2006

Using X+V to construct a non-proprietary speech browser for a public-domain SpeechWeb

Xiaoli Ma
University of Windsor

Using X+V to Construct a Non-Proprietary Speech Browser for a Public-Domain SpeechWeb

By

Xiaoli Ma

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

2006

Your file Votre référence
ISBN: 978-0-494-17031-1
Our file Notre référence
ISBN: 978-0-494-17031-1

ABSTRACT

A SpeechWeb is a collection of hyperlinked speech applications that are distributed over the Internet. Users access the speech applications through remote browsers, which accept human-voice-input and return synthesized-voice-output. In previous research, a new architecture (LRRP) has been proposed, which is ideally suited for building a Public-Domain SpeechWeb. However, a non-proprietary speech browser is needed for this architecture. In this thesis, we have solved several limitations of X+V, a programming language for developing Multimodal applications, and we have used X+V to build a viable Public-Domain SpeechWeb browser. Our browser has the following properties: real-time human-machine speech interaction; ease of installation and use; acceptable speech-recognition accuracy in a suitable environment; no cost, non-proprietary, ease of distribution; use of common communication protocol – CGI; ease of creation of speech applications; possibility to deploy on mobile devices.

[Keywords: SpeechWeb, Public-Domain, LRRP architecture, speech recognition, SpeechWeb browser, X+V, XHTML+Voice, Opera, Multimodal, distributed system, mobile speech application.]

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. Kai Hildebrandt, Dr. Dan Wu and Dr. Jianguo Lu 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

List of Tables

List of Figures

Chapter 1: Introduction

1.1 A Public-Domain SpeechWeb

According to Frost [Frost05], a SpeechWeb is a collection of hyperlinked speech applications that are distributed over the Internet. A SpeechWeb is similar to the conventional web in many ways. However, it differs in that users access the speech applications through browsers, which accept human-voice-input and return synthesized-voice-output. Rather than "clicking" on hyperlinks, users issue spoken requests to be transferred to hyperlinked applications.

Frost [Frost05] has made a case for a Public-Domain SpeechWeb to be constructed by non-expert of computer science, using non-proprietary or freely available software, common communication protocols, and conventional web servers.

1.2 The LRRP Architecture

Frost and Su [FAB04] have developed an architecture that claimed to be well suited for the development of a Public-Domain SpeechWeb. The architecture is called Local Recognition Remote Processing (LRRP) architecture. Users begin a session with the SpeechWeb by opening the local speech browser and asking to be connected to a remote application. A grammar is downloaded from the application and is used to tailor the local speech-recognizer for that application. The user speaks a question or command, which is recognized locally and then transmitted as text to the remote application. The application processes the question/command and returns a response as text, which is converted to synthesized voice that is output on the end-user device. It is claimed that the LRRP architecture has a number of advantages compared to

existing architectures for distributed speech applications. More description about these advantages are in Chapter 3.

1.3 Shortcomings of VXML as a SpeechWeb Browser

Frost and Su [FAB04] have shown how a browser for a Public-Domain SpeechWeb can be built as a single VXML page (VXML is a standard programming language for developing speech applications). The page, which is executed on end-user devices using freely available VXML interpreters, takes care of local speech recognition, the sending of recognized input to remote applications, and the synthesis of the voice output. The advantage of using VXML is that recent developments in speech-recognition technology, which are incorporated into VXML interpreters, are available to SpeechWeb users. In addition, VXML has the capability to send text and receive responses from applications running on remote servers. However, VXML has three shortcomings which limit its use in SpeechWeb:

- Grammars cannot be changed dynamically in a single VXML page. Consequently Su and Frost had to integrate the page with Java objects to fetch grammars from remote applications and rewrite the VXML page locally after every execution of a hyperlink. This requires the Java runtime environment to be available locally, precluding the use of the SpeechWeb Browser on lightweight end-user devices such as handheld PCs or cell phones.
- Even without the Java runtime environment, VXML interpreters are relatively demanding of resources and none are yet available for lightweight end-user devices.
- The freely available VXML interpreters that were accessible in the first few years of VXML's existence are no longer supported.

Consequently, we decided to investigate the use of the newer X+V protocol as a mechanism for building the browser for a Public-Domain SpeechWeb.

1.4 X+V

The X+V protocol is a relatively new protocol. It was developed by IBM and Opera, and accepted by W3C in December 2001. X+V includes a subset of VXML elements and supports multi-modal interaction with computers and lightweight end-user devices. X+V is currently supported by the Opera web browser that is freely available, and can be installed on PCs, and handhelds such as Sharp Zaurus 5500-5600.

1.5 Thesis Statement

The thesis to be investigated is that:

"A viable speech browser for a Public-Domain SpeechWeb can be implemented using X+V."

By "viable" we mean –

- *Real-time human-machine speech interaction.*
- *High speech-recognition accuracy in a suitable environment.*
- *Ease of installation and use.*
- *Completely free availability to all users.*
- *Ease of implementation for speech applications.*
- *Possible to be deployed on handheld devices after small changes;*
- *Implements the LRRP architecture so that it is ideally suited for building a Public-Domain SpeechWeb.*

1.6 Why proof of the thesis is important?

The thesis is important because it will facilitate the creation of a Public-Domain SpeechWeb and allow browser for the SpeechWeb to execute on lightweight end-user devices with freely available software.

1.7 Why the thesis is not obvious?

X+V is a very new protocol and very little is known about it. The demonstration applications from IBM are very simple and do not include examples in which grammars are changed dynamically. The speech-recognition accuracy of the Opera X+V interpreter cannot be determined from the demonstration applications from IBM as the example languages are very small.

1.8 How the thesis is proven by demonstration

Various designs were investigated. These included, first, a "Single Page" design which is similar to that of Frost and Su [FAB04] in which the X+V page is rewritten whenever a hyperlink is followed and the speech-recognition grammar needs to be changed. However, instead of using Java objects, which require the runtime environment, the new single-page design uses client-cookies to store small amounts of data. The X+V page is reloaded when connecting to a new CGI interpreter. It has to reload the page, because the parser in Opera environment only parses the VXML parts of X+V once, at the first time it loads the page. Looping through a single page cannot change anything, even though variables may appear to have been reset. Unfortunately, the Single-Page design did not work properly in all situations because of a problem with concurrency between updating the changes to variables in the X+V page and the VXML parsing. This concurrency problem is beyond the control of the application (See details in Chapter 6).

Second, a "Multiple Page" design was investigated in which an X+V browser page is associated with every remote application. These pages are identical for all applications except for the location of the recognition grammar and an application-dependent initial greeting which is spoken when the application is first contacted (See details in Chapter 6).

The "Multiple Page" design was successful and allowed the SpeechWeb Browser to execute on end-user devices with only one requirement – that the Opera browser had to be installed.

The Multiple-page X+V SpeechWeb Browser was tested with a number of applications. Response times and recognition-accuracy levels were documented.

Since Opera is freely available, and can be installed on lightweight end-user devices, and that response times and recognition-accuracy were acceptable, the thesis has been proven by successfully demonstrated.

Chapter 2: Requirements For a Public-Domain SpeechWeb Browser

2.1 Overview

A public-domain SpeechWeb is "a collection of hyperlinked speech applications, developed by individuals, which are distributed over the Internet and which are accessible by spoken commands and queries that are input through remote end-user devices" [FAB04].

A SpeechWeb is based on the Internet, and it is similar to a conventional web. Users can access SpeechWeb applications using a SpeechWeb Browser residing on their computers or lightweight devices.

2.2 Requirements for a Public-Domain SpeechWeb Browser

A number of requirements need to be satisfied for a Public-Domain SpeechWeb Browser to be widely employed in order to provide speech access to distributed hyperlinked knowledge sources on SpeechWeb:

1. **Ease of installation and use**

   A SpeechWeb Browser needs to be easy to install and execute on conventional PCs, laptops, cellphones, and other available lightweight end-user devices. The SpeechWeb Browser should be designed for users who may only have basic knowledge about using computer software. The installation and execution process should be as easy as using regular software.

2. **Acceptable accuracy in the speech recognition process**

Speech-recognition accuracy should be good in most environments. This requires close attention to choosing the speech-recognition engine for the SpeechWeb Browser.

Most of today's speech-recognition engines cannot be deployed on lightweight devices such as cell phones, because the recognition process has relatively high demands on the resources (CPU power and memory). VoiceSignal Company provides voice-dial and voice-message services in today's cellphones, where the speech-recognition engine is installed locally in the user's cell phone. Unfortunately, the engine is not grammar-based and not identical for all speech applications, which results in it being unsuitable for a SpeechWeb Browser.

3. **Common protocols in the data transmission.**

A common protocol, such as CGI (Common Gateway Interface) protocol should be used to support the data transmission between the SpeechWeb Browser and hyperlinked knowledge sources. This is necessary in order to share the knowledge sources with other applications, even with those applications that are not providing services on the SpeechWeb. In addition, CGI is supported by most web server software.

4. **Easy to build speech applications.**

As indicated in the paper "*Call for a Public-Domain SpeechWeb*" [Frost05], one of the reasons constraining the growth of a public-domain SpeechWeb is the difficulty that non-experts have in creating their own speech applications, and then hyperlinking them in a SpeechWeb.

In today's conventional web, many people can build their own HTML websites easily. Our SpeechWeb Browser should allow non-experts to create their speech applications and access them with our SpeechWeb

Browser, a process as easy as building HTML pages.

5. **Non-proprietary or freely available software.**

In a consideration for it to be widely used by all users, the SpeechWeb Browser needs to be completely free, and only contain non-proprietary components.

Most companies ask for several hundred US dollars to provide a speech-recognition server engine, which is too expensive for a single SpeechWeb user. Moreover, using proprietary components in SpeechWeb Browser may cause a problem if the company changes its policies.

Chapter 3: The LRRP Architecture

3.1 Overview

The paper "*Call for a Public-Domain SpeechWeb*" [Frost05] shows that "the technology is available to build a public-domain SpeechWeb, and that if existing architectures for distributed speech processing are augmented with a slight variation of one of them, then all that remains is required participation of the public."

In this chapter, I briefly discuss some existing architectures, and then introduce a well-suited architecture for a public-domain SpeechWeb which is proposed in [Frost05].

3.2 Existing architectures

In the last several years, speech applications have been used in some areas, such as call centers, website commercial advertisements, among others. Different architectures and technologies have been developed to provide speech interaction between users and distributed speech applications.

The following discussion of the architectures and their shortcomings follows in [Frost05], and summarized in [Su05]:

3.2.1 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 [HT95]. More sophisticated versions process the downloaded web pages and provide spoken summaries and allow some limited form of content querying.

However, the architecture has four shortcomings as listed in [Frost05]:

1. Most HTML pages are constructed for visual browsing, and pages are usually structured to take advantage of the users' ability to scan the page in two dimensions.
2. Keyword and phrase-matching techniques are not well-suited to speech access owing to the large number of words and phrases that could be input by the user. This results in low speech-recognition accuracy.
3. The architecture does not facilitate the development of applications in which a computation is required in response to a user's spoken input.
4. Speech-grammar is unable to be derived from the HTML page directly. Developers are difficult to implement.

3.2.2 Networks of Hyperlinked VXML Pages.

The second architecture involves the use of *networks of hyperlinked VXML pages*. VXML [Lucas00] 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.

However, two factors needs to be considered:

1. Many speech applications require some degree of natural-language processing as it is difficult for users to speak formal languages such as SQL. Natural language processors and their associated knowledge sources can be very large and are best executed on high-powered remote servers rather than VXML browsers running on lightweight end-user devices.

2. Developers should be able to create their speech applications in whatever language they like and not have to embed their application in a VXML page.

3.2.3 Telephone Access to Remote Speech Applications

This architecture is often used in today's call centers. Users phone in and speak to applications running on the center server machines. Speech recognition and application processing is carried out at the call center. VXML is emerging as the standard for implementing call-center applications. Some recent technologies use software to translate VXML pages that are stored on the call-center web servers to speech dialogues that are accessed by remote client telephones.

However, as [Frost05] noted, the call-center architecture has three limitations that constrain its use as basis for the development of a public-domain SpeechWeb:

1. In order to obtain adequate speech-recognition accuracy for applications with large input languages, user voice profiles would have to be stored at each call center, or stored at the user site and transferred to the call center every time that the user contacts a remote speech service. Neither of these options is practical owing to the fact that voice profiles are very large.
2. Application providers have to employ specialized software to allow their applications to be accessed from remote telephones.
3. Call centers have to provide speech-recognition capabilities including maintenance of user voice profiles for sophisticated applications.

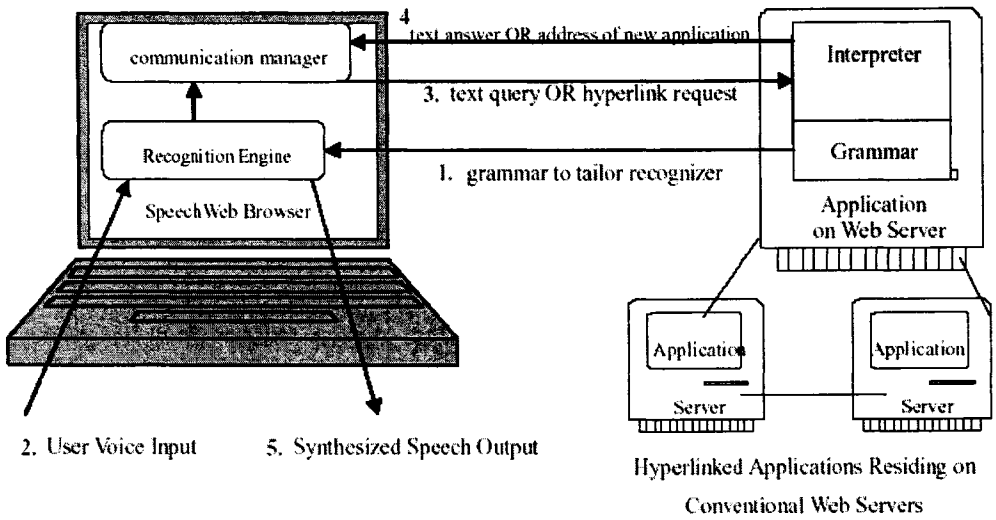
3.3 The LRRP architecture.

Given the physical location for doing speech-recognition and data-processing for the existing architectures, we may summarize them into two categories: **L**ocal **R**ecognition and **L**ocal **P**rocessing (**LRLP**), and **R**emote **R**ecognition and **R**emote **P**rocessing (**RRRP**).

Frost [FAB04] has proposed a new architecture which is claimed to be well-suited for a public-domain SpeechWeb. It is called **L**ocal **R**ecognition and **R**emote **P**rocessing (**LRRP**) architecture.

As described in [Frost05]: "In the LRRP architecture, speech applications and their associated recognition grammars are stored on regular web servers. Speech browsers, that include a speech-recognition engine, reside on PCs or lightweight end-user devices. **1.)** When a user first accesses a remote speech application through the speech browser, a speech-grammar is downloaded to the local device, and used to tailor the speech recognizer for that application. **2.)** When a user speech utterance is recognized to a text string by the speech-recognition engine, **3.)** it is sent to the remote server application for processing. **4.)** The result is returned to the local device, **5.)** and output in synthesized voice. Speech recognition is local and application data processing is remote. When a user asks to be connected to another speech application, the web address of that application is returned to the browser which begins by downloading the new speech-grammar. "

Figure 1. the structure of the LRRP architecture for SpeechWeb:



(The numbers indicate the process order in a simple speech interaction.)

Figure 1. The LRRP SpeechWeb architecture [FAB04]

3.4 Advantages of LRRP architecture.

The LRRP architecture is designed to cover shortcomings in the existing architecture to build a public-domain SpeechWeb. As described in [Frost05], the LRRP architecture has the following advantages:

1. Service providers can maintain their applications on conventional web (Internet) servers.
2. Applications can be written in any language provided that input and output conform to the web communication protocol being used.
3. Non-experts can create simple applications as scripts that return canned answers to user queries, whereas more advanced developers can create complex applications that make use of natural-language processors and database residing on powerful server-side machines.
4. Client-side speech-recognition accuracy is improved through the downloading of application-specific grammars and the use of locally maintained user voice profiles.

5. Can be readily integrated with the conventional web: speech applications reside on conventional web servers, communication uses conventional web protocols, and client-side speech browsers could be implemented as stand-alone interfaces on lightweight devices or as plug-ins to conventional web browsers running on PCs.
6. Communication is through text, which is efficient and appropriate when users access the SpeechWeb through a commercial wide-area wireless network.
7. SpeechWeb Browsers can execute on ultra-lightweight devices.

All that is required for LRRP architecture is two components residing on end-user devices: a grammar-based speech-recognition engine and a dialog manager (speech browser), both of which control human-machine speech interaction, communication of text to remote severs, and transfer from one remote speech-application to another.

Chapter 4: Problems in the Previous VXML SpeechWeb Browser

4.1 Overview

VoiceXML (VXML) has been around for several years. VXML 1.0 was released on May 7th, 2000 [W3C00] and VoiceXML 2.0 was first released on October 23rd, 2001 [W3C01] and recommended by W3C on March 16th, 2004[W3C04]. VoiceXML 2.1 was released recently on the VoiceXML Forum, but is still working as a draft in W3C. VXML has become a standard language for building speech applications, and is often used in call-centers.

In the past 2 years (2004-2005), Frost and Su have conducted research on using VXML to build a speech browser for a public-domain SpeechWeb. They have successfully built a speech browser as a single VXML page together with three Java objects, which is freely downloadable from their website (http://sol.cs.uwindsor.ca/~speechweb/oldversion.htm). In order to distinguish from other versions of the SpeechWeb Browser, it is called VXML-Java version in this thesis report, since it requires Java objects to support it.

This VXML-Java version SpeechWeb Browser has been successfully tested, with distributed sample speech-applications on PCs and laptops using wired and wireless connections to local and wide-area networks [Frost05].

The VXML-Java version SpeechWeb Browser takes advantage of all of the benefits of VXML including the ready availability of VXML interpreters for a wide range of devices, and improvements in speech-recognition technology that are being quickly integrated into new versions of VXML interpreters.

4.2 The VXML-Java SpeechWeb Browser structure.

Su [Su05] has investigated three different approaches before coming to the final one. He was trying to achieve the dynamic change of speech-grammars in a single VXML page, which is a very difficult technical problem in VXML version 2.0 and earlier. The three approaches are: 1) Using multiple VXML forms in the single VXML page. 2) Rewrite the speech-grammar file and reload it. 3) Using subdialogs in VXML pages. Unfortunately, all of these three approaches failed. This problem has resulted in the final design of the browser which must use three Java objects to support it.

The following is the detailed structure of the VXML-Java version of SpeechWeb Browser described in [Su05].

"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.
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. 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. 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."

4.3 Shortcomings of using VXML to build a SpeechWeb Browser

There are a number of shortcomings in using VXML to build a SpeechWeb Browser.

1. VXML version 1.0/2.0 does not support a dynamic change of speech-grammars. In order to achieve this goal, Su had to use other

components to support the browser, such as a database grammar system or Java objects. In the VXML-Java version of SpeechWeb Browser, it requires three Java objects, and these Java objects require a JVM environment to run. This means that a Java runtime environment has to be installed in the end-users local device before running the speech browser system. This demands more resources on the device, which is not suitable for handheld devices such as cell phones.

2. VXML interpreters are relatively demanding of resources and none are yet available for lightweight end-user devices. Most of these VXML interpreters are designed to work on a high-resource web server machine and provide services over the network, which is suitable for the RRRP architecture (call centers), but not for the LRRP architecture.
3. Free downloadable VXML interpreters are no longer available. A VXML server will cost several hundreds to thousands of dollars depending on the facilities. It is impossible to install a copy for every user.

The above shortcomings have greatly constrained the ability of VXML in building a LRRP speech browser for SpeechWeb, and make the VXML-Java LRRP speech browser impossible to deploy in lightweight devices.

Chapter 5: A Novel Use of X+V to Create a SpeechWeb Browser

5.1 Introduction to X+V

The X+V stands for **X**HTML+**V**oice, which is a markup language for developing multimodal applications. It has been developed by researchers from IBM, Opera Software, and Motorola, and is designed for building web clients to support visual and spoken interaction. The first X+V version 1.0 was approved by W3C at Dec. 21, 2001 [XV01]. The latest version is 1.2.

X+V combines XHTML and a subset of useful elements from VoiceXML 2.0. XHTML is essentially HTML 4.0 adjusted to comply with the rules of XML, which is the current standard for building Web pages. VoiceXML was one of the first XML-based languages developed in W3C, which provides an easy, standardized format for building speech-based applications. In X+V, the subset of elements from VXML gives it a great ability to do some basic control of the speech interaction between the user and application.

X+V version 1.1 is supported by two freely available multimodal browsers -- Opera and NetFront, which both have a build-in speech-recognition engine from IBM. Some versions of Opera and NetFront are supported in lightweight devices such as Sharp Zaurus 5500 - 5600. The X+V interpreters in Opera and NetFront, have relatively low demands on resource compare to a complete VXML interpreter, and are available to be deployed into mobile devices as the mobile provider requested, which is the greatest advantage of using X+V.

Multimodal access is the ability to combine multiple modes or channels in the same interaction or session. The methods of input include speech recognition,

keyboard, touch screen, and stylus. Depending on the situation and the device, a combination of input modes will make using a small device easier [IBM]. X+V is designed for building multimodal applications.

Section 4.3 has discussed that VXML cannot be used to build a viable Public-Domain SpeechWeb Browser. But by using X+V, with the ability of developing multimodal access applications, support from low-resource demand and completely freely available interpreters (Multimodal browsers), it is possible for us to build one.

5.2 Limitations of X+V

X+V is a useful language to build multimodal applications. It seems, with the multimodal access ability, X+V should be already widely known and used by today. However, a search for X+V papers on the Internet will find only few groups of people who are using it. Why has X+V not been widely used by current developers especially considering how quickly they adapt to new technologies? Some problems in difficulties of using X+V could be one reason.

The following is a list of limitations of X+V I have found:

Limitation 1. Some important VXML elements are not supported in X+V, such as the VXML 'goto' element.

Limitation 2. All voice dialogs in X+V are executed in a pre-defined sequence order. There is no available build-in element to perform an iterated dialog-control action. The recursive method is also not available in X+V.

Limitation 3. X+V supports JavaScript in coding, but there are some conflicts in use of JavaScript, VXML elements, and XHTML objects together. Some code normally works fine in regular

JavaScript-XHTML environment, but has problems in JavaScript-XHTML-VXML (X+V).

Limitation 4. X+V only supports a few types of speech-grammar languages.

Limitation 5. There is a lack of documentation, and there is no well-organized forum for public X+V application developers on the Internet.

5.3 New solution to some of the X+V limitations

In Section 5.2, a list of problems in the current version of X+V has been given. However, some of these problems are possibly solvable. The following are my solution to some of these limitations:

5.3.1 Limitation 1: Some important VXML elements are missing, such as 'goto' element.

Some of the missing elements from VXML can actually be replaced by other elements directly. Even for those ones which cannot be replaced, we can still try to achieve the same goal and functionality in logic by combining use of other elements and tools.

The 'goto' element is one of the most useful elements in VXML that are missing from X+V. In VXML 2.0, it is used to 1) transition to another form item in the current form, 2) transition to another dialog in the current document, or 3) transition to another document. It also normally used to perform an iterated dialog-control.

The following is my solution to achieve the same goal as using the 'goto' element.

Solution:

The solution can be separated into 2 situations: transition to another form/dialog in the current document, and transition to another document.

