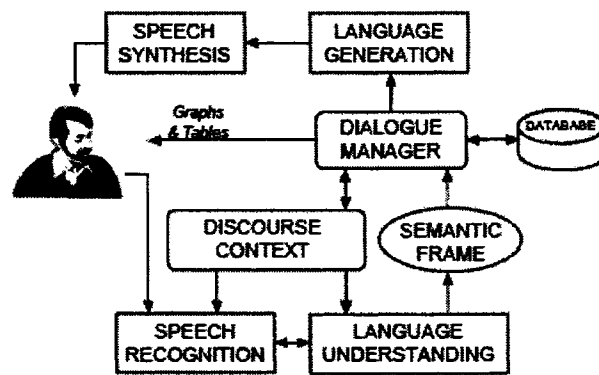


Figure 1. An example structure of a spoken dialogue system [Glass, 1999, Fig.1, pp2]

As you can see in figure 1, in the center of the diagram is the dialogue manager. It connects the speech recognition mechanism, the speech generation mechanism as well as the database. It is like a brain that controls the whole spoken dialogue system. In this survey, our main focus is on the design and structure of the dialogue manager, which Glass [1999] claims to be one of the most basic design decisions in creating a spoken-dialogue system.

The design of the spoken-dialogue manager or dialogue-management design is also simply known as dialogue design.

3 Spoken-Dialogue Design

One of the essential aspects of spoken-dialogue design is to create a user-friendly environment that provides pleasant and easy interactions between users and systems. "Pleasant and easy interaction" often refers to fewer misunderstandings or non-understandings during the interactions. Because human-to-human speech interactions often have misunderstandings and non-understandings because of different reasons, it is virtually impossible not to have these problems in human-machine speech interactions. According to Churcher [1997], "It is important that although your system may fail, it doesn't mean that the dialogue should, too." Therefore, a good dialogue manager should be able to identify errors and adopt a

strategy which recovers the dialogue efficiently.

According to the different strategies and methods used to control the dialogues with the user, spoken-dialogue design can be classified into different types/approaches.

3.1 Graph and finite-state-based design

“Graphs and finite state models are the simplest of dialogue managers. They are ideally suited to applications where the dialogue structure is the same as the task structure.” [Churcher, 1997]

McTear [2002] describes how finite state-based systems work: “In a finite state-based system the user is taken through a dialogue consisting of a sequence of predetermined steps or states. The dialogue flow is specified as a set of dialogue states with transitions denoting various alternative paths through the dialogue graph. Generally systems with state-based dialogue control restrict the user’s input to single words or phrases that provide responses to carefully designed system prompts. A major advantage of this form of dialogue control is that the required vocabulary and grammar for each state can be specified in advance, resulting in more constrained speech recognition and language understanding.” This sequential approach is also described in [Levin, et al. 1999].

Graph and finite-state-based dialogue control are relatively easy and straightforward to design. And their behaviors are predictable [Goddeau, et al. 1996].

This kind of dialogue design has been successfully used (e.g. [Nielsen and Baekgaard 1992] and [Muller and Runger 1993]). However, there are major disadvantages in graph and finite state-based system. The dialogue systems, as Churcher [1997] describes, have received criticism over their lack of flexibility both to deviations in the dialogue structure, and also in their applicability to other domains, because they restrict user inputs to pre-determined words and phrases. Making correction of misrecognized items is difficult as well as inhibiting the user’s opportunity to take the initiative and ask questions or introduce new topics.

A detailed example of a finite-state-based dialogue control system is described in McTear [2002, pp.96], Nuance automatic banking system.

3.2 Form-based design

Goddeau et al [1996] proposed an alternative approach to design the dialogue control system, which was intended to overcome the limitations of the finite-state-based design. The new approach, called form-based design, is based on the notion of filling in an electronic form, or “E-form”. Using this approach designed dialogue systems are intended to enable both user-initiated and system-initiated interactions, to improve robustness both to system errors and changes in user goals, and to allow construction of system whose behavior is predictable and verifiable.

Goddeau et al [1996] also presents a detailed example of an E-form-based dialogue planner in an application used to search a database of used car advertisements. The authors claim that the E-form structure seems to be an effective mechanism for accumulating and canonicalizing the constraints articulated by the user. It is also a convenient framework for specifying the combination of new information with dialogue context.

3.3 Task-based design

Allen et al [1996] designed a spoken-dialogue system using task-based design. The task-based design is to measure how well the system does at helping the user solve the problem. The two most telling measures are time-to-completion and the quality of the final solution. The authors presented a travel-by-train route-planning system and claim that the task-based approach does set a new standard for spoken-dialogue systems. It would allow them to address a large number of research issues in the future in a more systematic way, supported by empirical evaluation.

Rudnicky et al [1999] referred to the work of Allen et al [1996] and claimed to have a similar idea that “the role of the computer is to support problem solving activity on the part of the user, with the user having primary initiative in the process,

in the sense of setting goals and judging the acceptability of solutions. The role of the system is to provide information for decisions, propose solutions and highlight potential constraint violations.” Therefore, the structure of the “form” is not always predictable in advance. Moreover, for example, in a travel-planning system, a significant portion of the interactions in the domain deals with the consequences of constraints imposed by the user’s goals (which may change or evolve over the course of a session) and the possibilities afforded by the domain (which similarly may change over the course of a session). Rudnický et al [1999] claim to believe that the task-based dialogue-management approach they use will scale with the complexity of the underlying domain and the tasks that people expect to be able to perform.

3.4 Agenda-based design

Based on the task-based design presented in [Rudnický, et al. 1999], Rudnický and Xu [1999] proposed an agenda-based design approach.

The agenda-based approach addresses the problem of dialogue management in complex problem-solving tasks by treating the task at hand as one of cooperatively constructing a complex data structure, a product, and uses this structure to guide the task. In the agenda-based design, there are two new data structures: an agenda to replace a fixed form and a dynamic product (a form structure that could be dynamically constructed) that could evolve over the course of a session. “In the agenda-based system, the product is represented as a tree, which reflects the natural hierarchy, and order, of the information needed to complete the task. The agenda is an ordered list of topics, represented by handlers that govern some single item or some collection of information. The agenda specifies the overall “plan” for carrying out a task.” [Rudnický and Xu, 1999]

Rudnický and Xu [1999] claim that the agenda mechanism can be adapted easily to less-complex domains that might be implemented as a standard form-based system. However, they did not know how well the technique would succeed for domains of complexity comparable to travel planning but with different task structure.

3.5 Schema-based design

Constantinides et al [1998] state that the form-based design approach, such as that of [Goddeau, et al. 1996], leads to the question of whether such an approach could be adapted for domains that may require more sophisticated dialogue management and proposed a schema-based design approach for spoken dialogues.

Constantinides et al [1998] used implementations of two dialogue systems, a personal-calendar system and an air-travel-planning system as examples to describe two mechanisms that can provide more active system participation than form-based approaches in dialogue control. The schema-based design uses two mechanisms, a modified stack that allows the system to track multiple topics, and a form-specific schema that allows the system to deal with tasks that involve completion of multiple forms.

3.6 Object-Orientation-based design

O'Neill and McTear [2000] present an object-oriented approach to design dialogue-management systems. The authors claim that, in software engineering, object-orientation has proven to be an effective means of separating the generic from the specialized, and letting the specialized inherit the generic. This can be applied to dialogue design, because in spoken dialogues, certain skill-sets and patterns of dialogue evolution are common to many different contexts; regardless of the business domain of the transaction, there are certain set-pieces that are common to all such transactional dialogues.

O'Neill and McTear [2000] have used Prolog++, combined with the object-oriented approach with a self-organizing, mixed-initiative dialogue strategy to build a prototype of a travel-planning system. In the design, a generic dialogue functionality – turn-taking, confirmation strategy was used. In their later work, [O'Neill and McTear 2002], the relationship between generic confirmation strategies and domain-specific heuristics for furthering transactions was explored.

O'Neill et al [2002] have created a new dialogue-management system using Java, which is based on the intuitive programming style of the Prolog++ systems. The Java dialogue manager expands the domain-specific heuristics to encompass not only the interaction between the system and the user, but also the interaction between the dialogue manager and a database.

O'Neill and McTear [2000] claim that the object-oriented approach and methodology presents a practical approach to the task of turning the processes observed in natural spoken dialogues into maintainable extensible software systems.

3.7 Dialogue design overview

We have seen different dialogue design approaches. The following table gives an overview:

Date	Paper	Graph	Form	Task	Agenda	Schema	O.O.
1996	Allen, et al.			X			
1996	Goddeau, et al.		X				
1997	Churcher, et al.	X		X			
1998	Constantinides, et al.					X	
1998	McTear	X					
1999	Dahlbäck & Jönsson			X			
1999	Lin, et al.				X		
1999	O'Neill & McTear						X
1999	Rudnick, et al.			X			
1999	Rudnick & Xu				X		
1999	O'Neill & McTear						X
2000	O'Neill & McTear						X
2002	McTear	X	X	X	X		

The above table gives a clear view of research papers containing the topics of different dialogue design approaches. All the papers are listed in section 8, the annotated bibliography.

Graph and finite-state-based approaches are the simplest and most straightforward approaches to the design of dialogue managers. However, they lack flexibility. For example, they restrict user input and inhibit the user's opportunity to take the initiative and ask questions or introduce new topics.

Form-based and task-based approaches are designed to overcome the limitations of the graph and finite-state-based approaches. They both enable user initiative during a spoken dialogue.

The agenda-based approach is an improvement based on the form-based approach. It is a more dynamic approach that can create forms at runtime and evolve over the course of a session.

The schema-based approach is also an improvement based on the form-based approach. It can be adapted to domains that require sophisticated dialogue management such as a travel-planning system.

The object-orientation-based design is relatively a new approach which takes advantage of the inheritance property in object-orientation. Regardless of the domain of different dialogue systems, there are some common generic components which can be specified in implementation. Therefore, this approach treats different dialogue systems as a maintainable extensible software system.

4 Dialogue-Management-System Development Methodologies

According to McTear [2002], "Developing a spoken dialogue system involves deciding on the tasks that the system has to perform in order to solve a problem interactively with a human user; specifying a dialogue structure that will support the performance of the task; determining the recognition vocabularies and language structures that will be involved; and designing and implementing a solution that meets these criteria."

There are different methods to develop a dialogue-management system. According to Churcher et al [1997], a popular approach is to use iterative Wizard of Oz experiments to elicit the language and refine the interaction between system and

user. It is typical for human/machine dialogues, where the system needs to control the dialogue to a certain extent.

Wizard of Oz experiments are often used to determine the user's behavior towards a system when the system is still being developed [Fraser and Gilbert 1991]. Inspired by the film, Wizard of Oz, a human simulates the role of the computer, providing answers using a synthesized voice, and the user is made to believe that he or she is interacting with an actual, automated system.

Churcher et al [1997] described the process of the Wizard of Oz experiment. The simulation is controlled with carefully-designed scenarios, in which the user has to find out one or more pieces of information from the system. The iterative process continues until two objectives are reached: the users find the interaction comfortable and user's response can be anticipated. By tailoring the system's question so that the user's response is limited, language closure can be achieved.

According to McTear [2002], there is a major disadvantage in the Wizard of Oz experiment, because "It is difficult for a human experimenter to behave exactly as a computer would, and to anticipate the sorts of recognition and understanding problems that might occur in the real system." McTear suggested the "System in the Loop" method may be used to overcome this disadvantage. For example: "A system with limited functionality is used to collect data. The system might incorporate on the first cycle speech recognition and speech understanding modules, but the main dialogue management component may still be missing. On successive cycles additional components can be added and the functionality of the system increased, thus permitting more data to be collected. It is also possible to combine this method with the Wizard of Oz method, in which the human wizard simulates those parts of the system that have not yet been implemented."

An example of the use of Wizard of Oz to study how humans handle recognition errors before an actual dialogue system is built can be found in [Skantze, 2003].

5 Toolkits for Developing Spoken-Dialogue Systems

The development of a spoken dialogue system is a complex process. It involves designing and integrating different dialogue components and technologies. It would be a formidable task to build and integrate a dialogue system from scratch [McTear, 2002].

Fortunately, many different toolkits are available that support the construction of spoken-dialogue systems. According to McTear [2002], some of the toolkits are for commercial use, such as the Nuance Developers' Toolkit (Nuance Communications), SpeechWorks, Natural Language Speech Assistant (NLSA) (Unisys Corporation), SpeechManiaTM: A Dialogue Application Development Toolkit (Philips Speech Processing), and Vocalis SpeechWareTM.

Some of them were developed mainly to support academic research and to support the teaching of spoken-language technology, such as the Generic Dialogue System Platform (CPK, Denmark), CU Communicator system by University of Colorado, GULAN – An Integrated System for Teaching Spoken Dialogue Systems Technology (CCT/KTH), and the CSLU toolkit (Center for Spoken Language Understanding at the Oregon Graduate Institute of Science and Technology).

The following table is an overview of some currently available toolkits.

Toolkit	Developer	Academic	Commercial
Generic Dialogue System Platform	Center for PersonKommunikation (CPK)		X
CSLU toolkit	Oregon Graduate Institute of Science and Technology	X	
CU Communicator	University of Colorado	X	
GULAN	Royal Institute of Technology in Stockholm (KTH)	X	
IBM WebSphere Voice Server	IBM		X
Natural Language Speech Assistant	Unisys Corporation		X
Nuance Developers' Toolkit	Nuance Communications		X

SpeechMania™	Philips		X
Vocalis SpeechWare™	Vocalis Ltd.		X

5.1 CSLU Toolkit

Sutton, et al [1998] describe the CSLU Toolkit, developed by the Center for Spoken Language Understanding at the Oregon Graduate Institute of Science and Technology since 1993. The purpose of developing the toolkit was to incorporate state-of-the-art spoken-language technology into a portable, comprehensive and easy-to-use software environment. The toolkit integrated learning materials, authoring tools and core technologies such as speech recognition, text-to-speech synthesis, facial animation and speech reading.

The toolkit includes a graphically-based authoring environment, Rapid Application Developer (RAD), for designing and implementing simple spoken dialogue systems using a finite-state-based dialogue model [McTear, et al. 2000]. McTear [1998] claimed that the finite-state-based models have been criticized because of their inflexibility as well as their inability to model complex dialogues. However, Sutton, et al [1998] claimed that RAD makes it possible for people with little or no knowledge of speech technology to develop speech interfaces and applications in a matter of minutes.

The CSLU Toolkit is free-of-charge for non-commercial use. It is available for download at <http://www.cslu.ogi.edu/toolkit/>.

There are many other toolkits available which one can choose for developing a spoken-dialogue system. However, most of them are only commercially available. For academic research, CSLU Toolkit should be considered.

