VoiceXML and its Role in a Multimodal Web

Ajit Paul Singh, Guangwu Liu {ajit|guangwu}@cs.ualberta.ca

March 1, 2001

Abstract

VoiceXML is an XML application for representing human-computer speech dialogs. This document will present an overview of the features of VoiceXML and provide an example of the major markup elements.

Introduction

With the increasing popularity of wireless Internet access has come the need for new browsing paradigms. Conventional interactive browsing (*i.e.* viewing a page layout with some form filling) is unsuitable in many situations where the user is visually impaired or requires the unconstrained use of their arms and legs. Furthermore, far more people have ready access to a telephone than to the Internet. Finally, there are many situations where it is desirable to have access to the same information from either a conventional web page or a telephone.

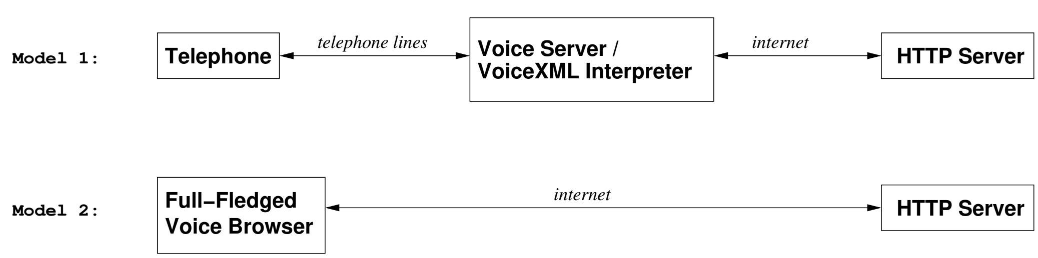
What is VoiceXML

VoiceXML is an extensible markup language for representing human-computer dialogs [1]. Although HTML can be defined as a markup language for representing human-computer dialogs it does so visually: a visual browser, mouse, and keyboard are presupposed. VoiceXML instead presupposes the existence of a *voice browser* capable of speech recording, voice recognition, speech synthesis, audio playback, and possibly a small keypad and/or viewing window. A voice browser could be a computer with a microphone, soundcard, and appropriate software; a voice browser could be a cellular telephone. In both cases some of the functionality of the voice browser may be implemented on the server-side[1].

As of February 2001 the latest VoiceXML specification is version 1.0.

VoiceXML and the Web

The most obvious relationship between VoiceXML and the web is the delivery mechanism. VoiceXML pages can be delivered to a voice browser using the same transport mechanisms as HTML pages. Essentially VoiceXML is a specific type of XML document, which can be viewed as structured data in a text file [2]. Like any other type of XML document it has a document type definition (DTD) [3, Appendix B]. Most importantly, using XSLT one can transform an XML document into HTML or VoiceXML as required[2].



[1]Usually VoiceXML systems implement the speech recognition and synthesis components on the server-side.

[2]Converting XML to VoiceXML is not automatic. Depending on the DTD of the XML document a transformation must be crafted by the developer. These issues lie outside the main concerns of this paper and will not be explored further.

Confusion often stems from the fact that there are two models for using VoiceXML documents. The first model has a VoiceXML interpreter which parses VoiceXML documents, contains the necessary grammars, voice recognition, speech synthesis, and audio playback capabilities to convert a VoiceXML documents to speech and convert spoken responses of the user to text. All the user needs a telephone to access such systems. In the second model all the processing of a VoiceXML document is done on the client side. In both cases a standard web server (*e.g.* Apache) serves VoiceXML documents.However, since VoiceXML software is rapidly evolving and has a small current user base model 1 is used almost exclusively.

Structure of a VoiceXML Document

VoiceXML documents are almost always interactive. The user desires some information and must somehow inform the web application what it is looking for. If the document was non-interactive one could just record the message as an audio file and let people download the file for playback. This interaction can be characterized in terms of *dialogs* or conversations between the user and web application. These snippets of interaction can be linked together. Thus the human-computer interaction can be characterized as a finite state machine with dialogs as states and links as dialog transitions [3].

There are two kinds of dialogs: *forms* and *menus*. Forms define interactions designed to get an answer from the user without explicitly enumerating all the possible choices. Menus define interactions to get an answer from the user after explicitly enumerating all the possible choices. Forms are analogous to text fields in HTML documents; Menus are analogous to pulldown menus in HTML documents. Unlike HTML form fields, all answers to fields in a dialogs must conform to certain rules as specified by a grammar (see page 3).

