VoiceXML

Matti Airas

November 29, 2000

Contents

1 Introduction

VoiceXML is an XML-based language 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 [10].

Today, voice applications are abundant, ranging from simple information retrieval systems, e.g. weather services, to automated mail order and ticket reservation systems. While widely used, the development systems are proprietary and re-implementation of applications using a system of different vendor is costly. Even worse, the interaction model of the service may not match well the established commodity systems.

The goal of XML is to make the well-established client-server architecture of WWW applications available to interactive voice response applications. This will make it easier to treat telephony applications as an alternative user interface for ordinary WWW or mobile services. By defining the data and interaction with XML, standard tools can be used, reducing product development time and simplifying development.

2 Architectural Model of VoiceXML

The VoiceXML architecture is modeled to closely match that of a web service client-server model. See figure 1 for an overview of the architectural model. A document server, e.g. a web server acts as an architectural backbone, handling requests from the VoiceXML interpreter and providing it documents. The VoiceXML interpreter handles the user inputs together with VoiceXML interpreter context. The VoiceXML interpreter parses the actual VoiceXML documents and the state information encoded within, while the VoiceXML interpreter context responds to certain system-wide pre-defined events, e.g. special escapes for reaching human operator, adjusting output volume or speech synthesis characteristics.

The VoiceXML interpreter controls the implementation platform together with VoiceXML interpreter context. The implementation platform is responsible for the actual hardware interaction, text-to-speech synthesis and speech recognition. The platform generates events according to user actions like spoken or DTMF input, or disconnect. The events are then handled as defined in the VoiceXML document by the VoiceXML interpreter, or if the interaction is not defined in the VoiceXML interpreter, by the VoiceXML interpreter context.

3 VoiceXML Language Overview

This section relies heavily on VoiceXML 1.0 specification document [10], and if not otherwise mentioned, the information can be assumed to be taken from there.

3.1 Application

An *application* is a set of documents which share the same *application root document*. The application root document is loaded whenever the user enters

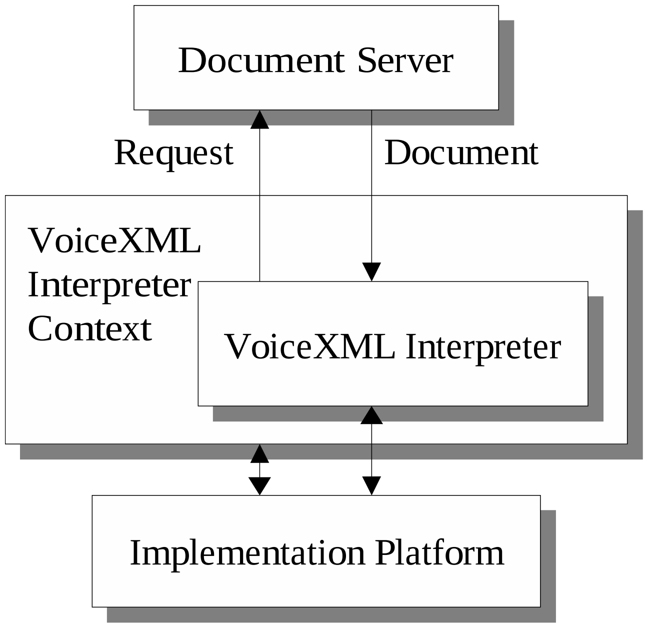

Figure 1: VoiceXML Architecture

a document within the application. The root document remains loaded as long as the user stays in that application.

While loaded, the variables defined in the application root document, *application variables* are available to other documents within the same application. The grammars of the root document can also be set to remain active throughout the lifespan of the application.

3.2 Sessions

A session begins when a user begins to interact with the VoiceXML interpreter context, continues as long as user interacts with the interpreter context and ends when any of the parties requests it.

In real-world-terms, a session usually equals to a phone call or similar contact with the computer running the VoiceXML server.

3.3 Dialogs and Subdialogs

Dialogs define an interaction between the user and the application, in which data is collected from user input. The data can either be stored to variables for further analysis (forms) or a transition to a new dialog may be performed (menu).

Subdialogs are dialogs, into which a function call like transition can be made. After completing the dialog, program execution is returned to the caller.

Subdialogs can be used to create a reusable library of dialogs that can be shared throughout the application or even many applications.