6 Examples of Spoken Dialogue Systems

Dahlbäck and Jönsson [1999] state that instead of using the conventional task-based approach to handle the cases where the user's initial information request is underspecified, and where therefore the system needs to ask the user about the

missing pieces of information, how the architecture of the modularized LIN-LIN dialogue manager can be augmented to handle these cases. They advocate use of a separate knowledge module called an Information Specification Form for managing these cases. They conclude that in dialogue system, it is important to divide information into knowledge sources so that they can be used for various purposes.

Lin, et al [1999] observed that although many spoken-dialogue systems have been successfully developed for some specific subject domain, not too many of them are able to handle dialogue across multiple subject domains efficiently and intelligently. Therefore, they presented a distributed-agent architecture for multi-domain spoken dialogue systems, and cooperative spoken-dialogue agents with high domain extensibility. Based on a similar idea, Wang [2002] has proposed an interface for the development of web-based multimodal dialogue systems.

Rahim, et al [1999] have built the W99 system, which is a spoken -dialogue system used in ASUR'99 registration, checking paper status and limited information access. The dialogue-design strategy is based on naturalness, ease-of-use and robustness. These achieved abilities are demonstrated using sample dialogue sessions between the system and users. The system was evaluated by the ratings of users. The W99' system achieved overall rating of 3.2 out of 5, and concept accuracy of 74.8%. They claimed that the ratings are clear indication that the W99 system was a successful application using spoken-dialogue systems.

Singh, et al [2002] proposed a new dialogue-design policy, the reinforcement learning approach, for automatically optimizing a dialogue policy using the NJFun system. The NJFun system is a database of activities in New Jersey which can be accessed using telephone. It is a real-time spoken dialogue system. They claimed that by using the reinforcement learning approach to construct a training version of the NJFun spoken-dialogue system, there are significant improvements in both the reward measure for which the optimization was performed and in several other objective reward measures even though the test policy was not optimized. However, there are no improvements for a set of subjective measures.

7 Conclusions

There has been much development of dialogue design in recent years. There are many different design approaches and strategies in developing spoken-dialogue systems. Among the different approaches, there isn't a best one, but a most-suitable one for a particular dialogue system. Some are easy to design and implement for a simple dialogue system, but have limitations. Some can handle more complex dialogues, but they are relatively harder to design. McTear [2002] predicts that the studies of different approaches to dialogue management in relation to the requirements of an application will go on. For example, where state-based methods are applicable and in which circumstances more complex approaches are required.

So far, most spoken-dialogue systems are telephone-based. Now, new web-based distributed spoken-dialogue systems are under development. VoiceXML has become the industry standard for this type of web-based application. It provides researchers with a new environment for developing spoken-dialogue technologies.

There are some commercially-available Software Developer Kits for creating spoken-dialogue systems. However, the CSLU Toolkit is a good choice for developing a simple dialogue system for academic and personal use. It is free as well.

8 Annotated Bibliography

- [1] Allen, J. F., Miller, B. W., Ringger, E. K. and Sikorski, T. A Robust System for Natural Spoken Dialogue, Proceedings of the 1996 Annual Meeting of the Association for Computational Linguistics (ACL'96), June 1996. pp. 62-70, 1996.

In this paper, the authors address the problem that even there has been much research on dialogue, only very few robust dialogue systems have been built. The authors' goal was to demonstrate that it was feasible to construct robust spoken natural dialogue systems. This paper describes a spoken dialogue system that specifically addresses the issue of robust interpretation of speech in the presence of recognition errors and presents an evaluation of the system using time-to-completion and the quality of the final solution. Based on the authors' previous dialogue models, the bottom-up rather than top-down approach is used in the design of their TRAINS-95, a travel route

planning system. The authors claim the task-based approach does set a new standard for spoken dialogue systems. It would allow them to address a large number of research issues in the future in a more systematic way, supported by empirical evaluation.

- [2] Churcher, G. E., Atwell, E. S. and Souter, C. Dialogue Management Systems: a Survey and Overview, Report 97.6, School of Computer Studies, University of Leeds, 1997.

This survey provides an overview of the current issues and techniques for the modeling of dialogues using a computer. Instead of single shot question and answer interaction between user and computer, a dialogue system is a more personal affair establishing the exact requirements from a user. In this paper, the authors describe different dialogue management approaches and present different ways to evaluate a dialogue management system. Some special issues relating to a dialogue management system such as the coding of a dialogue structure and recovery strategies are discussed in the paper as well.

- [3] Constantinides, P.C., Hansma, S., Tchou, C. and Rudnicky, A.I. A Schema Based Approach to Dialog Control, Proceedings of 5th International Conference on Spoken Language Processing (ICSLP'98), Sydney, Australia, November 1998.

In this paper, the authors address the problem that frame-based approaches lack active system participation in complex tasks. Therefore the authors use implementations of two dialogue systems, a personal calendar system and an air travel planning system as examples to describe two mechanisms that can provide more active system participation than frame-based approaches in dialogue control. The two mechanisms are a modified stack that allows the system to track multiple topics, and form-specific schema that allow the system to deal with tasks that involve completion of multiple forms. The authors claim that the results from system use indicate that the majority of callers are able to complete meaningful dialogues and the system overall achieved 82% task completion.

- [4] Dahlbäck, N. and Jönsson, A. Knowledge sources in spoken dialogue systems, Proceedings of 6th European Conference on Speech Communication and Technology (Eurospeech'99), Budapest, Hungary, September 1999.

This paper describes the authors' idea of finding a middle ground between the computational linguistics approach of general computational models for all kinds of dialogues and the one-shot designs approach for particular systems used by speech community. The authors present that instead of using conventional task model to handle the cases where the user's initial information request is underspecified, and where therefore the system needs to ask the user about the missing pieces of information, how the architecture of the modularized LIN-LIN dialogue manager can

be augmented to handle these cases. The authors advocate use of a separate knowledge module called an Information Specification Form for managing these cases. The authors conclude that in dialogue system, it is important to divide information into knowledge sources so that they can be used in various purposes. However, there is still more work needs to be done in order to establish a useful taxonomy of dialogue types, making it possible in advance to specify which kinds of knowledge is required to be implemented in a particular system under development. The authors also conclude that their modular approach has the advantage of making it possible to re-use the core of the system in porting it to another application domain, without having to incorporate aspects of dialogue management not required in the new situation.

- [5] Glass, J. R. Challenges for Spoken Dialogue Systems, In Proceedings of the 1999 IEEE ASRU Workshop, 1999.

In this paper, the author gives an overview of the existing spoken dialogue systems and describes some existing spoken dialogue systems such as the JUPITER weather information system developed by MIT, the DARPA spoken language systems developed in the US, the Esprit SUNDIAL system developed in Europe and so on. The author also discussed some key issues in dialogue design such as the studying of the human conversations, modeling of the dialogue phenomenon, implementation strategies and so on. Dialog systems are relied on spoken language technologies such as the speech recognition and language understanding. Despite the success in the existing dialogue systems, the author claims that there is still much work needs to be done for improving interaction with users.

- [6] Goddeau, D., Meng, H., Polifroni, J., Seneff, S. and Busayapongchai, S. A form-based dialogue manager for spoken language applications, Proceedings of 4th International Conference on Spoken Language Processing (ICSLP'96), Pittsburgh, PA, October, 1996, 701-704, 1996.

In this paper, the authors address the problem that in some existing spoken dialogue systems are hard to ported to a different domain and difficult to predict system behavior or modify the algorithm. They describe a form-based dialogue manager which is a dialogue planning algorithm based on the notion of filling in an electronic form, or "E-form". The authors claim this form-based approach is an alternative to the finite state-based approach. The example used in this paper is an E-form-based dialogue planner in an application used to search a database of used car advertisements. This dialogue system intended to enable both user-initiated and system-initiated interactions, to improve robustness both to system errors and changes in user goals, and to allow the construction of systems whose behavior is predictable and verifiable. The authors claim that the E-form structure dialogue system achieved the goal of supporting mixed-initiated interaction and promoting predictable and verifiable dialogue managers. The authors also present their future work as well.

- [7] Heeman P, Johnston M, Denney J and Kaiser E. Beyond structured dialogues: Factoring out grounding, Proceedings of 5th International Conference on Spoken Language Processing, Dec 1998, Sydney, Australia, 863-866. 1998.

In this paper, the authors address the problem that structured dialogue models are currently the only tools for easily building spoken dialogue systems. However, this approach has shortcomings such as the grounding behavior between the user and the system. The authors advocate factoring out the grounding behavior from structured dialogue models by using a general purpose dialogue manager that accounts for this behavior. The authors claim that their alternative approach not only simplifies the specification of dialogue, but also allows more powerful mechanisms of grounding to be employed. It allows much richer dialogue systems to be built while maintaining domain independence and the ease of development that the structured dialogue approach offers.

- [8] Levin, E., Pieraccini, R., Eckert, W., DiFabrizio, G. and Narayanan, S., Spoken Language Dialogue: From Theory to Practice, IEEE Automatic Speech Recognition and Understanding Workshop, Keystone, Colorado, 12-15 December 1999.

This paper discusses an approach for building a spoken language dialogue system. A model of sequential decision process is introduced in detail in this paper. The authors claim that, this model can be used in designing both the overall general architecture of the dialogues system and implementing a mixed initiative dialogue strategy.

- [9] Lin, B., Wang, H. and Lee, L. A Distributed Architecture for Cooperative Spoken Dialogue Agents with Coherent Dialogue State and History, the 1999 International Workshop on Automatic Speech Recognition and Understanding (ASRU99) December 12-15, 1999.

In this paper, the authors address the problem that although many spoken dialogue systems have been successfully developed for some specific subject domain, not too many of them are able to handle dialogue across multiple subject domains efficiently and intelligently. Therefore, this paper presents distributed agent architecture for multi-domain spoken dialogue systems, and cooperative spoken dialogue agents with high domain extensibility. The dialogue system is still under development, the results from initial experiments, the authors claim, are encouraging.

- [10]McTear M. F. Modelling spoken dialogues with state transition diagrams: experiences with the CSLU toolkit, Proceedings of 5th International Conference on Spoken Language Processing, Dec 1998, Sydney, Australia, 1223-1226. 1998.

This paper evaluates the applicability of state transition diagrams for modeling different types of spoken dialogue with the help of CSLU toolkit. The author uses a graphically-based authoring environment (RAD, Rapid Application Developer) in the CSLU toolkit to construct simple dialogue system. In RAD, dialogue objects are

linked together to create a finite-state dialogue model. This paper evaluates the strengths of finite-state dialogue models by examining the types of task for which they are appropriate as well as the technological implications of their use, and determines the class of dialogues that can be modeled using finite-state methods. The author concludes that the choice of a finite-state dialogue model for a particular task is determined by many factors. It is determined not only by the nature of the task but also by dependencies between the information items in the dialogue, the methods used for verification, and the performance of the speech recognition and understanding modules.

- [11]McTear, M. F., Allen, S., Clatworthy, L., Ellison, N., Lavelle, C. and McCaffery, H. Integrating Flexibility into a Structured Dialogue Model: Some Design Considerations, Proc 6th International Conference on Spoken Language Processing, Oct 2000, Beijing, China, Vol. 1, 110-113, 2000.

This paper describes attempts to make existing structured dialogue models more flexible using the Center for Spoken Language Understanding (CSLU) toolkit to develop more flexible dialogue control and its result. The author claims that the structured dialogue models in the existing commercial spoken dialogue toolkit is restricted to single words or phrases input. Although they are easy to design and re-use, as the dialogue becomes more complex, they can cause combinatorial explosion of dialogue states, because these models are not flexible. Five systems are built and tested. The results of evaluation shows that the recognition accuracy for longer unconstrained strings spoken input at initial system prompt is only between 10% to 20%, however in these error cases, the new dialogue control mechanism is able to direct dialogues into more constrained states and improves the accuracy to almost 100%. The author claims that these results show the new dialogue control system is more flexible.

- [12]McTear, M. F. Spoken Dialogue Technology: Enabling the Conversational User Interface, ACM Computing Surveys, Volume 34, Issue 1, pp. 90 – 169, 2002.

This paper is a survey. It describes the main components of the spoken dialogue technology – speech recognition, language understanding, dialogue management, communication with an external source such as a database, language generation, speech synthesis, and shows how these components can be integrated into a spoken dialogue system. This paper presents some well-known dialogue systems in detail, explores different system architectures considers issues of specification, design and evaluation. The author also reviews some dialogue development toolkits such as CSLU in this paper.

- [13]O'Neill, I. M. and McTear, M. F. An Object-oriented Approach to the Design of Dialogue management functionality, Proceedings of EACL '99, Bergen, 8-11 June, pp. 23-29, 1999.

This paper presents the idea of using object-orientation in dialogue design. The

system discussed in this paper is implemented using Prolog++. It combines the object-oriented approach with a self-organizing, mixed-initiative dialogue strategy. The system makes use of the advantage that object-orientation is an effective means of separating the generic from the specialized and letting the specialized inherit the generic. The authors claim that the system is successfully tested and has recently been amended to allow the confirmation strategy to come into play.

- [14] O'Neill, I. M. and McTear, M. F. Object-Oriented Modelling of Spoken Language Dialogue Systems. Natural Language Engineering, Best Practice in Spoken Language Dialogue System Engineering, Special Issue, Volume 6 Part 3 October 2000. Cambridge University Press. ISSN: 1351-3249, 2000.

In this paper, the authors address that in spoken dialogues, certain skillsets and patterns of dialogue evolution are common to many different contexts; regardless of the business domain of the transaction, there are certain set-pieces that are common to all such transactional dialogues. Therefore, their idea is to use the object-oriented approach. This paper shows how established object-oriented modeling techniques can be used in creation of spoken dialogue management system. In the dialogue design, the authors use generic dialogue functionality – turn taking, confirmation strategy. Domain-specific components have access to generic dialogue functionality, which is inheritance. Authors claim that the object-oriented approach and methodology outlined in this paper presents a practical approach to the task of turning the processes observed in natural spoken dialogues into maintainable extensible software systems.

- [15] Rahim, M., Pieraccini, R., Eckert, W., Levin, E., Di Fabrizio, G., Riccardi, G., Lin, C. E. and Kamm, C. W99 – A Spoken Dialogue System for the ASRU'99 Workshop, The 1999 International Workshop on Automatic Speech Recognition and Understanding (ASRU'99), 1999.

This paper describes the W99 system, which is a spoken dialogue system used in ASUR'99 registration, checking paper status and limited information access. The paper describes the architectures of the system and its dialogue design. The dialogue design strategy is based on naturalness, ease-of-use and robustness. These achieved abilities are demonstrated using sample dialogue sessions between the system and users. The system is evaluated by the ratings of users. The W99' system achieved overall rating of 3.2 out of 5, and concept accuracy of 74.8%. The authors claim that the ratings are clear indication that W99 system is a successful application using spoken dialogue systems.

