VoiceXML Overview

What is VoiceXML?

VoiceXML (or simply VXML) stands for Voice Extensible Markup Language. Such as HTML is a markup language for creating distributed textual/visual applications, VoiceXML is an XML based markup language for creating distributed voice applications. VoiceXML is a standard based on XML that allows Web applications and content to be accessed by a phone. You can develop speech-based telephony applications using VoiceXML.

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 replies with another VoiceXML document to continue the user's session with other dialogs.

VoiceXML is a markup language that:

- Minimizes client/server interactions by specifying multiple interactions per document.
- Shields application authors from low-level, and platform-specific details.
- Separates user interaction code (in VoiceXML) from service logic (e.g. CGI scripts).
- Promotes service portability across implementation platforms. VoiceXML is a common language for content providers, tool providers, and platform providers.
- 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.

Scope of VoiceXML

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

- Output of synthesized speech (text-to-speech).
- Output of audio files.
- Recognition of spoken input.

- Recognition of DTMF input.
- Recording of spoken input.
- Control of dialog flow.
- Telephony features such as call transfer and disconnect.

The language provides means for collecting character and/or spoken input, assigning the input results 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).

Architectural Model

The distributed application model of the Voice browser (Vxi) for Asterisk assumed by VoiceXML is a system based on three levels with the following architecture:

When a call is received, the implementation platform (telephony platform) sends an event to the VoiceXML interpreter, which in turn looks in its context for the URI of the initial document to fetch. A request is then sent to the document server for the initial document. The document server in turn sends the document to the VoiceXML interpreter that interprets the document and executes it appropriately using the services of the telephony platform. The interpretation may result in a message to be prompted to the caller through the implementation platform or the VoiceXML interpreter making additional document requests to the document server. Under the document's control, the VoiceXML interpreter directs the implementation to:

- Send prompts, messages, or other audio/video material to the user
- Accept numeric input that the user enters by DTMF (Touch Tone©) signals)
- Accept voice input and recognize the words by receiving grammar data dynamically
- Simply record voice and video input
- Send the user's information to a Web site or other Internet server
- Receive information from the Internet, and pass it to the user

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 the 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 the VoiceXML interpreter itself, as specified by the VoiceXML document, while others are acted upon the interpreter context. VoiceXML applications are comprised of a collection of documents. You will recall that a document is the equivalent of an HTML page, and encapsulates one or more dialogs. Execution of a dialog typically involves presentation of information to the caller, along with collection of input from that caller. Transition from one dialog to another is controlled by the currently executing dialog. Transition occurs via the <goto > or < submit > tags.

Application Model

Two application models exist to create a VoiceXML service. While simpler applications may use static VoiceXML pages, nearly all rely on dynamic VoiceXML page generation using an application server.

Static Applications

The simplest VoiceXML application is simply going to be a set of VoiceXML pages, returned by a web server to the VoiceXML platform. These pages are interpreted to control the interaction with the user.

NOTE: The pages can be hosted on a Web Server or stored and referenced directly in the local disk.

Below is a Static VoiceXML page example:

```
<?xml version="1.0"?>
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml">
  <form>
   <block>
    <prompt>
    Hello world!
    </prompt>
   </block>
  </form>
</vxml>
```

Dynamic Applications

Dynamic applications function in much the same way as Static ones, with the exception that some or all of the pages are generated dynamically, by a server-based technology such as JSP/JavaBeans, ASP/ActiveX, ColdFusion, or scripting languages such as perl. Typically, HTTP is used as the transport protocol for fetching VoiceXML pages. An overview of this model is shown below.

In a well-architected web application, the voice interface and the visual interface share the same back-end business logic. VoiceXML dynamic applications use same frameworks as Web applications. The content generated can be the result of several interactions with a database, a directory or an application interface (billing system, xml/http server...).

Below is a Dynamic VoiceXML page example:

The first snippet shows a typical VoiceXML file, called snack.vxml. It includes a snacks field that defines the grammar listing the available snacks. When the user responds with one or more of the snack options, the response is stored in the snacks field. Then the response is submitted to the dynamic JavaServer Page, called SnackList.jsv.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE vxml PUBLIC "-//W3C//DTD VOICEXML 2.1//EN"
"vxml20-1115.dtd">
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml">
   <meta name="GENERATOR" content="WebSphere Voice Toolkit" />
   <form id="getSnack">
        <field name="snacks">
            <grammar version="1.0" mode="voice"
                     type="application/srgs+xml"
                     root="snackfood">
                <rule id="snackfood" scope="public">
                   <one-of>
                      <item>crackers</item>
                      <item>ice cream</item>
                      <item>pretzels</item>
                      <item>chips</item>
                      <item>cookies</item>
                   </one-of>
                </rule>
            </grammar>
            <prompt>What is your favorite snack?</prompt>
         </field>

        <filled>
            <submit next="SnackList.jsv" namelist ="snacks"/>
        </filled>
        </form>
</vxml>
```

The second snippet, called SnackList.jsv, illustrates how it accepts the response. In this example, the conditional logic inside the block tag directs the program to respond with one of two responses: “You selected…” and adds the choice or choices from the user, or responds with “Do you like snacks?”

```
<?xml version="1.0" encoding="iso-8859-1"?>

<% response.setContentType("text/x-vxml"); %>
<%! String[] snacks; %>

<jsp:useBean id="SnackBean" class="food.SnackBean">
  <jsp:setProperty name="SnackBean"
    property="snacks"
    param="faveSnacks" />
</jsp:useBean>

<vxml version="1.0" lang="en_US">
  <form>
    <block>
                 <%   snacks = SnackBean.getSnacks();
                 if (snacks != null) {
```

```
      out.println("You selected: ");
      for (int i = 0; i < snacks.length; i++) {
        out.println (" "+snacks[i]+" ");
      }
    } else
      out.println ("Do you like snacks?");
                %>
    </block>
  </form>
</vxml>
```