Situation 1: Go to another dialog within the same document

The solution uses the *'reprompt'* element as the key, and uses *'throw'* and *'catch'* elements to perform an event control which handles the goto action. Notice, the *'reprompt'* element will push out the next available voice message from the circular-queue, which will force the VXML parser go to the dialog (which has the next available voice message). To do this:

1. Declare a boolean variable for each dialog field that you have, and use it in the *'cond'* attribute. You can either declare these variable in a JavaScript or VXML code, since javascript variables shared with VXML in X+V. Also please note that for each dialog there, it must contain an available *'prompt'* voice message element used at the beginning, but the message could be empty.

In sample code 1, there are three voice dialogs -- dialogA, dialogB, dialogC in the document, and their corresponding declared variables -- 'runA', 'runB', 'runC'.

Please note that all necessary coding statements in the current step of the solution are highlighted in the sample code.

```
<script type="text/javascript>
   var runA=true;
   var runB=true;
   var runC=true;
</script>
.....
<vxml:field name="dialogA" cond="runA==true">
   <vxml:prompt>messageA</vxml:prompt>
   .....
</vxml:field>

<vxml:field name="dialogB" cond="runB==true">
   <vxml:prompt>messageB</vxml:prompt>
   .....
</vxml:field>

<vxml:field name="dialogC" cond="runC==true">
   <vxml:prompt>messageC</vxml:prompt>
   .....
</vxml:field>
.....
```

Sample code 1, 'goto' another dialog within the same document, step 1.

In sample code 1, three boolean variables -- *runA, runB, runC* are declared and assigned to be *true* in a JavaScript. And a conditional check on them is done for every VXML dialog (VXML '*field*' element).

2. Use *'throw'* element to throw an event at where 'goto' element would normally be used, and catch this event in the upper level VXML form. Notice, any event name can be used, but suggested to be '*goto.DialogName*' since it has clearly indicated the event-action purpose.

For example: After processing the dialogA, the program want to jump to dialogC, instead of going to dialogB then dialogC in the sequential order. Then

add the event throw and catch coding statements to the program as in the sample code 2:

```
<script type="text/javascript>
   var runA=true;
   var runB=true;
   var runC=true;
</script>
.....
<vxml:field name="dialogA" cond="runA==true">
<vxml:prompt>messageA</vxml:prompt>
   .....
   <vxml:filled>
      <vxml:throw event="goto.dialogC"/>
   </vxml:filled>
</vxml:field>

<vxml:catch event="goto.dialogC">
.....
</vxml:catch>

<vxml:field name="dialogB" cond="runB==true">
<vxml:prompt>messageB</vxml:prompt>
   .....
</vxml:field>

<vxml:field name="dialogC" cond="runC==true">
<vxml:prompt>messageC</vxml:prompt>
   .....
</vxml:field>
.....
```

Sample code 2, 'goto' another dialog within the same document, step 2.

In sample code 2, an event – *'goto.dialogC'* is thrown from the *dialogA,* and it is caught in upper lever VXML form.

3. Within the catch event block, set the values of the dialog corresponding conditional boolean variable to false for all the dialogs located before the *'goto'* destination dialog, but set to true for itself. Then use *'clear'* element to clear the *'goto'* destination dialog if it has been processed before. Finally, use *'reprompt'* element to go to the dialog.

After the step 3, the code would be written as in sample code 3:

```
<script type="text/javascript>
   var runA=true;
   var runB=true;
   var runC=true;
</script>
.....
<vxml:field name="dialogA" cond="runA==true">
<vxml:prompt>messageA</vxml:prompt>
   .....
   <vxml:filled>
      <vxml:throw event="goto.dialogC"/>
   </vxml:filled>
</vxml:field>

<vxml:catch event="goto.dialogC">
   <vxml:assign name="runA" expr="false"/>
   <vxml:assign name="runB" expr="false"/>
   <vxml:assign name="runC" expr="true"/>
   <vxml:clear namelist="dialogC"/>
   <vxml:reprompt/>
</vxml:catch>

<vxml:field name="dialogB" cond="runB==true">
<vxml:prompt>messageB</vxml:prompt>
   .....
</vxml:field>

<vxml:field name="dialogC" cond="runC==true">
<vxml:prompt>messageC</vxml:prompt>
   .....
</vxml:field>
.....
```

Sample code 3, 'goto' another dialog within the same document, step 3.

In the sample code 3, the variables – *runA, runB* are set to be *false,* so that *dialogA* and *dialogB* won't be able to available to run. But the variable – *runC* is assigned to be *true*, and the *dialogC* will be the only available dialog to process.

Situation 2: Go to the dialog in another document.

The solution to this situation is a little more complicated, we have to pass variables to another document. The key used to solve this problem is JavaScript and cookie objects.

1. Call a JavaScript function to save the dialog name (which is located in another document) into a cookie before going to the next document.

In the following sample code 4, the JavaScript function *gotoDialog()* has saved the dialog name(*dialogC*) into a cookie named *'gotoDialog',* then load the new page – *voicePage_next.xml.*

```
<script type="text/javascript>
  var runA=true;

  function gotoDialog()
  {
    setCookie("gotoDialog", "dialogC", expireTime);

    /* Loads the next document. */
    window.location="voicePage_next.xml";

    return "";
  }

  function setCookie(c_name,value,expiredays)
  {
    var exdate=new Date();
    exdate.setDate(expiredays);
    document.cookie=c_name+ "=" +escape(value)+ ((expiredays==null) ? "" : ";
                    expires="+exdate);
  }
</script>
.....
<vxml:form>
<vxml:field name="dialogA" cond="runA==true">
<vxml:prompt>messageA</vxml:prompt>
  .....
  <vxml:filled>
    <vxml:throw event="goto.dialogC"/>
  </vxml:filled>
</vxml:field>
.....
<vxml:catch event="goto.dialogC">
  <vxml:prompt><vxml:value expr="gotoDialog()"/></vxml:prompt>
</vxml:catch>
</vxml:form>
.....
```

Sample code 4, 'goto' a dialog within another document, step 1.

2. Within the next page, execute a javascript to load the dialog name from the cookie, and set its dialog running condition check variable to true, but other

dialogs to false. Continue with the previous example, inside of voicePage_next.xml would be as in sample code 5:

```
<script type="text/javascript>
  var runC=false;
  var runB=false;
  var dialogName = getCookie("gotoDialog");
  if(dialogName=="dialogB")
    runB=true;
  else if(dialogName=="dialogC")
    runC=true;

  function getCookie(c_name)
  {
    if (document.cookie.length>0)
    {
      c_start=document.cookie.indexOf(c_name + "=");
      if (c_start!=-1)
      {
        c_start=c_start + c_name.length+1 ;
        c_end=document.cookie.indexOf(";",c_start);
        if (c_end==-1) c_end=document.cookie.length
          return unescape(document.cookie.substring(c_start,c_end));
      }
    }
    return null;
  }
</script>
.....

<vxml:form>
    <vxml:field name="dialogB" cond="runB==true">
    .....
    </vxml:field>

    <vxml:field name="dialogC" cond="runC==true">
    .....
    </vxml:field>
</vxml:form>
.....
```

Sample code 5, 'goto' a dialog within another document, step 2.

5.3.2 Limitation 2: No iterated dialog-control, or recursive methods.

Without a build-in iterated dialog-control element, in X+V, all dialogs will be run in a pre-defined sequential order. The order is the place order they are in the document, dialogs run from the top to the bottom of the document. After it finishes processing the last dialog (the bottom one), all vxml-voice processes are done. This is impropriate when building a more complex application. This limitation of X+V must be solved before X+V can be used.

Solution:

In section 5.3.1, a solution has been provided to achieve VXML *'goto'* element by using other elements and tools. The solution to Limitation 2 becomes quite easy by updating the solution from Section 5.3.1, Situation 1. The following are the steps to achieve iterated dialog-control:

1. Declare a variable to control the entry of loop, set it to false to exit from the loop.

2. After finishing all the processes in the last dialog (the bottom one), perform a true-false check on loop-entry control variable. If it has value true, then throw an event which forces the voice process to go back to the first dialog (the top one).

In sample code 6, three voice dialogs – *dialogA, dialogB, dialogC* – will be iterated running until *'continueLoop'* set to be *false*.

```
<script type="text/javascript>
   var runA=true;
   var runB=true;
   var runC=true;
   var continueLoop=true;
</script>
.....
<vxml:form>
<vxml:field name="dialogA" cond="runA==true">
   <vxml:prompt>messageA</vxml:prompt>
   ...
</vxml:field>

<vxml:field name="dialogB" cond="runB==true">
   <vxml:prompt>messageB</vxml:prompt>
   .....
</vxml:field>

<vxml:field name="dialogC" cond="runC==true">
      <vxml:prompt>messageC</vxml:prompt>
      ...
      <vxml:filled>
         ...
         <vxml:if cond="continueLoop==true">
            <vxml:throw event="loop.dialogA"/>
         </vxml:if>
      </vxml:filled>
</vxml:field>

<vxml:catch event="loop.dialogA">
   <vxml:assign name="runA" expr="true"/>
   <vxml:clear namelist="dialogA dialogB dialogC"/>
   <vxml:reprompt/>
</vxml:catch>
</vxml:form>
.....
```

Sample code 6, solution to achieved an iterated dialog control

In sample code 6, a loop entry variable has been declared in the JavaScript. And a *'loop.dialogA'* event will be thrown when the *dialogC* is about to be finished. The event will be caught in the upper level *catch* statement and force

the process to goto the *dialogA* again.

5.3.3 Limitation 3: Difficulties in using JavaScript, XHTML, and VXML objects together

In X+V, the VXML part has a very strong process control based on JavaScript and XHTML, but JavaScript and XHTML are not really designed for working with this subset of VXML. There are some difficulties to using them together with VXML. In some cases, the program code does not work properly as it normally would when using JavaScript and XHTML alone.

Problem 1:

VoiceXML has no element designed for calling a JavaScript function. Although *'var'* and *'assign'* elements may used to call JavaScript, they can only be used inside of 'filled' or 'catch' elements. They cannot be used to call a JavaScript function directly at the beginning of a dialog.

Solution:

The key to solve this problem is the VXML *'prompt'* element (or *'block'* element) and *'value'* element. The *'prompt'* element is used to output a voice message to the user. It can be used almost anywhere within a VXML form (except within a speech grammar definition). '*value*' can be a child of *'prompt'* element, and it can call a JavaScript in its attribute. Notice, the JavaScript function has to return an empty string, otherwise it will be uttered.

Sample code 7 has provided a call to JavaScript functions – *fun1(), fun2(),* at the beginning of a VXML form(within *'block'* statement); at the beginning of a dialog(the first prompt voice-output within *dialogA*); at the end of a dialog process (within the *'filled'* statement in *dialogA*); and within an event

thrown/catch or subdialog process.

```
<script type="text/javascript>
  function fun1()
  {
    ......
    return "";
  }

  function fun2()
  {
    ......
    return "";
  }
</script>
......
......
<vxml:form>
<block><vxml:value expr="fun1()"/></block>
<vxml:field name="dialogA">
  <vxml:prompt><vxml:value expr="fun2()"/></vxml:prompt>
  <vxml:grammar>...</vxml:grammar>
  <vxml:filled>
    <vxml:prompt><vxml:value expr="fun1()"/></vxml:prompt>
  </vxml:filled>
  <vxml:catch event="help nomatch noinput">
    <vxml:prompt><vxml:value expr="fun2()"/></vxml:prompt>
  </vxml:catch>
</vxml:field>

<vxml:subdialog name="subdialogA">
  <vxml:prompt><vxml:value expr="fun1()"/></vxml:prompt>
  ......
</vxml:subdialog>
</vxml:form>
......
```

Sample code 7, calls a JavaScript function within VXML part

Problem 2:

In X+V, the VXML part can access a JavaScript variable directly, but JavaScript cannot see or access a variable which is declared in VXML. This is a problem because some variables can only be declared in X+V, such as the important VXML 'field' variables.

Solution:

Declare a JavaScript variable which corresponds to the VXML variable, then use the VXML '*assign*' element to pass the value to it.

```
<script type="text/javascript">
  var dialog_AA;
</script>

<vxml:form>
...
<vxml:field name="dialogA">
  <vxml:filled>
    <vxml:assign name="dialog_AA" expr="dialogA"/>
  </vxml:filled>
</vxml:field>
...
</vxml:form>
.....
```

Sample code 8, access a VXML declared variable within a JavaScript code

In sample code 8, the value of VXML declared variable *dialogA* has been passed to a JavaScript variable *dialog_AA*.

Problem 3:

In X+V, when using DOM function (JavaScript) '*innerHTML*' to assign any

HTML tag (e.g. '
') to another variable, it does not work properly as it does on a regular DOM and XHTML web page. As sample code 9, in X+V, the statement – *objTable.rows[0].cells[0].innerHTML="
"* will not change a line in the table (*objTable.rows[0].cells[0]*), as '*
*' should normally do:

```
<html>
...
<script type="text/javascript">
showMsg("Welcome");
showMsg("<br/>");
showMsg("How are you today?");
showMsg("<br/>");

function showMsg(msg)
{
   var objTable=document.getElementById("textMsg");
   objTable.insertRow(0);
   objTable.rows[0].insertCell(0);

   if(msg== "<br/>")
      objTable.rows[0].cells[0].innerHTML="<br/>";
   else
      objTable.rows[0].cells[0].innerHTML=" ";
}
</script>
.....
<body>
<table id="textMsg">
</table>
</body>
</html>
```

Sample code 9, problem of using *
* tag within an *innerHTML* assignment

Solution:

We have to avoid using 'innerHTML' in this case. In the above example, instead of using *'innerHTML'* to assign '*
*' to '*objTable.rows[0].cells[0]*' object, we should create an '*br*' object, and then append it to '*objTable.rows[0].cells[0]*' as a child. Sample code 10 has highlighted the code to avoid use *innerHTML*

statement.

```
<script type="text/javascript">
showMsg("Welcome");
showMsg("<br/>");
showMsg("How are you today?");
showMsg("<br/>");

function showMsg(msg)
{
   var objTable=document.getElementById("textMsg");
   objTable.insertRow(0);
   objTable.rows[0].insertCell(0);

   if(msg== "<br/>")
   {
      var objBR = document.createElement("br");
      objTable.rows[0].cells[0].appendChild(objBR);
   }
   else
      objTable.rows[0].cells[0].innerHTML=" ";
}
</script>
.....
<body>
<table id="textMsg">
</table>
</body>
.....
```

Sample code 10, solution to avoid use of *
* tag within an *innerHTML* assignment

Problem 4:

In X+V, the only function which supports a dynamic speech output is '*<prompt src="#_id" />*'. But the source of the speech has to point to an existing HTML tag in the file which has the *id*, such as '* data *', and most likely the speech data will be shown on the screen and visible to the user. It cannot point to a variable directly.

Solution:

Point to a JavaScript variable indirectly by the following:

1. Do not use '*<prompt src="#_id"/>*, but regular '*<prompt></prompt>*', and keep inside of it empty.
2. Use '*<value expr="var_name">*' to point to a variable, and use it inside (as a child of) the above '*<prompt></prompt>*'.

In sample code 11, the voice-output message from the *prompt* is linked to a JavaScript variable – *output.*

```
<script type="text/javascript">
var output="hello";
</script>
<vxml>
.....
<prompt><value expr="output"/></prompt>
.....
</vxml>
```

Sample code 11, solution to dynamic voice-output message

Problem 5:

The voice output from VXML cannot suspend the processing of JavaScript, XHTML, and even VXML itself. They all run simultaneously with the voice output. This is good in efficiency in the most of situations. But, consider this situation, we want to output a voice to the user first, then go to another document. Unfortunately, you will find out that, you can only hear the beginning part of the voice output, which is then interrupted because Javascript has called to load another document.

In a program written as sample code 12, the user will not be able to hear a complete voice-output sentence: "*We are going to next page":* The program calls a JavaScript function – *loadNext("nextpage.xml")* to load a new page before finishing the VXML voice-output process.

```
...
<vxml:form>
<vxml:field name="dialogA">
  ...
  <vxml:filled>
    ...
    <!—Comments: Speech output -->
    <vxml:prompt>We are going to next page</vxml:prompt>
    <!—Comments: Call JavaScript function to load nextpage.xml -->
    <vxml:value expr="loadNext('nextpage.xml')"/>
  </vxml:filled>
</vxml:field>
...
</vxml:form>
...
<script type="text/javascript">
  function loadNext(nextPage)
  {
    window.location=nextPage;
    return "";
  }
</script>
.....
```

Sample code 12, complete VXML process before going to another page

Solution:

The key of this problem is to use the VXML event – "*vxmldone*", which is the only one can be catch by XHTML listener. "*vxmldone*" is an event generated automatically by the X+V interpreter when the whole VXML part, which including voice output process, is done.

Solution steps:

1. Using XHTML "*ev:listener*" object to register the Javascript for handling the "*vxmldone*" event. As the above example, this Javascript will call to load the next document (nextpage.xml).

2. Throw an exception to exit from all VXML dialogs, which will generate the '*vxmldone*' event automatically by X+V interpreter, instead of calling Javascript directly right after the voice output element '*<prompt>*'.

Sample code 13 is the solution code to solve this problem. The program would throw a self-declared VXML event (*finish.all)* in the case that it wants to call a JavaScript function to go to another page, but do nothing (exit from all VXML dialogs) in its catch statement. Then, use a JavaScript to catch X+V build-in VXML event – *"vxmldone"*, and to go to another page.

```
<html>
...
<vxml:form>
<vxml:field name="dialogA">
  ...
  <vxml:filled>
    ...
    <vxml:prompt>We are going to next page</vxml:prompt>
    <vxml:throw event="finish.all"/>
  </vxml:filled>
</vxml:field>
...
<vxml:catch event="finish">
  <!-- Do NOT need to do anything in here! -->
</vxml:catch>
...
</vxml:form>
...
<script type="text/javascript" id="gotoNextPage" declare="declare">
  window.location = "nextpage.xml";
</script>
.....
<body id="page.body">
...
<ev:listener ev:observer="page.body" ev:event="vxmldone" ev:handler="#gotoNextPage"
ev:propagate="stop" />
...
</body>
</html>
```

Sample code 13, solution to complete VXML process before going to another page

5.4 Conclusion of X+V usability

X+V has several limitations that make it difficult to use for application developers. However, the solutions in Section 5.3 show, these limitations are fixable, and do not reduce the great value of X+V. It is still the best tool to build a speech browser for a Public-Domain SpeechWeb.

Chapter 6: Investigation of the Use of X+V for a SpeechWeb Browser

6.1 Overview

We have investigated several approaches toward using X+V to build a speech browser for a Public-Domain SpeechWeb. At the beginning, we were trying to build the browser as a single X+V page. Unfortunately, the Single-Page design did not work properly in all situations because of a concurrency problem that is beyond the control of X+V application developers.

From the problem which appeared in the implementation of the Single-Page design, we have figured out that based on today's technologies, it is impossible to build a SpeechWeb Browser as a single X+V page which works reliably in all situations. Consequently, we developed a Multiple-Page design.

We begin with a short description of the Single-Page design followed by a detailed description of the successful Multiple-Page design.

6.2 Single-Page Design

The most difficult problem in building a SpeechWeb Browser as a single X+V page is the runtime dynamic change of speech-grammars. A single page SpeechWeb Browser has to change the speech-grammar dynamically everytime it connects to another remote application. After investigation and questions to the Opera company, there is a fact that the Opera X+V interpreter only parses the static VXML parts of X+V once, including the speech-grammar, and it is at the first time it loads the page. After this one-time load, the speech-grammar is parsed and saved into flash memory. Everytime the dialog

needs the speech-grammar, the X+V interpreter just goes to the memory and picks it up directly without looking at the document page again. Looping through a single page cannot change the grammar, even though the attribute of grammar element has been reset. This leaves no choice but to save the changes (new speech-grammar file location) somewhere (client's cookies) and reload/refresh the whole X+V page, then try to change the VXML data (the attribute of the grammar element) before the interpreter loads it.

6.2.1 Components and workflow of a Single-Page Design SpeechWeb Browser

A Single-Page design SpeechWeb Browser system has following components:

1. A single X+V page which contains following subcomponents:
   a) One iterated VXML dialog to recognize human-voice-input and to give synthesized-voice-output.
   b) XHTML objects to provide visual access to the browser for users.
   c) JavaScript functions to perform data checking, logical processing, control of cookies, and AJAX-technology powered Internet communication. Both VXML part and XHTML part performances greatly rely on these JavaScript functions.
2. Remote applications distributed over the SpeechWeb. Each remote applications contains the following four subcomponents:
   a) The knowledge source of the application; it could be program(s) or knowledge/data base system(s).
   b) A Common Gateway Interface (CGI) script to handle the communication between the local server application knowledge sources and remote user single X+V page.
   c) A speech-grammar file for the application.
   d) A small '.sihlo' file, which provides the basic information about the application: 1) a greeting message of the application as text string, 2)

the location of the CGI script, and 3) the location of the speech-grammar file.

3. A high speech-recognition accuracy, low-resource demands, completely freely available Opera X+V interpreter from Opera Software, which is the pre-required environment to run the browser.

Briefly, a Single-Page SpeechWeb Browser works as follows:

1. The user loads the browser in Opera (just like loading an HTML page in Netscape), and connects to a remote application possibility distributed over the SpeechWeb. JavaScript functions will validate the input and network communication to avoid a cross-domain security error from Opera.
2. The browser downloads the speech-grammar from a remote server, and uses it to tailor the interaction with the user. The user's voice-input is recognized and transmit to a text string locally by the speech-recognition engine that is build-in within Opera. Using the latest network communication technology – AJAX, the text string is sent to the remote application to do the data processing, and retrieve returned data and provide it to the user using synthesized-voice-output. Both speech-recognized user's input and remotely returned synthesized-voice-output are displayed on the browser page, which is clearly visible to the user.
3. When the user requests connecting to another remote application, the browser will save the Internet location of the new application and its speech-grammar files into cookies, then reload the browser and connect to the new application.
4. When reloading the browser, JavaScript changes the VXML grammar attribute right after the page document load, and tries to complete this task before the X+V interpreter parsed the VXML part of X+V. Then the whole process goes to step 2 again.

5. The user can terminate the session at anytime by closing the browser page on Opera

6.2.2 Problems in the Single-Page Design

As described in the workflow step 4 from section 6.2.1, we can see a concurrency problem in the Single-Page Design SpeechWeb Browser. Both the JavaScript code and VXML part interpreter will automatically start running right after the page document is loaded. If JavaScript successfully makes the change before the VXML part is parsed by the interpreter, then the process works as desired. Otherwise, the browser will load the speech-grammar file from the incorrect location (actually, it will try to load it from the browser pre-defined default location).

6.2.3 Single-Page Design conclusion

Unfortunately, Single-Page Design has failed to work in all situations. Even though the possibility of its success may be increased by a good order of the location of the VXML part code in the X+V document, but it still related to other uncertain factors in Opera and X+V. The result of the Single-Design is unacceptable.

6.3 Multiple-Page Design

During the investigation of the Single-Page Design, we have found out that by today's technology, it is impossible to build a speech browser for a Public-Domain SpeechWeb as a single X+V page. Therefore, we investigated a Multiple-Page design:

Instead of using a single X+V page, and dynamically changing the value at runtime for all application dependent variables, we are going to have a copy of

the X+V page associated with and located with each remote application. The size of the copy of the X+V page is less than 18.5KB. Most of the data on these copies of the X+V page are identical to each other except 3 lines which give the greeting message, the Internet location, and the speech-grammar file location for the associated application.