- [16] Rudnicky, A. I., Thayer, E., Constantinides, P., Tchou, C., Shern, R., Lenzo, K., Xu, W. and Oh, A. Creating natural dialogs in the Carnegie Mellon Communicator system, Proceedings of 6th European Conference on Speech Communication and Technology (Eurospeech'99), Budapest, Hungary, September 1999.

This paper describes the dialogue design in the Carnegie Mellon Communicator, which is a travel planning system. The authors present the Mellon Communicator system and use task-based dialogue management in the system. The authors claim to believe that the task-based dialogue management approach they use will scale with the complexity of the underlying domain and the tasks that people expect to be able to perform.

- [17]Rudnicky, A. I., and Xu, W. An Agenda-Based Dialog Management Architecture for Spoken Language Systems. In International Workshop on Automatic Speech Recognition and Understanding (ASRU), 1999.

Based on the author's previous work on task-based dialogue design, in this paper, they proposed an agenda-based dialogue management architecture. The agenda-based approach addressed the problem dialogue management in complex problem-solving tasks by treating the task at hand as on of cooperatively constructing a complex data structure, a product, and uses this structure to guide the task. The authors claim that the agenda mechanism can be adapted easily to less-complex domains that might be implemented as a standard form-based system. However, they did not know how well the technique would succeed for domains of complexity comparable to travel planning but with different task structure.

- [18]Singh, S., Litman, D., Kearns, M. and Walker, M. Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System, Journal of Artificial Intelligence Research (JAIR), 2002.

The authors address the problem that I designing a dialogue management policy, one has to make many nontrivial design choices. However, it is often difficult to assess the ramifications of these choices. The authors' idea is to use a reinforcement learning approach. They present this approach for automatically optimizing a dialogue policy using the NJFun system. The NJFun system is a database of activities in New Jersey which can be accessed using telephone. It is a real-time spoken dialogue system. The reinforcement learning approach is a new dialogue policy design proposed in this paper. Authors claim that by using the reinforcement learning approach to construct a training version of the NJFun spoken dialogue system, there are significant improvements in both the reward measure for which the optimization was performed and in several other objective reward measures even though the test policy was not optimized. However, there are no improvements for a set of subjective measures.

- [19]Skantze, G Exploring Human Error Handling Strategies: Implications for Spoken Dialogue Systems, ISCA (International Speech Communication Association) Tutorial and Research Workshop on Error Handling in Spoken Dialogue Systems, 2003.

Miscommunication is inevitable in the human-computer dialogue. Therefore, this paper describes an investigation of error handling in spoken dialogue system using a modified Wizard of Oz method. The focus is on how humans handle speech recognition errors before the dialogue system is built. The goal is that instead of testing a specific system design, the study is to find suitable error handling strategies for the spoken dialogue system. The author presents the experiment results and suggests that for early detection of errors and deciding reactions can use different knowledge sources such as confidence score, syntactic structure and context. The author also claimed the experiment results suggest using a good domain model and robust parsing techniques to pose relevant questions to users when non-understandings occur in spoken dialogue systems.

[20] Sutton, S., Cole, R.A., de Villiers, J., Schalkwyk, J., Vermeulen, P., Macon, M., Yan, Y., Kaiser, E., Rundle, B., Shobaki, K., Hosom, J.P., Kain, A., Wouters, J., Massaro, M. and Cohen, M., Universal Speech Tools: the CSLU Toolkit, Proceedings of the International Conference on Spoken Language Processing page 3221-3224, Sydney, Australia, November 1998.

This paper presents the CSLU Toolkit developed by Center of Spoken Language Understanding at the Oregon Graduate Institute of Science and Technology. The purpose of developing the toolkit is to incorporate state-of-the-art spoken-language technology into a portable, comprehensive and easy-to-use software environment. The toolkit is for developing spoken dialogue systems. The authors described the components and functionalities of the toolkit. The toolkit is free-of-charge for personal and educational use and it is available for download.

9 Bibliography

- 1) Allen, J. F., Miller, B. W., Ringger, E. K. and Sikorski, T. A Robust System for Natural Spoken Dialogue, Proceedings of the 1996 Annual Meeting of the Association for Computational Linguistics (ACL'96), June 1996. pp. 62-70. 1996.
- 2) Bernsen, N. O., Dybkjær, L. and Dybkjær H. User Errors in Spoken Human-Machine Dialogue, ECAI Workshop on Dialogue Processing in Spoken Language Systems: 14-28, 1996.
- 3) Carlson, R. and Hunnicutt, S. The natural language component – STINA, STL-QPSR 1/1995, pp. 28-49, 1995.
- 4) Carlson, R. The Dialog Component in the Waxholm System. Proceedings of the Twente Workshop on Language Technology (TWLT11) Dialogue Management in

- Natural Language Systems, University of Twente, the Netherlands, 209-218, 1996.
- 5) Churcher, G. E., Atwell, E. S. and Souter, C. Dialogue Management Systems: a Survey and Overview, Report 97.6, School of Computer Studies, University of Leeds, 1997.
 - 6) Constantinides, P. C., Hansma, S., Tchou, C. and Rudnicky, A. I. A Schema Based Approach to Dialog Control, Proceedings of 5th International Conference on Spoken Language Processing (ICSLP'98), Sydney, Australia, November 1998.
 - 7) Dahlbäck, N. and Jönsson, A. Knowledge sources in spoken dialogue systems, Proceedings of 6th European Conference on Speech Communication and Technology (Eurospeech'99), Budapest, Hungary, September 1999.
 - 8) Dybkjær, L. Bernsen, N. O. and Dybkjær, H. A Methodology for Diagnostic Evaluation of Spoken Human-Machine Dialogue. International Journal of Human Computer Studies/Knowledge Acquisition 1997.
 - 9) Dybkjaer, L. and Bernsen, N. O. Usability Issues in Spoken Language Dialogue Systems, Natural Language Engineering 6 (Parts 3/4): 243-272, 2000.
 - 10) Fraser, N., and Gilbert, G. N. Simulating Speech Systems. Computer Speech and Language 5, 81-99, 1991.
 - 11) Glass, J. R. Challenges for Spoken Dialogue Systems, In Proceedings of the 1999 IEEE ASRU Workshop, 1999.
 - 12) Goddeau, D., Meng, H., Polifroni, J., Seneff, S. and Busayapongchai, S. A form-based dialogue manager for spoken language applications, Proceedings of 4th International Conference on Spoken Language Processing (ICSLP'96), Pittsburgh, PA, October, 1996, 701-704. 1996.
 - 13) Goddeau, D. and Pineau, J. Fast Reinforcement Learning of Dialog Strategies, IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP). Istanbul, Turkey. June 2000.
 - 14) Hansen, B. Novick, D. and Sutton, S. Systematic design of spoken-dialogue interfaces, Conference on Human Factors in Computing Systems (CHI'96), 157-164, 1996.
 - 15) Heeman P., Johnston M., Denney J. and Kaiser E. Beyond structured dialogues: Factoring out grounding, Proceedings of 5th International Conference on Spoken Language Processing, Dec 1998, Sydney, Australia, 863-866. 1998.

- 16) Heisterkamp, P. and McGlashan, S. Units of dialogue management: an example, Proceedings of 4th International Conference on Spoken Language Processing (ICSLP'96), Pittsburgh, PA, October, 1996.
- 17) Knight, S., Gorrell, G., Rayner, M., Milward, D., Koeling, R. and Lwein, I., Comparing Grammar-Based and Robust Approaches to Speech Understanding, A Case Study Proceedings of Eurospeech, 2001.
- 18) Levin, E., Pieraccini, R., Eckert, W., DiFabrizio, G. and Narayanan, S., Spoken Language Dialogue: From Theory to Practice, IEEE Automatic Speech Recognition and Understanding Workshop, Keystone, Colorado, 12-15 December 1999.
- 19) Lin, B., Wang, H. and Lee, L. A Distributed Architecture for Cooperative Spoken Dialogue Agents with Coherent Dialogue State and History, The 1999 International Workshop on Automatic Speech Recognition and Understanding (ASRU99) December 12-15, 1999.
- 20) Lindberg, B., Andersen, B., Baekgaard, A., Broendsted, T., Dalsgaard P. and Kristiansen J. An Integrated Dialogue Design and Continuous Speech Recognition System Environment, International Conference on Spoken Language Processing (ICSLP), ISBN 0888648065, pp. 1653-1656, 1992.
- 21) McTear M. F. Modelling spoken dialogues with state transition diagrams: experiences with the CSLU toolkit, Proceedings of 5th International Conference on Spoken Language Processing, Dec 1998, Sydney, Australia, 1223-1226, 1998.
- 22) McTear, M. F., Allen, S., Clatworthy, L., Ellison, N., Lavelle, C. and McCaffery, H. Integrating Flexibility into a Structured Dialogue Model: Some Design Considerations, Proc 6th International Conference on Spoken Language Processing, Oct 2000, Beijing, China, Vol. 1, 110-113, 2000.
- 23) McTear, M. F. Spoken Dialogue Technology: Enabling the Conversational User Interface. ACM Computing Surveys, Volume 34, Issue 1, pp. 90 – 169, 2002.
- 24) Minker, W., Haiber, U., Heisterkamp, P. and Scheible, S. Intelligent Dialog Overcomes Speech Technology Limitations: The SENECA Example, Proceedings of the 8th international conference on Intelligent user interfaces pp. 267-269, 2003.
- 25) Muller, C. and Runger, F. Dialogue Design Principles – Key for Usability of Voice Processing, Proceedings of the 3rd European Conference on Speech, Communication and Technology (EUROSPEECH '93), pp. 934-946, 1993.

- 26) Najjar, L. J., Ockerman, J. J., and Thompson, J. C. User interface design guidelines for speech recognition applications, IEEE VRAIS 98 workshop - Interfaces for wearable computers, 1998.
- 27) Nielsen, P. B., and Baekaard, A. Experience with a Dialogue Description Formalism for Realistic Applications, Proceedings of the International Conference on Spoken Language Processing (ICSLP '92), pp. 719-722, 1992.
- 28) O'Neill, I. M. and McTear, M. F., An Object-oriented Approach to the Design of Dialogue management functionality, Proceedings of EACL '99, Bergen, 8-11 June, pp. 23-29. 1999.
- 29) O'Neill, I. M. and McTear, M. F. Object-Oriented Modelling of Spoken Language Dialogue Systems. Natural Language Engineering, Best Practice in Spoken Language Dialogue System Engineering, Special Issue, Volume 6 Part 3 October 2000. Cambridge University Press. ISSN: 1351-3249, 2000.
- 30) O'Neill, I. M., and McTear, M. F., A Pragmatic Confirmation Mechanism for an Object-Based Spoken Dialogue Manager, Proceedings of ICSLP-2002, Vol. 3, 2045-2048, 2002.
- 31) O'Neill, I. M., Hanna, P., Liu, X. and McTear, M. The Queen's Communicator: An Objectoriented Dialogue Manager. In Proceedings of 8th International Conference on Speech Communication and Technology (Eurospeech2003) 593-596, 2003.
- 32) Polifroni, J., Seneff, S., Glass, J. and Hazen, T. J. Evaluation Methodology for a Telephone-based Conversational System, Proceedings of the First International Conference on language Resources and Evaluation, pp. 42-50, Granada, Spain, May 1998.
- 33) Rahim, M., Pieraccini, R., Eckert, W., Levin, E., Di Fabbriizio, G., Riccardi, G., Lin, C. E. and Kamm, C. W99 – A Spoken Dialogue System for the ASRU'99 Workshop, The 1999 International Workshop on Automatic Speech Recognition and Understanding (ASRU'99), 1999.
- 34) Rudnický, A. I., Thayer, E., Constantinides, P., Tchou, C., Shern, R., Lenzo, K., Xu, W. and Oh, A. Creating Natural Dialogs in the Carnegie Mellon Communicator System, Proceedings of 6th European Conference on Speech Communication and Technology (Eurospeech'99), Budapest, Hungary, September 1999.
- 35) Rudnický, A. I., and Xu, W. An Agenda-Based Dialog Management Architecture

for Spoken Language Systems. In International Workshop on Automatic Speech Recognition and Understanding (ASRU), 1999.

- 36) Singh, S., Litman, D., Kearns, M. and Walker, M. Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System, Journal of Artificial Intelligence Research (JAIR), 2002.
- 37) Skantze, G. Exploring Human Error Handling Strategies: Implications for Spoken Dialogue Systems, ISCA (International Speech Communication Association) Tutorial and Research Workshop on Error Handling in Spoken Dialogue Systems. 2003.
- 38) Sutton, S., Cole, R.A., de Villiers, J., Schalkwyk, J., Vermeulen, P., Macon, M., Yan, Y., Kaiser, E., Rundle, B., Shobaki, K., Hosom, J.P., Kain, A., Wouters, J., Massaro, M. and Cohen, M., Universal Speech Tools: the CSLU Toolkit, Proceedings of the International Conference on Spoken Language Processing page 3221-3224, Sydney, Australia, November 1998.
- 39) Yasuda, N. Dohsaka, K. Aikawa, K. Spoken dialogue control based on a turn-minimization criterion depending on the speech recognition accuracy, SIGdial2001, 210-213, 2001.
- 40) Wang K., SALT: a Spoken Language Interface for Web-Based Multimodal Dialog Systems, in Proc. ICSLP-2002, Denver, CO, 2002.
- 41) Zadrozny, W., Wolf, C., Kambhatla, N., Ye, Y. Conversation Machines for Transaction Processing, AAAI/IAAI 1998: 1160-1166, 1998.