Dialogs can be extremely complex. To simplify development a complex dialog may be decomposed into a hierarchy of dialogs and subdialogs. Programmatically subdialogs are like function calls. Linguistically subdialogs represent a conversation in the context of a larger dialog. The purpose of a subdialog is to elicit information needed by one agent to complete the containing dialog. From a design perspective there are common dialogs which could easily be reused – login, search engine queries, etc. Such dialogs could be collected into a library of commonly used dialogs. Subdialogs can be implemented in separate .vxml files.

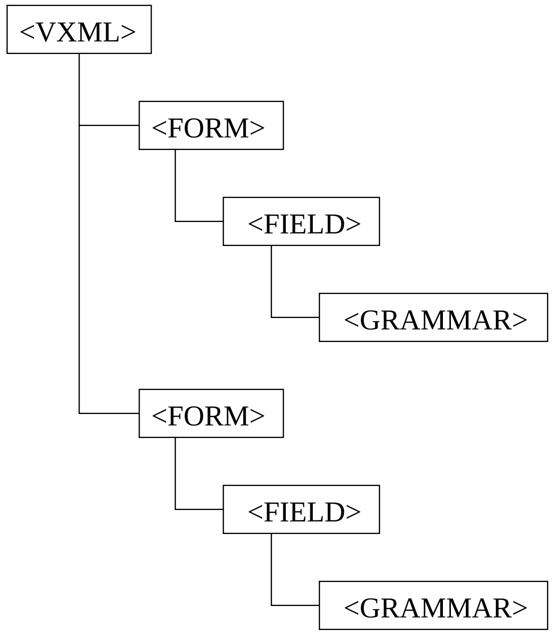


Figure: Structure of an example VoiceXML document with two forms, each containing one field [7].

Flow of Control

The default behavior is to sequentially go through a VoiceXML file. To get more complex behavior <goto next="#say_Goodbye"> where say_Goodbye is the label to a dialog in the file or <goto next="aDocument.vxml"> where aDocument.vxml is another VoiceXML document can be used. Moreover a rudimentary exception handling mechanism (similar to C++) exists. Events can be explicitly on implicitly (in the case of no user response after k seconds, requests for help, etc.) thrown. Event can be caught and handled as desired.

Grammars

Speech grammars specify the set of answers to a dialog. Since spoken answers must map to values that can be sent to a web server (via a query string for example) a speech grammar also specifies mappings from possible answers to string values or attribute-value pairs. These values are important as they can be manipulated by Javascript code inside a VoiceXML document or by a server-side CGI. This allows dynamic creation and customization of VoiceXML documents.

The specification does not require any particular grammar format. However, the Java Speech Grammar Format [4], which provides a way to code grammar for context free languages, is commonly used.

Grammars are critical to the feasibility of VoiceXML. When the user utters statements they must be converted into text. Natural language recognition over general sentences is computationally intensive and somewhat inaccurate. However dialogs provide scope to the conversation. The user could say anything, but is expected to say only a small range of things. With such constraints voice recognition is computationally easier and generally more accurate. Current directions in W3C speech grammar specifications tend toward representations that mix context-free grammars with N-gram stochastic grammars[3].

In addition to speech grammars DTMF (telephone keypad tones) grammars are possible. In this case a sequence of keypad tones is used to answer a dialog.

An example pair of grammars in the Nuance GSL grammar format is provided below [6]. It represents a speech grammar and a DTMF grammar used to recognize yes or no responses by the user. To answer yes the user can say any one of the yes-phrases or enter 1 on the keypad (if available). To answer no the user can say any one of the no-phrases or enter 2 on the keypad (if available).

```
[LIB_YES_NO:str DTMF_YES_NO:str] {<option $str>}

LIB_YES_NO [

[ yes yeah yup sure okay correct right ( ?( ?yes that's ) [ right correct ] ) ( ?yes it is ) (
you got it ) ( yes i do ) ( yes i would ) ( yes it is correct ) ] {return("yes")}

[ no nope incorrect ( no way ) ( no it isn't ) ( ?no [ it's that's ] not [ correct right ] ) (
?no it isn't ) ( ?no it is not ) ( ?no it's not ) ( no i don't ) ( no i do not ) ( no i wouldn't
) ] {return("no")} ]

DTMF_YES_NO [ dtmf-1 {return("yes")} dtmf-2 {return("no")} ]
```

Not all dialogs share the same grammar. Thus we scope grammars to certain dialogs or sets of documents. This allows for systems where there is a global grammar containing recognition for statements like “help” or “exit system” while simultaneously allowing a local grammar for the current dialog. VoiceXML also allows for modal dialogs (*i.e.* the only grammar is the local dialog grammar; the dialog must be answered before you can continue). The last type of grammar is a link grammar. This is scoped to an entire document. When a phrase is uttered that matches the a link grammar the flow of control shifts to a document specified by the <link> tag of that grammar. This allows the conversations where the user shifts focus.