6.3.1 Components of Multiple-Page Design SpeechWeb Browser

A Multiple-Page Design SpeechWeb Browser contains the following components:

1. An optional browser menu X+V page located in the user local device, which can help the user to connect to a remote application. Users may also connect to the remote application associated X+V page directly if they know its location (URL address). This optional browser menu X+V page contains the following subcomponents:
   a) One iterated VXML dialog to recognize human-voice-input and to give synthesized-voice-output.
   b) XHTML objects to provide visual access to the browser for the user if required.
   c) JavaScript functions to perform data checking, logical processing, control of cookies, and AJAX-technology powered Internet communication. Both VXML part and XHTML part performances greatly rely on these JavaScript functions.
2. Remote applications distributed over the SpeechWeb. Each remote application contains four subcomponents:
   a) The knowledge source of the application. It could be a program or a knowledge/data base system.
   b) A Common Gateway Interface (CGI) script to handle the communication between the local server application knowledge sources and remote user single X+V page.

c) A speech-grammar file for the application.

d) A single copy of the identical X+V browser page with the three lines changed on it for the application. These X+V pages for remote applications are very similar to each other except that each containing 3 specific information about the application: 1) a greeting message for the application as a text string, 2) the location of the CGI script, and 3) the location of the speech-grammar file. Other identical subcomponents are similar to the Single-Page Design (section 6.2.1) version, except that there are no cookies used; and no data requesting on '.sihlo' file when connecting to a new application. The file size is less than 18.5KB.

3. A high speech-recognition accuracy, low-resource demands, completely free available Opera X+V interpreter from Opera Software, which is the pre-required environment to run the browser.

6.3.2 Multiple-Page Design structure

The following is the structure of Multiple-Page design SpeechWeb Browser:

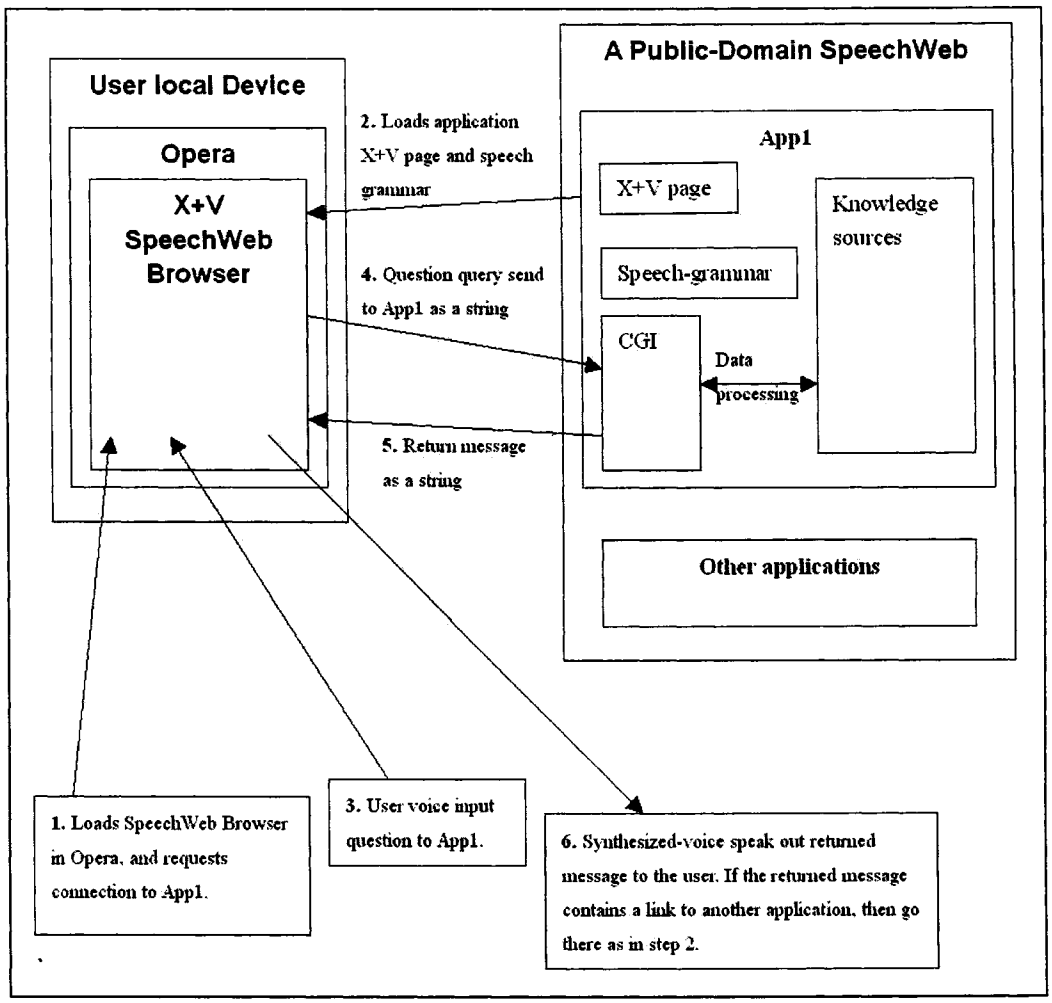


Figure 2. The structure of the Multiple-page Design

A simple Multiple-Page design SpeechWeb session will have following steps:

1. Open Opera. Then either visit the SpeechWeb Browser Menu page or give the location to a remote X+V page to it.
2. It will automatically download the startup X+V page and speech-grammar file to the local device. The user may start talking now.
3. The user gives a question/request to SpeechWeb Browser.

4. The speechWeb Browser will recognize and convert user's voice-input to a string query, and send it to the remote application CGI script.
5. The CGI script will do the data processing and return a response message to the SpeechWeb Browser as a string.
6. The SpeechWeb Browser speaks out the returned message to the user as a synthesized-voice-output.
7. If the returned message contains a link to another remote application, the SpeechWeb Browser will connect to the new application as in step 2. Otherwise, the user can keep asking questions as in step 3.

6.3.3 Comparison of a SpeechWeb Browser built as a single and multiple X+V pages

Our Multiple-Page Design SpeechWeb Browser has successfully passed the performance test, which can provide a reliable service in all circumstances. The following is a comparison of a SpeechWeb Browser building as a single X+V page and as multiple application associated X+V pages:

Table 1: Comparison of Single-Page design with Multiple-Page design

	A single X+V page	***Multiple X+V pages***
General performance	Unreliable.	Reliable.
Required components on user's local device	1. Opera 2. A single X+V page (<21.3KB)	Opera only.
Required components for remote application	1. Knowledge sources. 2. CGI script. 3. Speech-grammar file. 4. A ".sihlo" file. (<1KB)	1. Knowledge sources. 2. CGI script. 3. Speech-grammar file. 4. An application associated X+V browser page. (<18.5KB)
Network communication(s) when connecting to a new application	1. Retrieves data from remote ".sihlo" file. 2. Reloads and connects to the new application.	One step only: Connects and loads the new application X+V page directly.
Client Cookies	Required	Not required.

From the comparison of the single X+V page design and the multiple application-associated X+V pages design, it has clearly shown that the Multiple-Page Design is a better design to build SpeechWeb Browser, except that the copy of the X+V browser page has to be downloaded each time the user hyperlinks to a new application. However, the page size is less than 18.5KB, so the download time is negligible.

6.3.4 Multiple-Page Design conclusion

Our Multiple-Page Design SpeechWeb Browser has successfully passed all performance tests (See chapter 8). It has accomplished everything we wanted it to do. The Multiple-Page Design was successful and allowed the new SpeechWeb Browser to execute on end-user devices with only one requirement – that the free Opera be installed.

Chapter 7: The New SpeechWeb Browser

7.1 Overview

Our new SpeechWeb Browser is built based on Multiple-Page design (section 6.3). The new SpeechWeb Browser can execute on end-user devices with only one requirement – that the free Opera be installed.

We now describe the new browser in detail.

7.2 The end user interface

The new SpeechWeb Browser takes the advantage of multimodal access, which allows users' access through both XHTML graphical user interface and VXML speech interface.

After using Opera to load the SpeechWeb browser menu page, sighted users can see a graphical interface as regular conventional website. However, visual access is not required as all functionality can also be achieved through speech.

Figure 3. GUI of the SpeechWeb Browser Menu Page.

At the same time, the browser will speak a voice introduction: "Please say the name of the Sihlo that you wanna talk to, or input the URI for your own SpeechWeb application."

Users may choose a sample application (Sihlo) to connect to by either clicking on the link or just saying it. They could also load any other SpeechWeb application by typing the URL into the "Load your own speech application:" text field. We have not yet provided the ability to "speak" a URL for a starting application.

After connecting to a remote application, a Graphical User Interface will be loaded. And the greeting message from the application will be both said and displayed to the user. The following is the GUI shown after connected to one of the sample application (Judy). The greeting message "hi, my name is judy" is

spoken and displayed to the user at the same time.

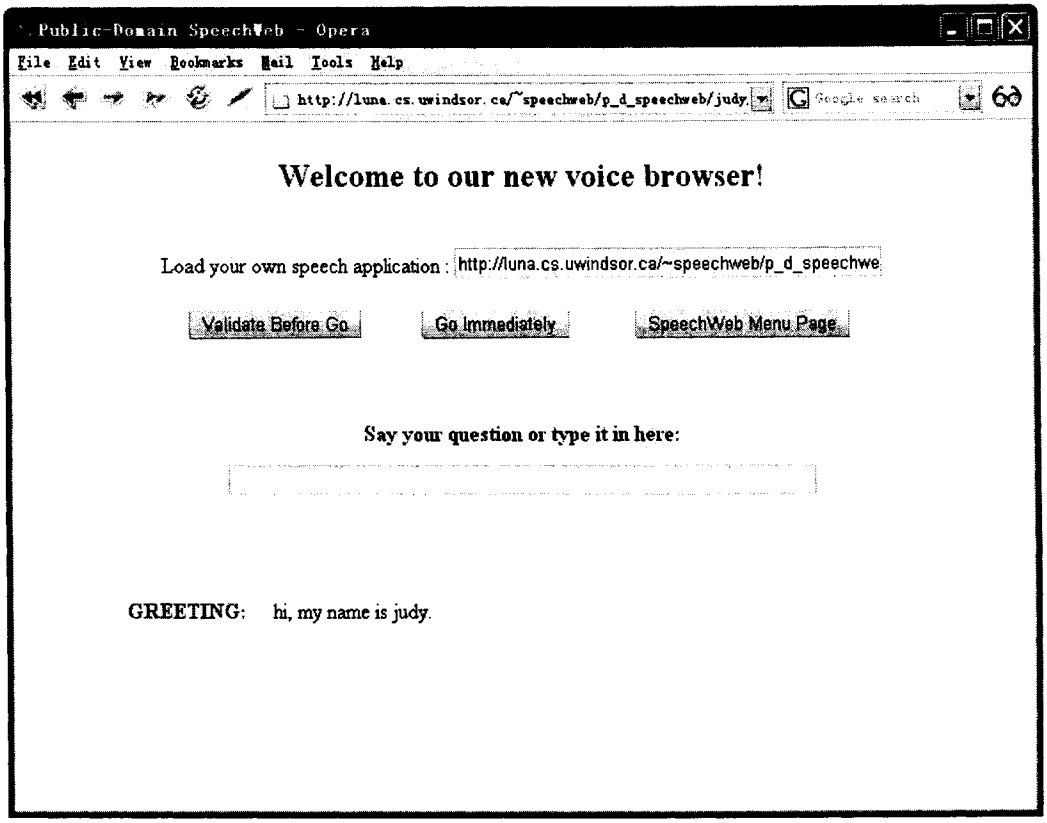


Figure 4. "Judy" is loaded

Now, users can start the conversation. They can ask their question by speaking it, or by typing it into "say your question or type it in here:" text field. Either way, the input question query will be displayed on the page and sent to the remote Judy application immediately. The response message returned from the remote application will also be displayed on the page. The following is a sample conversion with Judy:

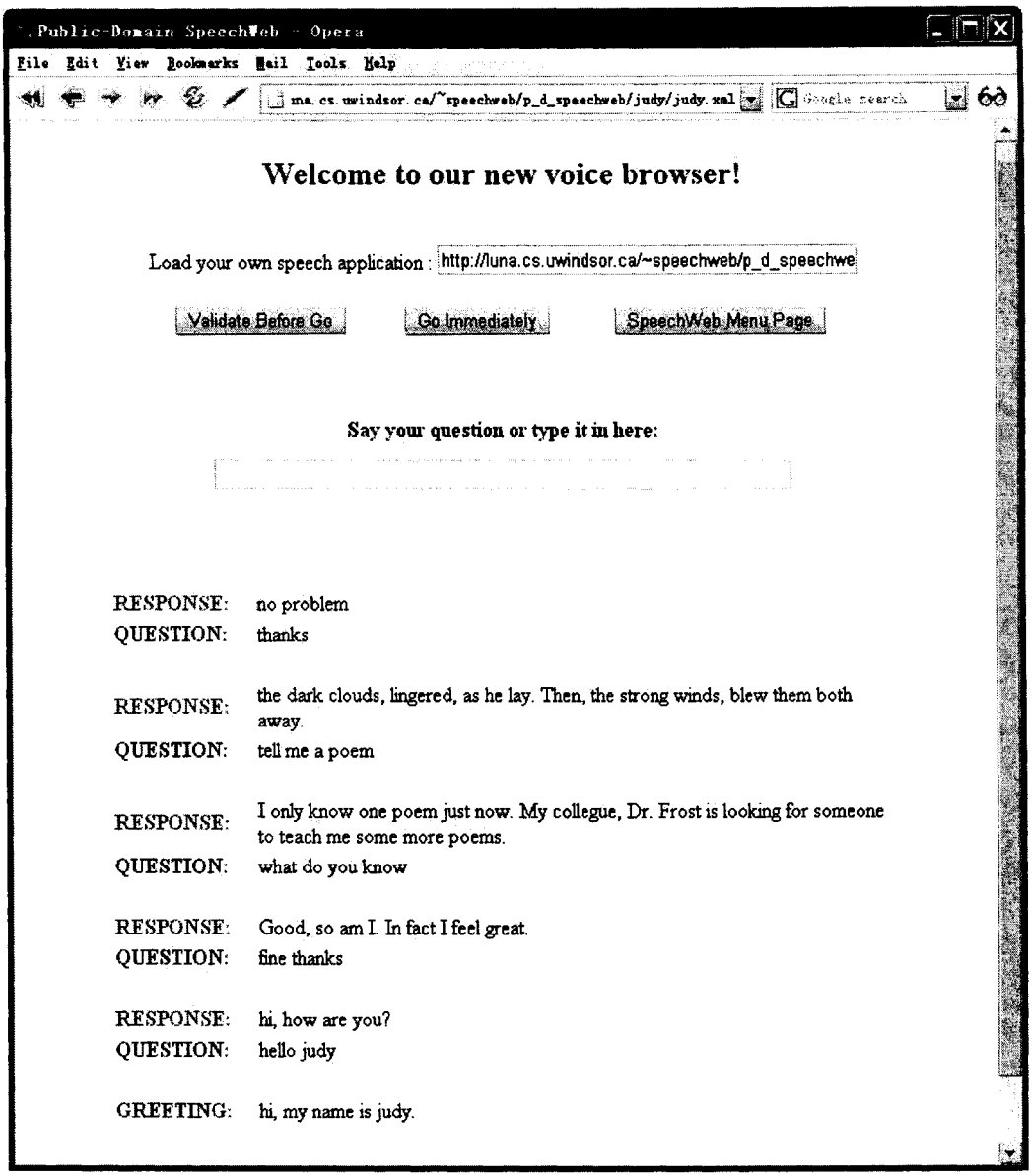


Figure 5. A Sample Conversation with Judy.

Note that the order displayed above is the reverse time-order of the session due to the fact that the page scrolls down.

7.3 A sample session user input/computer response

The following is another sample session user input / computer response with the SpeechWeb, which involves transition from one application to another:

Please note that, as we mentioned before, if the user knows the X+V page URL location of a remote speech application (Sihlo), then he/she can connect to it directly by type the URL into Opera, without going through the browser menu page. Note that this session is described in an ascending time order.

The user begins by opening Opera, and loads the SpeechWeb Browser menu page.

Browser menu page: **Please say the name of the sihlo that you wanna talk to. Or input the URL. for your own speechweb application.**
Browser menu page introduction message is spoken to the user right after it is loaded.

User: **I wanna talk to Monty.**
The user's voice input is recognized by menu page speech-grammar.

Browser menu page: **You are transferring to Monty now.**
The location of the X+V page for Monty is pre-stored in the browser menu page, so connects to it immediately.

Monty Sihlo: **hi, I am Monty, I know a joke.**
Greeting message from Monty is spoken to the user right after its X+V page is downloaded (The speech-grammar file comes with the X+V page).

User: **how old are you**
The user's voice input is recognized using the Monty speech-grammar and send it to remote Monty Sihlo.

Monty Sihlo: **Eighteen or nineteen. But I am not sure if that is years or days.**
Response message from Monty Sihlo is returned and spoken to the user.

User: **..... (No speaking.)**
User gives no input.

The browser: **Sorry, I don't understand, can you say it again?**
The browser catches no-input event, and output a message to the user.

User: **tell me a joke.**

Monty Sihlo: **Did you hear 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.**

User: **can I talk to solar man**
User requests a transfer to another Sihlo (remote application) called solar man. This change is similar to following a hyperlink on a regular web page.

Monty Sihlo: **yes, here he is.**
When the user requested a transfer to another Sihlo, the response returned actually contains two parts: 1) the response message that will be spoken to the user ("yes, here he is"), and 2) a link to the new Sihlo's X+V page, which is used to connecting to the new Sihlo right after spoken the response message.

Solarman Sihlo: **hi, I am solar man**

User: **what do you know**

Solarman Sihlo: **Not much I am afraid. I am just beginning to learn. I know a bit about the planets, the moons, and the people who discovered them. My master will teach me some more when he gets another grant**

User: **how many moons orbit mars**

Solarman Sihlo: **2**

User: **do you like Chinese food**

The browser: **Sorry, I don't understand, can you say it again?**
The user's input has no match with any choices in the speech-grammar definition. A no-match error is caught by the browser, and a help message is output to the user.

User: who discovered Titania

Solarman Sihlo: **Herschel.**

Etc.

Figure 6, a sample conversation session with SpeechWeb applications

Chapter 8: Analysis of the New SpeechWeb Browser

8.1 Overview

Our goal is to prove a viable speech browser, which meets the SpeechWeb Browser requirements, and can be constructed using X+V. In this chapter, we analyze our new X+V SpeechWeb Browser with respect to the properties required in a Public-Domain SpeechWeb:

1. Clarity in design.
2. Ease of installation and use.
3. Ease of distribution.
4. Use of common communication protocol.
5. Ease of creation and deployment for speech applications.
6. Capabilities.
7. Efficiency and speed.

8.2 Clarity in design

As we described in Chapter 3, the LRRP architecture is ideally suited for building a Public-Domain SpeechWeb. From the comparisons with other existing architectures, it shows that LRRP architecture resolves several of their limitations. By using this architecture in the design, our new SpeechWeb Browser improves performance in efficiency and capabilities.

Another issue in design is whether we should construct the SpeechWeb as a single X+V page or multiple application-associated X+V pages, to achieve the best performance. During our investigation as described in Chapter 6, we have proved that a reliable X+V SpeechWeb Browser can be built as multiple application-associated pages, by demonstration.

8.3 Ease of installation and use for end users

There's only one component required to be installed on the end-user's device -- Opera, which is completely free and downloadable from the Internet. Thanks to the Opera Software company, after downloading Opera to their device, users may install and enable its voice-features in a few simple steps (Please check the SpeechWeb User Manual document [Appendix I] for details.). Any person with basic knowledge of software downloading and installation can do it easily. After installing Opera, users can open our SpeechWeb Browser immediately.

Our new SpeechWeb Browser has provided Multimodal access to SpeechWeb applications. Users may either say or type their question, then hear and see the response message from the remote application. All the interaction processes are done by the browser automatically, including speech recognition, network communication, remote data processing, voice-output & message-display, transition to another application, etc.

8.4 Ease of distribution.

With the benefits of ease of installation and use of our SpeechWeb Browser, plus the complete free available environment (Opera), we can guarantee the ease of its distribution. This is not only for PC users, but also for mobile users. Our current version of X+V SpeechWeb Browser is designed for Windows OS Opera, but it only requires a few changes before deploying it on a mobile version of Multimodal browser (Opera v7, NetFront), which is also free and downloadable from the Internet.

Our new SpeechWeb Browser can be distributed freely as it contains no proprietary components.

8.5 Use of common communication protocol

In our new SpeechWeb Browser, JavaScript AJAX objects are used to implement network communication manager. AJAX is a new technology that allows sending HTTP requests to and receiving responses from a remote server/application without refreshing the page. Our browser currently uses the CGI-BIN protocol to communicate with remote applications. The CGI-BIN protocol is easy to implement, is widely used in most web-based applications, and is supported by most web server software.

8.6 Ease of creation and deployment of SpeechWeb applications

Because our SpeechWeb browser uses the CGI-BIN protocol, the remote application can be deployed on any Internet web server as long as it supports the CGI-BIN protocol. Also, the remote application knowledge-source (data-processing program) can be coded in any language as long as it is able to receive a request and return a response as text string, which means it is able to communicate with its CGI script through the CGI-BIN protocol.

An application-tailored X+V page is also needed for every application. As we described in Section 6.3.1, the X+V pages are identical to each other except each contains 3 specific lines of information about the application: 1) a greeting message of the application as text string, 2) the location of the CGI script, and 3) the location of the speech-grammar file.

Please see Appendix II for more details about how to create your own SpeechWeb application.

8.7 Capabilities

As we described in Chapters 5, 6, 7, our new SpeechWeb browser has following properties:

1. Non-proprietary software.
2. Uses common communication protocol.
3. Free distribution.
4. Possibility of deployment on handheld devices.
5. Ease of installation and use.
6. Ease of implementation for SpeechWeb applications.
7. Multimodal access.
8. Real-time human-machine speech interaction.
9. Acceptable speech-recognition accuracy in a suitable environment.

The only restriction on the browser is that it has to obey the JavaScript cross-domain security issue on Opera. The issue requires that all AJAX network communication must done within same Internet domain, which means the application-associated X+V browser page must be located in the same Internet domain with its associated CGI script file. This restriction is not a big problem since the X+V browser page file size is less than 18.5KB, which is extremely small comparing to today's web-server hard-disk capability. This restriction can be solved by applying a Digital Certificate for our browser program, which will unlock the cross-domain security issue on Opera.

8.8 Efficiency and speed

Our browser is designed for high performance in efficiency and speed. The LRRP architecture has a great potential in reducing the communication cost and improving speech-recognition accuracy.

Our browser uses AJAX, which is the latest web-application technology, to control the network communication. AJAX technology provides a great improvement in reducing the redundancy of network communication, which only updates the minimum required information for every communication.

Our SpeechWeb Browser provides a real-time human-machine interaction, with high speech-recognition accuracy in a suitable environment.

8.8.1 Network communication cost

By using the LRRP architecture, our SpeechWeb Browser has the lowest network communication cost compared to most other speech applications (text transmission comparing to voice transmission). The speech-recognition process is done locally on the user's device, and only the recognized text string will be sent to the remote server to process. The number of words in the text depends on the definition given in the speech-application grammar, but normally it is about 1-10 words.

8.8.2 Speech-recognition accuracy

The accuracy of the speech-recognition process is difficult to measure. It depends on the user's accent, the environment noise, and the quality of speech-input microphone. Our SpeechWeb Browser uses Opera which has a built-in speech-recognition engine developed by IBM. The speech-recognition engine in Opera is very accurate as we used in a regular room environment. More information about the speech-recognition accuracy of Opera can be found from Opera Software [OperaS].

