Volume 6, Issue 11, November 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

VOICE-XML - An Innovative Approach in Recent Communication Media

Rakhi Akhare[1], Smita Ambarkar[2], Monica Mangla[3]
Assistant Professor, L.T.C.O.E., Koparkhairane, Navi Mumbai,
Maharashtra, India

Abstract: ***VoiceXML is a language for creating voice-user interfaces, particularly for the telephone. It uses speech recognition and touchtone (DTMF keypad) for input, and pre-recorded audio and text-to-speech synthesis (TTS) for output. It is based on the Worldwide Web Consortium's (W3C's) Extensible Markup Language (XML), and leverages the web paradigm for application development and deployment. By having a common language, application developers, platform vendors, and tool providers all can benefit from code portability and reuse. With VoiceXML, speech recognition application development is greatly simplified by using familiar web infrastructure, including tools and Web servers. Instead of using a PC with a Web browser, any telephone can access VoiceXML applications via a VoiceXML "interpreter" (also known as a "browser") running on a telephony server. Whereas HTML is commonly used for creating graphical Web applications, VoiceXML can be used for voice-enabled Web applications.***

Keywords: TTS, DTMF, ASR

I. INTRODUCTION

One popular type of application is the *voice portal*, a telephone service where callers dial a phone number to retrieve information such as stock quotes, sports scores, and weather reports. Voice portals have received considerable attention lately, and demonstrate the power of speech recognition-based telephone services. These, however, are certainly not the only application for VoiceXML. Other application areas, including voice-enabled intranets and contact centers, notification services, and innovative telephony services, can all be built with VoiceXML.By separating application logic (running on a standard Web server) from the voice dialogs (running on a telephony server), VoiceXML and the voice-enabled Web allow for a new business model for telephony applications known as the *Voice Service Provider*. This permits developers to build phone services without having to buy or run equipment. While originally designed for building telephone services, other applications of VoiceXML, such as speech-controlled home appliances, are starting to be developed.

1.1 Goals of VOICEXML:

VoiceXML's main goal is to bring the full power of web development and content delivery to voice response applications, and to free the authors of such applications from low-level programming and resource management. It enables integration of voice services with data services using the familiar client-server paradigm. A voice service is viewed as a sequence of interaction dialogs between a user and an implementation platform. The dialogs are provided by document servers, which may be external to the implementation platform. Document servers maintain overall service logic, perform database and legacy system operations, and produce dialogs. A VoiceXML document specifies each interaction dialog to be conducted by a VoiceXML interpreter. User input affects dialog interpretation and is collected into requests submitted to a document server. The document server may reply with another VoiceXML document to continue the user's session with other dialogs.

VoiceXML is a markup language that:

1. Minimizes client/server interactions by specifying multiple interactions per document. Shields application authors from low-level, and platform-specific details.
2. Separates user interaction code (in VoiceXML) from service logic (CGI scripts).
3. Promotes service portability across implementation platforms. VoiceXML is a common language for content providers, tool providers, and platform providers.
4. Is easy to use for simple interactions, and yet provides language features to support complex dialogs.

While VoiceXML strives to accommodate the requirements of a majority of voice response services, services with stringent requirements may best be served by dedicated applications that employ a finer level of control.

[1] Typeset names in 8 pt Times Roman, uppercase. Use the footnote to indicate the present or permanent address of the author.

1.2 Scope of VOICEXML:

The language describes the human-machine interaction provided by voice response systems, which includes:

1. Output of synthesized speech (text-to-speech).
2. Output of audio files.
3. Recognition of spoken input.
4. Recognition of DTMF input.
5. Recording of spoken input.
6. Telephony features such as call transfer and disconnect.

The language provides means for collecting character and/or spoken input, assigning the input to document-defined request variables, and making decisions that affect the interpretation of documents written in the language. A document may be linked to other documents through Universal Resource Identifiers (URIs).

II. CREATING A BASIC VOICEXML DOCUMENT

VoiceXML is an extensible markup language (XML) for the creation of automated speech recognition (ASR) and interactive voice response (IVR) applications. Based on the XML tag/attribute format, the VoiceXML syntax involves enclosing instructions (items) within a tag structure in the following manner:

< element_name attribute_name="attribute_value">

.....contained items.....