3.4 Forms

Forms are key elements of VoiceXML documents. Forms define an interaction with user, in which a prompt is played out and the response of the user is interpreted according to given rules, *the grammar*. In addition to fields, forms may as well have declarations of non-field item variables, event handlers and "filled" actions, which are blocks of procedural logic that execute when certain combinations of field items are filled in.

Forms are interpreted by an implicit form interpretation algorithm (FIA). The form interpretation algorithm visits the first field item whose guard condition is not met. The default condition just tests whether the field variable is set, thus making the fields to be prompted in order.

- selecting and playing out prompts
- receiving user input, either in response to the form, or a generic event (e.g. "help") and
- interpreting any met "filled" actions.

Below is an example of a simple form.

```
<form>
  <field name="maincourse">
    <prompt>Please select an entree. Today, we re featuring <enumerate/></prompt>
    <option dtmf="1" value="fish"> swordfish </option>
    <option dtmf="2" value="beef"> roast beef </option>
    <option dtmf="3" value="chicken"> frog legs </option>
    <filled>
      <submit next="/cgi-bin/maincourse.cgi"
        method="post" namelist="maincourse"/>
    </filled>
  </field>
</form>
```

3.4.1 Form Items

Form items are elements of form, which may be visited during the form interpretation. They all have result variables, which are set when that item is filled. The also have guard conditions, which control the entry to that item. Usually the guard item just blocks entry if the field item variable is set. Form items may be divided to field items and control items.

Field items define a field item variable that is set according to user input. Field items have prompts that say what the user should say or key in, grammars that define allowed inputs, and event handlers that process any resulting events. Also "filled" elements which state an action to be taken after the field item variable is set, may be defined.

Control items define either blocks of procedural statements used for prompting and computation, but not collecting input, or a block of initial interaction with user in mixed initiative forms, which are used to ask the user for input that matches the form level grammar.

Fields specify input items to be gathered from the user. Possible field values may be defined by a built-in grammar, a user-defined grammar or by an option list. Arbitrary field values, for instance an audio recording, are supported as well.

Built-in grammars support entries of boolean (yes/no) values, dates, digits, currencies, numbers, phone numbers and times.

Explicit grammars can be specified using URLs pointing to the grammar document or as in-line. A simple grammar specification example follows.

```
<field name="flavor">
  <prompt>What is your favorite flavor?</prompt>
  <help>Say one of vanilla, chocolate, or strawberry.</help>
  <grammar type="application/x-jsgf">
    vanilla {van} | chocolate {choc} | strawberry {straw}
  </grammar>
  <dtmf type="application/x-jsgf"> 1 {van} | 2 {choc} | 3 {straw} </dtmf>
</field>
```

Mixed initiative forms are forms in which user may fill the whole form without waiting for separate prompts. In these, the `<initial>` element is used to instruct the user to enter information required in the form. In a typical mixed initiative form, the following conversation might occur:

> C: Welcome to the Driving Directions by Phone. Please say something like “from Atlanta Georgia to Toledo Ohio”.
>
> P: Albuquerque to Seattle.
>
> C: Please say something like “from Atlanta Georgia to Toledo Ohio”.
>
> P: From Albuquerque to Seattle.
>
> C: I’m sorry, I still don’t understand. From which city are you leaving?
>
> etc.

Control items include support for transferring the call to a third party, performing actions when form item variables are filled, entering subdialogs or platform specific extensions, entering metadata, and setting parameters for subdialogs or platform objects.

3.5 Variables

VoiceXML variables are in all respects equivalent to ECMAScript variables. The only difference is, variable names beginning with underscore are reserved for internal use.

Variables are declared by `<var>` elements:

```
<var name="vegetable" expr="'cucumber'"/>
<var name="phone"/>
```

They can also be declared by form items:

```
<field name="vegetables" type="number">
  <prompt>How many vegetables would you like?</prompt>
</field>
```

Depending on where variables are defined, they have different *scopes* of visibility. There are five different scopes, namely session, application, document, dialog and anonymous scope. For example, a document variable is visible throughout that VoiceXML document, but not within the whole application, let alone the session.

Variables are referenced in `cond` and `expr` attributes:

```
<if cond="city == 'LA'">
  <assign name="city" expr="'Los Angeles'"/>
<elseif cond="city == 'Philly'"/>
  <assign name="city" expr="'Philadelphia'"/>
<elseif cond="city == 'Constantinople'"/>
  <assign name="city" expr="'Istanbul'"/>
</if>

<assign name="var1" expr="var1 + 1"/>

<if cond="i > 1">
  <assign name="i" expr="i-1"/>
</if>
```

The expression language used in variable references is ECMAScript.