8.8.3 Time complexity

To start the browser with a new remote application:

Time complexity = O(size of X+V page + size of speech-grammar for the application);

= O(constant + size of speech-grammar);

= O(size of speech-grammar)

Recognition of user's speech:

The time complexity is affected by two factors: the size of speech-grammar and the length of user's input.

- O(size of grammar)
- O(length^3), for an ambiguous grammar

Transmission of text query to a remote sever and return of the answer:

Time complexity = O(length of query + length of answer)

Please check the program code in Appendix IV.

8.8.4 Speed

Our SpeechWeb Browser has been tested on a laptop with the following specification:

CPU: Pentium M ,1.6MHz,

Memory: 512M

Operation system: Windows XP SP2

Microphone: IBM ThinkPad T40 built-in

Internet connection: a stantard high-speed cable Internet connection with upload-speed up to 640Kbps and download-speed up to 7 Mbps [Cogeco Cable, 2006]

SpeechWeb application: Solarman Sihlo, which has the largest

speech-grammar file and knowledge source program in our samples.

Speech-recognition speed:

Speech-input:

The longest question tested with the Solarman Sihlo was: "*Who discovered Titania and Titania and Titania and Titania and Titania and Titania and Titania and Titania and Titania and Titania*".

Result:

The speech-recognition process appears to be instantaneous. The process time is extremely fast and cannot be measured with a clock.

SpeechWeb Browser speed:

We have calculated the browser start-up speed and human-machine speech interaction speed by running the SpeechWeb Browser 20 times on 2 different days. Table 2 contains the experimented results:

"Load SpeechWeb End User Interface" is: the period from user start to load a X+V page on Opera, until the SpeechWeb interface is fully loaded (the first sound of the welcome voice message). Time measured using a regular timer.

"Speech Interaction" is: the period start from the user's input (request) is recognized, until the answer is returned from remote and uttering to the user.

Table 2: SpeechWeb Browser Speed Experiment Result (in *seconds*)

	Load SpeechWeb End User Interface	Speech Interaction
Average	0.92	0.218
Minimum	0.7	0.150
Maximum	1.4	0.291

From Table 2, we can see that it takes about 1 second to load the X+V browser page. This result is acceptable, because we only need to load the X+V page when we are connecting to a new SpeechWeb application. After the application's X+V page is loaded, a user-Sihlo speech interaction can be done in about 0.2 sec, which is very fast comparing to human response.

Please note that the speed for application transition (the process to transfer from current Sihlo to another SpeechWeb application) is the speech interaction time + the time to load a new application X+V page.

Overall, our X+V SpeechWeb Browser runs at a very fast speed on an average-configured PC equipped with a home standard high-speed Internet connection.

Chapter 9: Use, Implementation and Documentation

9.1 New SpeechWeb Browser user manual

A "User Manual" for our new SpeechWeb Browser is provided in Appendix I. It includes all information about where and how to install Opera, how to enable the voice-feature in Opera, and how to use our SpeechWeb Browser.

9.2 Manual for creating SpeechWeb applications.

A manual on how to create a SpeechWeb application in few steps, together with an example, is given in Appendix II.

9.3 New SpeechWeb Browser website

The introduction to our new SpeechWeb Browser website is given in Appendix III. This website was recently created by Dr. Frost and Xiaoli Ma, and welcome to have more visitors. [SpeechWeb]

Copies of the manuals, the X+V browser, and X+V browser introduction page are available at the website.

9.4 Program code

Appendix IV shows the complete program code of our new SpeechWeb Browser in both the single-page version and the multiple-page version. Please note that the single-page version does not work reliably in all situations as we described in Section 6.2.2. All code is original and was written by the author of this thesis.

Chapter 10: Conclusions and Future Work

10.1 What has been achieved?

The Thesis Statement was that:

"A viable speech browser for a Public-Domain SpeechWeb can be implemented using X+V."

The thesis statement has been proven in demonstration by:

1. Investigating the new Multimodal programming language X+V.
2. Solving several limitations of X+V.
3. Investigating the LRRP architecture.
4. Investigating different potential designs for a new speech browser.
5. Identifying a design with good potentials.
6. Successfully implementing and testing the new design, and the resulting browser.
7. Analyzing the new SpeechWeb Browser with respect to its "viability", showing:
   - Real-time human-machine speech interaction.
   - High speech-recognition accuracy in a suitable environment.
   - Ease of installation and use.
   - Completely free availability to all users.
   - Ease of implementation for speech applications.
   - Possibility of deployment on handheld devices after small changes.
   - Compatibility with the LRRP architecture which is ideally suited for building a Public-Domain SpeechWeb.

10.2 Suggestions for future work

As we discussed in Chapter 8, our current SpeechWeb Browser could be updated to work on mobile version of Multimodal browsers – Opera v7 and/or NetFront, and distributed to mobile users.

In Section 8.7, we have discussed the restriction that our browser must obey the cross-domain security issue on Opera. This problem can be solved by applying for a Digital Certificate for our SpeechWeb Browser program, which could unlock the restriction.

Also, our current SpeechWeb Browser does not have the ability to let the user to speak the URL. This could be implemented in our browser, or constructed as a SpeechWeb application.

We can also extend the current simple browser start-up page with list of useful SpeechWeb sites/pages when the SpeechWeb has grown, so that the user can be familiar with SpeechWeb easily.

Bibliography

[CC06] Cogeco Cable, High Speed Internet – Packages and Pricing – Cogeco Cable – Ontario – Canada, http://www.cogeco.ca/en/high-speed-internet-_o.html

[FAB04]. 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.

[Frost05] Frost, Richard, "A Call for a Public-Domain SpeechWeb", *Communication of the ACM*, Volume 48, Issue 11 (2005) 45-49

[HT95] 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.

[IBM] IBM "Multimodal Overview" *IBM's Multimodal Website* http://www-306.ibm.com/software/pervasive/multimodal/

[Lucas00] Lucas, B., "VoiceXML for web-based distributed conversational applications", *Communications of the ACM* 43 (9), 2000, 53-57.

[Manuals] Xiaoli Ma, "User and Developer Manual of new SpeechWeb Browser" http://sol.cs.uwindsor.ca/~speechweb/p_d_speechweb/Multi-page Version Public-domain SpeechWeb Browser.pdf

[OperaS] Opera Software, "Opera Software Website" http://www.opera.com

[Su05] Su, Li, "Using VXML to Construct a Speech Browser for a Public-Domain SpeechWeb", *Master's Thesis*, School of Computer Science, University of Windsor, ON, Canada, 2005.

[SF05] Li Su, and Richard A. Frost, "A Novel Use of VXML to Construct a Speech Browser for a Public-Domain SpeechWeb" *Canadian Conference on AI 2005, 401-405*

[SpeechWeb] Xiaoli Ma and Richard A. Frost, "Website of New X+V Public-Domain SpeechWeb Browser" http://sol.cs.uwindsor.ca/~speechweb/p_d_speechweb/

[W3C00] W3C, "Voice eXtensible Markup Language (VoiceXML.) version 1.0"

W3C Recommendation 05 May 2000, http://www.w3.org/TR/voicexml/ , 2000.

[W3C01] 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

[W3C04] W3C, "Voice Extensible Markup Language (VoiceXML) Version 2.0" *W3C Working Recommendation 16 March 2004,*
http://www.w3.org/TR/voicexml20/, 2004

[XV01] Axelsson, J. Cross, C. Ferrans, J. McCobb, G. Raman, T.V. Wilson, L. "XHTML+Voice Profile 1.0", *W3C Note 21 December 2001,*
http://www.w3.org/TR/xhtml+voice/, 2001.

APPENDIX I

User Manual
Of Our New Public-Domain
SpeechWeb Browser

March, 2006

Supervisor: ***Dr. Richard A. Frost***

Student: ***Ma Xiaoli (William)***

Table of Contents

I. Installation Instructions

1.1 System Requirement

This X+V voice browser needs an "Opera for Windows" version 8 or higher to be installed on the user's windows system.

Recommended computer system requirements:

Operating System: Windows 2000 or XP.

CPU: Pentium 166MHz processor.

Memory: 32 MB of RAM.

Hard disk: 50 MB free disk space.

A speaker and a microphone, which are compatible with your computer system.

These system requirements may be updated without additional notification.

1.2 Install Opera 8 for Windows

Steps to install Opera 8 into your Windows system:

1. Go to the download page at http://opera.com/download/.
2. Select Opera 8 for Windows to download.
3. Double click on the Opera installation file you downloaded
4. The welcome screen tells you that: "You are about to install Opera". Press "Next >"
5. Accept the software license agreement. Press "I Accept"
6. Install Opera in the suggested directory. Press "Next >"
7. Create icons and shortcuts. Press "Next >"
8. The information is complete and the installation will begin. Press "Install" (Files are copied to your hard drive.)

9. Opera can start automatically after installation. Press "Finish"

If you have any trouble installing Opera 8 on your Windows system, please go to Opera Forum at http://my.opera.com/community/forums/forum.dml?id=2.

1.3 Install Voice Feature for Opera 8

Steps to install voice feature for Opera 8:

1. Download Opera 8 at http://opera.com/download/. Install Opera 8 for Windows into your computer.
2. Select "Tools" > "Preferences" > "Advanced" > "Voice". Enable the voice option.
3. After enabling the option, you will be asked to confirm that you want to install the voice libraries. Select confirm.
4. Voice is available after downloading Opera voice libraries.
5. To initiate a voice command, Press the "Voice" button on the View toolbar, or press the Scroll Lock key on your keyboard. Then say your command or query. After issuing, release the button.
6. You can also customize the "Voice key to talk", and "Talk key mode" in "Tools" > "Preferences" > "Advanced" > "Voice".

If you have any trouble to set up the voice feature into Opera, please go to Opera Voice Forum at http://my.opera.com/community/forums/forum.dml?id=95,

II. User Manual

2.1 Start the SpeechWeb Browser

Notice, Opera 8 for Windows with enabled voice feature has to be installed before using our X+V SpeechWeb Browser.

Steps to start the X+V SpeechWeb Browser:

1. Start your pre-installed Opera 8 for Windows. Make sure you have enabled voice feature.
2. Input the following URI into your Opera 8 web browser: http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/menu/demo_menu.xml Then Opera 8 will load the X+V SpeechWeb Browser Menu Page for you.
3. Make sure you have opened your sound speaker and set the volume high enough. You will hear a greeting voice message after you have successfully started the X+V SpeechWeb Browser.

Please note that SpeechWeb Browser also requires a microphone to allow the user to talk to the voice page.

2.2 Use the SpeechWeb Browser

In our SpeechWeb Browser, you can either talk to one of our sample voice pages (Sihlos), or ask SpeechWeb Browser to load your own voice page.

2.2.1 Talking to sample voice pages.

After successfully starting the SpeechWeb Browser, it loads the menu page of SpeechWeb Browser. You can choose one of the sample Sihlos to talk to. You may choose the Sihlo by clicking on their link, or simply saying it to menu

page(Please refers to Section 2.3 for the list of queries you can say in SpeechWeb Menu Page.) After you have chosen from one of the sample Sihlos, SpeechWeb will transfer you to that Sihlo, and you can start talking to it. You can also find the list of available speech input for Judy, Monty and Solarman in Section 2.3

Notice, you need to hold the "Voice" button on the Opera View toolbar or the Talk Key ('Scroll Lock' key by default) on your keyboard when you are speaking. Release the button after speaking.

2.2.2 Talking to your own SpeechWeb application.

If you have your own SpeechWeb application, you can input the URI of its X+V page into the text input-field, to load and talk to it. The text input-field is at the top of the SpeechWeb Menu Page and every sample Sihlos (*judy, monty, Solarman).* Please refer to the Developer's Manual (in the Appendix II) for how to create your own SpeechWeb application in 4 easy steps.

2.3 Available Speech Input List

There is one menu page and three interpreters in the SpeechWeb Browser default sample application that you can talk to.

2.3.1 SpeechWeb Menu Page

In the menu page, you can say a choice from Judy, Monty, or Solarman to start talking. The possible query you can say is much more flexible.

For example:

"Can I talk to Judy"

"I wanna speak to Monty, please"

"Could you please transfer me to Solarman."

"I would like to speak with Judy, please."

"Please transfer to Monty, thanks"

"Judy, please"

There are more queries you can say. Simply try it by yourself. But, please note that, you have to include the name of the Sihlo you would like to talk to in your query, which means you have to say Judy, Monty, or Solarman in your query.

2.3.2 Judy

List of available speech input for Judy:

| hello

| hello there

| hello judy

| goodbye

| goodbye judy

| fine thanks

| thanks

| thanks judy

| yes please

| what is your name

| who are you

| where do you live

| what do you know

| how old are you

| who made you

| what is your favorite band

| who is the vice president at the university of windsor

| who is the president at the university of windsor

| who is the president of sun microsystems canada

| 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 monty

| can I talk to monty

| can I talk to judy

| who is solar man

| can I talk to solar man;

2.3.3 Monty

List of available speech input for Monty:

hello

| hello there

| hello monty

| goodbye

| goodbye monty

| fine thanks

| thanks

| thanks monty

| yes please

| what is your name

| who are you

| where do you live

| what do you know

| how old are you

| who made you

| what is your favorite band

| who is the vice president at the university of windsor

| who is the president at the university of windsor

| who is the executive dean of science at the university of windsor

| who is the dean of science at the university of windsor

| tell me a poem

| know any poems

| tell me a joke

| know any jokes

| who is judy

| can I talk to judy

| who is solar man

| can I talk to monty

| can I talk to solar man;

2.3.4 Solarman

Solarman can answer much more complicated question compare to Judy and Monty. You can ask question 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"

APPENDIX II

Developer Manual
Of Our New Public-Domain
SpeechWeb Browser

May, 2006

Supervisor: ***Dr. Richard A. Frost***

Student: ***Ma Xiaoli (William)***

I. Components needed for your SpeechWeb application

1. Original knowledge source / program.
2. A CGI script.
3. A speech-grammar file.
4. A X+V page.

(Sometimes, we also refer to a SpeechWeb application as *"Sihlo"*. Just make it different from other applications.)

III. How to create your own SpeechWeb application

The following is the four steps to create a SpeechWeb application:

1. Create a knowledge source / program.

You can use any programming language to create your knowledge source program. The program has only one requirement: it can receive a request and return a response as a string, no matter how the data processing is done inside the program. You can easily create it by modifying your conventional web application knowledge sources.

For example, the following is a small program written in Miranda, *judy.m:*

```
interpret "hello" = "hi there. My name is Judy"
interpret "hello there" = "Hello, how are you?"
interpret "hello judy" = "hi, how are you?"

interpret "goodbye" = interpret "goodbye judy"

interpret "goodbye judy" = "goodbye. Who do you want to talk to?"


interpret "fine thanks" = "Good, so am I. In fact I feel great."
interpret "thanks" = "no problem"
interpret "thanks judy" = "no problem at all"
interpret "yes please" = "yes please? What did you say? I was working on a new poem."
interpret "what is your name" = "My name is Judy."
interpret "who are you"
    = "My name is Judy. I know about poems."
interpret "where do you live"
     = "I live in a warm computer. "
         ++ "In good old Lambton Tower. University of Windsor."
interpret "what do you know"
    = "I only know one poem just now. My collegue, Dr. Frost"
       ++ " is looking for someone to teach me some more poems."
interpret "how old are you"
  = "What? That is a bit cheeky. I am younger than stuffy old Solar man."
interpret "who made you"
    = "I. B. M. and Opera Software made my ears and vocal chords. William Ma connected
my "
       ++ "ears to my brain, and Doctor Frost, my colleague, made "
       ++ "my brain"

interpret "what is your favorite band"
    = "ARE. E. EM. They are a really cool band"

interpret "who is the vice president at the university of windsor"
  = "No idea, never been outside of Lambton Tower."

interpret "who is the president at the university of windsor"
  = "How should I know, this is the first time I have ever got to meet"
     ++ " anyone important."

interpret "who is the dean of science at the university of windsor"
    = "I know. Dr. Fryer. He helped get me more space for my memory."
```

```
interpret "tell me a poem"
        = "the dark clouds, lingered, as he lay."
          ++ " Then, the strong winds, blew them both away."

interpret "tell me a joke" = "do not know any jokes. But my friend, Monty, does"

interpret "know any jokes" = "No, but my friend, Monty does."

interpret "who is monty"
     = "Monty is my friend. He is a student"
       ++ " at the university of Windsor."


interpret "can I talk to judy"
                    ="LINK=yes. here she is;"
                      ++
"SIHLO=http://sol.cs.uwindsor.ca/~speechweb/p_d_speechweb/judy/judy.xml;"

interpret "can I talk to monty"
                    ="LINK=yes. here he is;"
                      ++
"SIHLO=http://sol.cs.uwindsor.ca/~speechweb/p_d_speechweb/monty/monty.xml;"

interpret "can I talk to solar man"
                    ="LINK=yes. here he is;"
                      ++
"SIHLO=http://sol.cs.uwindsor.ca/~speechweb/p_d_speechweb/solarman/solarman.xml;"


interpret x = "sorry, got no poem for that one"

sh_answer x = interpret (drop 9 x)
```

2. Create a CGI script.

You will need to create a CGI script to handle the communication between your knowledge source program with the outside. This CGI script file needs to recieve users' HTTP-request, and pass it to the program; also pass the returned response (from the program) back to the user.

For example, the following is the CGI script file *(judy.cgi)* for Judy:

```
#!/bin/csh -f

setenv HOME 'luna.cs.uwindsor.ca/fac3/richard/public_html/judy:$HOME'
setenv PATH 'luna.cs.uwindsor.ca/lapps1/mira:$PATH'


echo "Content-Type:text/plain"
echo "

setenv v "`luna.cs.uwindsor.ca/bin/cat`"
luna.cs.uwindsor.ca/lapps1/mira/bin/mira
luna.cs.uwindsor.ca//stu2/xing4/public_html/compile/judy.m << zzz
(sh_answer "$v")
/q
zzz
```

3. Create a JSGF speech-grammar file for your application.

You can find many tutorials about how to create a JSGF speech-grammar file from the Internet, here is one of them:

http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/.

The easiest way to create this speech-grammar file is to simply list all of the possible input for your program. The following is the speech-grammar file used for "*Judy Sihlo*":

```
grammar vxmljudy;

public <s> =   <simple>;


<simple> = yes
         | no
         | go back
         | hello
         | hello there
         | hello judy
         | goodbye
         | goodbye judy
         | fine thanks
         | thanks
         | thanks judy
         | yes please
         | what is your name
         | who are you
         | where do you live
         | what do you know
         | how old are you
         | who made you
         | what is your favorite band
         | who is the vice president at the university of windsor
         | who is the president at the university of windsor
         | who is the president of sun microsystems canada
         | 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 monty
         | can I talk to monty
         | can I talk to judy
         | who is solar man
         | can I talk to solar man;
```

If you are a skillful speech application developer, then you can create a more complicated JSGP speech-grammar. The following is the speech-grammar file used in our SpeechWeb Browser menu page; it can recognize 549 different combinations of speech-input sentences.

```
grammar speechweb;
public <speechweb> = <start> <name> [please | thanks] {$= $name;};

        <start> = <sub_i> <action>
                | <sub_you> [please] <transfer_to>
                | NULL;

        <sub_i> = i wanna
                | i want to
                | i like to
                | i hope to
                | i would like to
                | can i
                | may i
                | shall i;

         <sub_you> = could you
                   | would you
                   | NULL;

         <action>= talk to | speak to | talk with | speak with | <transfer_to>;

         <transfer_to>= transfer [me] to;

         <name>   = judy | monty | solarman;
```

Here are some possible sentences that the menu page can recognize:

"Can I talk to Judy"

"I wanna speak to Monty, please"

"Could you please transfer me to Solarman."

"I would like to speak with Judy, please."

"Please transfer to Monty, thanks"

"Judy, please"

4. Create the X+V page.

Before starting to create the X+V page, you need to have a copy of the identical X+V page, "*voicepage.xml*". It can be downloaded at http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/voicepage.xml (Furthermore, you also need to know the URL of your CGI script and speech-grammar file.)

The following are the 3 changes that need to be done on your copy of *"voicepage.xml":*

1.) Greeting message.

At the beginning of the *"voicepage.xml"*, you can find the following code:

```
<!-- Please modify the value of variable sv_greeting, and cgiLink to fit your application. -->
<script type="text/javascript">
/** The greeting message that will say to the user, only at the first time the user visits this
page. **/
var sv_greeting ="";
/** The link to your CGI interpreter location. Notice, you have to place the CGI interpreter
program with this page in the same domain to prevent a cross-domain security error.**/
var cgiLink="http://";
</script>
```

Change the value of the *sv_greeting* variable to your application greeting message. This message will be uttered automatically after the page is loaded. The following is the sample code on *Judy.xml* after the change:

```
<!-- Please modify the value of variable sv_greeting, and cgiLink to fit your application. -->
<script type="text/javascript">
/** The greeting message that will say to the user, only at the first time the user visits this page. **/
var sv_greeting ="hi, my name is judy.";
/** The link to your CGI interpreter location. Notice, you have to put the CGI interpreter program with this page in the same domain to prevent a cross-domain security error.**/
var cgiLink="http://";
</script>
```

2.) CGI script location.

Just next to the *sv_greeting* variable (the greeting message), you will find the variable called *cgiLink*. You should assign your CGI script URL to this variable.

The following is the *judy.xml* after changing the greeting message and the URL of its CGI script:

```
<!-- Please modify the value of variable sv_greeting, and cgiLink to fit your application. -->
<script type="text/javascript">
/** The greeting message that will say to the user, only at the first time the user visits this page. **/
var sv_greeting ="hi, my name is judy.";
/** The link to your CGI interpreter location. Notice, you have to put the CGI interpreter program with this page in the same domain to prevent a cross-domain security error.**/
var cgiLink="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/judy/judy.cgi";
</script>
```

3.) Speech-grammar file location.

There's only one dialog-field (*'vxml:field'*) inside of VXML form (*'vxml:form'*) in the *voicepage.xml* document. You only need to change the *'src'* attribute of the *'vxml:grammar'* element which is the first child element of *'vxml:field'*. You can easily find it at the beginning of the document and next to the *sv_greeting and*

cgiLink variables' declaration.

The following is the *voicepage.xml* before the change:

```
<!-- VoiceXML form. -->
<vxml:form id="vxml_form">
<vxml:field name="st_field" xv:id="voice_input" modal="true">
    <!-- NOTICE!!! PLEASE MODIFY THE VALUE OF 'src' ATTRIBUTE IN THE NEXT
LINE <grammar> ELEMENT TO YOUR GRAMMAR FILE LOCATION.-->
    <vxml:grammar type="application/x-jsgf"
        src="http://" />
```

The following is the code after the change for Judy, *judy.xml:*

```
<!-- VoiceXML form. -->
<vxml:form id="vxml_form">
<vxml:field name="st_field" xv:id="voice_input" modal="true">
    <!-- NOTICE!!! PLEASE MODIFY THE VALUE OF 'src' ATTRIBUTE IN THE NEXT
LINE <grammar> ELEMENT TO YOUR GRAMMAR FILE LOCATION.-->
    <vxml:grammar type="application/x-jsgf"
        src="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/judy/judy.jsgf/" />
```

III. Aware issues

Please also be aware of the following issues:

1. The knowledge source program should be an executable file, not a non-compiled source code file.
2. Make sure you give a read and execute permission to the public for your CGI script file. The UNIX command to change the permission is: *chmod (e.g. chmod 755 judy.cgi)*
3. Avoid using similar-sounding words to be the starting word of possible input choices in the knowledge-source program and the speech-grammar

file. For example, you can use "what time" and "which place" instead of "when" and "where".

