Introduction of VoiceXML

Outline
- Overview
  - What’s Is VoiceXML
  - Why use VoiceXML
  - Heritage and Evolution
- Internet Application Architecture
- Voice Application in VoiceXML
- Future Solution of VoiceXML

What’s Is VoiceXML(1)
- Often described as
  - a dialog markup language
  - a FSA
- 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 conversation

What’s Is VoiceXML(2)
- XML-based
- Web application semantics (SSL, HTTP, cookies, cache, etc)
- Call control semantics (even richer feature included VoIP, SIP)
- Spoken input and DTMF
- Directed and Mixed initiative dialogues

Why use phone and voice
- Main I/O device via phone
  - Far more phones than PCs, esp. in Europe
  - Can’t use a PC while handing some task
- WML/XHTML better than voice but…
  - Not available on legacy phone
  - Small screen restricted than voice output
  - Small keypad more difficult than voice input
- Use the right interface for the Task

Which services suited to Voice Applications?
- Information retrieval
  - News, traffics, stocks
- e-Transactions
  - Customer Service: call center
  - Financial: stock trading
  - Catalog browsing
- Telephone services
- Intranet
- Unification
  - Personal agents, unified messaging
**Heritage and Evolution**

- Internet Application Architecture (1)
- Internet Application Architecture (2)
- Compare to traditional graphic browsing
- Anatomy of a Voice Application
- Dialog (1)

### Internet Application Architecture (1)

**Phone/PC Platform**
- Say “hotspot” word, or submit form
- Click “hotspot” text/images, or submit form

- **Hyperlinking**
  - VoiceXML pages, speech recognition grammars, sound files
  - HTML pages, images, Java applet, Active Object

- **Retrieved from Server**
  - Streaming audio and synthesized text spoken according to mrak-up tags
  - Text and image composed according to mark-up tags

- **Browser Output**
  - VoiceXML/HTML Language
  - Voice Graphical Feature

### Internet Application Architecture (2)

**Anatomy of a Voice Application**

- Application Declaration
- “Document scope” Variables, Grammars, Scripts and Event Handlers

- **Forms, Fields, and Transitions**
  - Dialog
  - Subdialog
  - Transitions
  - POSTing

- **Forms vs. Menus**
  - Form: the `<Form>` tag
  - Collect value for a set of field item variables. Field may specify grammar what input is allowable.
  - Ex. `<form>`
  - `<field name = “menu_choice”>`
  - `<grammar src=”g_fdesk.xml” type= “application/grammar+xml”>`
  - `<prompt>`
  - `<filled>`
  - `<field>`
  - `</form>`
Dialog (2)

- Menu: the <Menu> tag
  - Present the user with choice of option that transition to another dialog based on that choice.
  - Shorthand to a form containing a single anonymous field.
  - Ex.

    ```xml
    <menu id="main_menu" dtmf="true">
      <prompt>…</prompt>
      <choice next="1.vxml">…</choice>
      <choice next="2.vxml">…</choice>
      …
    </menu>
    ```

Dialog (3)

- Directed vs. Mixed Initiative
  - Directed: (Both Form and Menu)
    - The simplest and most common type, form item are executed in sequential order to implement interaction. Also call "Computer directed".
  - Mixed Initiative: (Only in Form)
    - Both computer and human direct the conversation. Must have one or more `<initial>` form item and one or more form-level grammar.
    - More than one field item can be filled as a result of single user utterance.

Directed Form

```xml
<form id="weather_info">
  <block>Welcome to the weather information service.</block>
  <field name="state">
    <prompt>what state?</prompt>
  </field>
  <field name="city">
    <prompt>what city?</prompt>
  </field>
  <block>
    <submit next="/servlet/weather" namelist="city state"/>
  </block>
</form>
```

Mixed Initiative Form

```xml
<form id="weather_info">
  <grammar src="cityandstate.grxml" type="application/grammar+xml"/>
  <block>
    <prompt bargein="false">Welcome to …</prompt>
  </block>
  <initial name="start">
    <help>…</help>
  </initial>
  <field name="state">
    <prompt>what state?</prompt>
    <help>…</help>
  </field>
  <field name="city">
    <prompt>what city?</prompt>
    <help>…</help>
  </field>
  <field name="go_ahead" type="boolean" modal="true">
    <prompt>hear the forecast for <value expr="city" />, <value expr="state" />?</prompt>
    <filled>
      <if cond="go_ahead">
        <submit next="/servlet/weather" namelist="city state"/>
      </if>
      <clear namelist="start city state go_ahead"/>
    </filled>
  </field>
</form>
```