There are several standard session variables defined. These can be used to transmit information like the caller ID, information of the originating line (payphone, mobile phone, prison etc.).

3.6 Grammars

3.6.1 Speech Grammars

A speech grammar specifies a set of utterances which the user may speak, and gives corresponding string values or a set of attribute-value pairs to describe the information or action.

VoiceXML specification 1.0 does not define a grammar format, neither does it require support of a particular grammar format.

The `<grammar>` element is used to define an inline grammar or an external grammar. An inline grammar is specified by the content of the grammar element, while an external grammar is specified by `src` attribute of the `<grammar>` tag.

3.6.2 DTMF Grammars

Data input may as well be performed using DTMF codes generated by pressing the telephone keys. This is supported with DTMF grammars. The definition of the DTMF grammar is performed by using the `<dtmf>` tag. As with speech grammars, no DTMF grammar formats are defined, and the grammars may be inline or external.

3.7 Event handling

Events may be thrown by the implementation platform or by the document interpreter. The platform throws events when the user does not respond, does not respond intelligibly, says some common predefined phrase like “help”, etc. The interpreter throws events when it encounters a semantic error in the document, or when it encounters a ¡throw¿ element.

Elements are caught by *catch elements*, which include `<catch>`, `<error>`, `<help>`, `<noinput>`, and `<nomatch>`. An element inherits the catch elements from its ancestors, if they are not defined within the current context. Thus, common event handling behaviour can be specified at any level of the VoiceXML application.

3.7.1 Event Types

Events may be divided into pre-defined and application-defined events. Events are also subdivided into plain events and error events. Error events are named so that multiple levels of granularity is supported–events may be caught by specific name or by a common prefix. For example, pre-defined events include `cancel`, `telephone.disconnect.hangup`, and `exit`, among a few others. The pre-defined errors include `error.semantic`, and `error.noauthorization`, among others.

3.7.2 Throwing and catching events

Events may be thrown explicitly with `<throw>`. They may be pre-defined ones:

```
<throw event="nomatch"/>
```

Also application-defined elements may be thrown:

```
<throw event="mil.warroom.nuclear.novegetable"/>
```

Elements may be caught with `<catch>` element:

```
<form id="launch_missiles">
  <field name="password">
    <prompt>What is the code word?</prompt>
    <grammar>rutabaga</grammar>
    <help>It is the name of an obscure vegetable.</help>
    <catch event="nomatch noinput" count="3">
      <prompt>Security violation!</prompt>
      <submit next="apprehend_felon" namelist="user_id"/>
    </catch>
  </field>
  <block>
    <goto next="#get_city"/>
  </block>
</form>
```

Some events have implicit default catch elements defined. For example, an `error` event by default exits the interpreter, and `help` events replay the prompt to the user.

3.8 Prompts

The prompt element controls the output of synthesized speech and prerecorded audio. A simple prompt example follows.

```
<prompt>Please say your name.</prompt>
```

The `<prompt>` tags may be omitted, if there are no prompt attributes to be defined and if the prompt contains no speech markup. Below are examples of such prompts:

```
Please say your name.
<audio src="say_your_name.wav"/>
```

Prompts may have special speech markup to indicate prosody, breaks or instructions how a specific word or phrase should be spoken:

```
<prompt> This is <emp>also</emp> computer-generated text.
  <break size="medium"/> Do you like it? </prompt>

<prompt>You are calling <value expr="home_num" class="phone"/></prompt>
<prompt>You are calling
  <sayas class="phone">312-555-1212</sayas>
</prompt>
```

In the current specification of VoiceXML, the text-to-speech engine may ignore some or all of the speech markup.