Appendix II

SpeechWeb Browser Implementation Manual

Table of Contents

I. Installation Instructions

1.1 System Requirement

1.2 Install IBM WebSphere Voice Server SDK

1.3 Setup SDK

1.4 Install the SpeechWeb Browser

II. User Manual

2.1 Start the SpeechWeb Browser

2.2 Use the SpeechWeb Browser

2.2.1 Connect to the server sihlo

2.2.2 Change the Default

2.3 Speak to the SpeechWeb Browser

2.4 Voice Input List

2.4.1 Judy Input

2.4.2 Monty Input

2.4.3 Solarman Input

2.5 Trouble Shooting

III. Browser Documentation

3.1 About SpeechWeb Browser

3.2 SpeechWeb Architecture

3.2.1 Local Machine

3.2.2 Remote Sihlo

3.3 SpeechWeb Browser Architecture

3.3.1 Initialization State

3.3.2 Running State

3.3.3 Ending State

3.4 Source Code

3.4.1 *default.sihlo*

3.4.2 *start.vxml*

3.4.3 *t-1.vxml*

3.4.4 *HTTPpost.java*

3.4.5 *Post.java*

3.4.6 *Rewrite.java*

3.4.7 *Start.java*

IV. Application Developer Manual

4.1 About This Manual

4.2 Remote Sihlo

4.2.1 Sihlo Header File

4.2.2 Grammar File

4.2.3 Interpreter Application

V. Protocols and Restrictions

5.1 Protocols

5.2 Restrictions

5.2.1 Local Machine

5.2.2 Remote Sihlo

VI. Program listing

6.1 *default.sihlo*

6.2 *start.vxml*

6.3 *t-1.vxml*

6.4 *HTTPpost.java*

6.5 *Post.java*

6.6 *Rewrite.java*

6.7 *Start.java*

Installation Instructions ¶

1.1 System Requirements

A computer equipped with the following minimum hardware and software is required:

- Intel Pentium^(R) 366 MHz processor or equivalent
- 128 MB RAM
- 290 MB disk space for each language to be installed, which includes:
 - 30 MB for installing the Sun^(R) Java Runtime Environment (Sun JRE) 1.3.1
 - 80 MB in the Windows system directory
 - 130 MB disk space in the destination installation directory for installing each language selected
- 28 MB in the installation destination directory for caching, logging, and tracing
- A Microsoft Windows 2000 or Windows XP compatible, 16-bit, full-duplex sound card (with a microphone input jack) with good recording quality
- Microsoft Windows 2000 or Windows XP operating system

After you have installed the SDK product, you can remove the downloaded packages and the extracted installation program files.

1.2 Install IBM WebSphere Voice Server SDK

Download IBM WebSphere Voice Server SDK US English Version: *vssdkinstall_launcher.exe* and *vssdkinstall_en.exe* from the "Downloads" page in the SpeechWeb homepage at <http://davinci.newcs.uwindsor.ca/~speechweb> unpack both of them to a same directory.

¶ Based on the Installation Manual of IBM WebSphere Voice Server SDK 1.3.1

If you already have installed a version of JRE newer than 1.3.1, you have to **uninstall it first**. Then you can install JRE 1.3.1 from the file `\install\jre131\j2re-1_3_1-win-i.exe` (this path is relative to the directory where you unpacked the installation package.)

Then execute the file `setup.exe` in the unpacked directory to start the installation wizard for the SDK.

NOTE:

1. You must install the SDK on a local hard drive. You cannot install to a local diskette or to a ZIP drive, CD-ROM, or remote drive.
2. Should a message be displayed during the installation process and the message is not completely visible, enlarge the dialog window by clicking and dragging one of the four corners until the entire message is visible.
3. The name of the folder where you install the SDK cannot contain any accented characters (for example, é, û, and Ç).
4. Follow the instructions in the installation wizard to install the SDK.
5. The default destination directory is: **C:\Program Files\VoiceServerSDK**. It is highly recommended that the SDK is installed in the default directory. If you choose to specify a different directory, the following restrictions apply:

The destination path should not include non-alphanumeric characters (for example, ~!@#%&*), accented characters (for example, é, û, and Ç), or non-ASCII characters. You can only use ASCII characters, alphanumeric characters, and space.

The destination path should not be more than five directory levels. For example:

<windrive>:\Program Files\two\three\four\VoiceServerSDK

When the installation wizard has finished installing the SDK, restart your computer.

1.3 Setup SDK

Start the Audio Setup program to configure your microphone and speakers by selecting Start -> Programs -> IBM WebSphere Voice Server SDK, then select the Audio Setup program. Follow the instructions.

1.4 Install SpeechWeb Browser

Download SpeechWeb Browser *swb_1.0.zip*, or self-extracting file *swb_1.0.exe*, from the "Downloads" page in the SpeechWeb homepage at:

<http://davinci.newcs.uwindsor.ca/~speechweb>

Unpack either one of them to any directory. The default directory is **C:\SpeechWeb**.

If your IBM Voice Server SDK is NOT installed in the default directory: **C:\Program Files\VoiceServerSDK**, you have to modify the file *run.bat*.

1. Open *run.bat* by using a text editor.
2. Change the content of the batch file:

"c:\progra~1\voices~1\bin\vsaudio_en_US.bat start.vxml"

to "<windrive>:\<your SDK directory>\bin\vsaudio_en_US.bat start.vxml"

Execute *run.bat* to start the SpeechWeb Browser.

SpeechWeb Browser uses Java objects. Depends on the speed of your PC, it maybe slow when it first starts.

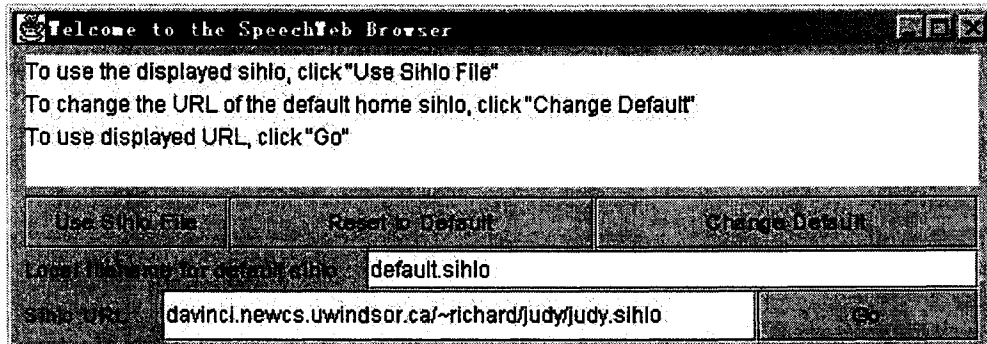
User Manual

2.1 Start the SpeechWeb Browser

Note: Please Setup your IBM Voice Server SDK before running the SpeechWeb Browser (see section 1.4).

Go to your SpeechWeb Browser folder where you unpack the zip file *swb_1.0.zip* or self-extracting file *swb_1.0.exe*. Double click the file "*run.bat*" to start the SpeechWeb Browser and wait for the browser to be fully loaded. Depends on the speed of your computer, it may take a few seconds to load the browser.

Please make sure your computer speaker is ON. After the Browser is loaded you will see a DOS prompt window and an interface window. The computer would have a voice output "Welcome to the SpeechWeb Browser". The interface window looks like the following:



2.2 Use the SpeechWeb Browser

2.2.1 Connect to the server sihlo

1. When the browser starts, it loads a default local sihlo information file: "*default.sihlo*". Its content is displayed in the "Shilo URL" field. To use the default you can either click "Go" or "Use Sihlo File". Then the browser will connect to the server sihlo.

2. If you have your own local sihlo information file, you can open it by typing the filename in the text field “Local filename for default sihlo” and press the Enter key.

Note: the local sihlo information file has to be in the same directory of the SpeechWeb Browser installation folder. You have to type the full filename including the extension name in order to use it.

After the Enter key is pressed, the file content will be displayed in the “**Sihlo URL**” text field. If the file is valid, you can either click “Go” or “Use Sihlo File” to connect to the server. If the file is not valid, it will be indicated in the text field and text area.

3. You can simply type the server sihlo address in the “**Sihlo URL**” text field and click “Go” to connect to the server. You do **NOT** need to type “http://” in front of the sihlo address.

You can click “Reset to Default” any time to open the default local sihlo information file: “*default.sihlo*”.

2.2.2 Change the Default

It is not recommended to change the default sihlo information file, unless it is necessary.

If the button “Change Default” is clicked, the browser will open the file “*default.sihlo*” in the local directory and attempt to rewrite it. If the “*default.sihlo*” file does not exist, the browser will attempt to write a new one.

There are four steps in changing the default:

1. Click “Change Default”, an input window will open.

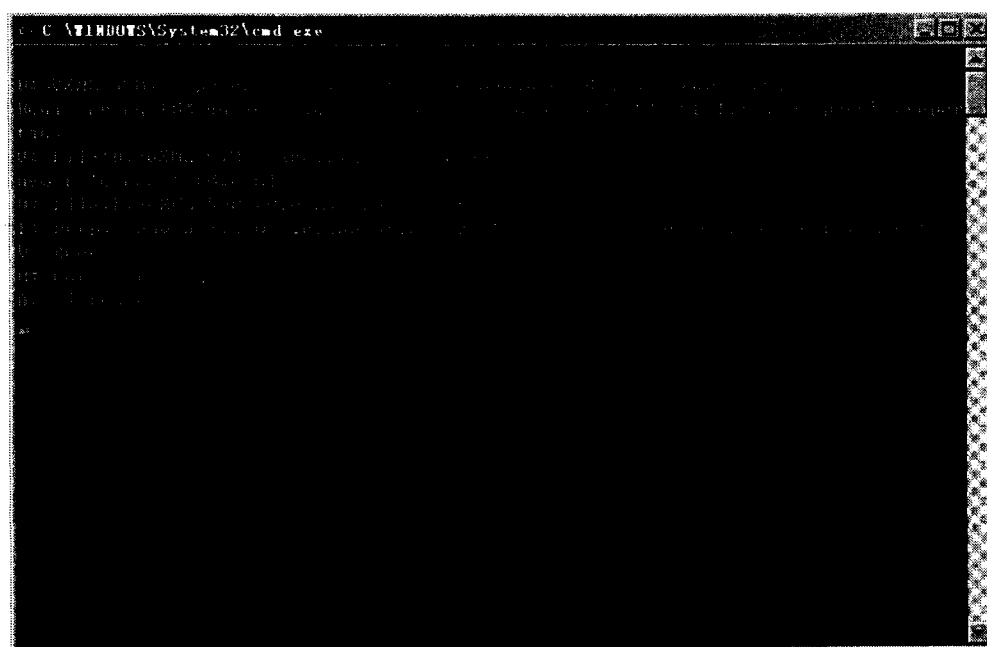
2. The original default URL is displayed. You can type the new URL in the text field and click OK to continue. You can click Cancel to quit the process.
3. When a new URL is entered, a new window will popup and ask you to confirm the new address. Choose “Yes” to confirm the change. Choose “No” to correct the change. Choose “Cancel” to quit the process.
4. When the change is confirmed, an information window will display the changed address. Just click OK to continue.

2.3 Speak to the SpeechWeb Browser

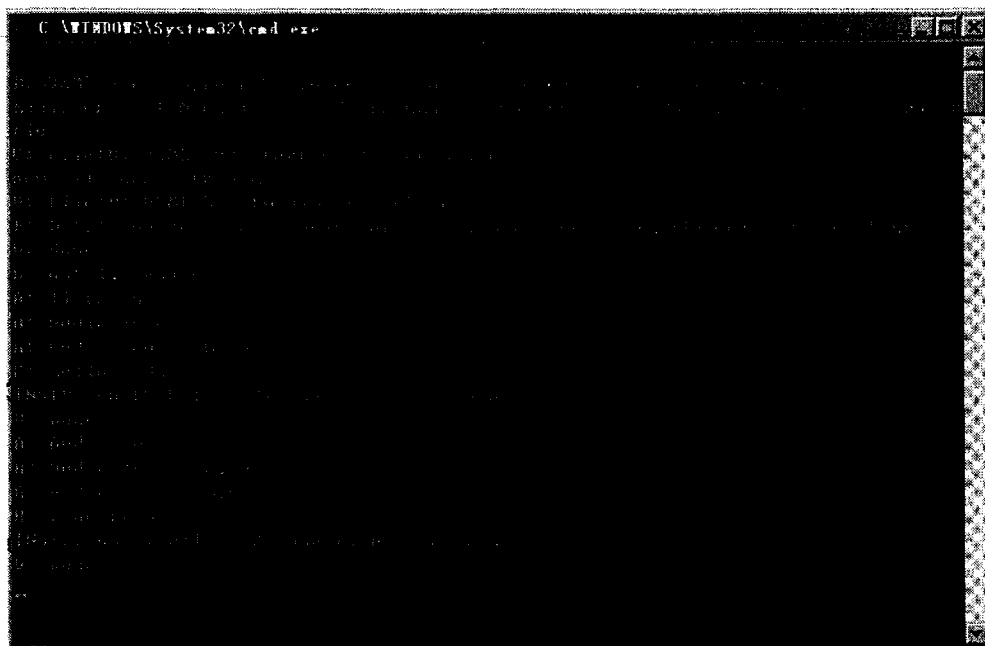
After entering the correct server sihlo information, you can click “Go” or “Use Sihlo File” button to start voice browsing. The interface window disappears.

In the following example, we will use server **sihlo Judy** to demonstrate how to speak to the SpeechWeb Browser.

If the browser successfully connects to the server, the browser will initialize speech by saying “You are talking to (**Server Name**) now”, in this case the browser will say “You are talking to Judy now”. The DOS prompt looks like the following:



Notice the last line in the DOS prompt, it say “listening”. That means the browser is listening to the input if there is any. Now you can speak to the browser.



Circled in red are the sentences or phrases that are recognized by the browser. Circled in blue are the process of sending the user voice input to the server `sihlo` and getting the answer back. The browser speaks out the answers. However, the answers are not displayed in the DOS prompt.

Notice the **“Audio level”** displayed in the DOS prompt. That is the indication of the volume of your voice input. The ideal audio level is between 0.3 and 0.7. If it says “Too loud” or “Too quiet”, it will affect the recognition accuracy of your speech. You can change the position of your microphone and the distance between you and the microphone to get the ideal audio level. If that still does not work, you should setup your microphone again (See section 1.3).

There are three cases where the browser does not accept the voice input:

1. When the browser does not recognize a voice input, it will say "Sorry, I didn't understand" and display "not understood" in the DOS prompt.
2. Sometimes, even if the browser displays your voice input sentence in the DOS

prompt, but it still says “Sorry, I didn't understand.” That means the browser recognizes part of your input, but it is not confident enough to confirm it.

3. In some cases, the browser displays “still speaking” and has no respond. You can just simply repeat your sentence and continue.

For the list of questions and sentences you can say for a voice input, please refer to the next section.

If your input is hardly recognized, you may refer to section 2.5 for trouble shooting.

2.4 Voice Input List

This section lists the possible phrases and sentences user can say as a voice input.

2.4.1 Basic Command

Following are basic commands can be applied to all sihlos”

- “quiet” to ask the browser to stop talking
- “help” to ask for specific instruction of using the sihlo, if available
- “go back” to talk to the previous sihlo, if available
- “can I speak to (Sihlo Name)” to talk to a new sihlo
- “goodbye” to quit the programme

2.4.2 Judy Input

List of voice inputs for Judy sihlo:

hello

hello there

hello judy

goodbye

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

know any poems

go back

tell me a joke

know any jokes

who is monty

can I talk to monty

can I talk to judy

who is solar man

can I talk to solar man

2.4.3 Monty Input

List of voice inputs for Monty sihlo:

hello

go back

hello there

hello monty
goodbye
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 monty
can I talk to solar man

2.4.4 Solarman Input

Solarman sihlo is more complex than other two sihlos.

