ON THE CONSTRUCTION OF A VOICEXML VOICE BROWSER

Chih-Hsing Hsu, Miaw-Ru Hsu, Cher-Yao Yang, Sen-Chia Chang

Computer Communication Research Laboratories (CCL)
Industrial Technology Research Institute (ITRI)
hsujs@itri.org.tw

ABSTRACT
In this paper, we introduce the construction of our VoiceXML-compliant voice browser that is able to interpret and execute VoiceXML documents. The platform of our voice browser contains a VoiceXML interpreter, a speech recognition (SR) engine, a text-to-speech (TTS) engine, and a computer-telephony-integrated (CTI) user interface. We adopt an open source called OpenVXI as the VoiceXML interpreter. Our main effort for developing the voice browser is to incorporate OpenVXI with our SR, TTS, and CTI modules. In order to test whether our voice browser conforms to the VoiceXML specifications, we build three VoiceXML applications including stock quotes querying, automatic call-transfer, and news-reading systems. Besides, we develop a system that allows users to register their own VoiceXML documents in our voice gateway system via Web. They may then call to our voice gateway system and browse their VoiceXML documents by using our voice browser.

1. INTRODUCTION
What is VoiceXML? It's an XML language for writing Web pages you use phone to interact with by listening to spoken prompts, and control by means of spoken input.[1-2] VoiceXML brings the Web to telephones. There are an increasing number of voice portal systems which you can call into and try out for yourself. Several sites also offer free hosting for VoiceXML. This allows users to build and use their own VoiceXML applications.

VoiceXML isn’t HTML. The difference can be seen from Fig 1.1. HTML was designed for visual Web pages and lacks the control over the user-application interaction that is needed for a speech-based interface. With speech you can only hear one thing at a time (kind of like looking at a newspaper with a times 10 magnifying glass). VoiceXML has been carefully designed to give authors full control over the spoken dialog between the user and the application. The application and user take it in turns to speak: the application prompts the user, and the user in turns responds. VoiceXML documents describe:

- voice prompts (synthetic speech)
- playback of audio files
- speech recognition & grammars
- touch tone (DTMF) key grammars & presses
- recording of voice file
- control of dialog flow
- telephony control (call transfer and hang up)

VoiceXML makes it easy to rapidly create new applications and shields developers from the low level and implementation details. It separates user-interaction from service logic. The W3C VoiceXML 2.0 specification is the definitive reference to VoiceXML. You can also find other related work on W3C’s voice browser activity page.[4]

Using VoiceXML can develop many suitable speech applications. The following are some examples.

- Information retrieval: weather, stock quotes
- E-Commerce: banking, accounting
- Call Center Service: Auto attendant
- Unification: Personal Agent, Personal Portal

2. COMPONENTS AND EXAMPLES OF VOICEXML

In VoiceXML 1.0 specifications, many XML elements were defined for developing speech applications. The following is a simple VoiceXML document example.

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<form>
  <field name = "breakfast">
    <prompt> May I help you </prompt>
    <grammar> toast | hotdog | ice cream </grammar>
    <filled>
      <submit next =http://www.food.com/breakfast.cgi
               namelist = "breakfast" />
    </filled>
  </field>
</form>
```

The example specified a voice dialog between human and computer. The following is a possible dialog example

C: May I help you
H: hotdog
...
This example document first define the language decoding of the document (iso-8859-1), and the following is a form for describing the voice dialog template between user and computer. A dialog turn is specified by a `<field>` element. Each field can be filled by voice or DTMF input. There is also a `<prompt>` element to prompt the user and a `<grammar>` element to specify the speech grammar the user can say. When a field is filled, the `<filled>` element is executed. In this example, the recognition result is submitted to another CGI program for further processing.

Besides for `<form>` element, simple voice dialog can use `<menu>` element to construct the VoiceXML document. The following is a menu example.

```xml
<?xml version="1.0" encoding="ISO8859-1" ?>
<vxml version="1.0">
  <menu>
    <prompt> please say your choice </prompt>
    <choice next="sports.vxml"> sport </choice>
    <choice next="weather.vxml"> weather </choice>
    <choice next="news.vxml"> news </choice>
  </menu>
</vxml>
```

When user says one the choice, the corresponding VoiceXML document is loaded for execution.

There are two types of voice dialogs in VoiceXML. Computer-directed voice dialog executes the dialog flow based on the appearance order of form or menu. The dialog flow is fixed. On the contrary, mixed-directed voice dialog can change the dialog flow dynamically based on the user-input. The following is a mixed-directed voice dialog example.