4. You have to place your X+V page together with the knowledge-source program under the same Internet domain.
5. You can rename the X+V page *"voicepage.xml"* to any name you want but keep the extension to be *".xml"*.
6. If you have heard that *"there is an error in this application"* right after the browser is loaded, then it means you have given an incorrect speech-grammar file URL. Make sure you have *"http://"* at the beginning of the URL.
7. If a *SYSTEM ERROR* message appears after you have given your first question/request to the browser, then it could be caused by one or more of the following reasons:
   a) No Internet connection. Please double-check your Internet connection, and make sure your firewall does NOT block it.
   b) Invalid cross-domain connection [1]. To fix this problem you have to place your X+V page within the same Internet domain of your CGI script file. If you don't know what is meant by "same Internet domain", then simply place the X+V page and the CGI script file in the same folder/directory on your server.
   c) Either an incorrect CGI script URL or the CGI script file in that URL does not exist. Double-check the spelling of the CGI script URL (*cgiLink*), and make sure it starts with a *"http://"*.

[1], "About Cross-Frame Scripting and Security"

http://msdn.microsoft.com/workshop/author/om/xframe_scripting_security.asp

APPENDIX III New Public-Domain SpeechWeb Browser Website

We have created an Internet website for our new Public-Domain SpeechWeb Browser. It is the first place you should visit if you really want to try our SpeechWeb Browser. In there, you can find all the information about it, including the instruction to how to install it, links to sample SpeechWeb applications, and both User Manual and Developer Manual.

You are always welcome to visit our website, and give us any comments to our work. Our new Public-Domain SpeechWeb Browser website is at http://sol.cs.uwindsor.ca/~speechweb/p_d_speechweb/. You can use any regular web browser to open it, such as Microsoft IE, Firefox, Opera, etc.

The following is the URL to our SpeechWeb home page, which has links to both old and new version of our SpeechWeb Browser: http://sol.cs.uwindsor.ca/~speechweb/.

APPENDIX IV

Multiple-Page Version

Identical X+V page, voicepage.xml:

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//VoiceXML Forum//DTD XHTML+Voice 1.2//EN"
"http://www.voicexml.org/specs/multimodal/x+v/12/dtd/xhtml+voice12.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:vxml="http://www.w3.org/2001/vxml"
      xmlns:ev="http://www.w3.org/2001/xml-events"
      xmlns:xv="http://www.voicexml.org/2002/xhtml+voice"
      xml:lang="en-US">

<!--*****************************************************************************************
Date: May. 2006
Developer: Ma, Xiaoli(William)
Architecture: LRRP (Dr.Frost, University of Windsor, Canada)
*****************************************************************************************-->

<!--***********************************
Note:
This is an incompleted SpeechWeb voice page, which is not associated with any speech
applications.
In order to complete, you need to edit three variables -- 'sv_greeting', 'cgiLink', and 'src'
attribute in the 'vxml:grammar'.
Please read the "Developer's Maual" document for details, downloadable at our website
http://sol.cs.uwindsor.ca/~speechweb/p_d_speechweb/index.html
***********************************-->

<head>

<!-- Please modify the value of variable sv_greeting, and cgiLink to fit your application. -->
<script type="text/javascript">
/** The greeting message that will say to the user, only at the first time the user visits this
page. **/
var sv_greeting ="";
/** The link to your CGI interpreter location. Notice, you have to place the CGI interpreter
```

```
program with this page in the same domain to prevent a cross-domain security error.**/
var cgiLink="http://";
</script>

<!-- VoiceXML form. -->
<vxml:form id="vxml_form">
<vxml:field name="st_field" xv:id="voice_input" modal="true">
    <!-- NOTICE!!! PLEASE MODIFY THE VALUE OF 'src' ATTRIBUTE IN THE NEXT
LINE <grammar> ELEMENT TO YOUR GRAMMAR FILE LOCATION.-->
    <vxml:grammar type="application/x-jsgf" src="http://" />
    <!-- The following greeting will only speak out when user connects to a new interpreter.
-->
      <vxml:prompt cond="sayGreetings==true"><vxml:break
time="500ms"/><vxml:value expr="sv_greeting"/><vxml:value
expr="updateShowFrame('GREETING: '+sv_greeting);"/></vxml:prompt>
    <vxml:filled>
 <!--**********************************
  This "filled" element will be run after user speech input has recognized.
  Inside this element, first step, i have assign the user input to the variable 'question',
because VoiceXML code can access a JavaScript defined variable, but JavaScript can not
see a VoiceXML defined variable.
  Then, in the next step, i call a JavaScript function "runCode()' to proceed AJAX submit
process.
**********************************-->
      <vxml:assign name="question" expr="st_field"/>
      <!--**********************
        Calls to javascript mainControl() function to do the logical process based on user
voice input.
      *************************-->
      <vxml:assign name="javacode" expr="mainControl();"/>
      <vxml:prompt><vxml:break time="300ms"/><vxml:value
expr="answer"/></vxml:prompt>
<!-- If the answer is not a link to next interpreter, then repeat the voice dialog. -->
      <vxml:if cond="isLink==false">
        <vxml:throw event="repeat.st_field"/>
      </vxml:if>
    </vxml:filled>
    <vxml:catch event="nomatch noinput">
      <vxml:prompt>Sorry, I don't understand, can you say it again?</vxml:prompt>
      <vxml:reprompt/>
    </vxml:catch>
    <vxml:catch event="help">
      No help is available! Restart the dialog!
      <vxml:clear namelist="st_field"/>
```

```
      <vxml:reprompt/>
    </vxml:catch>
</vxml:field>

<!-- Catch the 'repeat.st_field' event. -->
<vxml:catch event="repeat.st_field">
  <vxml:clear namelist="st_field"/>
<!-- Restart the voice form without change the speech grammar. -->
  <vxml:reprompt/>
</vxml:catch>
</vxml:form>


<script type="text/javascript">
/******* Declare global variables shared by JavaScript and VoiceXML *********/
var sayGreetings=true;
var defaultGreetingMsg="Hi, i'm ready to talk now.";
/*** The location of next remote speech-application/CGI-application interpeter. ***/
var nextPage="";
/*** Question query recognized from user's speech (request). ***/
var question="";
/*** Answer query returned from remote CGI interpter (response). ***/
var answer="";
var answerRecieved=false;
/*** Answer query contains a link to next CGI interpreter. ***/
var isLink=false;
var gotoNext= false;
/*** This variable needed for VXML to call JavaScript code. ***/
var javacode="";
/*** menu page of the demo public-domain speechweb. ***/
var
startPage="http://sol.cs.uwindsor.ca/~speechweb/p_d_speechweb/menu/demo_menu.xm
l";

if(sv_greeting=="")
  sv_greeting=defaultGreetingMsg;


/*****************************************************************************************
This is the main control function to the whole question submit and answer retrived
procedures.
It will call submitReq() method to send the question to the CGI program.
then it will check the answer whether it is a link to new CGI program or a simply answer
string.
```

```
if it is a link to another interpreter, then retrieve the data from there,
and call the 'changeData' function to change the neccessary information for the next
round dialog.
**********************************************************************************************/
function mainControl()
{
  updateShowFrame("QUESTION: "+question+"<br/>");

  answer="";
  answerRecieved=false;
  isLink=false;
  sayGreetings=false;

  /* call submitReq() method to send the question to the CGI program. */
  submitReq("POST", cgiLink);
  /** Cannot recieve data from CGI interpreter. Network problem. **/
  if(answerRecieved==false)
    return "-1";
  answer = getAnswer(xmlhttp.responseText);

  /******* Check whether the recieved answer is a link or not. And, assign the result to the
global variable isLink. ******/
  checkAnswer(xmlhttp.responseText);

  /****** if the answer is not a link, then show the answer to the user and return. ******/
  if(!isLink)
  {
    gotoNext=false;
    updateShowFrame("RESPONSE: "+answer+"<br/>");
    return "1";
  }

  nextPage=getNextInterpreter(xmlhttp.responseText);

  updateShowFrame("RESPONSE: "+answer+"<br/><br/>");

  if(gotoNext==true)
    window.location=nextPage;

  return "1";
}


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

```
This function returns the substring that has to be spoken as a result of the user's question.
Same procedure is applied for extracting the content to be spoken out.
**********************************************************************************************/
function getAnswer(answer)
{
  var ex=answer;
  var index;
  if((ex.indexOf('LINK=',0)) == -1)
    return ex;
  ex= ex.slice(5);
  index = ex.indexOf(";",0);
  ex = ex.substring(0,index);
  return ex;
}


/**********************************************************************************************
This function uses AJAX, it will submit the question to the given URI if it use a 'POST'
method.
Or, it will retrieve data from the given URI if it use a 'GET' method.
**********************************************************************************************/
function submitReq(method, url)
{
  /***** Initialize AJAX XMLHttpRequest object. ****/
  xmlhttp=new XMLHttpRequest();
  /******
  Assign a event listener to the 'onreadystatechange' event.
  Different listerner assigned depends on a 'GET' or a 'POST' method.
  ******/
  if(method=="GET")
    xmlhttp.onreadystatechange=stateChange_GET;
  else
    xmlhttp.onreadystatechange=stateChange_POST;


  /** Check whether the url involves a cross-domain security error before send the
request. **/
  if(isCrossDomain(url)==true)
  {
    /** if method is 'GET', it means this function is called from loadPage() function to
validate a user input URL. **/
    if(method=="GET")
      alert("Cannot validate input URL since it involves a cross-domain security issue.
Load URL immediately.");
```

```
    /** if method is not 'GET', which means 'POST' method, it means this method is called
from main control to submit a question query to the interpreter. **/
    else
      updateShowFrame("SYSTEM ERROR: An error which against the web browser
cross-domain security issue. Your CGI interpreter has to be placed in the same domain
with this voice page."+
                      "Please contact to your application provider to fix this problem.
\n"+"Your CGI interpreter location: "+ url+"    Current voice page host domain:
"+window.location.host);
    answerRecieved=true;
    answer="An error which against the web browser cross-domain security issue has
occured. Please check the error message to continue.";
    return;
  }

  /** Open the connect, sychronized.   ***/
  xmlhttp.open(method,url,false);

  if(method=="GET")
    xmlhttp.send();
  else
  {
    xmlhttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    xmlhttp.send("question="+question);
  }
}

function stateChange_POST()
{
  /******* if xmlhttp shows loaded   ******/
  if (xmlhttp.readyState==4)
  {
    if (xmlhttp.status==200 || xmlhttp.status==304)
    {
      answerRecieved = true;
    }
    else
    {
      answerRecieved = false;
      xmlhttp.responseText = "";
    }
  }
}
```

```
function stateChange_GET()
{
  /******* if xmlhttp shows loaded  ******/
  if (xmlhttp.readyState==4)
  {
    if (xmlhttp.status==200 || xmlhttp.status==304)
    {
      answerRecieved = true;
    }
    else
    {
      answerRecieved = false;
      xmlhttp.responseText = "";
    }
  }
}

/** Validate the given url with the current page domain(hostname), to see whether they are
in the same domain or cross-domain(different domain). **/
function isCrossDomain(url)
{
  var domain = url;
  var i = domain.indexOf("//");
  if(i==-1)
    return false;

  domain = domain.slice(i+2);

  var k = domain.indexOf("/");
  if(k!=-1)
    domain = domain.slice(0, k);
  else
    return true;

  var host = window.location.hostname;

  if(host==domain)
    return false;
  else
    return true;

}

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

```
****
Check whether there is a occurrence of '=' character in the answer, which means a link
existed in it.
And, assign the result to the global variable 'isLink'.
**********************************************************************************************
**/
function checkAnswer(answer)
{
  if((answer.indexOf('LINK=',0))== -1)
    isLink=false;
  else
    isLink=true;

  return isLink;
}

/**********************************************************************************************
************
if the answer is a link, this function will return the next interpreter's URI as a string.
Otherwise, return "-1".
**********************************************************************************************
*************/
function getNextInterpreter(answer)
{
  var loc;
  var ex = answer;
  var index;

/****** Check if the answer is a link to next speech application interpreter. It should never
be evaluated as true, otherwise error. ******/
  if(!isLink)
    return "-1";

/**********************************************************************************************
******************
If the answer is a link, then its formation should be: "LINK=_answer;SIHLO=_location;".
e.g. Question send to judy.cgi: "can i talk to solar man".
      Answer recieved from judy.cgi: "LINK=yes. here he
is;SIHLO=http://sol.cs.uwindsor.ca/~speechweb/p_d_speechweb/judy/judy.xml"
**********************************************************************************************
*****************/
/****** 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
answer-query 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(";",0);
    loc = ex.substring(0,index);

    return loc;
}

/** Update the text area in the HTML and show message on it. **/
function updateShowFrame(message)
{
  var objTable = document.getElementById("logFrame");

  objTable.insertRow(0);
  objTable.rows[0].insertCell(0);
  objTable.rows[0].insertCell(1);
  var cell0 = objTable.rows[0].cells[0];

  var cell1 = objTable.rows[0].cells[1];
  cell1.align="left";
  cell0.align="left";
  cell0.width="105";
  if(message.indexOf("SYSTEM ERROR: ")!=-1)
  {
    var objFont = document.createElement("font");
    objFont.color="red";
    objFont.size="-1";
    objFont.appendChild(document.createElement("b"));
    objFont.firstChild.innerHTML = message.slice(0, message.indexOf(":")+1);
    cell0.appendChild(objFont);
```

```
      var objFont2 = document.createElement("font");
      var objIta = document.createElement("i");
      objFont2.color="black";
      objFont2.size="-1";
      objFont2.appendChild(objIta);
      cell1.appendChild(objFont2);
      objIta.innerHTML=message.slice(message.indexOf(":")+1);
    }else
    {
      var index = message.indexOf(":");
      var ex=message.slice(0,index+1);
      var objFont = document.createElement("font");
      if(ex.indexOf("QUESTION:")!=-1)
        objFont.color= "blue";
      else if(ex.indexOf("RESPONSE:")!=-1)
        objFont.color="green";
      else
        objFont.color="purple";

      objFont.appendChild(document.createElement("b"));
      cell0.appendChild(objFont);
      objFont.firstChild.innerHTML = ex;
      cell1.appendChild(document.createElement("font"));
      cell1.firstChild.innerHTML = message.slice(index+1);
    }
    /****** Insert a table row as an empty line after a response and greeting message.
***********/
    if(message.indexOf("QUESTION")==-1)
    {
      objTable.insertRow(0);
      objTable.rows[0].insertCell(0);
      objTable.rows[0].colspan="2";
      objTable.rows[0].cells[0].innerHTML = "<br/> ";
    }

    return "";
}

/** Load user's application. **/
function loadPage(checkInput)
{
    /** Get user's input. **/
    var loc = document.getElementById("id_nextPage").value;
```

```
  /** if user's input is empty, then return a error message. **/
  if(loc=="")
  {
    updateShowFrame("SYSTEM ERROR: Please input the URL to your voice page in
the above text field. It can not be empty!" );
  }
  /** if user input is not empty, and user asked to validate URL before go. **/
  else if(checkInput==true)
  {
    submitReq("GET", loc);
    /** if the valicating process return a false as result, which means invalid URL. **/
    if(answerRecieved==false)
    {
      if(xmlhttp.status==404)
        updateShowFrame("SYSTEM ERROR: Unable to load your voice page. File
does not exist at: "+ loc );
      else
        updateShowFrame("SYSTEM ERROR: Unable to load your voice page. Network
problem, error code: "+xmlhttp.status+". Please check your internet connection.");
    }
    /** if user's input is not empty, it is a valid URL to next page. **/
    else
      window.location=loc;
  }
  /** if user's input is not empty, and user asked to load URL page immediately. **/
  else
    window.location=loc;
}

function menuPage()
{
  window.location=startPage;
}

function processQuestion()
{
  gotoNext = true;
  question = document.getElementById("id_questionField").value;
  document.getElementById("id_questionField").value="";
  mainControl();
}

</script>
```

```
<!--*******************************************************
   The following script will only be run after a 'vxmldone' event is thrown after the
VoiceXML form finish all its process.
   It also means that the answer returned from interpreter contains a link to next interpreter,
so it needs to go there.
*******************************************************-->
<script type="text/javascript" id="gotoNextPage" declare="declare">
   window.location=nextPage;
</script>

<title>SpeechBrowser</title>
</head>
<body id="page.body">
<center><h2>Welcome to our new voice browser!</h2></center>
<br/>
<center>
<table>
<tr><td colspan="6">Load your own speech application : <input type="text"
id="id_nextPage" size="50"
value="http://sol.cs.uwindsor.ca/~speechweb/p_d_speechweb/menu/demo_menu.xml"/>
<br/><br/></td></tr>
<!-- Call loadPage() function to setup the interprter and speech-grammar location
according to the above input text field value; -->
<tr><td colspan="2"><input type="button" name="submitValidate" value="Validate Before
Go" onclick="loadPage(true)"/></td>
<td colspan="2"><input type="button" name="submitGo" value="Go Immediately"
onclick="loadPage(false)"/></td>
<td colspan="2"><input type="button" name="menuGo" value="SpeechWeb Menu Page"
onclick="menuPage()"/></td></tr>
<tr><td colspan="6"><br/><br/><br/><b>Say your question or type it in
here:</b></td></tr>
<tr><td colspan="6"><form onsubmit="processQuestion(); return false;"><input
type="text" size="70" name="questionField" id="id_questionField"
value=""/></form></td></tr>
</table>
<br/><br/>
<table id="logFrame" width="600"></table><br/>
<br/><br/><br/>
</center>
</body>
<!-- Call a script to reload the vxml form when the current vxml form has done its process.
-->
<ev:listener ev:observer="page.body" ev:event="vxmldone" ev:handler="#gotoNextPage"
ev:propagate="stop" />
```

```
<!-- Load 'vxml_form' when the page.body loaded. -->
<ev:listener ev:observer="page.body" ev:event="load" ev:handler="#vxml_form"
ev:propagate="stop" />
</html>
```

SpeechWeb Menu Page, "demo_menu.xml"

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//VoiceXML Forum//DTD XHTML+Voice 1.2//EN"
"http://www.voicexml.org/specs/multimodal/x+v/12/dtd/xhtml+voice12.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:vxml="http://www.w3.org/2001/vxml"
      xmlns:ev="http://www.w3.org/2001/xml-events"
      xmlns:xv="http://www.voicexml.org/2002/xhtml+voice"
      xml:lang="en-US">

<!--**************************************************************************************
Date: March. 2006
Developer: Ma, Xiaoli(William)
Architecture: LRRP (Dr.Frost, University of Windsor, Canada)
**************************************************************************************-->

<head>
<!-- VoiceXML form. -->
<vxml:form id="vxml_form">
<vxml:var name="sayit" expr="true"/>
<vxml:field name="st_field" xv:id="voice_input" modal="true">
                <!-- The following grammar contains 549 different combinations of possible
speech-input. 549=(8*6+3*2*2+1)*3*3; -->
                <vxml:grammar>
                    <![CDATA[
                        #JSGF V1.0;
                        grammar speechweb;
                        public <speechweb> = <start> <name> [please | thanks] {$=
$name;};

                                    <start> = <sub_i> <action>
```

```
                    | <sub_you> [please] <transfer_to>
                    | NULL;

            <sub_i> = i wanna
                    | i want to
                    | i like to
                    | i hope to
                    | i would like to
                    | can i
                    | may i
                    | shall i;

            <sub_you> = could you
                      | would you
                      | NULL;

            <action>= talk to | speak to | talk with | speak with |
<transfer_to>;

            <transfer_to>= transfer [me] to;

            <name>   = judy | monty | solarman;
        ]]>
    </vxml:grammar>
  <vxml:prompt cond="sayit==true">Please say the name of the sihlo that you wanna
talk to.
  <vxml:break time="500ms"/>Or input the U. R. L. for your own speechweb
application.</vxml:prompt>
  <vxml:filled>
    <vxml:assign name="name" expr="st_field" />
    <vxml:if cond="st_field=='judy'">
      You are transfering to Judy now.
    <vxml:elseif cond="st_field=='monty'"/>
      You are transfering to Monty now.
    <vxml:elseif cond="st_field=='solarman'"/>
      You are transfering to Solarman now.
    </vxml:if>
  </vxml:filled>
```

```
    <vxml:catch event="nomatch noinput">
      <vxml:assign name="sayit" expr="false"/>
      <vxml:prompt>Please choose from Judy,<vxml:break time="200ms"/>
Monty,<vxml:break time="200ms"/> or Solarman!</vxml:prompt>
      <vxml:prompt>For example, say <vxml:break time="500ms"/>i wanna talk to
Judy</vxml:prompt>
      <vxml:reprompt/>
    </vxml:catch>
    <vxml:catch event="help">
      <vxml:assign name="sayit" expr="false"/>
      <vxml:prompt>Who do you wanna talk to? Please choose from Judy,<vxml:break
time="300ms"/> Monty,
      <vxml:break time="300ms"/> or Solarman!</vxml:prompt>
      <vxml:prompt>For example, say <vxml:break time="500ms"/>i wanna talk to
Judy</vxml:prompt>
      <vxml:reprompt/>
    </vxml:catch>
</vxml:field>
</vxml:form>

<script type="text/javascript">

var name="";
var loc="";

function loadPage(checkInput)
{
  /** Get user's input. **/
  loc = document.getElementById("id_nextPage").value;

  /** if user's input is empty, then return a error message. **/
  if(loc=="")
  {
    alert("SYSTEM ERROR: Please input the URL to your voice page in the above text
field. It can not be empty!" );
  }
  /** if user input is not empty, and user asked to validate URL before go. **/
  else if(checkInput==true)
```

```
  {

    submitReq("GET", loc);
    /** if the valicating process return a false as result, which means invalid URL. **/
    if(answerRecieved==false)
    {
      if(xmlhttp.status==404)
        alert("SYSTEM ERROR: Unable to load your voice page. File does not exist at:
"+ loc );
      else
        alert("SYSTEM ERROR: Unable to load your voice page. Network problem, error
code: "
        +xmlhttp.status+". Please check your internet connection.");
    }
    /** if user's input is not empty, and it is a valid URL to next page. **/
    else
      window.location=loc;
  }
  /** if user's input is not empty, and user asked to load URL page immediately. **/
  else
    window.location=loc;
}

/**********************************************************************************************
This function uses AJAX, it will submit the question to the given URI if it use a 'POST'
method.
Or, it will retrieve data from the given URI if it use a 'GET' method.
**********************************************************************************************/
function submitReq(method, url)
{
  /***** Initialize AJAX XMLHttpRequest object. ****/
  xmlhttp=new XMLHttpRequest();
  /******
  Assign a event listener to the 'onreadystatechange' event.
  Different listerner assigned depends on a 'GET' or a 'POST' method.
  ******/
  if(method=="GET")
    xmlhttp.onreadystatechange=stateChange_GET;
```

```
  else
    xmlhttp.onreadystatechange=stateChange_POST;

  /** Check whether the url involves a cross-domain security error before send the
request. **/
  if(isCrossDomain(url)==true)
  {
    alert("Cannot validate input URL since it involve a cross-domain security issue. Load
URL immediately.");
    answerRecieved=true;
    return;
  }


  /** Open the connect, sychronized.   ***/
    xmlhttp.open(method,url,false);


  if(method=="GET")
    xmlhttp.send();
  else
  {
    xmlhttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    xmlhttp.send("question="+question);
  }
}

function stateChange_POST()
{
  /******* if xmlhttp shows loaded   ******/
  if (xmlhttp.readyState==4)
  {
    if (xmlhttp.status==200 || xmlhttp.status==304)
    {
      answerRecieved = true;
    }
    else
    {
```

```
        answerRecieved = false;
        alert("SYSTEM ERROR: Network problem. Unable to retrieve data from:" + loc+".\n
Error code:"
        + xmlhttp.status);
        xmlhttp.responseText = "";
      }
    }
}

function stateChange_GET()
{
   /******* if xmlhttp shows loaded   ******/
   if (xmlhttp.readyState==4)
   {
      if (xmlhttp.status==200 || xmlhttp.status==304)
      {
         answerRecieved = true;
      }
      else
      {
         answerRecieved = false;
         alert("SYSTEM ERROR: Problem retrieving data from:" + loc+".\n Error code:" +
xmlhttp.status);
         xmlhttp.responseText = "";
      }
   }
}

/**
Validate the given url with the current page domain(hostname),
to see whether they are in the same domain or cross-domain(different domain).
**/
function isCrossDomain(url)
{
   var domain = url;
   var i = domain.indexOf("//");
   if(i==-1)
      return false;
```

```
  domain = domain.slice(i+2);

  var k = domain.indexOf("/");
  if(k!=-1)
    domain = domain.slice(0, k);
  else
    return true;

  var host = window.location.hostname;

  if(host==domain)
    return false;
  else
    return true;
}

</script>

<script type="text/javascript" id="gotoNextPage" declare="declared">

window.location="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/"+name+"/"+n
ame+".xml";
</script>

<title>Public-Domain SpeechWeb Menu Page</title>
</head>
<body id="page.body">
<center><h2>Welcome to our new Public-Domain SpeechWeb Browser!</h2></center>
<br/>
<center>
<table>
<tr><td colspan="6">Load your own speech application :
<input type="text" id="id_nextPage" size="50"
value="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/judy/judy.xml"/>
<br/><br/></td></tr>
<!-- Call loadPage() function to setup the interprter and speech-grammar location
according to the above input text field value; -->
```

```
<tr><td colspan="3"><input type="button" name="submitValidate" value="Validate Before
Go" onclick="loadPage(true)"/></td>
<td colspan="3"><input type="button" name="submitGo" value="Go Immediately"
onclick="loadPage(false)"/></td></tr>
<tr><td colspan="6"><br/><br/><br/><br/></td></tr>
<tr><td colspan="6" align="left">
<h3>Choose the Sihlo to start talking: </h3>
</td></tr>
<tr><td colspan="2"><a
href="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/judy/judy.xml">Judy</a></t
d>
<td colspan="2"><a
href="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/monty/monty.xml">Monty<
/a></td>
<td colspan="2"><a
href="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/solarman/solarman.xml">
Solar Man</a></td></tr>
<tr><td colspan="6"><br/><br/><br/><br/></td></tr>
<tr><td colspan="6" align="left"><h3>Installation Document and User
Manual:</h3></td></tr>
<tr><td colspan="6"><a
href="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/doc/Multi-page Version
Public-domain SpeechWeb Browser.pdf">
 Public-domain SpeechWeb Browser User Manual</a></td></tr>

</table>
<br/><br/>
<table id="logFrame" width="60%"></table><br/>
<br/><br/><br/><br/>
</center>
</body>
<!-- Call a script to goto next page when the current voice page is done. -->
<ev:listener ev:observer="page.body" ev:event="vxmldone" ev:handler="#gotoNextPage"
ev:propagate="stop" />
<!-- Load 'vxml_form' when the page.body loaded. -->
<ev:listener ev:observer="page.body" ev:event="load" ev:handler="#vxml_form"
ev:propagate="stop" />

</html>
```