You can give the similar voice inputs as in Judy and Monty. You can also ask questions about the planets, the moons, and the people who discovered them in the solar system. For example:

“How many moons orbit Mars”

“Which moons orbit Jupiter”

“Who discovered Titania”

2.5 Trouble Shooting

There are several problems you may run into when using the SpeechWeb Browser.

The following is the list of possible problems and their solutions.

1. The browser does not run.

a) If you did not install the IBM voice SDK in the default folder, you need to change the *run.bat* file in the SpeechWeb Browser folder. Change the content to:

"<windrive>:\<your SDK directory>\bin\vsaudio_en_US.bat start.vxml"

b) Make sure you have installed the Java Runtime Environment 1.3.1 properly and you have not installed a later version of JRE in your computer after installing 1.3.1

c) If you already have a JRE with a version later than 1.3.1, you have to uninstall it before you can successfully install the JRE 1.3.1

2. The input voice is hardly recognized.

a) This version of SpeechWeb Browser is designed for recognizing standard US English. The browser may have a hard time in recognizing your speech if you have a heavy accent.

b) Every time you change the physical operating environment, for example, move the computer to a different place, it is recommended that you should run the SDK voice setup again (see section 1.3).

c) When you are dictating, please make sure you say every word clearly.

d) If all of about adjustment still does not change the recognition accuracy, you

may need to consider changing your microphone to one with a better quality.

3. Cannot connect to a server sihlo.

- a) Please make sure your Internet connection is working.
- b) Please make sure your computer does not have a firewall that is not blocking the connection between the SpeechWeb Browser and the Internet.
- c) Please make sure your server sihlo address is correct

Browser Documentation

3.1 About SpeechWeb Browser

SpeechWeb is hyperlinked speech-accessible knowledge sources that are distributed over the Internet.

Dr. Richard Frost and Mr. Sanjay Chitte have already built a prototype SpeechWeb Browser in Java in 1999. This browser allows the user to access remote interpreters, databases and query processors. It uses proprietary IBM API's and therefore cannot be distributed.

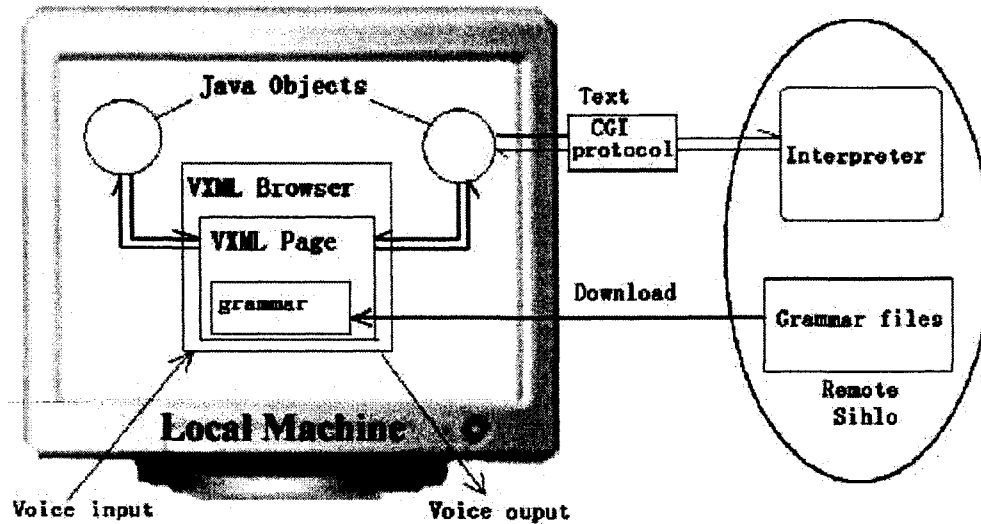
This version of the SpeechWeb Browser is based on the old version. We re-engineered the browser in VXML (Voice Extensible Markup Language). The SpeechWeb Browser uses IBM Websphere Voice Server SDK 3.1 as a voice recognition application and the VXML browser. The SDK was downloaded from IBM.com for free.

3.2 SpeechWeb Architecture

There are some existing speech recognition and processing architectures serve the same purpose of accessing web sources using speech. For example, speech interfaces to the conventional web, RRRP (Remote Recognition & Remote Processing) architecture like telephone access to remote VXML applications, LRLP (Local Recognition & Local Processing) architecture like downloadable hyperlinked VXML pages.

However, they all have disadvantages compare to SpeechWeb architecture. Because SpeechWeb uses LRRP (Local Recognition & Remote Processing) architecture, it has better recognition accuracy and faster speed. [Frost, R. A., N. Abdullah, K. Bhatia, S. Chitte, F Hanna, M. Roy, Y. Shi and L. Su, "LRRP SpeechWebs", 2004]

The diagram below is an example of SpeechWeb Browser connects to a remote sihlo and accesses remote resources.



There are two parts in the SpeechWeb System, the local machine and the remote sihlo.

3.2.1 Local Machine

Local machine can be a PC or a mobile phone with speech-to-text and text-to-speech capability.

Local machine contains a VXML Browser. The VXML pages run in the VXML Browser. The VXML Browser has voice recognition capability. When the VXML Browser runs, it needs grammar for voice recognition. The grammar file is downloaded from a remote sihlo where the web resources are located. Then the VXML Browser takes voice input and translates the input into text. It sends the text through couple Java objects using CGI Protocol to the remote sihlo. The sihlo processes the text input and sends the answer back to the Java objects. The answer is also in the text format. The answer is sent back from the remote sihlo to the Java objects, to the VXML Browser and the VXML Browser speaks the answer out using text-to-speech capability.

3.2.2 Remote Sihlo

Remote sihlo can be a regular web server with CGI Protocol support. It contains an application for processing questions, a header file and at least one grammar file.

Please see Application Developer Manual in section 4 for detailed documentation on how to develop a remote sihlo for the SpeechWeb

3.3 SpeechWeb Browser Architecture

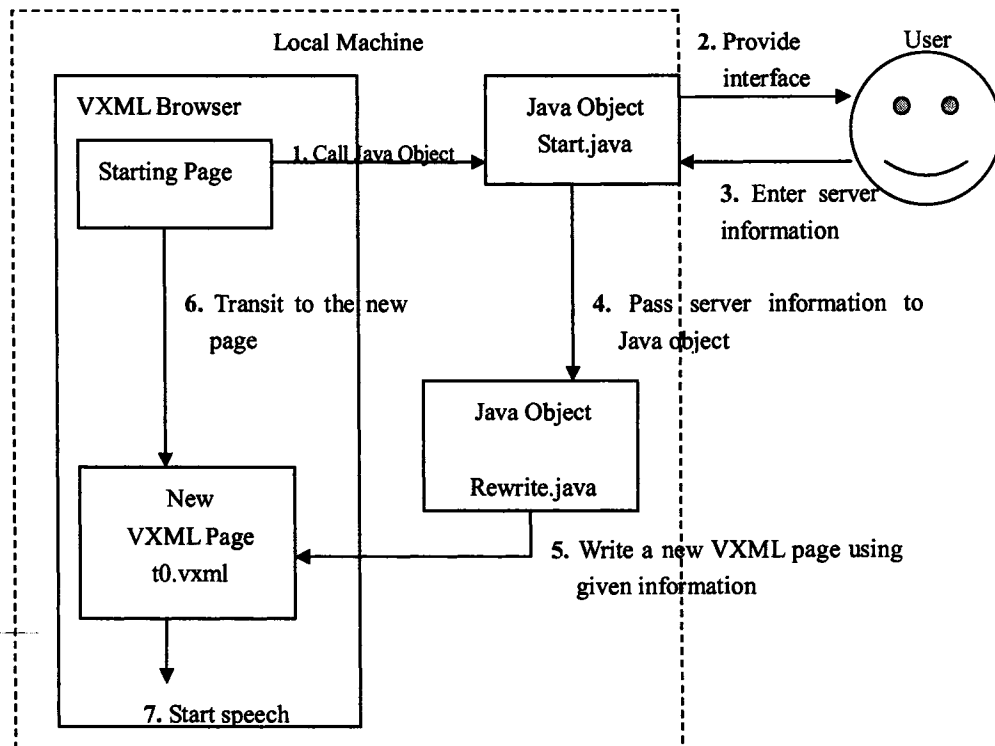
There are three states in running the SpeechWeb Browser. They are initialization, running and ending states.

In the initialization state, the browser provides an interface for the user. The interface displays the default sihlo information. User can also change the default. After correct sihlo information is entered, user can start talking to the browser.

In the running state, the browser uses only voice to communicate with users. It takes user voice input and gives voice output.

In the ending state, the browser cleans the memory and dynamically created files, then it shuts down.

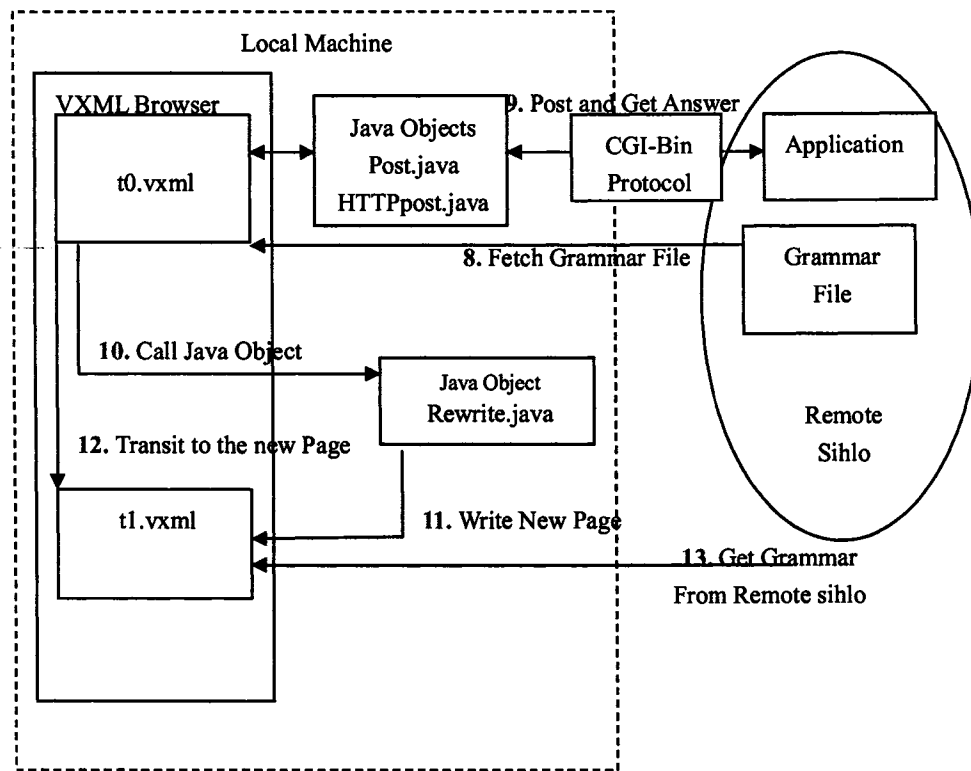
3.3.1 Initialization State



There are 7 steps in the initialization state.

1. After the browser starts, it runs a starting page *start.vxml* to initiate an interface by calling a Java object *Start.java*.
2. *Start.java* provides an interface to user shown in section 2.1
3. User enters correct sihlo information and click “Go” or “Use Sihlo File” button on the interface.
4. As soon as “Go” or “Use Sihlo File” button is clicked, the sihlo information is gathered and sent to another Java object *Rewrite.java*
5. Based on a default VXML page, *Rewrite.java* uses given sihlo information and writes a new VXML page called *t0.vxml*.
6. After the new page is created, the starting VXML page transits to the new page.
7. Browser initialize speech and user can start talking to the browser.

3.3.2 Running State



There are following steps in the running state:

8. After loading the first page *t0.vxml*, the VXML browser downloads a grammar file from the remote sihlo and initiate speech.
9. Then the user can talk to the browser, ask questions and get answers back through the Java class objects “*Post*” and “*HTTPpost*”. The questions are processed and answers are produced in the sihlo application. The communication protocol used between the Java objects and the sihlo application is CGI Protocol.
10. When the user wants to talk to a new sihlo, he can ask for it. The browser then extracts the new sihlo URL, calls another Java class object called “*Rewrite*” and passes the URL address and other necessary information as variables to the Java object.
11. Based on the variables passed from the VXML browser, the Java class object “*Rewrite*” creates a new VXML page with the new sihlo information. Importantly, it saves the new page with a predefined name, which is different from the first page. In this case, the new page is called “*t1.vxml*”.

12. After the new page is written, the browser transits to the new page.
13. Then the VXML browser repeats the steps 8 and 9. Therefore, user can start talking to the new VXML page with different application and grammar.
14. If different remote sihlos are needed, the browser repeats the steps 10, 11 and 12 again and again. The new page will always be named with an incremented page number so that it is different from all other existing pages.
15. In some case, if user says “Go back”, the browser will transit to the previous VXML page and repeat step 8 and 9. Therefore, user can talk to the previous sihlo if available.

3.3.3 Ending State

If user wants to quit the programme, he just needs to say “Goodbye” to the browser. Browser deletes all the VXML pages created in the hard drive and cached files in the memory then shuts down.

3.4 Source Code

The source code consists of seven parts, they are:

default.sihlo

start.vxml

t-1.vxml

HTTPpost.java

Post.java

Rewrite.java

Start.java

For the full source code, please see section 6.

3.4.1 *default.sihlo*

This text file contains only ONE line. It is the full URL address of the remote sihlo.

For example:

davinci.newcs.uwindsor.ca/~richard/judy/judy.sihlo

Make sure the address does not have space symbol anywhere in the line and the end of the address does not have a “\n” symbol.

3.4.2 *start.vxml*

When the VXML browser starts, it runs this file. This file calls the method “test” in Java object “Start”. The Java object creates a graphic user interface.

The browser transits to *t0.vxml* page, after user confirms the remote sihlo address.

3.4.3 *t-1.vxml*

This file is the heart of the SpeechWeb Browser. However, *t-1.vxml* is only a template file. It is never used by the VXML browser.

The line “<!-- break -->” in the file is used as a note for the Java object *Rewrite.java* to stop reading this file and do necessary changes to the following lines after the “break”. Then the Java object writes a new file with all the necessary changes. The new file runs in the VXML browser and is connected to Java objects *Post.java* and *Rewrite.java*.

User voice input is recognized and translated into texts. The recognized sentences are passed to the Java object *Post.java* alone with other necessary information such as the remote sihlo address. The answer is returned from the Java object and spoken out by the VXML browser.

Java object *Rewrite.java* is used to write a new VXML file which is different from the current one with new sihlo information.

ECMA Scripts are embedded in the VXML file. They are used for string processing.

3.4.4 *HTTPpost.java*

The class is designed to perform a CGI based post method primarily used to send queries and receive responses during the client-server communication.