Subdialog

- A mechanism for decomposing dialogs or creating reusable components.
- Subdialog and callee dialog are independent, even in the same doc. Variables are not shared with the called subdialog:
  - All variable bindings applied in the subdialog context are lost on return to the calling context.
- Invoked doc provides the results using <return>, and values are accessed through the name attribute which on the <subdialog>.
- Events thrown in a subdialog are treated by local event handlers within its context.
- The <param> in <subdialog> specify the parameters to pass to the subdialog. But must be declared using <var> elements in it.
  - If no corresponding <param> is specified to <var>, an expr attribute is used as a default value, or undefined

```xml
<var name="account_number"/>
<var name="home_phone"/>
<subdialog name="accountinfo" src="acct_info.vxml#basic">
  <filled>
    <assign name="account_number" expr="accountinfo.acctnum"/>
    <assign name="home_phone" expr="accountinfo.acctphone"/>
  </filled>
</subdialog>
```

Customer Service Application (app.vxml)

```xml
<grammar src="account.vxml" type="application/grammar+xml"/>
```

```xml
<grammar src="accountinfo.vxml#basic">
  <filled>
    <assign name="account_number" expr="accountinfo.acctnum"/>
    <assign name="home_phone" expr="accountinfo.acctphone"/>
  </filled>
</subdialog>
```
Account Information Subdialog (acct_info.vxml)

```xml
<?xml version="1.0"?>
<vxml version="2.0">
<form id="basic">
  <field name="acctnum" type="digits">
    <prompt> What is your account number? </prompt>
  </field>
  <field name="acctphone" type="phone">
    <prompt> What is your home telephone number? </prompt>
    <filled> return namelist="acctnum acctphone" /
  </filled>
  </field>
</form>
</vxml>
```

Intradocument Transitions

- As each dialog executes, it determines the next dialog and that within the same document. Compare to the “anchors” in Html using ‘#’

  ```xml
  <var name="hi" expr="'Hello World!'"/>
  <form>
    <block>  <value expr="hi"/>
    <goto next="#say_goodbye"/>
    </block>
  </form>
  <form id="say_goodbye">
    <field name="answer" type="boolean">
      <prompt>Shall we say <value expr="application.bye"/>?
    </prompt>
    <filled>
      <if cond="answer"> <exit/> </if>
      <clear namelist="answer"/>
    </filled>
    </field>
  </form>
  </vxml>
  ```

Transitions

- Intradocument Transition
- Root to Leaf Within Application
- Leaf to Leaf Within Application
- Leaf to Root Within Application
- Root to Root

Inter-Application Transitions

- Cause the application root context to be initialized with the next application's root document.

Root document (app-root.vxml)

```xml
<?xml version="1.0"?>
<vxml version="2.0">
  <var name="bye" expr="'Ciao'"/>
  <link next="operator _xfer.vxml">
    <grammar>
      <rule id="root" scope="public">operator</rule>
    </grammar>
    <link>
      </link>
  </link>
</vxml>
```

Leaf document (leaf.vxml)

```xml
<?xml version="1.0"?>
<vxml version="2.0" application="app-root.vxml">
  <form id="say_goodbye">
    <field name="answer" type="boolean">
      <prompt>Shall we say <value expr="application.bye"/>?
    </prompt>
    <filled>
      <if cond="answer"> <exit/> </if>
      <clear namelist="answer"/>
    </filled>
    </field>
  </form>
</vxml>
```

Root to Leaf Within Application

- Within same application
- Occurs in root doc and the target doc's application attribute's value resolves to the same absolute URI as the name of the current application.
- The application root document and its context are preserved.

Ex.

Doc 1 specifies a transition to URI B, yields doc 2. Doc 2's application attribute equals URI A. The root is doc 1 with its context preserved.
Leaf to Leaf Within Application

- Within same application
- Occurs in a leaf doc and the target doc’s application attribute’s value resolves to the same absolute URI as the name of the current application.
- The application root doc and its context are preserved.

Ex. Doc 2 specifies a transition to URI C, which yields another leaf doc. Its application attribute also equals URI A. Root is doc 1 with context preserved.

Leaf to Root Within Application

- Within same application
- Occurs in leaf doc and the target doc’s absolute URI is the same as the name of the current application.
- The current application root document is preserved when the transition is caused by <choice>, <goto>, or <link>
- The root context is initialized when a <submit> causes the leaf to root transition.

Ex. Doc 3 specifies a transition to URI A using a <choice>, <goto>, or <link>. Doc 1 is used with its root context intact.

Root to Root

- Occurs when the current doc is a root doc and the target doc is a root doc, i.e. it does not have an application attribute.
- The root context is initialized with the new application root doc, even if the doc are the same or have the same application name.