Prompts may have audio clips intermingled with synthesized speech:

```
<prompt>
  Welcome to the Bird Seed Emporium.
  <audio src="http://www.birdsounds.example/thrush.wav"/>
  We have 250 kilogram drums of thistle seed for
  <sayas class="currency">299.95 euro</sayas>
  plus shipping and handling this month.
  <audio src="http://www.birdsounds.example/mourningdove.wav"/>
</prompt>
```

If the audio file cannot be played, the contents of audio element is played instead. If the content is empty, an appropriate error event is thrown. This makes it possible to define alternate rendering of prompts:

```
<prompt>
  <audio src="welcome.wav"><emp>Welcome</emp> to Voice Portal.</audio>
</prompt>
```

Usually the user may interrupt, or “barge-in” on a prompt. This speeds up conversations and follows good user-interface design guidelines, but sometimes such behaviour might not be suitable. For example, it may be desired to force the user to listen to all of a warning or an advertisement. This may be done by setting the `bargein` attribute to false:

```
<prompt bargein="false"><audio src="eatspam.wav"/></prompt>
```

Prompts may also be set to change with each attempt. Information-requesting prompts may become shorter after a few tries under the assumption that the user is becoming more familiar with the task. Help messages, on the other hand, may become more detailed, or prompts can change randomly just to make the interaction less monotonic and annoying.

Here is an example of changing forms using the `count` attribute:

```
<form id="tapered">
  <block>
    <prompt bargein="false">Welcome to the ice cream survey.</prompt>
  </block>
  <field name="flavor">
    <grammar>vanilla|chocolate|strawberry</grammar>
    <prompt count="1">What is your favorite flavor?</prompt>
    <prompt count="3">Say chocolate, vanilla, or strawberry.</prompt>
    <help>Sorry, no help is available.</help>
  </field>
</form>
```

3.9 Executable Content

VoiceXML documents may contain *executable content*, which refers to blocks of simple procedural logic. Such logic appears in `<block>` form items, `<filled>` actions in forms and fields, and in event handlers.

Executable contents can contain variable definitions and assignments, conditional execution and branches, definitions of ECMAScript functions and statements. Below are some examples of executable content:

```
<if cond="total > 1000">
  <prompt>This is way too much to spend.</prompt>
  <throw "com.xyzcorp.acct.toomuchspent"/>
</if>

<nomatch count="1">
  To open the pod bay door, say your code phrase clearly.
</nomatch>
<nomatch count="2">
  <prompt> This is your <emp>last</emp> chance. </prompt>
</nomatch>
<nomatch count="3">
  Entrance denied.
  <exit/>
</nomatch>
```

4 Example Implementation Platforms

4.1 IBM WebSphere Voice Server

IBM offers the IBM WebSphere Voice Server product as part of their ViaVoice product family [2]. There are two main products in the Voice Server family.

Voice Server with ViaVoice technology supports Voice over IP (VoIP) technologies, enabling Internet-based voice applications. Voice Server for DirectTalk supports VoiceXML application deployment for traditional telephony servers.

The VoIP servers run on Windows NT and operate in a similar environment and scalability as contemporary web servers.

The Voice Server for DirectTalk runs on IBM's AIX based DirectTalk telephony servers. DirectTalk is intended for enterprise and operator-scale services: it supports up to 360 simultaneous calls per server, with ability to cluster servers. Since DirectTalk builds on existing technology and hardware, it is not exclusively VoiceXML – traditional state tables, as well as Java Beans are supported.

IBM offers Windows NT based SDK's for their VoiceXML technologies free of charge.

IBM only supports US English language at the moment.

4.2 New Lucent Speech Server

Lucent has a product called New Lucent Speech Server, which is a telephony server hardware platform running on Solaris x86 and Lucent's own server software [9]. The application environment supports VoiceXML, C++ and Java.

Lucent's Speech Server includes VoiceXML interpreter as well as text-to-speech and voice recognition software. No web server platform is included by default, although considering the wide deployment and matureness of other web server and XML solutions, that cannot be considered as a major drawback.

Lucent emphasizes traditional telephone network services in their product offerings.

4.3 Motorola Internet Exchange

Motorola offers the Motorola Internet Exchange (MIX) technology to implement WML, VoxML and VoiceXML services [5]. The emphasis on their solution is on the presentation format independence – services may be deployed both on WAP and as voice-based telephony services with little extra cost.

Motorola's solution is based on their own Aspira server architecture, of which they unfortunately submit very little concrete information.

Motorala targets a wide variety of applications and customers with their platform. Examples range from large alliance-scale telephone service systems to telecom operator services to applications internal to a single company, like voice-mail management systems.

4.4 Voice Application Portals

With the advent of standardized voice application markup languages, new voice application portals have emerged. One such company is Tellme Studio[6]. The company provides developers a possibility to develop and deploy VoiceXML applications with no charge. The applications can be accessed through the PSTN using Tellme Studio's exchange and selecting the custom extension number of the application. For charge, they will also provide companies their own 1-800 service numbers within the US.