3.4.5 *Post.java*

This class is designed to receive sihlo information and user query through those "set" methods. Then it calls Java class object *HTTPpost.java* to get answers back using sihlo information..

3.4.6 *Rewrite.java*

This class uses "set" methods to define all the global variables.

If the calling VXML page is the default "t-1.vxml", the "writeFile" method will write a new VXML file named "t0.vxml". Then if the calling page is "t0", the next page will be "t1", and the next one after "t1" will be "t2". In another word, if the current page is "t(n)", the new one will be "t(n+1)".

The method "goodbye" deletes all the newly created VXML pages in the stack, maximum 10 pages in total, from "t(n).vxml" to "t(n-9).vxml". If there are less than 10 pages, it deletes whatever is in the stack. Also, the programme will delete all the files that are created by this class, in case the user terminate programme illegally.

The method "goBack" tests if the VXML page can go back to a previous page. In another word, it tests if the previous page exists. If can go back, returns 0. If cannot go back, returns 1.

3.4.7 *Start.java*

This class uses JFrame to create a user interface. It reads the local sihlo address file *default.sihlo* which contains the remote sihlo address. User can input remote sihlo address in JTextFields manually as well.

User can change the default sihlo address.

After user enters address of the sihlo, the programme will have a simple check and see if the address is valid. Space is not allowed in any part of the input.

Application Developer Manual

4.1 About This Manual

This manual provides necessary information of developing a remote sihlo for the SpeechWeb.

In section 3.2, we mentioned that a remote sihlo has a header file, a grammar file and an interpreter. In the following sections we will give the details of how to develop a remote sihlo.

4.2 Remote Sihlo

A remote sihlo for the SpeechWeb can be just a regular web server with CGI support.

A remote sihlo contains a sihlo header file, a grammar file and an interpreter.

4.2.1 Sihlo Header File

The header file is similar to the sihlo address file in the local client machine (see section 3.4.1). They all have a file extension “*.sihlo*”. They are all text files.

However, the content of the files in local machine and remote sihlo are different. Here is an example of a remote sihlo header file:

```
GREETING=Hi, I am Judy. I know about poems;  
SERVER_NAME=davinci.newcs.uwindsor.ca;  
SIHLO_INTERPRETER=/~su5/judy/judy.cgi;  
SIHLO_GRAMMAR=/~su5/judy/h.cgi?judy.gram;
```

As you can see, the sihlo header file contains 4 lines and 4 lines only. Every line has to end with a semicolon. “GREETING=”, “SERVER_NAME=”, “SIHLO_INTERPRETER=” and “SIHLO_GRAMMAR=” have to be written in

exactly the same way and they are case sensitive. The order of the lines cannot be changed.

GREETING is used as a sihlo introduction. It will be spoken out by the SpeechWeb Browser to the user when user first connects to the sihlo. It can be just a simple greeting. It can be a brief description of the sihlo functions. Remember, all the sentences in GREETING must be in one line. The GREETING can be empty, like the following:

GREETING=;

SERVER_NAME cannot be empty. It is the URL address of the server. Like in the example, the server name is the full address of the Davinci server in University of Windsor. It can be a simple IP address as well.

SIHLO_INTERPRETER contains the full path of the interpreter application executable file in the server. It does not contain the server name.

SIHLO_GRAMMAR contains the full path of the grammar file. Notice that in the example, the path contains “h.cgi?”. However, in the remote sihlo, the actual filename for the grammar is “*vxmlyjudy.gram*”. In this version of SpeechWeb, we have to change the “vxmly” to “h.cgi?” in the sihlo header file. This issue is addressed more in section 5.

4.2.2 Grammar File

When SpeechWeb Browser connects to the remote sihlo, it will download the grammar file first. The grammar file provides an environment for speech recognition. It contains all the possible sentences users can say to the browser. It's very important that the grammar covers all the syntax and semantics for the speech recognition.

It seems that either the grammar is very big or the recognition accuracy is low.

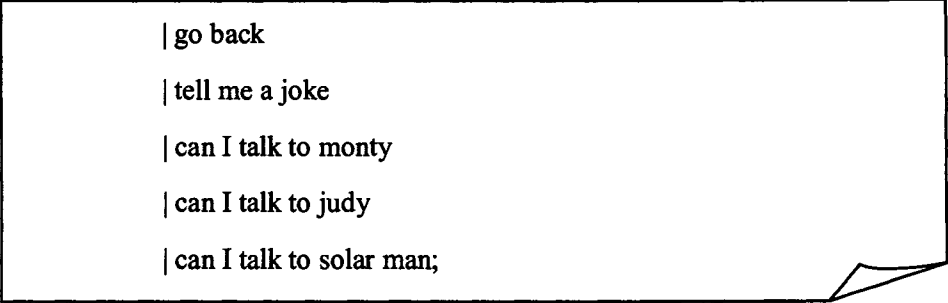
However, because the remote sihlo contains specific information for the user, for example, weather information, we can predict what kind of question the user will ask. Therefore, the size of the grammar is greatly reduced and the recognition accuracy is increased.

The grammar file is implemented using Java Speech Grammar Format (JSGF).

There is a simple example of JSGF grammar file.

vxmljudy.gram

```
grammar vxmljudy;  
  
public <s> = <simple>;  
  
<simple> = | hello  
          | hello judy  
          | goodbye  
          | 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 president at the university of windsor  
          | tell me a poem
```



```
| go back  
| tell me a joke  
| can I talk to monty  
| can I talk to judy  
| can I talk to solar man;
```

The first line: “**grammar vxmljudy;**” The grammar name is “*vxmljudy*”. It has to be the same as the filename “*vxmljudy*” and the file extension has to be “*.gram*”.

The last three lines, “**can I talk to ...**” is a way of asking a new sihlo. For example, if user asks “**can I talk to monty**”, the sihlo application will return the Monty sihlo address and information in the same format of the sihlo header file. The detail is discussed in section 5.

More complex grammar can be developed using JSGF.

4.2.3 Interpreter Application

The sihlo interpreter is used to process queries and produce answer. User asks a question or says a sentence; the SpeechWeb Browser recognizes it and translates it into text. The text is sent from the local machine to the remote sihlo using CGI Protocol. The interpreter processes the text and produces an answer or response in text format. It is sent back to the SpeechWeb Browser. The browser speaks the answer out.

The interpreter application can be implemented using any language. There is a simple example of implementing an interpreter using Shell Script in UNIX:

judy.cgi


```

#!/bin/csh -f
# this is a comment
echo "Content-Type:text/plain"
echo "

setenv v ``/bin/cat``
#echo $v

set reply = `echo $v | awk -F='{print $2 }`"
#echo $reply

#strip of the  + inserted by post
set replycase = `echo $reply | sed -e s/+/ /g`

switch ("$replycase")
    case "hello":
        echo "hello how are you"
        breaksw
    case "I am fine":
        echo "My name is Judy I am from Windsor"
        breaksw
    case "How are you":
        echo " I am fine thank you"
        breaksw
    default:
        echo "please repeat I don't understand"
        breaksw
endsw
end

```

As you can see in the example, there are only three sentences can be processed and only four different answers can be produced. Other than “hello”, “I am fine” and “How are you”, the answer will always be “please repeat I don’t understand.”

This is a simple way of implementing the interpreter application. We match each

question with an answer.

If user wants to talk to other sihlos and the current interpreter knows the sihlo address, the current interpreter should return the new sihlo header file address in the format of “LINK=;SIHLO=;”.

For example, if user says “can I talk to monty”, interpreter should return:

“LINK=yes, here he is;” ++

“SIHLO=davinci.newcs.uwindsor.ca/~su5/monty/monty.sihlo;”

“LINK=” is followed by the interpreter response. It will be spoken out in the SpeechWeb Browser. “SIHLO=” is followed by the full URL address of monty sihlo header file.

More complex programme and other programming languages can be used to implement bigger interpreter to handle more questions.

Protocols and Restrictions

5.1 Protocols

All the queries and answers are in text format.

If user asks for a new sihlo, the returned text from sihlo to local machine should be in the following format:

"LINK=;SIHLO=;"

LINK can be empty, SIHLO cannot.

LINK contains the interpreter response. It will be spoken out by the SpeechWeb Browser.

SIHLO contains the full URL address of the remote sihlo header file.

An example:

"LINK=yes. Here he is;"++
"SIHLO=davinci.newcs.uwindsor.ca/~su5/monty/monty.sihlo;"

5.2 Restrictions

5.2.1 Local Machine

Local sihlo address file *default.sihlo* contains one and only one line of text. It contains the full URL address of the remote sihlo header file.

5.2.2 Remote Sihlo

1. Sihlo header file, sihlo interpreter application executable and sihlo grammar file

must be in a same directory.

2. The interpreter and grammar path cannot contain space.
3. Sihlo header file filename must have an extension name “.*sihlo*”.
4. Interpreter name and header file name must be the same. For example, “*judy.sihlo*” and “*judy.cgi*”
5. Grammar filename must be “*vxml*” follow by sihlo name, and then follow by extension name “.*gram*”. For example, “*vxmljudy.gram*”
6. In the grammar path of the sihlo header file, instead of “*vxml*” in the grammar filename, we have to change it to “*h.cgi?*” For example, instead of:

SIHLO_GRAMMAR=/~su5/judy/vxmljudy.gram;

We use:

SIHLO_GRAMMAR=/~su5/judy/h.cgi?judy.gram;

Program listing

6.1 *default.sihlo*

davinci.newcs.uwindsor.ca/~richard/judy/judy.sihlo

6.2 *start.vxml*

```
<?xml version="1.0"?>
<vxml version="1.0" >
  <property name="caching" value="safe"/>
  <form id="init">
    <block>
      <prompt>Welcome to the speech web browser</prompt>
    </block>
    <!-- *****
start the interface by calling Java object Start.java
***** -->
    <object caching="safe" name="vxml_interface" classid="method://Start/test"
codetype="javacode">
      </object>

    <block>
      <goto next ="t0.vxml"/>
    </block>
  </form>

  <catch event="error">
    Error ocured, please try again.
  <exit/>
</catch>
</vxml>
```

6.3 *t-1.vxml*

```
<?xml version="1.0"?>
<vxml version="1.0" >
  <property name="caching" value="safe"/>
  <property name="confidence" value="1.0"/>
  <!-- *****declare global variables***** -->
```

```

<!-- break -->
<var name="counter" expr="-1" />
<var name="path" expr="/~richard/judy/judy.cgi" />
<var name="loc" expr="davinci.newcs.uwindsor.ca" />
<var name="gramPath" expr="/~richard/judy/vxmljudy.gram" />
<var name="name" expr="judy" />
<var name="greeting" expr="hi, my name is judy" />
<var name="sihlo" expr="URL/path/filename.sihlo" />
<!-- break -->

<!-- *****
"break" in this page is used as a note for the Java object to stop
reading this file and do necessary changes to the following lines
after the "break". Then the Java object will write a new file.
***** -->

<!-- *****Embedding ECMA script into VXML. ***** -->
<script>
<![CDATA[

/*****
This function returns the address of the server.
The one which has the required directories and files
for accessing the remote files.
*****/

function getSihlo(objectparam)
{
var loc;
var ex = objectparam;
var index;

/*****
checks if there is no occurrence of '=' in the
variable ex which indicates that the files are
the same and need not be changed.
*****/
if(ex.indexOf("=">0)) == -1)
return "-1";
// extracts the 'LINK=' substring from the string and assigns it to the variable ex
ex = ex.slice(5);
// gets the index position of ','
index = ex.indexOf(",>0);
index = index+1;

```

```

/*****
the string after the '=' and upto ';' are eliminated
because this is the content which is the result of
the user's input
*****/

```

```

loc = ex.substr(index);
/*****
eliminating 'SIHLO=' from the loc variable.
SIHLO contains the server address starting right
after '=' and ended by the delimiter ';'
*****/

ex = loc.slice(6);
index = ex.indexOf(";");
loc = ex.substring(0,index);
return loc;

}

```

```

/*****
This function returns the substring that has to be spoken as
a result of the user's input or say query. Same procedure is
applied for extracting the content to be spoken out.
*****/

```

```

function speakout(objectparam)
{
var ex=objectparam;

var index;
if((ex.indexOf("=")) == -1)
return ex;
ex= ex.slice(5);
index = ex.indexOf(";");
ex = ex.substring(0,index);
return ex;

}

```

```

/*****
This function returns the name of the file to which the grammar
and interpreter files change dynamically. This function is needed

```

to change the grammar files rather than interpreter files.

*****/

```
function getName(path)
```

```
{
```

```
var ex=path;
```

```
/******
```

When path is -1, in order for the function to be
synchronous with the other functions, the if construct
which returns nothing is needed.

```
*****/
```

```
if((ex.indexOf("/",0)) == -1)
```

```
return "";
```

```
var index;
```

```
var i;
```

```
index=ex.lastIndexOf(".");
```

```
i=ex.lastIndexOf("/");
```

```
i = i+1;
```

```
ex = ex.substring(i,index);
```

```
return ex;
```

```
}
```

```
/******
```

This function returns the complete path of the grammar
file in the remote server. The grammar path is extracted
from the interpreter path

```
*****/
```

```
function getGramPath(objectparam)
```

```
{
```

```
var path;
```

```
var ex= objectparam;
```

```
if((ex.indexOf("-",0)) == -1)
```

```
return "-1";
```

```
ex = ex.slice(5);
```

```
var index = ex.indexOf(";",0);
```

```
index = index+1;
```

```
ex = ex.substr(index);
```

```
ex = ex.slice(12);
```

```
index = ex.indexOf(";",0);
```

```
index = index+1;
```

```
ex = ex.substr(index);
```

```
ex = ex.slice(18);
```

```
index = ex.indexOf(";",0);
```

```
index = index+1;
```

```
ex = ex.substr(index);
```



```

ex = ex.slice(25);
index = ex.indexOf("h.cgi?",0);
path = ex.substring(0,index);
path = path.concat("vxml");
index = index+6;
ex = ex.substr(index);
index = ex.indexOf(";",0);
path = path.concat(ex.substring(0, index));
return path;
/*
var ex=path;
var name;
if(ex.indexOf("/") == -1)
return "";
var index;
var i;
index=ex.lastIndexOf(".");
i=ex.lastIndexOf("/");
i = i+1;
name = ex.substring(i,index);
ex = ex.substring(0, i);
*/
/*
<!-- break -->
*/
/*ex = ex.concat("h.cgi?");
ex = ex.concat("vxml");
ex = ex.concat(name);
ex = ex.concat(".gram");
return ex;
*/
}
function increase(a)
{
++a;
return a;
}

]]> </script>

<form id="start">
<block>
<prompt>you are talking to <value expr="name"/> now</prompt>
<prompt> <value expr="greeting"/> </prompt>