Multiple-Page version SpeechWeb Browser, sample application-associated X+V page:

Judy Sihlo, "judy.xml"

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//VoiceXML Forum//DTD XHTML+Voice 1.2//EN"
"http://www.voicexml.org/specs/multimodal/x+v/12/dtd/xhtml+voice12.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:vxml="http://www.w3.org/2001/vxml"
      xmlns:ev="http://www.w3.org/2001/xml-events"
      xmlns:xv="http://www.voicexml.org/2002/xhtml+voice"
      xml:lang="en-US">

<!--**********************************************************************************
Date: March. 2006
Developer: Ma, Xiaoli(William)
Architecture: LRRP (Dr.Frost, University of Windsor, Canada)
**********************************************************************************-->

<head>

<!-- Please modify the value of variable sv_greeting, and cgiLink to fit your application. -->
<script type="text/javascript">
/** The greeting message that will say to the user, only at the first time the user visits this
page. **/
var sv_greeting ="hi, my name is judy.";
/** The link to your CGI interpreter location.
Notice, you have to place the CGI interpreter program with this page in the same domain
to prevent a cross-domain security error.**/
var cgiLink="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/judy/judy.cgi";
</script>

<!-- VoiceXML form. -->
<vxml:form id="vxml_form">
<vxml:field name="st_field" xv:id="voice_input" modal="true">
    <!-- NOTICE!!! PLEASE MODIFY THE VALUE OF 'src' ATTRIBUTE IN THE NEXT
LINE <grammar> ELEMENT TO YOUR GRAMMAR FILE LOCATION.-->
```

```
    <vxml:grammar type="application/x-jsgf"
src="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/judy/judy.jsgf" />
    <!-- The following greeting will only speak out when user connects to a new interpreter.
-->
       <vxml:prompt cond="sayGreetings==true"><vxml:break
time="500ms"/><vxml:value expr="sv_greeting"/>
       <vxml:value expr="updateShowFrame('GREETING:
'+sv_greeting);"/></vxml:prompt>
    <vxml:filled>
 <!--**********************************
   This "filled" element will be run after user speech input has recognized.
   Inside this element, first step, i have assign the user input to the variable 'question',
   because VoiceXML code can access a JavaScript defined variable, but JavaScript can
not see a VoiceXML defined variable.
   Then, in the next step, i call a JavaScript function "runCode()' to proceed AJAX submit
process.
**********************************-->
       <vxml:assign name="question" expr="st_field"/>
       <!--**********************
         Calls to javascript mainControl() function to do the logical process based on user
voice input.
       ************************-->
       <vxml:assign name="javacode" expr="mainControl();"/>
       <vxml:prompt><vxml:break time="300ms"/><vxml:value
expr="answer"/></vxml:prompt>
<!-- If the answer is not a link to next interpreter, then repeat the voice dialog. -->
       <vxml:if cond="isLink==false">
         <vxml:throw event="repeat.st_field"/>
       </vxml:if>
    </vxml:filled>
    <vxml:catch event="nomatch noinput">
       <vxml:prompt>Sorry, I don't understand, can you say it again?</vxml:prompt>
       <vxml:reprompt/>
    </vxml:catch>
    <vxml:catch event="help">
       No help is available! Restart the dialog!
       <vxml:clear namelist="st_field"/>
       <vxml:reprompt/>
    </vxml:catch>
 </vxml:field>

 <!-- Catch the 'repeat.st_field' event. -->
 <vxml:catch event="repeat.st_field">
   <vxml:clear namelist="st_field"/>
```

```
<!-- Restart the voice form without change the speech grammar. -->
  <vxml:reprompt/>
</vxml:catch>
</vxml:form>


<script type="text/javascript">
/******* Declare global variables shared by JavaScript and VoiceXML *********/
var sayGreetings=true;
var defaultGreetingMsg="Hi, i'm ready to talk now.";
/*** The location of next remote speech-application/CGI-application interpeter. ***/
var nextPage="";
/*** Question query recognized from user's speech (request). ***/
var question="";
/*** Answer query returned from remote CGI interpter (response). ***/
var answer="";
var answerRecieved=false;
/*** Answer query contains a link to next CGI interpreter. ***/
var isLink=false;
var gotoNext= false;
/*** This variable needed for VXML to call JavaScript code. ***/
var javacode="";
/*** menu page of the demo public-domain speechweb. ***/
var
startPage="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/menu/demo_menu.x
ml";

if(sv_greeting=="")
  sv_greeting=defaultGreetingMsg;


/*****************************************************************************************
This is the main control function to the whole question submit and answer retrived
procedures.
It will call submitReq() method to send the question to the CGI program.
then it will check the answer whether it is a link to new CGI program or a simply answer
string.
if it is a link to another interpreter, then retrieve the data from there,
and call the 'changeData' function to change the neccessary information for the next
round dialog.
*****************************************************************************************/
function mainControl()
{
  updateShowFrame("QUESTION: "+question+"<br/>");
```

```
  answer="";
  answerRecieved=false;
  isLink=false;
  sayGreetings=false;

  /* call submitReq() method to send the question to the CGI program. */
  submitReq("POST", cgiLink);
  /** Cannot recieve data from CGI interpreter. Network problem. **/
  if(answerRecieved==false)
    return "-1";
  answer = getAnswer(xmlhttp.responseText);

  /*******
  Check whether the recieved answer is a link or not.
  And, assign the result to the global variable isLink.
  ******/
  checkAnswer(xmlhttp.responseText);

  /****** if the answer is not a link, then show the answer to the user and return. ******/
  if(!isLink)
  {
    gotoNext=false;
    updateShowFrame("RESPONSE: "+answer+"<br/>");
    return "1";
  }

  nextPage=getNextInterpreter(xmlhttp.responseText);

  updateShowFrame("RESPONSE: "+answer+"<br/><br/>");

  if(gotoNext==true)
    window.location=nextPage;

  return "1";
}


/*********************************************************************************************
This function returns the substring that has to be spoken as a result of the user's question.
Same procedure is applied for extracting the content to be spoken out.
*********************************************************************************************/
function getAnswer(answer)
{
```

```
  var ex=answer;
  var index;
  if((ex.indexOf('LINK=',0)) == -1)
    return ex;
  ex= ex.slice(5);
  index = ex.indexOf(";",0);
  ex = ex.substring(0,index);
  return ex;
}


/*************************************************************************************************
This function uses AJAX, it will submit the question to the given URI if it use a 'POST'
method.
Or, it will retrieve data from the given URI if it use a 'GET' method.
*************************************************************************************************/
function submitReq(method, url)
{
  /***** Initialize AJAX XMLHttpRequest object. ****/
  xmlhttp=new XMLHttpRequest();
  /******
  Assign a event listener to the 'onreadystatechange' event.
  Different listerner assigned depends on a 'GET' or a 'POST' method.
  ******/
  if(method=="GET")
    xmlhttp.onreadystatechange=stateChange_GET;
  else
    xmlhttp.onreadystatechange=stateChange_POST;


  /** Check whether the url involves a cross-domain security error before send the
request. **/
  if(isCrossDomain(url)==true)
  {
    /** if method is 'GET', it means this function is called from loadPage() function to
validate a user input URL. **/
    if(method=="GET")
      alert("Cannot validate input URL since it involves a cross-domain security issue.
Load URL immediately.");
    /**
    if method is not 'GET', which means 'POST' method,
    it means this method is called from main control to submit a question query to the
interpreter.
    **/
```

```
    else
      updateShowFrame("SYSTEM ERROR: An error which against the web browser
cross-domain security issue."
                    +" Your CGI interpreter has to be placed in the same domain with this
voice page."+
                        "Please contact to your application provider to fix this problem.
\n"
                        +"Your CGI interpreter location: "+ url+"    Current voice page
host domain: "
                        +window.location.host);
    answerRecieved=true;
    answer="An error which against the web browser cross-domain security issue has
occured. Please check the error message to continue.";
    return;
  }

  /** Open the connect, sychronized.   ***/
  xmlhttp.open(method,url,false);

  if(method=="GET")
    xmlhttp.send();
  else
  {
    xmlhttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    xmlhttp.send("question="+question);
  }
}

function stateChange_POST()
{
  /******* if xmlhttp shows loaded   ******/
  if (xmlhttp.readyState==4)
  {
    if (xmlhttp.status==200 || xmlhttp.status==304)
    {
      answerRecieved = true;
    }
    else
    {
      answerRecieved = false;
      xmlhttp.responseText = "";
    }
  }
}
```

```
function stateChange_GET()
{
  /******* if xmlhttp shows loaded   ******/
  if (xmlhttp.readyState==4)
  {
    if (xmlhttp.status==200 || xmlhttp.status==304)
    {
      answerRecieved = true;
    }
    else
    {
      answerRecieved = false;
      xmlhttp.responseText = "";
    }
  }
}

/** Validate the given url with the current page domain(hostname), to see whether they are
in the same domain or cross-domain(different domain). **/
function isCrossDomain(url)
{
  var domain = url;
  var i = domain.indexOf("//");
  if(i==-1)
    return false;

  domain = domain.slice(i+2);

  var k = domain.indexOf("/");
  if(k!=-1)
    domain = domain.slice(0, k);
  else
    return true;

  var host = window.location.hostname;

  if(host==domain)
    return false;
  else
    return true;

}
```

```
/**************************************************************************************************
****
Check whether there is a occurrence of '=' character in the answer, which means a link
existed in it.
And, assign the result to the global variable 'isLink'.
**************************************************************************************************
**/
function checkAnswer(answer)
{
  if((answer.indexOf('LINK=',0))== -1)
    isLink=false;
  else
    isLink=true;

  return isLink;
}

/**************************************************************************************************
************
if the answer is a link, this function will return the next interpreter's URI as a string.
Otherwise, return "-1".
**************************************************************************************************
************/
function getNextInterpreter(answer)
{
  var loc;
  var ex = answer;
  var index;

/******
Check if the answer is a link to next speech application interpreter.
It should never be evaluated as true, otherwise error.
******/
  if(!isLink)
    return "-1";

/**************************************************************************************************
******************
If the answer is a link, then its formation should be: "LINK=_answer;SIHLO=_location;".
e.g. Question send to judy.cgi: "can i talk to solar man".
      Answer recieved from judy.cgi: "LINK=yes. here he
is;SIHLO=http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/judy/judy.xml"
**************************************************************************************************
****************/
```

```
/****** 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
answer-query 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(";",0);
    loc = ex.substring(0,index);

    return loc;
}

/** Update the text area in the HTML and show message on it. **/
function updateShowFrame(message)
{
  var objTable = document.getElementById("logFrame");

  objTable.insertRow(0);
  objTable.rows[0].insertCell(0);
  objTable.rows[0].insertCell(1);
  var cell0 = objTable.rows[0].cells[0];

  var cell1 = objTable.rows[0].cells[1];
  cell1.align="left";
  cell0.align="left";
  cell0.width="105";
  if(message.indexOf("SYSTEM ERROR: ")!=-1)
  {
    var objFont = document.createElement("font");
    objFont.color="red";
    objFont.size="-1";
    objFont.appendChild(document.createElement("b"));
```

```
    objFont.firstChild.innerHTML = message.slice(0, message.indexOf(":")+1);
    cell0.appendChild(objFont);

    var objFont2 = document.createElement("font");
    var objIta = document.createElement("i");
    objFont2.color="black";
    objFont2.size="-1";
    objFont2.appendChild(objIta);
    cell1.appendChild(objFont2);
    objIta.innerHTML=message.slice(message.indexOf(":")+1);
  }else
  {
    var index = message.indexOf(":");
    var ex=message.slice(0,index+1);
    var objFont = document.createElement("font");
    if(ex.indexOf("QUESTION:")!=-1)
      objFont.color= "blue";
    else if(ex.indexOf("RESPONSE:")!=-1)
      objFont.color="green";
    else
      objFont.color="purple";

    objFont.appendChild(document.createElement("b"));
    cell0.appendChild(objFont);
    objFont.firstChild.innerHTML = ex;
    cell1.appendChild(document.createElement("font"));
    cell1.firstChild.innerHTML = message.slice(index+1);
  }
  /****** Insert a table row as an empty line after a response and greeting message.
***********/
  if(message.indexOf("QUESTION")==-1)
  {
    objTable.insertRow(0);
    objTable.rows[0].insertCell(0);
    objTable.rows[0].colspan="2";
    objTable.rows[0].cells[0].innerHTML = "<br/> ";
  }

  return "";
}

/** Load user's application. **/
function loadPage(checkInput)
{
```

```
  /** Get user's input. **/
  var loc = document.getElementById("id_nextPage").value;

  /** if user's input is empty, then return a error message. **/
  if(loc=="")
  {
    updateShowFrame("SYSTEM ERROR: Please input the URL to your voice page in
the above text field. It can not be empty!" );
  }
  /** if user input is not empty, and user asked to validate URL before go. **/
  else if(checkInput==true)
  {
    submitReq("GET", loc);
    /** if the valicating process return a false as result, which means invalid URL. **/
    if(answerRecieved==false)
    {
      if(xmlhttp.status==404)
        updateShowFrame("SYSTEM ERROR: Unable to load your voice page. File
does not exist at: "+ loc );
      else
        updateShowFrame("SYSTEM ERROR: Unable to load your voice page. Network
problem, error code: "
        +xmlhttp.status+". Please check your internet connection.");
    }
    /** if user's input is not empty, it is a valid URL to next page. **/
    else
      window.location=loc;
  }
  /** if user's input is not empty, and user asked to load URL page immediately. **/
  else
    window.location=loc;
}

function menuPage()
{
  window.location=startPage;
}

function processQuestion()
{
  gotoNext = true;
  question = document.getElementById("id_questionField").value;
  document.getElementById("id_questionField").value="";
  mainControl();
```

```
}

</script>

<!--*******************************************************
  The following script will only be run after a 'vxmldone' event is thrown after the
VoiceXML form finish all its process.
  It also means that the answer returned from interpreter contains a link to next interpreter,
so it needs to go there.
*******************************************************-->
<script type="text/javascript" id="gotoNextPage" declare="declare">
  window.location=nextPage;
</script>

<title>Public-Domain SpeechWeb</title>
</head>
<body id="page.body">
<center><h2>Welcome to our new voice browser!</h2></center>
<br/>
<center>
<table>
<tr><td colspan="6">Load your own speech application :
<input type="text" id="id_nextPage" size="50"
value="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/menu/demo_menu.xml"/
>
<br/><br/></td></tr>
<!-- Call loadPage() function to setup the interprter and speech-grammar location
according to the above input text field value; -->
<tr><td colspan="2"><input type="button" name="submitValidate" value="Validate Before
Go" onclick="loadPage(true)"/></td>
<td colspan="2"><input type="button" name="submitGo" value="Go Immediately"
onclick="loadPage(false)"/></td>
<td colspan="2"><input type="button" name="menuGo" value="SpeechWeb Menu Page"
onclick="menuPage()"/></td></tr>
<tr><td colspan="6"><br/><br/><br/><b>Say your question or type it in
here:</b></td></tr>
<tr><td colspan="6"><form onsubmit="processQuestion(); return false;">
<input type="text" size="70" name="questionField" id="id_questionField"
value=""/></form></td></tr>
</table>
<br/><br/>
<table id="logFrame" width="600"></table><br/>
<br/><br/><br/>
</center>
```

