COMP349
Spoken Language Dialogue Systems
VoiceXML 2.1

Rolf Schwitter
schwitt@ics.mq.edu.au
Today’s Program

• VoiceXML 2.1 and Beyond
• Elements in VoiceXML 2.1
• Fetching XML
• Looping through Executable Content
• Detect Barge-in During Prompt Playback
• Referencing Grammars Dynamically
• Referencing Scripts Dynamically
VoiceXML 2.1

- VoiceXML 2.1 is not a replacement for VoiceXML 2.0.
- It only discusses 8 elements.
- Two of these 8 elements are new ones.
- 6 elements are enhancements to existing VoiceXML elements.
- VoiceXML 2.1 is a W3C Recommendation since 19 June 2007.
- See: http://www.w3.org/TR/voicexml21/
VoiceXML 2.1 and Beyond

• Statement from the “Voice Browser” Activity web page:

VoiceXML 2.1 provides a small set of additional features. These will help developers to build even more powerful, maintainable and portable voice-activated services, with complete backwards compatibility with the VoiceXML 2.0 specification.

VoiceXML 3.0 is the next major release of VoiceXML. Its purpose is to provide powerful dialog capabilities that can be used to build advanced speech applications, and to provide these capabilities in a form that can be easily and cleanly integrated with other W3C languages. It will provide enhancements to existing dialog and media control, as well as major new features (e.g. modularization, a cleaner separation between data/flow/dialog, and asynchronous external eventing) to facilitate interoperability with external applications and media components.

• See: http://www.w3.org/Voice/#intro
<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Section</th>
<th>New/Enhanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;data&gt;</td>
<td>Fetches arbitrary XML data from a document server.</td>
<td>5</td>
<td>New</td>
</tr>
<tr>
<td>&lt;disconnect&gt;</td>
<td>Disconnects a session.</td>
<td>8</td>
<td>Enhanced</td>
</tr>
<tr>
<td>&lt;grammar&gt;</td>
<td>References a speech recognition or DTMF grammar.</td>
<td>2</td>
<td>Enhanced</td>
</tr>
<tr>
<td>&lt;foreach&gt;</td>
<td>Iterates through an ECMAScript array.</td>
<td>6</td>
<td>New</td>
</tr>
<tr>
<td>&lt;mark&gt;</td>
<td>Declares a bookmark in a sequence of prompts.</td>
<td>4</td>
<td>Enhanced</td>
</tr>
<tr>
<td>&lt;property&gt;</td>
<td>Controls platform settings.</td>
<td>5.1, 7</td>
<td>Enhanced</td>
</tr>
<tr>
<td>&lt;script&gt;</td>
<td>References a document containing client-side ECMAScript.</td>
<td>3</td>
<td>Enhanced</td>
</tr>
<tr>
<td>&lt;transfer&gt;</td>
<td>Transfers the user to another destination.</td>
<td>9</td>
<td>Enhanced</td>
</tr>
</tbody>
</table>
Fetching XML Data

• The `<data>` element corresponds to AJAX in HTML.
• When the `<data>` element is used, no transition from dialog to dialog is necessary, when information is sent or received.
• The `<data>` element
  – performs the equivalent of an HTTP request in place
  – fetches a block of XML data.
• VoiceXML can parse the result using ECMAScript and DOM.
Excursus: AJAX and HTML
Excursus: AJAX and HTML

- AJAX introduces an AJAX engine between the client and the server.
- The browser loads an AJAX engine at the start of a session.
- The JavaScript AJAX engine is responsible for
  - rendering the interface
  - communicating with the server.
- All user actions are handled by the AJAX engine.
- AJAX engine deals with simple actions (validating, editing, navigation).
- If the engine needs something from the server, requests are made asynchronously using XML (or text).
# Attributes of the `<data>` Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>src</code></td>
<td>The URI specifying the location of the XML data to retrieve.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>The name of the variable that exposes the DOM.</td>
</tr>
<tr>
<td><code>srcexpr</code></td>
<td>Like <code>src</code>, except that the URI is dynamically determined by evaluating the given ECMAScript expression when the data needs to be fetched. If <code>srcexpr</code> cannot be evaluated, an <code>error.semantic</code> event is thrown.</td>
</tr>
<tr>
<td><code>method</code></td>
<td>The request method: <code>get</code> (the default) or <code>post</code>.</td>
</tr>
<tr>
<td><code>namelist</code></td>
<td>The list of variables to submit. By default, no variables are submitted. If a <code>namelist</code> is supplied, it may contain individual variable references which are submitted with the same qualification used in the <code>namelist</code>. Declared VoiceXML and ECMAScript variables can be referenced.</td>
</tr>
<tr>
<td><code>enctype</code></td>
<td>The media encoding type of the submitted document. The default is <code>application/x-www-form-urlencoded</code>. Interpreters must also support <code>multipart/form-data</code> [RFC2388] and may support additional encoding types.</td>
</tr>
<tr>
<td><code>fetchaudio</code></td>
<td>See Section 6.1 of [VXML2]. This defaults to the <code>fetchaudio</code> property described in Section 6.3.5 of [VXML2].</td>
</tr>
<tr>
<td><code>fetchhint</code></td>
<td>See Section 6.1 of [VXML2]. This defaults to the <code>fetchhint</code> property described in Section 5.1.</td>
</tr>
<tr>
<td><code>fetchtimeout</code></td>
<td>See Section 6.1 of [VXML2]. This defaults to the <code>fetchtimeout</code> property described in Section 6.3.5 of [VXML2].</td>
</tr>
<tr>
<td><code>maxage</code></td>
<td>See Section 6.1 of [VXML2]. This defaults to the <code>maxage</code> property described in Section 5.1.</td>
</tr>
<tr>
<td><code>maxstale</code></td>
<td>See Section 6.1 of [VXML2]. This defaults to the <code>maxstale</code> property described in Section 5.1.</td>
</tr>
</tbody>
</table>
Example: Fetching XML Data