```

```

<goto next="#one"/>
</block>
</form>

<form id="one">

<field name="mode">

<!-- break -->
<grammar caching="safe" src="http://davinci.newcs.uwindsor.ca/~richard/h.sh?vxmlyudy.gram"/>

</field>
<!-- *****
This posts the users input to the Post.java
file and the results are stored in 'a'. It
posts through the parameter setQuestion and gets
the result through the getAnswer method which is
in the java file. The user input is stored in the
variable 'mode'. The value of the mode is sent
to setQuestion method in the java file through
the parameter setQuestion declared in the <param ...>
***** -->
<object name="a" classid="method://Post/getAnswer" codetype="javacode">
<param name="setQuestion" expr="mode"/>
<param name="setLocation" expr="loc"/>
<param name="setPath" expr="path"/>
</object>

<block>

<if cond="mode == 'goodbye'">
<goto next="#goodbye"/>
</if>

<if cond="mode == 'go back'">
<goto next="#back"/>
</if>

<var name="objectparam" expr="a"/>
<var name="answer" expr="-l"/>

<!-- *****
This calls the ECMA script function and store the

```

answer to a variable, in this case, it is stored
in variable "answer".

```
***** -->
<assign name="answer" expr="speakout(objectparam)"/>
```

```
<!-- *****
```

Speaks out the answer

```
***** -->
```

```
<prompt> <value expr = "answer"/> </prompt>
```

```
<!-- *****
```

Assigning the result of the ECMA script
functions to the above defined global variables
whose purpose is to share the server address
and the exact path of the required files to be
passed on and accessed by other VXML pages later on.

```
***** -->
```

```
<assign name="sihlo" expr= "getSihlo(objectparam)" />
```

```
<!-- *****
```

If the answer returned from the interpreter contains
a new link, it then transits to the transit form.
Otherwise, keep talking in the same form.

```
***** -->
```

```
<if cond="getSihlo(objectparam) != '-1'">
```

```
<goto next="#transit"/>
```

```
</if>
```

```
<if cond="sihlo == '-1'">
```

```
<goto next="#one"/>
```

```
</if>
```

```
</block>
```

```
</form>
```

```
<!-- *****
```

This form will call the Java object defined in Rewrite.java
it sets all the necessary parameters first through those
"set" methods. Then the "writeFile" method will write a new
VXML page before the browser transits to that page.

```
***** -->
```

```
<form id="transit">
```

```

<object name="w" classid="method://Rewrite/writeFile" codetype="javacode">
<param name="setCounter" expr="counter"/>
<param name="setSihlo" expr="sihlo"/>
</object>
</block>

<!-- break -->
<goto caching="safe" next="t0.vxml"/>
</block>
</form>

<!-- *****
When user says "goodbye", it will call the Java method "goodbye"
in Rewrite.java. The Java method deletes all the files in the
stack. Then it exits the VXML browser
***** -->
<form id="goodbye">

<object name="g" classid="method://Rewrite/goodbye" codetype="javacode">
<param name="setCounter" expr="counter"/>
</object>

</block>
<prompt> goodbye </prompt>
<exit/>
</block>
</form>

<!-- *****
When user says "Go back", it will call the Java method "back"
in Rewrite.java. The Java method check if there is a page
exists that we can go back to. If there is, it transits to
that page. Otherwise, keep talking.
***** -->
<form id="back">
<object name="flag" classid="method://Rewrite/goBack" codetype="javacode">
<param name="setCounter" expr="counter"/>
</object>
</block>
<if cond="flag == '1'">
<prompt> Cannot go back </prompt>
<goto next="#one"/>
</else/>

```

```

<!-- break -->
<goto caching="safe" next="t.vxml"/>
</if>
</block>
</form>

<!-- *****
Error handling. If any error occurs, the VXML browser will
transit to the starting page which is "start.vxml"
***** -->

<catch event="error">
Error occured, going back to the first page.
<goto next="start.vxml"/>
</catch>

</vxml>

```

6.4 HTTPpost.java

```

// DESCRIPTION SUMMARY OF CLASS:
/* The class is designed to perform a CGI based post method primarily
   used to send queries and recive responses during the
   client-server communication.
*/
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 String receiveData; //string representation of data recieved from silho
                               //when a query is sent to it

    // DESCRIPTION OF CONSTRUCTOR:
    /* The constructor takes in a host url name (or IP address) along with necessary
       port number and name of the file found at the given hostname. A query string
       (which is also provided as a constructor parameter) is then piped as an output
       stream to the remote script file found at the given host (script file given in
       the query itself). Upon answering the incoming query, the file then transmits

```

a new stream of data back as incoming input stream for our constructor to pickup.
The stream of data is then stored in a String variable named `recieveData`.

```

*/
public HTTPpost(String hostName, String fileName, String query ) throws Exception
{
    URL 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");

    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)
    {
        receiveData = nextline;
    }
    dis.close();
}

// DESCRIPTION OF METHOD:
/* This function returns the recieveData variable when called
*/
public String getResults ()
{
    return (receiveData) ;
}
}

```

6.5 *Post.java*

```

// DESCRIPTION OF CLASS:
/* This class is designed to receive sihlo information and user query
   through those "set" methods. Then it calls class object HTTPpost

```

```

        to get answers back.
    */

import java.net.*;
import java.io.*;
import java.io.IOException;

public class Post
{
    private String location="davinci.newcs.uwindsor.ca";
    private String file="/~su5/judy/judy.cgi";
    private String question="question=" ;
    public Post()
    {

    }
    public String getAnswer()
    {
        String res="";
        try{
            HTTPPost obj = new HTTPPost(location, file, question);
            res=obj.getResults();
            question="question=" ;
        }catch (Exception e){ e.printStackTrace(); }
        return ((String)res);
    }
    public void setQuestion(String que)
    {
        question += que;
    }

    public void setPath(String path)
    {
        if(path.compareTo("-1")==0)
            return;

        file = path;
    }
    public void setLocation(String loc)
    {
        if(loc.compareTo("-1")==0)
            return;

        location = loc;
    }

```

```

    }

    public static void main (String args[])
    {
        Post p = new Post();
        p.setQuestion("how old are you");
        System.out.println(p.getAnswer());
    }
}

```

6.6 Rewrite.java

// DESCRIPTION OF CLASS:

/* This class uses "set" methods to define all the global variables. If the calling VXML page is the default "t-1.vxml", the "writeFile" method will write a new VXML file named "t0.vxml". Then if the calling page is "t0", the next page will be "t1", and the next one after "t1" will be "t2". In another word, if the current page is "t(n)", the new one will be "t(n+1)".

The method "goodbye" deletes all the newly created VXML pages in the stack, maximum 10 pages in total, from "t(n).vxml" to "t(n-9).vxml". If there are less than 10 pages, it deletes whatever is in the stack.

Also, the programme will delete all the files that are created by this class, in case the user terminate programme illegally.

The method "goBack" tests if the VXML page can go back to a previous page. In another word, it tests if the previous page exists. If can go back, returns 0. If cannot go back, returns 1.

```

*/
import java.io.*;
import java.io.IOException;
import java.net.URL;
import java.net.URLConnection;

public class Rewrite
{
    private int counter;      //count the current VXML page number
    private String file="";   //interpreter path in the host
    private String location=""; //host address
    private String gramPath=""; //grammar path in the host
    private String name="";    //grammar/interpreter name
    private String greeting="";
    private String sihlo="";

```



```

private String stringB4Gram="";
public Rewrite()
{
}

// DESCRIPTION OF METHOD:
// Receive the page number from the calling VXML page
public void setCounter(String a)
{
    counter = Integer.parseInt(a);
}

// DESCRIPTION OF METHOD:
// Set the path of the interpreter file
public void setPath(String path)
{
    file = path;
}

// DESCRIPTION OF METHOD:
// Set the IP address of the interpreter
public void setLocation(String loc)
{
    location = loc;
}

// DESCRIPTION OF METHOD:
// Set the path of the grammar file
public void setGramPath(String gram)
{
    gramPath = gram;
}

// DESCRIPTION OF METHOD:
// Set the grammar name
public void setName(String n)
{
    name = n;
}

public void setGreeting(String n)
{
    greeting = n;
}

```

```

public String setSihlo(String a)
{
    sihlo = a;
    String in="", hostName="", fileName="";
    int index = -1;
    index = a.indexOf("/");
    if( index == -1)
        return "1";
    hostName = a.substring(0,index);
    fileName = a.substring(index);
    try{
        URL urlToGo = new URL ("http", hostName, fileName);
        URLConnection uc = urlToGo.openConnection();
        uc.setDoOutput(false);
        uc.setDoInput(true);
        uc.setAllowUserInteraction(false);
        uc.setRequestProperty("Content-type","text/plain");
        DataInputStream inp = new DataInputStream(uc.getInputStream());
        //if the file is empty, return.
        if( (in=inp.readLine()) == null )
            return "-1";
        index = in.indexOf("=");
        if( index == -1)
            return "-1";
        in = in.substring(index+1);
        index = in.indexOf(";");
        if( index == -1)
            return "-1";
        greeting = in.substring(0, index);

        //read sever URL
        if( (in=inp.readLine()) == null )
            return "-1";
        index = in.indexOf("=");
        if( index == -1)
            return "-1";
        in = in.substring(index+1);
        index = in.indexOf(";");
        if( index == -1)
            return "-1";
        location = in.substring(0, index);

        //read interpreter path
    }
}

```

```

    if( in=inp.readLine() == null )
        return "-1";
    index = in.indexOf("=");
    if( index == -1)
        return "-1";
    in = in.substring(index+1);
    index = in.indexOf(";");
    if( index == -1)
        return "-1";
    file = in.substring(1, index);

    //read grammar path
    if( in=inp.readLine() == null )
        return "-1";
    index = in.indexOf("=");
    if( index == -1)
        return "-1";
    in = in.substring(index+2);
    index = in.indexOf("h.cgi?");
    if( index == -1)
        return "-1";
    gramPath = in.substring(0, index);
    gramPath += "vxml";
    index += 6;
    in = in.substring(index);
    index = in.indexOf(";");
    if( index == -1)
        return "-1";
    gramPath += in.substring(0, index);

    //read the name of interpreter
    index = file.lastIndexOf("/");
    if( index == -1)
        return "-1";
    name = file.substring(index+1);
    index = name.indexOf(".");
    if( index == -1)
        return "-1";
    name = name.substring(0, index);
    file = "/" + file;
    gramPath = "/" + gramPath;
} catch (Exception ioe)
{
    ioe.printStackTrace();
}

```

```

        return "1";
    }
    return "0";
}

public void setStringB4Gram(String n)
{
    stringB4Gram = n;
}
// DESCRIPTION OF METHOD:
/* This method uses all the variables set by previous methods and write a new
VXML page. If the calling page is "t2.vxml", the new page will be "t3.vxml".
The method will read the old file line by line, when it reads the line that
contains string "<!-- break -->", it will stop and redefine some variables in
the new VXML page.
*/
public int writeFile()
{
    String in = "t" + counter + ".vxml"; // name of the calling page
    counter++;
    String out = "t" + counter + ".vxml"; // name of the new page
    String tmp, back="";
    while(name=="");
    try{
        // read from the default page t.vxml
        BufferedReader input = new BufferedReader(new FileReader(in));
        FileWriter output = new FileWriter(out);
        BufferedWriter bw = new BufferedWriter(output);
        while( (tmp=input.readLine()) != null )
        {
            bw.write(tmp);
            bw.newLine();
            if(tmp.compareTo("<!-- break -->") == 0 )
                break;
        }
        // Write global variables in the new file
        bw.write("<var name=\"counter\" expr=\"\""+
            counter+"\"/>");
        bw.newLine();
        bw.write("<var name=\"path\" expr=\"\""+
            file+"\"/>");
        bw.newLine();
        bw.write("<var name=\"loc\" expr=\"\""+

```

```

        location+"\"/>");
bw.newLine();
bw.write("<var name=\"gramPath\" expr=\"\""+
        gramPath+"\"/>");
bw.newLine();
bw.write("<var name=\"name\" expr=\"\""+
        name+"\"/>");
bw.newLine();
bw.write("<var name=\"greeting\" expr=\"\""+
        greeting+"\"/>");
bw.newLine();
bw.write("<var name=\"sihlo\" expr=\"\""+
        sihlo+"\"/>");
bw.newLine();
// skip all lines until next "break"
while( (tmp=input.readLine()) != null )
{
    if(tmp.compareTo("<!-- break -->") == 0 )
    {
        bw.write(tmp);
        bw.newLine();
        break;
    }
}
while( (tmp=input.readLine()) != null )
{
    bw.write(tmp);
    bw.newLine();
    if(tmp.compareTo("<!-- break -->") == 0 )
        break;
}
//add the part of string before the grammar filename
if(stringB4Gram.compareTo("") != 0)
{
    bw.write("*/");
    bw.newLine();
    bw.write("ex = ex.concat(\"\\");
    bw.newLine();
    input.readLine();
    input.readLine();
}
while( (tmp=input.readLine()) != null )
{
    bw.write(tmp);

```

```

        bw.newLine();
        if(tmp.compareTo("<!-- break -->") == 0 )
            break;
    }
    // Write grammar URL
    bw.write("<grammar caching=\"fast\" src=\"http://"+
        location+gramPath+"\"/>");
    bw.newLine();
    input.readLine();           //skip a line

    while( (tmp=input.readLine()) != null )
    {
        bw.write(tmp);
        bw.newLine();
        if(tmp.compareTo("<!-- break -->") == 0 )
            break;
    }
    // Write the next VXML page file name. For example, if the new page
    // is t3.vxml, the next page it will eventually transit to is t4.vxml
    counter++;
    bw.write("<goto next=\"t"+counter+".vxml\"/>");
    bw.newLine();
    input.readLine();           //skip a line
    while( (tmp=input.readLine()) != null )
    {
        bw.write(tmp);
        bw.newLine();
        if(tmp.compareTo("<!-- break -->") == 0 )
            break;
    }
    // Write the previous VXML page file name that the user can go back to.
    int i=counter-2;
    if(i >= 0)
    {
        bw.write("<goto next=\"t"+i+".vxml\"/>");
        bw.newLine();
    }else{
        bw.write("<goto next=\"t.vxml\"/>");
        bw.newLine();
    }
    input.readLine();           //skip a line
    while( (tmp=input.readLine()) != null )
    {
        bw.write(tmp);

```

```

        bw.newLine();
    }
    bw.flush();
    bw.close();
    input.close();
    output.close();
    File x = new File(out);
    // Delete every file that is every created by this Jave class object,
    // in case the user terminate the programme illegally.
    x.deleteOnExit();
} catch(IOException ioe)
{
    ioe.printStackTrace();
}
System.out.println("new file name: "+out );
// It keeps 10 pages in a stack, and delete the oldest page.
// The page t.vxml is always kept as a default page.
if( (counter-11) >= 0 )
{
    String old="t"+(counter-11)+".vxml";
    File f=new File(old);
    f.delete();
}
return 0;
}

```

// DESCRIPTION OF METHOD:

/* This method deletes all the newly created VXML pages in the stack, maximum 10 pages in total, from "t(n).vxml" to "t(n-9).vxml". If there are less than 10 pages, it deletes whatever is in the stack.