< /element_name>

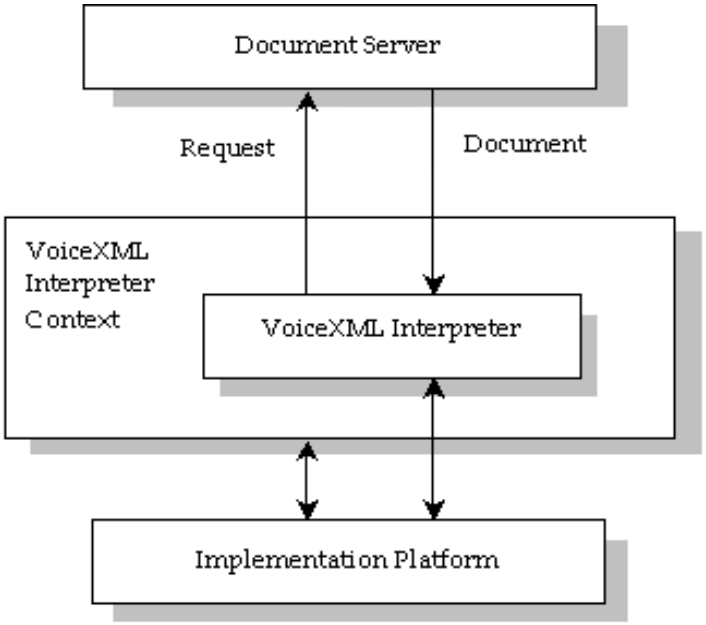
A VoiceXML application consists of one or more text files called documents. These document files are denoted by a ".vxml" file extension and contain the various VoiceXML instructions for the application. It is recommended that the first instruction in any document to be seen by the interpreter be the XML version tag:

< ?xml version="1.0"?>

The remainder of the document's instructions should be enclosed by the vxml tag with the version attribute set equal to the version of VoiceXML being used ("1.0" in the present case) as follows:

< vxml version="1.0">

2.1 Architectural Model:



A document server (e.g. a web server) processes requests from a client application, the VoiceXML Interpreter, through the VoiceXML interpreter context. The server produces VoiceXML documents in reply, which are processed by the VoiceXML Interpreter. The VoiceXML interpreter context may monitor user inputs in parallel with the VoiceXML interpreter. For example, one VoiceXML interpreter context may always listen for a special escape phrase that takes the user to a high-level personal assistant, and another may listen for escape phrases that alter user preferences like volume or text-to-speech characteristics.

The implementation platform is controlled by the VoiceXML interpreter context and by the VoiceXML interpreter. For instance, in an interactive voice response application, the VoiceXML interpreter context may be responsible for detecting an incoming call, acquiring the initial VoiceXML document, and answering the call, while the VoiceXML interpreter conducts the dialog after answer. The implementation platform generates events in response to user actions (e.g. spoken or character input received, disconnect) and system events (e.g. timer expiration). Some of these events are acted upon by the VoiceXML interpreter itself, as specified by the VoiceXML document, while others are acted upon by the VoiceXML interpreter context.

2.2 Implementation Platform Requirements:

This section outlines the requirements on the hardware/software platforms that will support a VoiceXML interpreter.

Document acquisition. The interpreter context is expected to acquire documents for the VoiceXML interpreter to act on. In some cases, the document request is generated by the interpretation of a VoiceXML document, while other requests are generated by the interpreter context in response to events outside the scope of the language, for example an incoming phone call.

Audio output. An implementation platform can provide audio output using audio files and/or using text-to-speech (TTS). When both are supported, the platform must be able to freely sequence TTS and audio output. Audio files are referred to by a URI. The language does not specify a required set of audio file formats.

Audio input. An implementation platform is required to detect and report character and/or spoken input simultaneously and to control input detection interval duration with a timer whose length is specified by a VoiceXML document.

- It must report characters (for example, DTMF) entered by a user.
- It must be able to receive speech recognition grammar data dynamically. Some VoiceXML elements contain speech grammar data; others refer to speech grammar data through a URI. The speech recognizer must be able to accommodate dynamic update of the spoken input for which it is listening through either method of speech grammar data specification.
- It should be able to record audio received from the user. The implementation platform must be able to make the recording available to a request variable.

Concepts. A VoiceXML document (or a set of documents called an application) forms a conversational finite state machine. The user is always in one conversational state, or dialog, at a time. Each dialog determines the next dialog to transition to. Transitions are specified using URIs, which define the next document and dialog to use. If a URI does not refer to a document, the current document is assumed. If it does not refer to a dialog, the first dialog in the document is assumed. Execution is terminated when a dialog does not specify a successor, or if it has an element that explicitly exits the conversation.

Dialogs and subdialogs There are two kinds of dialogs: forms and menus. Forms define an interaction that collects values for a set of field item variables. Each field may specify a grammar that defines the allowable inputs for that field. If a form-level grammar is present, it can be used to fill several fields from one utterance. A menu presents the user with a choice of options and then transitions to another dialog based on that choice.

A subdialog is like a function call, in that it provides a mechanism for invoking a new interaction, and returning to the original form. Local data, grammars, and state information are saved and are available upon returning to the calling document. Subdialogs can be used, for example, to create a confirmation sequence that may require a database query; to create a set of components that may be shared among documents in a single application; or to create a reusable library of dialogs shared among many applications.

Sessions A session begins when the user starts to interact with a VoiceXML interpreter context, continues as documents are loaded and processed, and ends when requested by the user, a document, or the interpreter context.