```
</body>
<!-- Call a script to reload the vxml form when the current vxml form has done its process.
-->
<ev:listener ev:observer="page.body" ev:event="vxmldone" ev:handler="#gotoNextPage"
ev:propagate="stop" />
<!-- Load 'vxml_form' when the page.body loaded. -->
<ev:listener ev:observer="page.body" ev:event="load" ev:handler="#vxml_form"
ev:propagate="stop" />
</html>
```

Monty Sihlo, "monty.xml"

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//VoiceXML Forum//DTD XHTML+Voice 1.2//EN"
"http://www.voicexml.org/specs/multimodal/x+v/12/dtd/xhtml+voice12.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:vxml="http://www.w3.org/2001/vxml"
      xmlns:ev="http://www.w3.org/2001/xml-events"
      xmlns:xv="http://www.voicexml.org/2002/xhtml+voice"
      xml:lang="en-US">

<!--**********************************************************************************
Date: March. 2006
Developer: Ma, Xiaoli(William)
Architecture: LRRP (Dr.Frost, University of Windsor, Canada)
**********************************************************************************-->

<head>

<!-- Please modify the value of variable sv_greeting, and cgiLink to fit your application. -->
<script type="text/javascript">
/** The greeting message that will say to the user, only at the first time the user visits this
page. **/
var sv_greeting ="hi, I am monty, i know a joke.";
/** The link to your CGI interpreter location.
Notice, you have to place the CGI interpreter program with this page in the same domain
to prevent a cross-domain security error.**/
var cgiLink="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/monty/monty.cgi";
</script>

<!-- VoiceXML form. -->
<vxml:form id="vxml_form">
```

```
<vxml:field name="st_field" xv:id="voice_input" modal="true">
    <!-- NOTICE!!! PLEASE MODIFY THE VALUE OF 'src' ATTRIBUTE IN THE NEXT
LINE <grammar> ELEMENT TO YOUR GRAMMAR FILE LOCATION.-->
    <vxml:grammar type="application/x-jsgf"
src="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/monty/monty.jsgf" />
    <!-- The following greeting will only speak out when user connects to a new interpreter.
-->
       <vxml:prompt cond="sayGreetings==true"><vxml:break
time="500ms"/><vxml:value expr="sv_greeting"/>
       <vxml:value expr="updateShowFrame('GREETING:
'+sv_greeting);"/></vxml:prompt>
    <vxml:filled>
 <!--**********************************
   This "filled" element will be run after user speech input has recognized.
   Inside this element, first step, i have assign the user input to the variable 'question',
   because VoiceXML code can access a JavaScript defined variable, but JavaScript can
not see a VoiceXML defined variable.
   Then, in the next step, i call a JavaScript function "runCode()' to proceed AJAX submit
process.
**********************************-->
       <vxml:assign name="question" expr="st_field"/>
       <!--**********************
          Calls to javascript mainControl() function to do the logical process based on user
voice input.
       ************************-->
       <vxml:assign name="javacode" expr="mainControl();"/>
       <vxml:prompt><vxml:break time="300ms"/><vxml:value
expr="answer"/></vxml:prompt>
<!-- If the answer is not a link to next interpreter, then repeat the voice dialog. -->
       <vxml:if cond="isLink==false">
          <vxml:throw event="repeat.st_field"/>
       </vxml:if>
    </vxml:filled>
    <vxml:catch event="nomatch noinput">
       <vxml:prompt>Sorry, I don't understand, can you say it again?</vxml:prompt>
       <vxml:reprompt/>
    </vxml:catch>
    <vxml:catch event="help">
       No help is available! Restart the dialog!
       <vxml:clear namelist="st_field"/>
       <vxml:reprompt/>
    </vxml:catch>
</vxml:field>
```

```
<!-- Catch the 'repeat.st_field' event. -->
<vxml:catch event="repeat.st_field">
  <vxml:clear namelist="st_field"/>
<!-- Restart the voice form without change the speech grammar. -->
  <vxml:reprompt/>
</vxml:catch>
</vxml:form>


<script type="text/javascript">
/******* Declare global variables shared by JavaScript and VoiceXML *********/
var sayGreetings=true;
var defaultGreetingMsg="Hi, i'm ready to talk now.";
/*** The location of next remote speech-application/CGI-application interpeter. ***/
var nextPage="";
/*** Question query recognized from user's speech (request). ***/
var question="";
/*** Answer query returned from remote CGI interpter (response). ***/
var answer="";
var answerRecieved=false;
/*** Answer query contains a link to next CGI interpreter. ***/
var isLink=false;
var gotoNext= false;
/*** This variable needed for VXML to call JavaScript code. ***/
var javacode="";
/*** menu page of the demo public-domain speechweb. ***/
var
startPage="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/menu/demo_menu.x
ml";

if(sv_greeting=="")
  sv_greeting=defaultGreetingMsg;


/**********************************************************************************************
This is the main control function to the whole question submit and answer retrived
procedures.
It will call submitReq() method to send the question to the CGI program.
then it will check the answer whether it is a link to new CGI program or a simply answer
string.
if it is a link to another interpreter, then retrieve the data from there,
and call the 'changeData' function to change the neccessary information for the next
round dialog.
**********************************************************************************************/
```

```
function mainControl()
{
  updateShowFrame("QUESTION: "+question+"<br/>");

  answer="";
  answerRecieved=false;
  isLink=false;
  sayGreetings=false;

  /* call submitReq() method to send the question to the CGI program. */
  submitReq("POST", cgiLink);
  /** Cannot recieve data from CGI interpreter. Network problem. **/
  if(answerRecieved==false)
    return "-1";
  answer = getAnswer(xmlhttp.responseText);

  /******* Check whether the recieved answer is a link or not. And, assign the result to the
global variable isLink. ******/
  checkAnswer(xmlhttp.responseText);

  /****** if the answer is not a link, then show the answer to the user and return. ******/
  if(!isLink)
  {
    gotoNext=false;
    updateShowFrame("RESPONSE: "+answer+"<br/>");
    return "1";
  }

  nextPage=getNextInterpreter(xmlhttp.responseText);

  updateShowFrame("RESPONSE: "+answer+"<br/><br/>");

  if(gotoNext==true)
    window.location=nextPage;

  return "1";
}


/*************************************************************************************************
This function returns the substring that has to be spoken as a result of the user's question.
Same procedure is applied for extracting the content to be spoken out.
*************************************************************************************************/
function getAnswer(answer)
```

```
{
  var ex=answer;
  var index;
  if((ex.indexOf('LINK=',0)) == -1)
    return ex;
  ex= ex.slice(5);
  index = ex.indexOf(";",0);
  ex = ex.substring(0,index);
  return ex;
}


/*************************************************************************************************
This function uses AJAX, it will submit the question to the given URI if it use a 'POST'
method.
Or, it will retrieve data from the given URI if it use a 'GET' method.
*************************************************************************************************/
function submitReq(method, url)
{
  /***** Initialize AJAX XMLHttpRequest object. ****/
  xmlhttp=new XMLHttpRequest();
  /******
  Assign a event listener to the 'onreadystatechange' event.
  Different listerner assigned depends on a 'GET' or a 'POST' method.
  ******/
  if(method=="GET")
    xmlhttp.onreadystatechange=stateChange_GET;
  else
    xmlhttp.onreadystatechange=stateChange_POST;


  /** Check whether the url involves a cross-domain security error before send the
request. **/
  if(isCrossDomain(url)==true)
  {
    /** if method is 'GET', it means this function is called from loadPage() function to
validate a user input URL. **/
    if(method=="GET")
      alert("Cannot validate input URL since it involves a cross-domain security issue.
Load URL immediately.");
    /**
    if method is not 'GET', which means 'POST' method,
    it means this method is called from main control to submit a question query to the
interpreter.
```

```
    **/
    else
      updateShowFrame("SYSTEM ERROR: An error which against the web browser
cross-domain security issue."
                  +" Your CGI interpreter has to be placed in the same domain with this
voice page."+
                      "Please contact to your application provider to fix this problem.
\n"
                      +"Your CGI interpreter location: "+ url+"    Current voice page
host domain: "+window.location.host);
    answerRecieved=true;
    answer="An error which against the web browser cross-domain security issue has
occured. Please check the error message to continue.";
    return;
  }

  /** Open the connect, sychronized.   ***/
  xmlhttp.open(method,url,false);

  if(method=="GET")
    xmlhttp.send();
  else
  {
    xmlhttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    xmlhttp.send("question="+question);
  }
}

function stateChange_POST()
{
  /******* if xmlhttp shows loaded   ******/
  if (xmlhttp.readyState==4)
  {
    if (xmlhttp.status==200 || xmlhttp.status==304)
    {
      answerRecieved = true;
    }
    else
    {
      answerRecieved = false;
      xmlhttp.responseText = "";
    }
  }
}
```

```
function stateChange_GET()
{
  /******* if xmlhttp shows loaded   ******/
  if (xmlhttp.readyState==4)
  {
    if (xmlhttp.status==200 || xmlhttp.status==304)
    {
      answerRecieved = true;
    }
    else
    {
      answerRecieved = false;
      xmlhttp.responseText = "";
    }
  }
}

/** Validate the given url with the current page domain(hostname),
to see whether they are in the same domain or cross-domain(different domain). **/
function isCrossDomain(url)
{
  var domain = url;
  var i = domain.indexOf("//");
  if(i==-1)
    return false;

  domain = domain.slice(i+2);

  var k = domain.indexOf("/");
  if(k!=-1)
    domain = domain.slice(0, k);
  else
    return true;

  var host = window.location.hostname;

  if(host==domain)
    return false;
  else
    return true;

}
```

```
/**************************************************************************************************
****
Check whether there is a occurrence of '=' character in the answer, which means a link
existed in it.
And, assign the result to the global variable 'isLink'.
**************************************************************************************************
**/
function checkAnswer(answer)
{
  if((answer.indexOf('LINK=',0))== -1)
    isLink=false;
  else
    isLink=true;

  return isLink;
}

/**************************************************************************************************
************
if the answer is a link, this function will return the next interpreter's URI as a string.
Otherwise, return "-1".
**************************************************************************************************
*************/
function getNextInterpreter(answer)
{
  var loc;
  var ex = answer;
  var index;

/******
Check if the answer is a link to next speech application interpreter.
It should never be evaluated as true, otherwise error.
******/
  if(!isLink)
    return "-1";

/**************************************************************************************************
******************
If the answer is a link, then its formation should be: "LINK=_answer;SIHLO=_location;".
e.g. Question send to judy.cgi: "can i talk to solar man".
      Answer recieved from judy.cgi: "LINK=yes. here he
is;SIHLO=http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/judy/judy.xml"
**************************************************************************************************
****************/
```

```
/****** 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
answer-query 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(";",0);
    loc = ex.substring(0,index);

    return loc;
}

/** Update the text area in the HTML and show message on it. **/
function updateShowFrame(message)
{
  var objTable = document.getElementById("logFrame");

  objTable.insertRow(0);
  objTable.rows[0].insertCell(0);
  objTable.rows[0].insertCell(1);
  var cell0 = objTable.rows[0].cells[0];

  var cell1 = objTable.rows[0].cells[1];
  cell1.align="left";
  cell0.align="left";
  cell0.width="105";
  if(message.indexOf("SYSTEM ERROR: ")!=-1)
  {
    var objFont = document.createElement("font");
    objFont.color="red";
    objFont.size="-1";
    objFont.appendChild(document.createElement("b"));
```

```
    objFont.firstChild.innerHTML = message.slice(0, message.indexOf(":")+1);
    cell0.appendChild(objFont);

    var objFont2 = document.createElement("font");
    var objIta = document.createElement("i");
    objFont2.color="black";
    objFont2.size="-1";
    objFont2.appendChild(objIta);
    cell1.appendChild(objFont2);
    objIta.innerHTML=message.slice(message.indexOf(":")+1);
  }else
  {
    var index = message.indexOf(":");
    var ex=message.slice(0,index+1);
    var objFont = document.createElement("font");
    if(ex.indexOf("QUESTION:")!=-1)
      objFont.color= "blue";
    else if(ex.indexOf("RESPONSE:")!=-1)
      objFont.color="green";
    else
      objFont.color="purple";

    objFont.appendChild(document.createElement("b"));
    cell0.appendChild(objFont);
    objFont.firstChild.innerHTML = ex;
    cell1.appendChild(document.createElement("font"));
    cell1.firstChild.innerHTML = message.slice(index+1);
  }
  /****** Insert a table row as an empty line after a response and greeting message.
**********/
  if(message.indexOf("QUESTION")==-1)
  {
    objTable.insertRow(0);
    objTable.rows[0].insertCell(0);
    objTable.rows[0].colspan="2";
    objTable.rows[0].cells[0].innerHTML = "<br/> ";
  }

  return "";
}

/** Load user's application. **/
function loadPage(checkInput)
{
```

```
  /** Get user's input. **/
  var loc = document.getElementById("id_nextPage").value;

  /** if user's input is empty, then return a error message. **/
  if(loc=="")
  {
    updateShowFrame("SYSTEM ERROR: Please input the URL to your voice page in
the above text field. It can not be empty!" );
  }
  /** if user input is not empty, and user asked to validate URL before go. **/
  else if(checkInput==true)
  {
    submitReq("GET", loc);
    /** if the valicating process return a false as result, which means invalid URL. **/
    if(answerRecieved==false)
    {
      if(xmlhttp.status==404)
        updateShowFrame("SYSTEM ERROR: Unable to load your voice page. File
does not exist at: "+ loc );
      else
        updateShowFrame("SYSTEM ERROR: Unable to load your voice page. Network
problem, error code: "
        +xmlhttp.status+". Please check your internet connection.");
    }
    /** if user's input is not empty, it is a valid URL to next page. **/
    else
      window.location=loc;
  }
  /** if user's input is not empty, and user asked to load URL page immediately. **/
  else
    window.location=loc;
}

function menuPage()
{
  window.location=startPage;
}

function processQuestion()
{
  gotoNext = true;
  question = document.getElementById("id_questionField").value;
  document.getElementById("id_questionField").value="";
  mainControl();
```

```
}

</script>

<!--*******************************************************
  The following script will only be run after a 'vxmldone' event is thrown after the
VoiceXML form finish all its process.
  It also means that the answer returned from interpreter contains a link to next interpreter,
so it needs to go there.
*******************************************************-->
<script type="text/javascript" id="gotoNextPage" declare="declare">
  window.location=nextPage;
</script>

<title>Public-Domain SpeechWeb</title>
</head>
<body id="page.body">
<center><h2>Welcome to our new voice browser!</h2></center>
<br/>
<center>
<table>
<tr><td colspan="6">Load your own speech application :
<input type="text" id="id_nextPage" size="50"
value="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/menu/demo_menu.xml"/
>
<br/><br/></td></tr>
<!-- Call loadPage() function to setup the interprter and speech-grammar location
according to the above input text field value; -->
<tr><td colspan="2"><input type="button" name="submitValidate" value="Validate Before
Go" onclick="loadPage(true)"/></td>
<td colspan="2"><input type="button" name="submitGo" value="Go Immediately"
onclick="loadPage(false)"/></td>
<td colspan="2"><input type="button" name="menuGo" value="SpeechWeb Menu Page"
onclick="menuPage()"/></td></tr>
<tr><td colspan="6"><br/><br/><br/><b>Say your question or type it in
here:</b></td></tr>
<tr><td colspan="6"><form onsubmit="processQuestion(); return false;">
<input type="text" size="70" name="questionField" id="id_questionField"
value=""/></form></td></tr>
</table>
<br/><br/>
<table id="logFrame" width="600"></table><br/>
<br/><br/><br/>
</center>
```

```
</body>
<!-- Call a script to reload the vxml form when the current vxml form has done its process.
-->
<ev:listener ev:observer="page.body" ev:event="vxmldone" ev:handler="#gotoNextPage"
ev:propagate="stop" />
<!-- Load 'vxml_form' when the page.body loaded. -->
<ev:listener ev:observer="page.body" ev:event="load" ev:handler="#vxml_form"
ev:propagate="stop" />
</html>
```

Solarman Sihlo, "solarman.xml"

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//VoiceXML Forum//DTD XHTML+Voice 1.2//EN"
"http://www.voicexml.org/specs/multimodal/x+v/12/dtd/xhtml+voice12.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:vxml="http://www.w3.org/2001/vxml"
      xmlns:ev="http://www.w3.org/2001/xml-events"
      xmlns:xv="http://www.voicexml.org/2002/xhtml+voice"
      xml:lang="en-US">

<!--**************************************************************************************
Date: March. 2006
Developer: Ma, Xiaoli(William)
Architecture: LRRP (Dr.Frost, University of Windsor, Canada)
**************************************************************************************-->

<head>

<!-- Please modify the value of variable sv_greeting, and cgiLink to fit your application. -->
<script type="text/javascript">
/** The greeting message that will say to the user, only at the first time the user visits this
page. **/
var sv_greeting ="hi, I am solar man.";
/** The link to your CGI interpreter location.
Notice, you have to place the CGI interpreter program with this page in the same domain
to prevent a cross-domain security error.**/
var
cgiLink="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/solarman/solarman.cgi"
;
</script>
```

```
<!-- VoiceXML form. -->
<vxml:form id="vxml_form">
<vxml:field name="st_field" xv:id="voice_input" modal="true">
    <!-- NOTICE!!! PLEASE MODIFY THE VALUE OF 'src' ATTRIBUTE IN THE NEXT
LINE <grammar> ELEMENT TO YOUR GRAMMAR FILE LOCATION.-->
    <vxml:grammar type="application/x-jsgf"
src="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/solarman/solarman.jsgf" />
    <!-- The following greeting will only speak out when user connects to a new interpreter.
-->
      <vxml:prompt cond="sayGreetings==true"><vxml:break time="500ms"/>
      <vxml:value expr="sv_greeting"/><vxml:value
expr="updateShowFrame('GREETING: '+sv_greeting);"/></vxml:prompt>
    <vxml:filled>
 <!--**********************************
   This "filled" element will be run after user speech input has recognized.
   Inside this element, first step, i have assign the user input to the variable 'question',
   because VoiceXML code can access a JavaScript defined variable, but JavaScript can
not see a VoiceXML defined variable.
   Then, in the next step, i call a JavaScript function "runCode()' to proceed AJAX submit
process.
**********************************-->
      <vxml:assign name="question" expr="st_field"/>
      <!--**********************
        Calls to javascript mainControl() function to do the logical process based on user
voice input.
      ************************-->
      <vxml:assign name="javacode" expr="mainControl();"/>
      <vxml:prompt><vxml:break time="300ms"/><vxml:value
expr="answer"/></vxml:prompt>
<!-- If the answer is not a link to next interpreter, then repeat the voice dialog. -->
      <vxml:if cond="isLink==false">
        <vxml:throw event="repeat.st_field"/>
      </vxml:if>
    </vxml:filled>
    <vxml:catch event="nomatch noinput">
      <vxml:prompt>Sorry, I don't understand, can you say it again?</vxml:prompt>
      <vxml:reprompt/>
    </vxml:catch>
    <vxml:catch event="help">
      No help is available! Restart the dialog!
      <vxml:clear namelist="st_field"/>
      <vxml:reprompt/>
    </vxml:catch>
</vxml:field>
```

```
<!-- Catch the 'repeat.st_field' event. -->
<vxml:catch event="repeat.st_field">
  <vxml:clear namelist="st_field"/>
<!-- Restart the voice form without change the speech grammar. -->
  <vxml:reprompt/>
</vxml:catch>
</vxml:form>


<script type="text/javascript">
/******* Declare global variables shared by JavaScript and VoiceXML *********/
var sayGreetings=true;
var defaultGreetingMsg="Hi, i'm ready to talk now.";
/*** The location of next remote speech-application/CGI-application interpeter. ***/
var nextPage="";
/*** Question query recognized from user's speech (request). ***/
var question="";
/*** Answer query returned from remote CGI interpter (response). ***/
var answer="";
var answerRecieved=false;
/*** Answer query contains a link to next CGI interpreter. ***/
var isLink=false;
var gotoNext= false;
/*** This variable needed for VXML to call JavaScript code. ***/
var javacode="";
/*** menu page of the demo public-domain speechweb. ***/
var
startPage="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/menu/demo_menu.x
ml";

if(sv_greeting=="")
  sv_greeting=defaultGreetingMsg;


/*****************************************************************************************
This is the main control function to the whole question submit and answer retrived
procedures.
It will call submitReq() method to send the question to the CGI program.
then it will check the answer whether it is a link to new CGI program or a simply answer
string.
if it is a link to another interpreter, then retrieve the data from there,
and call the 'changeData' function to change the neccessary information for the next
round dialog.
```

```
*****************************************************************************************/
function mainControl()
{
  updateShowFrame("QUESTION: "+question+"<br/>");

  answer="";
  answerRecieved=false;
  isLink=false;
  sayGreetings=false;

  /* call submitReq() method to send the question to the CGI program. */
  submitReq("POST", cgiLink);
  /** Cannot recieve data from CGI interpreter. Network problem. **/
  if(answerRecieved==false)
    return "-1";
  answer = getAnswer(xmlhttp.responseText);

  /******* Check whether the recieved answer is a link or not. And, assign the result to the
global variable isLink. ******/
  checkAnswer(xmlhttp.responseText);

  /****** if the answer is not a link, then show the answer to the user and return. ******/
  if(!isLink)
  {
    gotoNext=false;
    updateShowFrame("RESPONSE: "+answer+"<br/>");
    return "1";
  }

  nextPage=getNextInterpreter(xmlhttp.responseText);

  updateShowFrame("RESPONSE: "+answer+"<br/><br/>");

  if(gotoNext==true)
    window.location=nextPage;

  return "1";
}


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

```
function getAnswer(answer)
{
  var ex=answer;
  var index;
  if((ex.indexOf('LINK=',0)) == -1)
    return ex;
  ex= ex.slice(5);
  index = ex.indexOf(";",0);
  ex = ex.substring(0,index);
  return ex;
}


/*********************************************************************************************
This function uses AJAX, it will submit the question to the given URI if it use a 'POST'
method.
Or, it will retrieve data from the given URI if it use a 'GET' method.
*********************************************************************************************/
function submitReq(method, url)
{
  /***** Initialize AJAX XMLHttpRequest object. ****/
  xmlhttp=new XMLHttpRequest();
  /******
  Assign a event listener to the 'onreadystatechange' event.
  Different listerner assigned depends on a 'GET' or a 'POST' method.
  ******/
  if(method=="GET")
    xmlhttp.onreadystatechange=stateChange_GET;
  else
    xmlhttp.onreadystatechange=stateChange_POST;


  /** Check whether the url involves a cross-domain security error before send the
request. **/
  if(isCrossDomain(url)==true)
  {
    /** if method is 'GET', it means this function is called from loadPage() function to
validate a user input URL. **/
    if(method=="GET")
      alert("Cannot validate input URL since it involves a cross-domain security issue.
Load URL immediately.");
    /**
    if method is not 'GET', which means 'POST' method,
    it means this method is called from main control to submit a question query to the
```

```
interpreter.
      **/
      else
        updateShowFrame("SYSTEM ERROR: An error which against the web browser
cross-domain security issue."
                    +" Your CGI interpreter has to be placed in the same domain with this
voice page."+
                          "Please contact to your application provider to fix this problem.
\n"
                          +"Your CGI interpreter location: "+ url+"    Current voice page
host domain: "+window.location.host);
      answerRecieved=true;
      answer="An error which against the web browser cross-domain security issue has
occured. Please check the error message to continue.";
      return;
    }

    /** Open the connect, sychronized.   ***/
    xmlhttp.open(method,url,false);

    if(method=="GET")
      xmlhttp.send();
    else
    {
      xmlhttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
      xmlhttp.send("question="+question);
    }
}

function stateChange_POST()
{
    /******* if xmlhttp shows loaded   ******/
    if (xmlhttp.readyState==4)
    {
      if (xmlhttp.status==200 || xmlhttp.status==304)
      {
        answerRecieved = true;
      }
      else
      {
        answerRecieved = false;
        xmlhttp.responseText = "";
      }
    }
```

```
}

function stateChange_GET()
{
  /******* if xmlhttp shows loaded  ******/
  if (xmlhttp.readyState==4)
  {
    if (xmlhttp.status==200 || xmlhttp.status==304)
    {
      answerRecieved = true;
    }
    else
    {
      answerRecieved = false;
      xmlhttp.responseText = "";
    }
  }
}

/**
Validate the given url with the current page domain(hostname),
to see whether they are in the same domain or cross-domain(different domain).
**/
function isCrossDomain(url)
{
  var domain = url;
  var i = domain.indexOf("//");
  if(i==-1)
    return false;

  domain = domain.slice(i+2);

  var k = domain.indexOf("/");
  if(k!=-1)
    domain = domain.slice(0, k);
  else
    return true;

  var host = window.location.hostname;

  if(host==domain)
    return false;
  else
    return true;
```

```
}

/**************************************************************************************************
****
Check whether there is a occurrence of '=' character in the answer, which means a link
existed in it.
And, assign the result to the global variable 'isLink'.
**************************************************************************************************
**/
function checkAnswer(answer)
{
  if((answer.indexOf('LINK=',0))== -1)
    isLink=false;
  else
    isLink=true;

  return isLink;
}

/**************************************************************************************************
************
if the answer is a link, this function will return the next interpreter's URI as a string.
Otherwise, return "-1".
**************************************************************************************************
*************/
function getNextInterpreter(answer)
{
  var loc;
  var ex = answer;
  var index;

/******
Check if the answer is a link to next speech application interpreter.
It should never be evaluated as true, otherwise error.
******/
  if(!isLink)
    return "-1";

/**************************************************************************************************
******************
If the answer is a link, then its formation should be: "LINK=_answer;SIHLO=_location;".
e.g. Question send to judy.cgi: "can i talk to solar man".
      Answer recieved from judy.cgi: "LINK=yes. here he
```

```
is;SIHLO=http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/judy/judy.xml"
*****************************************************************************************************
*****************/
/****** 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
answer-query 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(";",0);
    loc = ex.substring(0,index);

    return loc;
}

/** Update the text area in the HTML and show message on it. **/
function updateShowFrame(message)
{
   var objTable = document.getElementById("logFrame");

   objTable.insertRow(0);
   objTable.rows[0].insertCell(0);
   objTable.rows[0].insertCell(1);
   var cell0 = objTable.rows[0].cells[0];

   var cell1 = objTable.rows[0].cells[1];
   cell1.align="left";
   cell0.align="left";
   cell0.width="105";
   if(message.indexOf("SYSTEM ERROR: ")!=-1)
   {
      var objFont = document.createElement("font");
```

```
    objFont.color="red";
    objFont.size="-1";
    objFont.appendChild(document.createElement("b"));
    objFont.firstChild.innerHTML = message.slice(0, message.indexOf(":")+1);
    cell0.appendChild(objFont);

    var objFont2 = document.createElement("font");
    var objIta = document.createElement("i");
    objFont2.color="black";
    objFont2.size="-1";
    objFont2.appendChild(objIta);
    cell1.appendChild(objFont2);
    objIta.innerHTML=message.slice(message.indexOf(":")+1);
  }else
  {
    var index = message.indexOf(":");
    var ex=message.slice(0,index+1);
    var objFont = document.createElement("font");
    if(ex.indexOf("QUESTION:")!=-1)
      objFont.color= "blue";
    else if(ex.indexOf("RESPONSE:")!=-1)
      objFont.color="green";
    else
      objFont.color="purple";

    objFont.appendChild(document.createElement("b"));
    cell0.appendChild(objFont);
    objFont.firstChild.innerHTML = ex;
    cell1.appendChild(document.createElement("font"));
    cell1.firstChild.innerHTML = message.slice(index+1);
  }
  /****** Insert a table row as an empty line after a response and greeting message.
**********/
  if(message.indexOf("QUESTION")==-1)
  {
    objTable.insertRow(0);
    objTable.rows[0].insertCell(0);
    objTable.rows[0].colspan="2";
    objTable.rows[0].cells[0].innerHTML = "<br/> ";
  }

  return "";
}
```

```
/** Load user's application. **/
function loadPage(checkInput)
{
  /** Get user's input. **/
  var loc = document.getElementById("id_nextPage").value;

  /** if user's input is empty, then return a error message. **/
  if(loc=="")
  {
    updateShowFrame("SYSTEM ERROR: Please input the URL to your voice page in
the above text field. It can not be empty!" );
  }
  /** if user input is not empty, and user asked to validate URL before go. **/
  else if(checkInput==true)
  {
    submitReq("GET", loc);
    /** if the valicating process return a false as result, which means invalid URL. **/
    if(answerRecieved==false)
    {
      if(xmlhttp.status==404)
        updateShowFrame("SYSTEM ERROR: Unable to load your voice page. File
does not exist at: "+ loc );
      else
        updateShowFrame("SYSTEM ERROR: Unable to load your voice page. "
        +"Network problem, error code: "+xmlhttp.status+". Please check your internet
connection.");
    }
    /** if user's input is not empty, it is a valid URL to next page. **/
    else
      window.location=loc;
  }
  /** if user's input is not empty, and user asked to load URL page immediately. **/
  else
    window.location=loc;
}

function menuPage()
{
  window.location=startPage;
}

function processQuestion()
{
  gotoNext = true;
```

```
  question = document.getElementById("id_questionField").value;
  document.getElementById("id_questionField").value="";
  mainControl();
}

</script>

<!--*******************************************************
  The following script will only be run after a 'vxmldone' event is thrown after the
VoiceXML form finish all its process.
  It also means that the answer returned from interpreter contains a link to next interpreter,
so it needs to go there.
*******************************************************-->
<script type="text/javascript" id="gotoNextPage" declare="declare">
  window.location=nextPage;
</script>

<title>Public-Domain SpeechWeb</title>
</head>
<body id="page.body">
<center><h2>Welcome to our new voice browser!</h2></center>
<br/>
<center>
<table>
<tr><td colspan="6">Load your own speech application :
<input type="text" id="id_nextPage" size="50"
value="http://luna.cs.uwindsor.ca/~speechweb/p_d_speechweb/menu/demo_menu.xml"/
>
<br/><br/></td></tr>
<!-- Call loadPage() function to setup the interprter and speech-grammar location
according to the above input text field value; -->
<tr><td colspan="2"><input type="button" name="submitValidate" value="Validate Before
Go" onclick="loadPage(true)"/></td>
<td colspan="2"><input type="button" name="submitGo" value="Go Immediately"
onclick="loadPage(false)"/></td>
<td colspan="2"><input type="button" name="menuGo" value="SpeechWeb Menu Page"
onclick="menuPage()"/></td></tr>
<tr><td colspan="6"><br/><br/><br/><b>Say your question or type it in
here:</b></td></tr>
<tr><td colspan="6"><form onsubmit="processQuestion(); return false;">
<input type="text" size="70" name="questionField" id="id_questionField"
value=""/></form></td></tr>
</table>
<br/><br/>
```

```
<table id="logFrame" width="600"></table><br/>
<br/><br/><br/>
</center>
</body>
<!-- Call a script to reload the vxml form when the current vxml form has done its process.
-->
<ev:listener ev:observer="page.body" ev:event="vxmldone" ev:handler="#gotoNextPage"
ev:propagate="stop" />
<!-- Load 'vxml_form' when the page.body loaded. -->
<ev:listener ev:observer="page.body" ev:event="load" ev:handler="#vxml_form"
ev:propagate="stop" />
</html>
```