*/

```

public int goodbye()
{
    String name="";
    int start = counter;
    int end = counter-9;
    if(start < 0)
        return 0;
    if(end < 0)
        end = 0;
    for(int i=start; i>=end; i--)
    {
        name = "t"+i+".vxml";
    }
}

```

```

        File f=new File(name);
        f.delete();
    }
    return 0;
}

// DESCRIPTION OF METHOD:
/* This method tests if the VXML page can go back to a previous page.
   In another word, it tests if the previous page exists. If can go back, return 0.
   If cannot go back, return 1.
*/
public int goBack()
{
    String page="t"+(counter-1)+".vxml";
    if(counter == 0)
        return 1;
    File b=new File(page);
    if(b.exists())
        return 0;
    else
        return 1;
}

// DESCRIPTION OF METHOD:
// main method used for testing
public static void main(String args[]) throws Exception
{
    String host="davinci.newcs.uwindsor.ca";
    String path="~richard/judy/judy.cgi";
    String gram="~richard/judy/vxmljudy.gram";
    String n="judy",g="hi, it's me.";
    Rewrite r = new Rewrite();
    r.setCounter("-1");
    //r.setPath(path);
    //r.setLocation(host);
    //r.setGramPath(gram);
    //r.setName(n);
    //r.setGreeting(g);
    r.setSihlo("davinci.newcs.uwindsor.ca/~su5/judy/judy.sihlo");
    r.writeFile();
    //r.goodbye();
    System.out.println("success");
} //end of main

```



```
} //end of class
```

6.7 Start.java

```
// DESCRIPTION OF CLASS:
```

```
/* This class uses JFrame to create a user interface. User can input remote server
information in JTextFields. After user enters information of the server, the
programme will have a simple check and see if the information is valid.
User cannot leave any JTextField empty. Space is not allowed in any part of the
input. For the interpreter file and grammar file, they must have a dot "." to
separate the filename and extension name. The grammar file must in the format of
"vxml" following grammar name, then the dot ".", then the extension "gram".
For example, a "Judy" grammar would be "vxmljudy.gram".
```

```
*/
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import java.net.URL;
import java.net.URLConnection;
import java.io.*;
import java.io.IOException;

public class Start extends JFrame
{
    private JPanel south, center, north;
    private JTextField address, path, interp, gram;
    private JTextArea display;
    private JLabel l1, l2;
    private JButton ok, reset, change, go;
    private String input="";
    private Container c;
    private String address1="", path1="", interp1="",
                    gram1="", name="", gramRule="", greeting="";
    private GridBagConstraints gbc;
    private GridBagLayout gb;

    public Start()
    {
        super("Welcome to the SpeechWeb Browser");
        c = getContentPane();
        c.setLayout( new BorderLayout() );
    }
}
```

```

north = new JPanel();
south = new JPanel();
gb = new GridBagLayout();
south.setLayout(gb);
gbc = new GridBagConstraints();
address = new JTextField(20);
address.setText("default.sihlo");
address.setEditable(true);
path = new JTextField(30);
path.setText(read("default.sihlo"));
path.setEditable(true);
ok = new JButton("Use Sihlo File");
change = new JButton("Change Default");
reset = new JButton("Reset to Default");
go = new JButton("Go");
display = new JTextArea(4, 48);
display.setText("To use the displayed sihlo, click \"Use Sihlo File\\n\"+
    \"To change the URL of the default home sihlo, click \"Change Default\\n\"+
    \"To use displayed URL, click \"Go\\n\" );
display.setEditable(false);
north.add(display);
c.add(north, BorderLayout.NORTH);
l1 = new JLabel("Local filename for default sihlo : ");
l2 = new JLabel("Sihlo URL : ");
gbc.gridx = 0;
gbc.gridy = 0;
gbc.gridwidth = 2;
gbc.gridheight = 1;
gbc.fill = GridBagConstraints.BOTH;
gb.setConstraints(ok, gbc);
south.add(ok);
gbc.gridx = 2;
gbc.gridy = 0;
gbc.gridwidth = 2;
gbc.gridheight = 1;
gbc.fill = GridBagConstraints.BOTH;
gb.setConstraints(reset, gbc);
south.add(reset);
gbc.gridx = 4;
gbc.gridy = 0;
gbc.gridwidth = 2;
gbc.gridheight = 1;
gbc.fill = GridBagConstraints.BOTH;
gb.setConstraints(change, gbc);

```

```

south.add(change);
gbc.gridx = 0;
gbc.gridy = 1;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gb.setConstraints(l1, gbc);
south.add(l1);
gbc.gridx = 3;
gbc.gridy = 1;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.fill = GridBagConstraints.BOTH;
gb.setConstraints(address, gbc);
south.add(address);
gbc.gridx = 0;
gbc.gridy = 2;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gb.setConstraints(l2, gbc);
south.add(l2);
gbc.gridx = 1;
gbc.gridy = 2;
gbc.gridwidth = 4;
gbc.gridheight = 1;
gbc.fill = GridBagConstraints.BOTH;
gb.setConstraints(path, gbc);
south.add(path);
gbc.gridx = 5;
gbc.gridy = 2;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.fill = GridBagConstraints.BOTH;
gb.setConstraints(go, gbc);
south.add(go);
c.add(south, BorderLayout.CENTER);

//When "Reset to Default" is clicked, reset everything to the default.
reset.addActionListener(
    new ActionListener()
    {
        public void actionPerformed( ActionEvent e )
        {
            address.setText("default.sihlo");
            path.setText(read("default.sihlo"));

```

```

        display.setText("To use the displayed sihlo, click \"Use Sihlo File\"\\n"+
        "To change the URL of the default home sihlo, click \"Change Default\"\\n"+
        "To use displayed URL, click \"Go\"");
    }
}

);

//Change the default sihlo information in the default.sihlo file
change.addActionListener(
    new ActionListener()
    {
        public void actionPerformed( ActionEvent e )
        {
            display.setText("Reseting default header file");
            address.setText("default.sihlo");
            write();
        }
    }
);

//Use the file displayed in the "Local filename for default sihlo" textfield if available
//to connect to the sihlo and start the speech
ok.addActionListener(
    new ActionListener()
    {
        public void actionPerformed( ActionEvent e )
        {
            String tmp = "";
            String answer = "",flag="";
            tmp = address.getText();
            if(tmp.compareTo("")==0){
                display.setText("Please input the sihlo header filename");
                return;
            }else{
                answer = read(tmp);
                if(answer.compareTo("CANNOT OPEN FILE")==0){
                    display.setText("Cannot open file: "+tmp);
                    return;
                }else if(answer.compareTo("INCORRECT FILE")==0){
                    display.setText("INCORRECT HEADER FILE: "+tmp);
                    return;
                }else{
                    flag = readRemote(answer);
                    if(flag.compareTo("1")==0){

```

```

        display.setText("Cannot connect to server");
        return;
    }else if(flag.compareTo("-1")==0){
        display.setText("Invalid server sihlo header file");
        return;
    }else{
        input = address1+"\n"+interp1+"\n"+gram1+"\n"+name;
    }
    }
    }
    }
    );

//read the entered file in the textfield if available
address.addActionListener(
    new ActionListener()
    {
        public void actionPerformed( ActionEvent e )
        {
            String file = address.getText();
            path.setText( read(file) );
        }
    }
);

//connect to the sihlo use the URL displayed in the "Sihlo URL" textfield
//and start the speech
go.addActionListener(
    new ActionListener()
    {
        public void actionPerformed( ActionEvent e )
        {
            String tmp = "";
            String flag="";
            tmp = path.getText();
            if(tmp.compareTo("")==0){
                display.setText("Please input the remote sihlo URL");
                return;
            }else{
                flag = readRemote(tmp);
                if(flag.compareTo("1")==0){
                    display.setText("Cannot connect to server");
                    return;
                }
            }
        }
    }
);

```

```

        }else if(flag.compareTo("-1")==0){
            display.setText("Invalid server sihlo header file");
            return;
        }else{
            input
greeting+"\n"+address1+"\n"+interp1+"\n"+gram1+"\n"+name;
        }
    }
}
);
setSize( 550, 190 );
show();
}

```

// DESCRIPTION OF METHOD:

/* This method uses a busy waiting to check if a correct input is entered. if true, it will create a new "Rewrite" object and pass all the information obtained from the interface. the "r.writeFile();" statement will create a new VXML file based on the information.

*/

```

public String test()
{
    input="";
    setVisible(true);
    while(input.compareTo("")==0)
    ;
    Rewrite r = new Rewrite();
    r.setCounter("-1");
    r.setPath(path1+interp1);
    r.setLocation(address1);
    r.setGramPath(path1+gram1);
    r.setName(name);
    r.setStringB4Gram(gramRule);
    r.setGreeting(greeting);
    r.writeFile();
    setVisible(false);
    return input;
}

```

// DESCRIPTION OF METHOD:

/* This method first checks if the file exists. It returns "CANNOT OPEN FILE" if the file cannot be found, returns "INCORRECT FILE" if the content of the header file does not match the required format. Otherwise, it returns

```

        the sihlo URL from the header file
    */
    public String read(String a)
    {
        String in;
        File f=new File(a);
        if(!f.exists())
            return "CANNOT OPEN FILE";
        try{
            // read the default header file
            BufferedReader inp = new BufferedReader(new FileReader(a));
            in=inp.readLine();
            //if the file is empty, return.
            if( in != null && !in.endsWith(".sihlo") )
                return "INCORRECT FILE";
        }catch(IOException ioe)
        {
            ioe.printStackTrace();
            return "-1";
        }

        return in;
    }
    // DESCRIPTION OF METHOD:
    /* This method first checks if the file exists. It returns "1" if the file cannot be
        be found, returns "-1" if the content of the header file does not match the protocol.
        Otherwise, it extracts information from the header file and returns "0"
    */
    public String readRemote(String a)
    {
        String in="", hostName="", fileName="";
        int index = -1;
        index = a.indexOf("/");
        if( index == -1)
            return "1";
        hostName = a.substring(0,index);
        fileName = a.substring(index);
        try{
            URL urlToGo = new URL ("http", hostName, fileName);
            URLConnection uc = urlToGo.openConnection();
            uc.setDoOutput(false);
            uc.setDoInput(true);
            uc.setAllowUserInteraction(false);
            uc.setRequestProperty("Content-type", "text/plain");

```

```

DataStream inp = new DataInputStream(uc.getInputStream());

//if the file is empty, return.
if( in=inp.readLine() == null )
    return "-1";
index = in.indexOf("=");
if( index == -1)
    return "-1";
in = in.substring(index+1);
index = in.indexOf(";");
if( index == -1)
    return "-1";
greeting = in.substring(0, index);

//read sever URL
if( in=inp.readLine() == null )
    return "-1";
index = in.indexOf("=");
if( index == -1)
    return "-1";
in = in.substring(index+1);
index = in.indexOf(";");
if( index == -1)
    return "-1";
address1 = in.substring(0, index);

//read interpreter path
if( in=inp.readLine() == null )
    return "-1";
index = in.indexOf("=");
if( index == -1)
    return "-1";
in = in.substring(index+1);
index = in.indexOf(";");
if( index == -1)
    return "-1";
interp1 = in.substring(1, index);

//read grammar path
if( in=inp.readLine() == null )
    return "-1";
index = in.indexOf("=");
if( index == -1)
    return "-1";

```



```

        in = in.substring(index+2);
        index = in.indexOf("h.cgi?");
        if( index == -1)
            return "-1";
        gram1 = in.substring(0, index);
        gram1 += "vxml";
        index += 6;
        in = in.substring(index);
        index = in.indexOf(",");
        if( index == -1)
            return "-1";
        gram1 += in.substring(0, index);

        //read the name of interpreter
        index = interp1.lastIndexOf("/");
        if( index == -1)
            return "-1";
        name = interp1.substring(index+1);
        index = name.indexOf(".");
        if( index == -1)
            return "-1";
        name = name.substring(0, index);
        path1 += "/";

    } catch(Exception ioe)
    {
        ioe.printStackTrace();
        return "1";
    }
    return "0";
}

// DESCRIPTION OF METHOD:
/* This method rewrites the file "default.sihlo" in the local directory using new
   information provided by the user. If the file does not exist, it will create one
   */
public void write()
{
    String in="default.sihlo", tmp="";
    String out="", confirm="";
    int ans=1;
    JFrame dis = new JFrame();
    try{
        tmp = read(in);

```

```

        if(tmp.compareTo("CANNOT OPEN FILE")==0 || tmp.compareTo("INCORRECT
FILE")==0)

            display.setText("Incorrect sihlo header file\n" +
                            "Please rewrite a new header file");

        while(ans == 1)
        {
            /* Method "showInputDialog" in Java 1.3 cannot have a predefined value
            displayed in the text field. However, Java 1.4 can.
            */
            //for Java 1.4 or higher
            //out = JOptionPane.showInputDialog("Enter the sihlo URL", tmp);
            //for Java 1.3
            out = JOptionPane.showInputDialog("Enter the sihlo URL\n"+
            "Default URL: "+tmp);
            if(out == null){
                out = tmp;
                break;
            }
            ans = JOptionPane.showConfirmDialog(null,
            "Default is going to be changed to:\n"+out);

            if(ans == 0){
                FileWriter output = new FileWriter("default.sihlo");
                BufferedWriter bw = new BufferedWriter(output);
                bw.write(out);
                bw.flush();
                bw.close();
                JOptionPane.showMessageDialog(null, "Default is changed to:\n"+out);
                path.setText(read("default.sihlo"));
            }
        }
    }catch(IOException ioe)
    {
        ioe.printStackTrace();
    }
}

// DESCRIPTION OF METHOD:
// main method used for testing
public static void main ( String args[] )
{
    Start app = new Start();

```

```

String output = app.test();
System.out.println(output);
app.addWindowListener(
    new WindowAdapter(){
        public void windowClosing( WindowEvent e)
        {
            System.exit( 0 );
        }
    }
);
}
}

```

VITA AUCTORIS

Li Su was born in 1980 in Chengdu, Sichuan, China. He graduated from high school in China and came to Canada in 1998. From there he went on to the University of Windsor, Ontario, Canada where he obtained a Bachelor of Science Honours degree in Computer Science in 2003. Currently, he is a candidate for the Master's degree in Computer Science at the University of Windsor and hopes to graduate in Fall 2005.