Example of a VoiceXML Document

In this section we will present an example VoiceXML modified fragment from [3] and explain the components step by step. Furthermore we will map this document to a conversation possible with a VoiceXML system processing this fragment.

[3] N-gram stochastic grammars are those which specify the expected relative frequency of phrases of length N in the grammar. This information can be used to increase the accuracy of natural language recognition. See [5] for further details.

The VoiceXML fragment is for a form that gets credit card information from the user (as part of a shopping cart checkout, for example) .The initial <block> element provides non-prompting information. The text between <block> tags is converted to speech by the text-to-speech (TTS) module of a VoiceXML implementation. This form is a simple directed form – fields are presented to the user sequentially.

Field: card_type

Every field contains a hidden variable with the same name as the name of the field. When the user provides a valid answer that text is bound to the hidden field item variable. These variables can be tested and manipulated by ECMAScript statements (*e.g.* JavaScript 1.3 statements).

This field contains a *tapered prompt*. The first time the prompt is read to the user he is asked "What kind of credit card do you have ?" If the prompt needs to be repeated a shorter prompt, "Type of card ?" is used. This is done under the presumption that as the user becomes more familiar with the request they prefer shorter, reminder-type prompts.

This field also contains an *inline grammar*. Grammars can be specified as an external file or encoded directly into the document. This grammar, for example, returns mastercard if "master" or "mastercard" is uttered. If the user's utterance does not conform to the grammar a system-specific message indicating it did not understand is displayed.

The <help> block is a catch statement. Whenever the user utters "help" or a like phrase the system throws a help event which is caught by this block. The <help> tag is semantically equivalent to <catch event="help">.

Field: card_num

This form field specifies a grammar using the TYPE attribute. Type digits corresponds to a built-in grammar for recognizing spoken digits.

The code between the <filled> tags is executed immediately after the user provides a valid response to the prompt. The job of code in the <filled> block is to verify the correctness of answer. It is analogous to verifying form fields in an HTML form using Javascript. In this case we check to see if the credit-card number has the right number of digits (which depends on what type of credit card is being used). If the credit card number is not valid the hidden form variable for this field is cleared and a nomatch event is thrown. Since no event handler has been provided the default nomatch handler is used – this causes the prompt to be repeated.

Note that variables and control flow are declared as elements in the markup language. Furthermore note that the attributes of these statements are ECMAScript expressions.

Field: expiry_date

No significant structural differences from the previous field. Notice that the since the value returned by the digits grammar is a string normal Javascript string manipulation functions can be used.

Field: confirm

Elements of the confirmation prompt (i.e. card type, number, expiry date) are filled in using the hidden variables of previous fields. The type boolean attribute for the field indicates the grammar is the set of confirm/cancel statements.

What is significant is that variables can be submitted using POST, or in this case GET, to a server-side script. This allows one to integrate VoiceXML documents into a larger web site.

Menu: anonymous

Menus lead to some action being chosen. There is no field filling per-se and thus a name attribute is not required. Menus consist of choices and an action. The action can be a URI to load or an event to throw when that <choice> element is picked.

The choices of a menu implicitly for a grammar for the dialog. In this case the user can choose to say “Cancel Order”, “Checkout”, or “Save cart and exit”. After a system specified period if no input is given (*i.e.* silence) a noinput event is thrown.

In this case no DTMF grammar is specified. It is possible to map each of the options to keypad tones as well as to voice commands. Both grammars can be active simultaneously.

Example Conversation

The VoiceXML fragment explained above allows the following conversation, modified from [3]:

C: We now need your credit card type, number, and expiration date.
C: What kind of credit card do you have?
H: Discover
C: I did not understand what you said. (a platform-specific default message.)
C: Type of card? (the second prompt is used now.)
H: Shoot. (treated as "help" by this platform)
C: Please say Visa, Master card, or American Express.
H: Uh, Amex. (this platform ignores "uh")
C: What is your card number?
H: One two three four wait
C: I did not understand what you said.
C: Card number?
H: (uses DTMF) 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 #
C: What is your card's expiration date?
H: one two oh one
C: I have Amex number 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 expiring on 1 2 0 1. Is this correct ?
H: Yes
C: Thank you. Which of the following actions do you wish to take ? Cancel Order, Checkout, or

Save cart and exit ?
H: <long pause>
C: Please say one of Cancel Order, Checkout, or Save cart and exit.
H: Checkout

<goes to checkout.vxml>

Additional Features of VoiceXML

Recorded Audio

If music needs to be played (or if a more natural voice is needed for certain static prompts) the developer may record the sound into a .wav file and use it in a VoiceXML document. This is done using the <audio src="...."> tag. This is analogous to placing an image in an HTML document.