Single-Page Version SpeechWeb Browser:

A single X+V page loaded in end-user's device, *"single_page_version.xml":*

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//VoiceXML Forum//DTD XHTML+Voice 1.2//EN"
"http://www.voicexml.org/specs/multimodal/x+v/12/dtd/xhtml+voice12.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:vxml="http://www.w3.org/2001/vxml"
      xmlns:ev="http://www.w3.org/2001/xml-events"
      xmlns:xv="http://www.voicexml.org/2002/xhtml+voice"
      xml:lang="en-US">

<!--*****************************************************************************
Date: Feb. 2006
Developer: Ma, Xiaoli(William)
Architecture: LRRP (Dr.Frost, University of Windsor, Canada)
*****************************************************************************-->

<head>
<!--***** Ask browser that do not cache any of page, grammar, document, and vxml object.
*****-->
<vxml:property name="grammarfetchhint" value="safe"/>
<vxml:property name="documentfetchhint" value="safe"/>
<vxml:property name="objectfetchhint" value="safe"/>

<script type="text/javascript">
/********* Load data if there's anything saved in the cookie. **************/
var gmr = getCookie("grammar");
if(gmr!=null)
  if(gmr!="")
  {
    var objgrammar = document.getElementsByTagName("grammar")[0];
    objgrammar.setAttribute("src", "solarman.jsgf");
    if(objgrammar==undefined)
    {
      window.location.reload();
    }else
      objgrammar.setAttribute("src", gmr);
  }

var sv_greeting = getCookie("greeting");
```

```
if(sv_greeting==null)
  sv_greeting="hi, my name is judy.";

var cgiLink = getCookie("interpreter");
if(cgiLink==null)
  cgiLink="http://luna.cs.uwindsor.ca/~richard/judy/judy.cgi";


/******* Declare global variables shared by JavaScript and VoiceXML *********/
var sv_loc = "";
var grammar="";
var cgiLinkPrev="";
var question="";
var answer="";
var answerRecieved=false;
var sayGreetings=true;
var isLink=false;
var sihloLoc = "";
var logMsg=""
var javacode="";
var error="";


/*************************************************************************************************
This is the main control function to the whole question submit and answer retrived
procedures.
It will call submitReq() method to send the question to the CGI program.
then it will check the answer whether it is a link to new CGI program or a simply answer
string.
if it is a link to another interpreter, then retrieve the data from there,
and call the 'changeData' function to change the neccessary information for the next
round dialog.
*************************************************************************************************/
function mainControl()
{


  if(sayGreetings==true)
    updateShowFrame("GREETING: "+sv_greeting+"<br/>");

  updateShowFrame("QUESTION: "+question+"<br/>");

  answer="";
  answerRecieved=false;
```

```
  isLink=false;
  error="";

  /* call submitReq() method to send the question to the CGI program. */
  submitReq("POST", cgiLink);
  answer = getAnswer(xmlhttp.responseText);

  /******* Check whether the recieved answer is a link or not. And, assign the result to the
global variable isLink. ******/
  checkAnswer(xmlhttp.responseText);

  /****** if the answer is not a link, then return quickly to speak out the answer to the user.
******/
  if(!isLink)
  {
    updateShowFrame("RESPONSE: "+answer+"<br/>");
    return "1";
  }

  cgiLinkPrev=cgiLink;
  cgiLink="http://"+getSihlo(xmlhttp.responseText);

  submitReq("GET", cgiLink);

  changeData();

  updateShowFrame("RESPONSE: "+answer+"<br/><br/>");

// alert("grammar:"+grammar+" answer:"+answer+" greeitng:"+sv_greeting+"
cgi:"+cgiLink);

  setCookie("grammar", grammar, 365);
  setCookie("greeting", sv_greeting, 365);
  setCookie("interpreter", cgiLink, 30);

// alert(getCookie("grammar")+" "+getCookie("answer")+" "+getCookie("greeting")+"
"+getCookie("interpreter"));

  return "1";
}

function setCookie(c_name,value,expiredays)
{
  var exdate=new Date();
```

```
  exdate.setDate(expiredays);
  document.cookie=c_name+ "=" +escape(value)+ ((expiredays==null) ? "" : ";
expires="+exdate);
}

function getCookie(c_name)
{
  if (document.cookie.length>0)
  {
    c_start=document.cookie.indexOf(c_name + "=");
    if (c_start!=-1)
    {
      c_start=c_start + c_name.length+1 ;
      c_end=document.cookie.indexOf(";",c_start);
      if (c_end==-1) c_end=document.cookie.length
        return unescape(document.cookie.substring(c_start,c_end));
    }
  }
  return null;
}


/*************************************************************************************************
This function returns the substring that has to be spoken as a result of the user's question.
Same procedure is applied for extracting the content to be spoken out.
*************************************************************************************************/
function getAnswer(answer)
{
  var ex=answer;
  var index;
  if((ex.indexOf('LINK=',0)) == -1)
    return ex;
  ex= ex.slice(5);
  index = ex.indexOf(";",0);
  ex = ex.substring(0,index);
  return ex;
}


/*************************************************************************************************
This function uses AJAX, it will submit the question to the given URI if it use a 'POST'
method.
Or, it will retrieve data from the given URI if it use a 'GET' method.
*************************************************************************************************/
```

```
function submitReq(method, url)
{
  /***** Initialize AJAX XMLHttpRequest object. ****/
  xmlhttp=new XMLHttpRequest();
  /******
  Assign a event listener to the 'onreadystatechange' event.
  Different listerner assigned depends on a 'GET' or a 'POST' method.
  ******/
  if(method=="GET")
    xmlhttp.onreadystatechange=stateChange_GET;
  else
    xmlhttp.onreadystatechange=stateChange_POST;

  /** Open the connect, sychronized.   ***/
  xmlhttp.open(method,url,false);

  if(method=="GET")
    xmlhttp.send();
  else
  {
    xmlhttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    xmlhttp.send("question="+question);
  }
}

function stateChange_POST()
{
  /******* if xmlhttp shows loaded   ******/
  if (xmlhttp.readyState==4)
  {
    if (xmlhttp.status==200 || xmlhttp.status==304)
    {
      answerRecieved = true;
    }
    else
    {
      answerRecieved = true;
      alert("Network Error: Problem retrieving data from:" + cgiLink+".\n Error code:" +
xmlhttp.status);
      xmlhttp.responseText = "";
      error="critical";
    }
  }
}
```

```
function stateChange_GET()
{
  /******* if xmlhttp shows loaded  ******/
  if (xmlhttp.readyState==4)
  {
    if (xmlhttp.status==200 || xmlhttp.status==304)
    {
      answerRecieved = true;
    }
    else
    {
      answerRecieved = true;
      alert("Problem retrieving data from:" + cgiLink+".\n Error code:" + xmlhttp.status);
      xmlhttp.responseText = "";
      error="critical";
    }
  }
}

/*****************************************************************************************************
****
Check whether there is a occurrence of '=' character in the answer, which means a link
existed in it.
And, assign the result to the global variable 'isLink'.
*****************************************************************************************************
**/
function checkAnswer(answer)
{
  if((answer.indexOf('LINK=',0))== -1)
    isLink=false;
  else
    isLink=true;

  return isLink;
}

/*****************************************************************************************************
************
This function will return the sihlo link as a string, if the answer is a link. Otherwise, return
"-1".
*****************************************************************************************************
************/
function getSihlo(answer)
```

```
{
  var loc;
  var ex = answer;
  var index;

/****** Check if the answer is a link to a sihlo file.******/
  if(!isLink)
    return "-1";

/*************************************************************************************************
******************
If the answer is a link, then its formation should be: "LINK=_answer;SIHLO=_location;".
e.g. Question send to judy.cgi: "can i talk to solar man".
      Answer recieved from judy.cgi: "LINK=yes. here he
is;SIHLO=luna.cs.uwindsor.ca/~richard/solarman/solarman.sihlo;"
*************************************************************************************************
*****************/
/****** 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(";",0);
   loc = ex.substring(0,index);
   return loc;

}

/*************************************************************************************************

*************************************************************************************************
```

```
***/
function changeData()
{

  /*********
  Sihlo file formation example:
GREETING=Hi, I am Monty. I know a joke;
SERVER_NAME=luna.cs.uwindsor.ca;
SIHLO_INTERPRETER=/~richard/monty/monty.cgi;
SIHLO_GRAMMAR=/~richard/monty/h.cgi?monty.gram;
  *********/

  /** Retrieve the greeting message from the response, and say it in next round dialog. **/
  sv_greeting=getGreeting(xmlhttp.responseText);
  if(sv_greeting==-1)
  {
    updateShowFrame("SYSTEM ERROR: Wrong formation in the file:" +
cgiLink+"\nPart: GREETING\nCorrect formation: GREETING=_yourGreetingMsg;");
    sv_greeting = "Hi, i'm ready to answer your question now.";
    error="non-critical-lowestlv";
  }
  else if(sv_greeting==-2)
  {
    updateShowFrame("SYSTEM ERROR: No greeting message is available in the
SHILO file.");
    sv_greeting = "Hi, i'm ready to answer your question now.";
    error="non-critical-lowestlv";
  }

  var server_name=getServerName(xmlhttp.responseText);
  if(server_name==-1)
  {
    updateShowFrame("SYSTEM ERROR: Wrong formation in the file:" +
cgiLink+"\nPart: SERVER_NAME\nCorrect formation:
SERVER_NAME=_yourServerLocation;\n Example: luna.cs.uwindsor.ca\n Note: No
'http://' needed");
    updateShowFrame("SYSTEM ERROR: Attempting to use current server host for next
interpreter :"+sv_loc);
    error="non-critical";
  }else
    sv_loc=server_name;

  var tmp = getInterpreter(xmlhttp.responseText);
  if(tmp==-1)
```

```
  {
    updateShowFrame("SYSTEM ERROR: Wrong formation in the file:" +
cgiLink+"\nPart: SIHLO_INTERPRETER\nCorrect formation:
SIHLO_INTERPRETER=_yourInterpreterLocation;\n Example:
SIHLO_INTERPRETER=/~richard/judy.cgi;");
    cgiLink = cgiLinkPrev;
    answer="Next interpreter is not available, stay in the current interpreter.";
    error="non-critical";
  }else
    cgiLink="http://"+sv_loc+tmp;

  if(changeGrammarLink(xmlhttp.responseText)==-1)
  {
    updateShowFrame("SYSTEM ERROR: Wrong formation in the file:" +
cgiLinkPrev+"\nPart: SIHLO_GRAMMAR\nCorrect formation:
SIHLO_GRAMMAR=_yourGrammarLocation;\n Example:
SIHLO_GRAMMAR=/~richard/judy.jsgf");
    updateShowFrame("SYSTEM ERROR: Continue use current grammar file for next
interpreter.");
    error="non-critical";
  }
}

/*******************************
This function returns ServerName if it existed in the response message, otherwise return
-1.
*********************************/
function getServerName(response)
{
  var ex=response;
  var start=ex.indexOf('SERVER_NAME=',0);
  if(start == -1)
    return -1;
  ex=ex.slice(start+12);
  var end=ex.indexOf(';',0);
  ex=ex.slice(0, end);
  return ex;
}

/******** get speech grammar location and assign it to 'grammar' global variable *******/
function changeGrammarLink(response)
{
  var tmpGrammar=getGrammarLink(response);
  if(tmpGrammar==-1)
```

```
    return tmpGrammar;
  else
    grammar="http://"+sv_loc+tmpGrammar;

/** Next line is used in application testing only, should be deleted from final product. **/
  grammar = modify(grammar);

  return "";
}

/** This method is used for application testing only. it should be deleted from final product/
**/
function modify(grammar)
{
  if(grammar.indexOf("monty")!=-1)
    return "monty.jsgf";
  else if(grammar.indexOf("solar")!=-1)
    return "solarman.jsgf";
  else
    return "judy.jsgf";
}

/****** Get next speech grammar link URI from response and return it. *******/
function getGrammarLink(response)
{
  var ex=response;
  var start=ex.indexOf('SIHLO_GRAMMAR=',0);
  if(start == -1)
    return -1;
  ex=ex.slice(start+14);
  var end=ex.indexOf(';',0);
  ex=ex.slice(0, end);
  return ex;
}

/** Get next interpreter URI from response and return it. **/
function getInterpreter(response)
{
  var ex=response;
  var start=ex.indexOf('SIHLO_INTERPRETER=',0);
  if(start == -1)
    return -1;
  ex=ex.slice(start+18);
  var end=ex.indexOf(';',0);
```

```
  ex=ex.slice(0, end);
  return ex;
}

/** Get next interpreter's greeting message from response and return it. **/
function getGreeting(response)
{
  var ex=response;
  var index;
  if((ex.indexOf('GREETING=',0)) == -1)
    return -1;
  ex= ex.slice(9);
  index = ex.indexOf(";",0);
  ex = ex.substring(0,index);
  /****** if greeting message is an empty string then return -2 as an error inditifier ********/
  if(ex=="")
    return -2;
  return ex;
}

/** Update the text area in the HTML and show message on it. **/
function updateShowFrame(message)
{
  logMsg +=message;

  var objTable = document.getElementById("logFrame");

  objTable.insertRow(0);
  objTable.rows[0].insertCell(0);
  objTable.rows[0].insertCell(1);
  var cell0 = objTable.rows[0].cells[0];

  var cell1 = objTable.rows[0].cells[1];
  cell1.align="left";
  cell0.align="left";
  cell0.width="105";
  if(message.indexOf("SYSTEM ERROR: ")!=-1)
  {
    var objFont = document.createElement("font");
    objFont.color="red";
    objFont.size="-1";
    objFont.appendChild(document.createElement("b"));
    objFont.firstChild.innerHTML = message.slice(0, message.indexOf(":")+1);
    cell0.appendChild(objFont);
```

```
    var objFont2 = document.createElement("font");
    var objIta = document.createElement("i");
    objFont2.color="black";
    objFont2.size="-1";
    objFont2.appendChild(objIta);
    cell1.appendChild(objFont2);
    objIta.innerHTML=message.slice(message.indexOf(":")+1);
  }else
  {
    var index = message.indexOf(":");
    var ex=message.slice(0,index+1);
    var objFont = document.createElement("font");
    if(ex.indexOf("QUESTION:")!=-1)
      objFont.color= "blue";
    else
      objFont.color="green";

    objFont.appendChild(document.createElement("b"));
    cell0.appendChild(objFont);
    objFont.firstChild.innerHTML = ex;
    cell1.appendChild(document.createElement("font"));
    cell1.firstChild.innerHTML = message.slice(index+1);
  }
  /****** Insert a row as an empty line after a response message. ***********/
  if(message.indexOf("RESPONSE:")!=-1)
  {
    objTable.insertRow(0);
    objTable.rows[0].insertCell(0);
    objTable.rows[0].colspan="2";
    objTable.rows[0].cells[0].innerHTML = "<br/> ";
  }
}

/** Setup the URI link for next interpreter, speech grammar from user input. And, setup a
default greeting message. **/
function setupLinks()
{
  setCookie("grammar", document.getElementById("id_grammarLink").value, 1);
  setCookie("greeting", "Hi, I'm ready to answer your question now.", 1);
  setCookie("interpreter", document.getElementById("id_cgiLink").value, 1);

  reloadVoiceForm(false);
```

```
  return "";
}

/** Reload VXML voice form. Check error messages if asked. **/
function reloadVoiceForm(errorCheck)
{
  if(errorCheck==true)
  {
    if(error=="non-critical")
      alert("Some non-critical errors occured in the process. Please look at the error
message and press ok to continue.");
    else if(error=="critical")
    {
      alert("Some critical errors occured in the process. Voice browser will continue from
the default page.");
      setCookie("grammar", "", 0);
      setCookie("greeting", "", 0);
      setCookie("interpreter", "", 0);
    }
    else if (error!="")
    {
      setTimeout("window.location.reload()", 2000);
    }
  }

  window.location.reload();

  return "";
}

</script>

<!-- VoiceXML form. -->
<vxml:form id="vxml_form">
<vxml:field name="st_field" xv:id="voice_input" modal="true">
    <vxml:grammar type="application/x-jsgf" src="judy.jsgf" />
    <!-- The following greeting will only speak out when user connects to a new interpreter.
-->
      <vxml:prompt cond="sayGreetings==true"><vxml:break
time="500ms"/><vxml:value expr="sv_greeting"/></vxml:prompt>
    <vxml:filled>
 <!--**********************************
  This "filled" element will be run after user speech input has recognized.
  Inside this element, first step, i have assign the user input to the variable 'question',
```

```
because VoiceXML code can access a JavaScript defined variable, but JavaScript can not
see a VoiceXML defined variable.
  Then, in the next step, i call a JavaScript function "runCode()' to proceed AJAX submit
process.
**********************************-->
      <vxml:assign name="sayGreetings" expr="false"/>
      <vxml:assign name="question" expr="st_field"/>
      <!--**********************
        Calls to javascript mainControl() function to do the logical process based on user
voice input.
      ************************-->
      <vxml:assign name="javacode" expr="mainControl();"/>
      <vxml:prompt><vxml:break time="300ms"/><vxml:value
expr="answer"/></vxml:prompt>
<!-- If the answer is not a link to next interpreter, then repeat the voice dialog. -->
      <vxml:if cond="isLink==false">
        <vxml:throw event="repeat.st_field"/>
      </vxml:if>
    </vxml:filled>
    <vxml:catch event="nomatch noinput">
      <vxml:prompt>Sorry, I don't understand, can you say it again?</vxml:prompt>
      <vxml:reprompt/>
    </vxml:catch>
    <vxml:catch event="help">
      No help is available! Restart the dialog!
      <vxml:clear namelist="st_field"/>
      <vxml:reprompt/>
    </vxml:catch>
</vxml:field>

<!-- Catch the 'repeat.st_field' event. -->
<vxml:catch event="repeat.st_field">
  <vxml:clear namelist="st_field"/>
<!-- Restart the voice form without change the speech grammar. -->
  <vxml:reprompt/>
</vxml:catch>

</vxml:form>

<!--*****************************************************
  The following script will only be run after a 'vxmldone' event is thrown after the
VoiceXML form finish all its process.
  It also means the answer returned from interpreter contains a link to next interpreter, so
it needs to reload the voice form in order to reload the speech-grammar file.
```

```
**********************************************************-->
<script type="text/javascript" id="reloadVoiceForm" declare="declare">

reloadVoiceForm(true);

function reloadVoiceForm(errorCheck)
{
  if(errorCheck==true)
  {
    if(error=="non-critical")
      alert("Some non-critical errors occured in the process. Please look at the error
message and press ok to continue.");
    else if(error=="critical")
    {
      alert("Some critical errors occured in the process. Voice browser will continue from
the default page.");
      setCookie("grammar", "", 0);
      setCookie("greeting", "", 0);
      setCookie("interpreter", "", 0);
    }
    else if (error!="")
    {
      setTimeout("window.location.reload()", 2000);
    }
  }

  window.location.reload();

  return "";
}
</script>

<title>SpeechBrowser</title>
</head>
<body id="page.body">
<center><h2>Welcome to our new voice browser!</h2></center>
<br/>
<center>
<table>
<tr><td>CGI link:</td>
<td><input type="text" name="cgiLink" id="id_cgiLink" size="30"
value="http://luna.cs.uwindsor.ca/~richard/judy/judy.cgi"/></td></tr>
<tr><td>Grammar link:</td>
<td><input type="text" name="grammarLink" id="id_grammarLink" size="30"
```

```
value="http://luna.cs.uwindsor.ca/~xing4/judy.jsgf"/></td></tr>
<!-- Call setupLinks() function to setup the interprter and speech-grammar location
according to the above input text field value; -->
<tr><td colspan="2"><input type="button" name="submit" value="Go"
onclick="setupLinks()"/></td></tr>

</table>
<br/><br/>
<table id="logFrame" width="60%"></table><br/>
<br/><br/><br/>
</center>
</body>

<!-- Call a script to reload the vxml form when the current vxml form has done its process.
-->
<ev:listener ev:observer="page.body" ev:event="vxmldone"
ev:handler="#reloadVoiceForm" ev:propagate="stop" />
<!-- Load 'vxml_form' when the page.body loaded. -->
<ev:listener ev:observer="page.body" ev:event="load" ev:handler="#vxml_form"
ev:propagate="stop" />

</html>
```

VITA AUCTORIS

Xiaoli Ma was born in 1981 in the district of ShenYang city, People's Republic of China. He earned his B.Sc. in Computer Science with option in Software Engineering in 2004 from University of Windsor, Ontario, Canada. Xiaoli Ma is currently a candidate for the Master's degree under the supervision of Dr. Richard A. Frost in the School of Computer Science at the University of Windsor, Ontario, Canada and expecting to graduate in Summer 2006.