A similar service is provided also by BeVocal [1]. Just like Tellme, BeVocal offers a free development environment for rapid deployment of VoiceXML based services.

An example of services offered in the BeVocal voice portal would be the Business Finder, which functions like a location-sensitive, voice-controlled yellow pages service, containing information of over one million US-based companies. The index may be searched using business categories like “Fast Food” or using brand names like “McDonalds”. The Business Finder service may be accessed using BeVocal’s toll-free number.

Another service example is Driving Directions, which gives driving instructions between two addresses using normal telephone.

Other services offered by BeVocal include stock quotes, weather and traffic reports and flight information.

Tellme, while providing business-oriented services similar to BeVocal, also has a multitude of entertainment services, including soap opera updates, voice-based blackjack, sports information and so on.

5 Related Technologies

5.1 WML

Wireless Markup Language (WML; the language used to render WAP application) provides a reasonably similar application framework as VoiceXML [7]. While the VoiceXML application consists of documents which contain dialogs, a WML application consists of stacks and cards, and navigation in the application is very similar to VoiceXML. Furthermore, the scripting language used in WML, WMLScript is based on JavaScript and ECMAScript.

Since the application structure and information bandwidth are so similar in VoiceXML and WML, and since the applications are intended to be used in similar environments, it should be very compelling to provide phone service counterparts of current WAP applications.

5.2 Java API Speech Markup Language

Java API Speech Markup Language is an XML language submitted to the W3C by Sun Microsystems. Its purpose is to define a language suitable for marking up speech for text-to-speech synthesizers, providing markup about the document format, as well as of prosody, pronounciation and emphasis of speech [3]. While VoiceXML provides similar features in a limited scale, the focus of JSML is much narrower, concentrating on the TTS markup.

5.3 Java API Speech Grammar Format

Java API Speech Grammar (JSGF) defines a standardized grammar to be used in limited-vocabulary speech recognition systems, such as those used in VoiceXML applications [4].

While JSGF is not officially related to VoiceXML, it acts as a natural counterpart to VoiceXML as VoiceXML does not specify any grammar language by itself. It is repeatedly used in the examples of VoiceXML specification document, hinting a strong relationship between the two languages.

5.4 TalkML

TalkML is an experimental XML language for voice browsers developed by HP labs [8]. It seems to be projected for very similar applications as VoiceXML, although it is meant to be interpreted in the client terminal, not in a centralized switch hardware like VoiceXML. Therefore the language is much more limited in scope than VoiceXML.

6 Conclusions

VoiceXML, while being a very young standard, already has a large number of supporting companies and a surprisingly good product support. Its main strengths lie on its ability to take advantage of the tried and established web server development model, which is also used in WAP applications. Furthermore, there is a clear need on the market for standardized platform for telephony application development, since by automating their telephone services companies can often find significant cost savings because of smaller customer service costs.

The main drawback of VoiceXML at the moment is its incomprehensivness. Many central aspects of the language, like the speech and DTMF grammars, are completely unspecified. Furthermore, specifications for such things as supported audio formats are quite lacking. The language also supports many different platform-specific extensions. All these loose ends may lead to a situation similar to different SQL dialects, in which the applications in principle are written in a common language, but the porting costs between different platforms are still considerable.

References

[1] Inc. BeVocal. Web pages, Nov 2000. http://www.bevocal.com/.

[2] Ibm viavoice web site. Web page, November 2000. http://www-4.ibm.com/software/speech/enterprise/ms_3.html.

[3] Sun Microsystems. Java api speech markup language. Web pages, August 1997. http://java.sun.com/products/java-media/speech/forDevelopers/JSML/.

[4] Sun Microsystems. Java api speech grammar format. Web pages, 1998. http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/.

[5] Inc. Motorola. Web pages, Nov 2000. http://mix.motorola.com/.

[6] Tellme Networks. Tellme studio home page. Web page, November 2000. http://www.tellme.com.

[7] The xml cover pages, November 2000. http://www.oasis-open.org/cover/sgml-xml.html.

[8] Dave Raggett. Introduction to talkml. Web pages, 2000. http://www.w3.org/Voice/TalkML/.

[9] Lucent Technologies. Web pages, Nov 2000. http://www.lucent.com/speech/products.html.

[10] Voicexml forum home page, October 2000. http://www.voicexml.org/.