Application An application is a set of documents sharing the same application root document. Whenever the user interacts with a document in an application, its application root document is also loaded. The application root document remains loaded while the user is transitioning between other documents in the same application, and it is unloaded when the user transitions to a document that is not in the application. While it is loaded, the application root document's variables are available to the other documents as application variables, and its grammars can also be set to remain active for the duration of the application

Grammers Each dialog has one or more speech and/or DTMF grammars associated with it. In machinedirected applications, each dialog's grammars are active only when the user is in that dialog. In mixed initiative applications, where the user and the machine alternate in determining what to do next, some of the dialogs are flagged to make their grammars active (i.e., listened for) even when the user is in another dialog in the same document, or on another loaded document in the same application. In this situation, if the user says something matching another dialog's active grammars, execution transitions to that other dialog, with the user's utterance treated as if it were said in that dialog. Mixed initiative adds flexibility and power to voice applications.

Events VoiceXML provides a form-filling mechanism for handling "normal" user input. In addition, VoiceXML defines a mechanism for handling events not covered by the form mechanism.

Events are thrown by the platform under a variety of circumstances, such as when the user does not respond, doesn't respond intelligibly, requests help, etc. The interpreter also throws events if it finds a semantic error in a VoiceXML document. Events are caught by catch elements or their syntactic shorthand. Each element in which an event can occur may specify catch elements. Catch elements are also inherited from enclosing elements "as if by copy". In this way, common event handling behavior can be specified at any level, and it applies to all lower levels.

Links A link supports mixed initiative. It specifies a grammar that is active whenever the user is in the scope of the link. If user input matches the link's grammar, control transfers to the link's destination URI. A <link> can be used to throw an event to go to a destination URI.

2.3 Supported audio file formats:

VoiceXML recommends that a platform support the playing and recording audio formats specified below. Note: a platform need not support both A-law and μ-law simultaneously.

Audio Format	MIME Type
Raw (headerless) 8kHz 8-bit mu-law [PCM] single channel.	audio/basic
Raw (headerless) 8kHz 8 bit A-law [PCM] single channel.	audio/x-alaw-basic
WAV (RIFF header) 8kHz 8-bit mu-law [PCM] single channel.	audio/wav
WAV (RIFF header) 8kHz 8-bit A-law [PCM] single channel.	audio/wav

2.4 Example:

The top-level element is <vxml>, which is mainly a container for dialogs. There are two types of dialogs: forms and menus. Forms present information and gather input; menus offer choices of what to do next. This example has a single form, which contains a block that synthesizes and presents "Hello World!" to the user. Since the form does not specify a successor dialog, the conversation ends.

Our second example asks the user for a choice of drink and then submits it to a server script:

```
<?xml version="1.0"?>
<vxml version="1.0">
<form>
<field name="drink">
<prompt>Would you like coffee, tea, milk, or nothing?</prompt>
<grammar src="drink.gram" type="application/x-jsgf"/>
</field>
<block> <submit next="http://www.drink.example/drink2.asp"/> </block>
</form>
</vxml>
```

A field is an input field. The user must provide a value for the field before proceeding to the next element in the form. A sample interaction is:

C (computer): Would you like coffee, tea, milk, or nothing?

H (human): Orange juice.

C: I did not understand what you said.

C: Would you like coffee, tea, milk, or nothing?

H: Tea

C: (continues in document drink2.asp)

III. APPLICATIONS OF VOICEXML

Below are a few examples in which VoiceXML applications can be used:

Voice portals: Just like Web portals, voice portals can be used to provide personalized services to access information like stock quotes, weather, restaurant listings, news, etc.

Location-based services: You can receive targeted information specific to the location you are dialing from. Applications use the telephone number you are dialing from.

Voice alerts (such as for advertising): VoiceXML can be used to send targeted alerts to a user. The user would sign up to receive special alerts informing him of upcoming events.

Commerce: VoiceXML can be used to implement applications that allow users to purchase over the phone. Because voice gives you less information than graphics, specific products that don't need a lot of description (such as tickets, CDs, office supplies, etc.) work well.

IV. CONCLUSION

VoiceXML is designed for creating audio dialogs that feature synthesized speech, digitized audio, recognition of spoken and DTMF key input, recording of spoken input, telephony, and mixed-initiative conversations. Its major goal is to bring the advantages of web-based development and content delivery to intera. With this we can conclude VoiceXML development in speech recognition application is greatly simplified by using familiar web infrastructure, including tools and Web servers. Instead of using a PC with a Web browser, any telephone can access VoiceXML applications via a VoiceXML "interpreter" (also known as a "browser") running on a telephony serverctive voice response applications.

REFERENCES

[1] Adam Hocek,David Cuddihy(2002);Definitive VoiceXML.
[2] James Larson,.;(2002) Introduction to Developing Speech Applications
[3] Bob C. Edgar, (2001) The VoiceXML Handbook.
[4] J.A.Larson,(2003),IEEE,"VoiceXML and the W3C speech inteface framework".
[5] Sharad Kumar Singh (2013),IJCA,"XML based interactive Voice Response System".
[6] MichaelMorrison, (1999),"XML Unleashed".
[7] http://www.w3.org/TR/2000/NOTE-voicexml-20000505.