Computer: Welcome to Joe's Tuna Shack store locator; if you tell me your zip code, I can help you find the closest shack.

Caller: 94086

Computer: 94086, got it. One moment while I look that up.

Computer: Berkeley, California. There are 4 shacks in Berkeley.

Say "that one" or the name of the street the shack is on for directions:

Computer: 500 Telegraph Avenue

Computer: 486 Dwight Way

Computer: 1719 Shattuck Avenue

Caller: That one

Caller: Got it. Let me get you directions to the shack at 1719 Shattuck Avenue.
Benefits of the `<data>` Element

- In VoiceXML 2.0
  - at least two separate dialogs are required:
    - an initial dialog that
      - greets the caller,
      - collects the zip code, and
      - sends it to the server,
    - another dialog that figures out address of interest.
- In VoiceXML 2.1 one single dialog does the job.
Example: Getting Zip Code and Address

```xml
<form>
  <var name="stores" expr="''"/>
  <block>
    <prompt>
      Welcome to Joe's Tuna Shack Store Locator
    </prompt>
  </block>
  <field name="zipcode" type="digits">
    <prompt>
      What's your zip code?
    </prompt>
    <filled>
      <data name="locations"
            src="http://shacklocs.com/cgi-bin/getlocs.pl"
            namelist="zipcode"/>
    </filled>
  </field>
</form>
```
Example: Getting Zip Code and Address

```html
<script>
  <! [CDATA[
  stores = new Array();

  var root = locations.documentElement;

  var addrs = root.getElementsByTagName( "address" );
  var nums = root.getElementsByTagName( "storenum" );

  for( var i = 0; i < addrs.length; i++ ){
    var addrNode = addrs.item(i);
    var numNode = nums.item(i);
  }

</CDATA[>
</script>
```
Example: Getting Zip Code and Address

```
stores[i].addr = addrNode.text;
stores[i].num = numNode.text;
stores[i].mark = i;
```

...
Example: Retrieved XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<locations>
  <location>
    <address> 500 Telegraph Avenue </address>
    <storenum> 880 </storenum>
  </location>
  <location>
    <address> 486 Dwight Way </address>
    <storenum> 237 </storenum>
  </location>
  <location>
    <address> 1719 Shattuck Avenue </address>
    <storenum> 101 </storenum>
  </location>
</locations>
```
The `<foreach>` Element

- The `<foreach>` element allows for listing items of an ECMAScript array object.
- For example, all available addresses:
  - Computer: 500 Telegraph Avenue
  - Computer: 486 Dwight Way
  - Computer: 1719 Shattuck Avenue
Example: Listing Street Addresses

```xml
<field name="getstore">
  <grammar .../>
  <prompt count="1">Ok, I've got a list of nearby stores.</prompt>
  <prompt>
    You can either say <emphasis level="strong">that one</emphasis> or say the shack address.
  </prompt>
  <prompt>
    <foreach item="st" array="stores">
      <value expr="st.addr"/><break time="250ms"/>
    </foreach>
  </prompt>
  <filled>
    ...
  </filled>
</field>
```
The `<mark>` Element in SSML

- The `<mark>` element was first defined in SSML.
- It places a marker into the text/tag sequence.
- In SSML the `<mark>` element has a required “name” attribute.
- The `<mark>` element can be used
  - to reference a specific location in the text/tag sequence,
  - to insert a marker into an output stream for asynchronous notification.
The `<mark>` Element in SSML

```xml
<?xml version="1.0"?>
<speak version="1.1" xmlns="http://www.w3.org/2001/10/synthesis"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
    http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
    xml:lang="en-US">

    Go from <mark name="here"/> here, to <mark name="there"/> there!