Ex. Doc 1 specifies a transition to its own URI A. The resulting doc 4 does not have an application attribute, so it is considered a root doc and the root context is initialized.

Inter-Application Transitions

- Two conditions always holds in interpretation:
  - The application root doc is loaded and the user is executing in it.
  - The application root doc and one other doc in the application, known as an application leaf document, are both loaded and the user is executing in the leaf doc.

Ex. Doc 4 specifies a transition to URI D yields doc 5 which application attribute is URI E and URI E produces the root doc 6. The root context is initialized from the content of doc 6. Also, Doc 3 specifies a transition to URI A. The cache check returns doc 4, so the root context is initialized. Even though this application and it’s root doc were used earlier in the session.

Form Interpretation Algorithm (FIA)

- Form initialization.
- Prompting and counting.
- Grammar activation and deactivation.
- Entering the form with an matched utterance while the user was visiting a different one.
- Leaving the form because the user matched another form, menu, or link ’s document-scoped grammar.
- Processing multiple field fills from one utterance
- Selecting the next form item to visit, and then processing it.
- Handle any events throw.

Grammar (1)

- XML Grammar Format.
  - `<grammar type="application/grammar+xml" src="http://www.grammar.example.com/date.grxml"/>
- Augmented BNF grammar format.
  - `<grammar type="application/grammar" src="http://www.grammar.example.com/date.grm"/>
- Java Speech Grammar Format.
  - `<grammar type="application/x-jsgf" src="http://www.grammar.example.com/date.gram"/>
Grammar (2)

- Inline Grammars (ABNF)
  - Ex. `<grammar mode="voice" type="application/grammar"> red | green | blue </grammar>`
- External Grammars (XML, JSGF, ABNF)
  - Ex. `<grammar type="application/grammar" src="http://www.grammar.example.com/date.grm"/>

Event

- `error`
- `cancel`
- `exit`
- `help`
- `noinput`
- `nomatch`
- `telephone.disconnect`

Event Default Catch

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Audio Provided</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>cancel</td>
<td>no</td>
<td>Don’t reprompt</td>
</tr>
<tr>
<td>error</td>
<td>yes</td>
<td>exit interpreter</td>
</tr>
<tr>
<td>exit</td>
<td>no</td>
<td>exit interpreter</td>
</tr>
<tr>
<td>help</td>
<td>yes</td>
<td>reprompt</td>
</tr>
<tr>
<td>noinput</td>
<td>yes</td>
<td>reprompt</td>
</tr>
<tr>
<td>nomatch</td>
<td>yes</td>
<td>reprompt</td>
</tr>
<tr>
<td>maxspeechtimeout</td>
<td>yes</td>
<td>reprompt</td>
</tr>
<tr>
<td>telephone.disconnect</td>
<td>no</td>
<td>exit interpreter</td>
</tr>
<tr>
<td>all others</td>
<td>yes</td>
<td>exit interpreter</td>
</tr>
</tbody>
</table>

Structure of a VoiceXML page(1)

```
<?xml version="1.0" ?>
<vxml version="2.0">
<form>
  <block>
    <prompt>Hello,world.</prompt>
  </block>
  <disconnect/>
</form>
```

Structure of a VoiceXML page(2)

Dialog Feature (1)

- Menus
  - `<Menu>`
    - `id, scope, dtmf(Bool), accept(exact, approximate)`
  - `<choice>`
    - `next, event, expr, dtmf, accept, etc`
  - `<enumerate>`
    - `Ex. <enumerate>`
      - `For <value expr="_prompt"/>`, `press <value expr="_dtmf"/>`
      - `</enumerate>`
Dialog Feature (2)

- Form
  - `<field>`
    - Name, type, expr, cond, modal, slot
    - Built-in Grammar/Built-in Parameter:
      boolean, date, digit, currency, number, phone, time
  - `<block>`, `<initial>`
    - Name, expr, cond

- Subdialog
  - `<subdialog>`
    - Name, expr, cond, namelist, method, etc.
  - `<submit>`
    - Name, expr, namelist, method, etc.
  - `<return>`
    - Event, namelist

Dialog Feature (2)

- Link
  - `<link>`
    - Next, expr, event, dtmf, etc.

- Prompt
  - `<audio>`, `<value>`, SSML(say-as, prosody, etc)
  - Var, Assign (name, expr)
  - Clear (namlist)
  - If, Elseif, and Else

Future Solution of VoiceXML

- True Natural Language Recognition
- Multimodal Access to Application
  - Combining VoiceXML, WML, SMS
- Telematics (Telecommunication + Information)
- VIM (Video Instant Messaging)
- Wearable Computer
- PVA (Personal Virtual Assistant)