```xml
<ex>
  C: What weather information do you want?  
  U: Temperature.  
  C: What city?  
  U: Taipei  
  C: What weather information do you want in Taipei?  
  U: Temperature.  
</ex>
```

3. **OUR VOICE BROWSER**

A voice browser is constructed by integrating some speech processing techniques, i.e., speech recognition, speech synthesis, computer-telephony integration interface, and Internet access. Most of current voice browsers are tightly coupled with their own speech engine. However, OpenVXI that was developed by SpeechWorks provides an open environment for developers to integrate their speech engine to develop a voice browser [3]. Hence, we adopt the OpenVXI as the framework to developing our own voice browser.

### 3.1 OpenVXI

The system architecture of OpenVXI is shown in Fig. 3.1. It was composed of the following modules and APIs.

- **VoiceXML Interpreter**
  This module is used to retrieve VoiceXML pages from Internet for parsing and execution. The lower layer APIs such as speech recognition API, Prompt API, or Telephony API were called to complete the execution.

- **JavaScript Interpreter**
  This module is used to executing the JAVA script embedded in the VoiceXML document.

- **Speech Recognition API**
  This API sets define the interface between the VoiceXML interpreter and the speech recognition engine. In a VoiceXML document, when voice input or touch-tone input are required, the interpreter calls the speech recognition APIs.

- **Prompt API**
  This API sets define the interface between the VoiceXML interpreter and the speech synthesis engine. When voice output is required, interpreter calls the prompt API to synthesis the voice output.

- **Telephony API**
  This API sets define the interface between the VoiceXML interpreter and the Telephony interface. When telephony related function such as waiting for call, disconnecting a call, recording a voice file are required, the interpreter calls the Telephony API to complete this job.
Object API

<OBJECT> element is an extended module for VoiceXML. By using <OBJECT>, platform dependent functions can be added to a voice browser. However, there is only syntax definition in VoiceXML. The specific functions were implement by voice browser developer. The Object API set defines the interface between the interpreter and the platform. If voice browser have special object functions, they must follow this API set to implement their platform dependent functions.

3.2 Integration of Our Speech Engine and OpenVXI

Based on the open-architecture of OpenVXI, our voice browser is implemented by integration of our speech kernel function and the APIs described in section 3.1.

Before the description of the integration we first inspect the execution procedure of the OpenVXI. The execution procedure can be divided as three parts.

- Platform Scope: processing the system initialization and finished procedures.
- Channel Scope: processing procedure for each channel.
- Session Scope: processing procedure for each call. It can be further divided as the following procedures.
  A. Document loop: loading a VoiceXML document for execution. For example, when a ‘goto’ element is executed, the document loop is performed.
  B. Dialog loop: selecting a ‘form’ or a ‘menu’ for execution from the same VoiceXML document.
  C. FIA Loop: FIA(Form Implementation Algorithm) is used to select the next element for visiting and execution. The FIA algorithm was described in details in VoiceXML 1.0 specification. The Speech Recognition API, Prompt API, Telephony API and JavaScript API were called in this step.

To integrate our kernel function into OpenVXI, our speech engine was designed to conform the OpenVXI’s API definition.

Grammar Parser Engine

Grammar parser API is used to parse the grammar specified in the VoiceXML document. The grammar can be embedded in a VoiceXML document or written in a separated grammar file. The grammar is converted to a finite state grammar for recognition by our speech recognition engine. For example the following grammar is converted to the finite state grammar shown in Fig.3.2.

<grammar>
$prefillern= please connect me to | connect me to
$name=John | Marry
public $prefiller+ $name {$name}
</grammar>

It is noticed that the grammar format in VoiceXML 1.0 is not a standard format. So the above grammar format is defined by our own convenience. While in VoiceXML 2.0, grammar format must follow the ABNF or XML grammar format.

Speech Recognition Engine

This kernel function used for speech recognition. Because in a VoiceXML document, multiple grammars are allowed to be enabled for recognition, we use a tree structure to connect all the grammars and only enabled grammar are marked as starting tree for recognition.

TTS Engine

This module implements the TTS function. The text in VoiceXML document is marked-up for producing versatile speech output. For example the following marked-up text produce emphasized speech output.

<pre>increase tone</pre>

Hence many parameters can be adjust in our TTS engine to produce the required speech quality. Tab. 3.1 shows some adjustable parameters for the requirements.

Computer-Telephony Integration (CTI) Engine

This module is used for telephony functions, such as ringing detection, disconnect-signal detection, voice playback and

<table>
<thead>
<tr>
<th>Level</th>
<th>Pitch (%)</th>
<th>Energy (%)</th>
<th>Pause (%)</th>
<th>Duration (%)</th>
<th>Syllable Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong</td>
<td>+9%</td>
<td>+10%</td>
<td>+50%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>moderate</td>
<td>+6.5%</td>
<td>+6%</td>
<td>+30%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>None</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>reduced</td>
<td>-3%</td>
<td>-3%</td>
<td>-15%</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Tab 3.1 Adjustable parameters of TTS Engine
recording.

- **Speaker Verification Engine**
  This module is used for speaker verification application. It is implemented as a Object function in the OpenVXI’s object API. Fig. 3.3 is the system architecture for integrating our speech engine and the OpenVXI.

![Fig. 3.3 Integration of our Speech Engine and OpenVXI](image)

### 4. CONFORMANCE TESTING

In order to test whether our voice browser conforms to the VoiceXML specifications, we built three VoiceXML applications including stock query, automatic call-transfer, and news reading systems. Besides, we developed a platform that allows users to register their own VoiceXML documents in our voice gateway system via Web, and we provided our voice browser for users to browse these documents. The system architecture is shown in Fig. 4.1. The services include:

- Auto Attendant service
- Stock quotes inquiry service
- News reading service

Each service was composed of a VoiceXML document and a CGI program. The main page of the voice portal system was controlled by a voice gateway page. When customer call to the voice portal system, he first say the service name, such as ‘auto attendant’, ‘stock’, or ‘news reader’ to access to relevant the VoiceXML application. And then he says the keyword to query information from database via the CGI program. The customer may also say ‘main menu’ to return the top-level entry of the voice portal system or other service name to switch to other service domain.

![Fig. 4.1 Our voice portal system](image)

### 5. CONCLUSION

VoiceXML has been proposed as a standard language to develop Internet speech applications. The properties of easy to use and high-level language make it gradually popular for developing speech applications. Besides, the distributed architecture of VoiceXML documents makes speech applications possible to be shared in the Internet. Hence, constructing a voice browser is necessary for executing VoiceXML applications. In this paper we integrate OpenVXI and our speech engine to construct our voice browser. For the conformance testing, we also build a voice portal system based on VoiceXML.

### 6. REFERENCES