Tags and functionality to allows applications to record the user's voice are also provided for in the specification.

Embedded Scripts

Just like in HTML it can be useful to insert client-side scripts into an HTML document. ECMAScript (Javascript 1.3) functions can be inserted into VoiceXML documents using the <script> tag and CDATA XML escapes. For example,

```
<script> <![CDATA[ function circ(radius) { return 2*Math.PI*radius; } ]]> </script>
```

Allows the circ() function to be use in control logic within a VoiceXML document.

Server-Side Objects

Platform specific functionality and user-defined extensions (*e.g.* Java classes) may be accessed with the <object> tag which gives the location of the external object and parameters to pass. For example, from [3]:

```
<object name="debit"
classid="method://credit_card/gather_and_debit"
data="http://www.recordings.example/prompts/credit/jesse.jar"/>
<param name="amount" expr="document.amt"/>
<param name="vendor" expr="vendor_num"/>
</object>
```

Caching

Just as caching is used to reduce delivery time for HTML documents so to is it used for VoiceXML documents. Audio recordings in documents, like large images in a document, are one of the main reasons for caching. The scheme used is similar to that of many HTML browsers: if the document is not already available or is older than the document on the server fetch; otherwise use the cached document[4].

Speech Markup

The text-to-speech (TTS) module should be fairly sophisticated to handle the requirements of VoiceXML. The specification allows one to provide speech markup – instructions on how to say a sentence. The develop can specify emphasis (<emp>) on certain words, pauses (<break>), prosody[5] (<pros>). There is also the ability to say text in a certain way (<sayas>). When we write down a statement like "$199" we read it as "One-hundred and ninety-nine dollars". <sayas> explicitly tells the system how to read back the phrase by giving the type of the phrase (*e.g.* currency, telephone number).

Although the interpreter must accept all speech markup, the TTS engine is not required to implement all the functionality needed to operationalize the markup.

Barge-in

Some users do not wish to finish listening to a prompt before providing a response. However, some prompts, like legal disclaimers, should be fully spoken to the user before they can continue. The developer can specify whether or not a prompt is interruptible.

[4]This describes safe caching. Some HTML browsers (and VoiceXML browsers unless told otherwise) will use a faster caching procedures that avoid file modification checks. This opens up the possibility for inconsistencies between the files in the interpreter and the files on the web server.

[5]Prosody defines the rate, volume, and pitch of speech.

Bibliography

[1] Ferrans, J. VoiceXML Tutorial: Introduction. Retrieved from the World Wide Web February 23rd, 2001: http://www.voicexml.org/tutorials/intro1.html

[2] World Wide Web Consortium. XML in 10 points. Retrieved from the World Wide Web February 28th, 2001: http://www.w3.org/XML/1999/XML-in-10-points

[3] World Wide Web Consortium. Voice eXtensible Markup Language (VoiceXML) version 1.0. W3C Note 05, May 2000. Retrieved from the World Wide Web February 23rd, 2001: http://www.w3.org/TR/voicexml/

[4] Sun Microsystems. Java Speech Grammar Format Specification, version 1.0. Retrieved from the World Wide Web February 24th, 2001: http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/index.html

[5] World Wide Web Consortium. Grammar Requirements for Voice Markup. Working Draft 23, December 1999. Retrieved from the World Wide Web February 27th, 2001: http://www.w3.org/TR/voice-grammar-reqs/

[6] Tellme.com. Grammar Library: YES_NO intrinsic grammar. Retrieved from the World Wide Web February 23rd, 2001: http://studio.tellme.com/library/grammar/yes_no_src.html

[7] Martin, D. Adapting Content for VoiceXML. Retrieved from the World Wide Web February 23rd, 2001: http://www.xml.com/pub/a/2000/08/23/didier/

[8] World Wide Web Consortium. "Voice Browser" Activity. Retrieved from the World Wide Web February 23rd, 2001: http://www.w3.org/Voice/

[9] World Wide Web Consortium. Extensible Markup Language. Retrieved from the World Wide Web February 27th, 2001: http://www.w3.org/XML/