</speak>
```
The `<mark>` Element in VoiceXML 2.1

- The `<mark>` element places a marker into the text/tag sequence.
- The `<mark>` element can be placed within a prompt.
- The VoiceXML interpreter keeps track of the marks.
- When the caller barges in
  - the last mark is returned
  - in a shadow variable
  - with the recognition result.
Example: `<mark> Element

```xml
<field name="getstore" type="digits">
  <grammar ... >
  <prompt count="1">
    Ok, I've got a list of nearby stores.</prompt>
  <prompt>
    You can either say
    <emphasis level="strong">that one</emphasis>
    or say the shack address.
  </prompt>
  <prompt>
    <foreach item="st" array="stores">
      <value expr="st.addr"/><break time="250ms"/>
      <mark nameexpr="st.mark"/>
    </foreach>
  </prompt>
</field>
```
Example: `<mark>` Element

```xml
<filled>
    <var name="idx" expr="0"/>
    <if cond="getstore$.markname != undefined">
        <assign name="idx" expr="getstore$.markname"/>
    </if>
    <prompt>
        Got it. Let me get you directions to shack number
        <value expr="stores[idx].num"/> at
        <value expr="stores[idx].addr"/>.
    </prompt>
    <exit/>
</filled>
</field>
```
The `<mark>` Attributes

The following attributes are available for the `<mark>` element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nameexpr</td>
<td>An ECMAScript expression which evaluates to the name of the mark when the prompt is queued. If nameexpr cannot be evaluated, an <code>error.semantic</code> event is thrown.</td>
</tr>
<tr>
<td>markname</td>
<td>The name of the mark last executed by the SSML processor before barge-in occurred or the end of audio playback occurred. If no mark was executed, this variable is undefined.</td>
</tr>
<tr>
<td>marktime</td>
<td>The number of milliseconds that elapsed since the last mark was executed by the SSML processor until barge-in occurred or the end of audio playback occurred. If no mark was executed, this variable is undefined.</td>
</tr>
</tbody>
</table>
Another `<mark>` Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml">

<var name="played_ad" expr="false"/>
<form>
  <field name="team">
    <prompt>
      <mark name="ad_start"/>
      Baseball scores brought to you by Elephant Peanuts. There's nothing like the taste of fresh roasted peanuts. Elephant Peanuts. Ask for them by name.
      <mark name="ad_end"/>
      <break time="500ms"/>
      Say the name of a team. For example, say Boston Red Sox.
    </prompt>
  </field>
</form>
</vxml>```
Another `<mark>` Example

```xml
<grammar type="application/srgs+xml" src="teams.grxml"/>

<filled>
  <if cond="typeof(team$.markname) == 'string' &&
    (team$.markname=='ad_end' ||
     (team$.markname=='ad_start' &&
      team$.marktime >= 5000))">
    <assign name="played_ad" expr="true"/>
  </if>
  <else/>
  <assign name="played_ad" expr="false"/>
</if>
</filled>
</field>
</form>

</vxml>
```
Referencing Grammars Dynamically

• As we already know, the `<grammar>` element specifies a
  — speech recognition grammar or
  — DTMF grammar.
• VoiceXML 2.1 allows for a “srcexpr” attribute.
• It determines a URI dynamically by evaluating an ECMAScript expression.
• The expression must be evaluated each time the grammar is used.
• An error.semantic event is thrown, if the value cannot be evaluated.
Example: Referencing Grammars Dynamically

```xml
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml">

<form id="get_address">
  <field name="country">
    <grammar type="application/srgs+xml" src="country.grxml"/>
    <prompt>Say a country.</prompt>
  </field>

  <field name="city">
    <grammar type="application/srgs+xml" srcexpr="country + '/cities.grxml'"/>
    <prompt>What city in <value expr="country"/>.</prompt>
  </field>

</form>
```
Example: Referencing Grammars Dynamically

```xml
<field name="street">
    <grammar type="application/srgs+xml"
        srcexpr="country + '/' + city + '/streets.grxml'"/>
    <prompt> What street in <value expr="city"/> are you looking for? </prompt>
</field>

<filled>
    <prompt> You chose <value expr="street"/> in <value expr="city"/> <value expr="country"/> </prompt>
    <exit/>
</filled>
</form>
</vxml>
```
Referencing Scripts Dynamically

• Also scripts can be referenced dynamically.
• For example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml">
  <var name="scripts_baseuri">
    <expr>'http://www.example.org/'</expr>
  </var>
  <form>
    <script srcexpr="scripts_baseuri + 'lib/util.js'"/>
  </form>
</vxml>
```
Take-Home Messages

- VoiceXML 2.1 discusses 8 elements.
- Two of these 8 elements are new:
  - the <data> element
  - the <foreach> element.
- The <data> element fetches XML data from a server.
- The <foreach> elements iterates through an ECMAScript array.
- The other elements are extensions of existing ones.
- Grammars and scripts can be referenced dynamically.