COMP349
Spoken Language Dialogue Systems
Executing VoiceXML

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

• Application Root Document
• One-Document Application
• Multi-Document Application
• Form Items
• Form Interpretation Algorithm
• Subdialogs
Application Root Document

• A VoiceXML application consists of one or more documents.
• These documents share an application root document.
• The application root document is (and remains) loaded
  – when the caller interacts with a document in the application
  – when the caller transitions between documents in the application.
• The application root document is unloaded
  – when the caller transitions to a document that is not in the application.
Application Root Document

- While the application root document is loaded
  - its variables are available to the other (leaf) documents
  - its grammars remain active for the duration of the application.
Transition between Documents

Transitioning between documents in an application
Executing a One-Document Application

- Normally, each document runs as an isolated application.
- Documents are composed of dialogs (= forms and menus).
- Document execution begins at the first dialog by default.
- As each dialog executes, it determines the next dialog.
- When a dialog does not specify a successor dialog, document execution stops.
Executing a One-Document Application

```xml
<?xml version = "1.0" encoding="UTF-8"?>
<vxml version = "2.0">
  <var name = "hello" expr = "'Hello World!'"/>
  <form>
    <block>
      <value expr = "hello"/>
      <goto next = "#say_goodbye"/>
    </block>
  </form>
  <form id = "say_goodbye">
    <block>
      Goodbye!
    </block>
  </form>
</vxml>
```
Executing a Multi-Document Application

• If you want a multi-document application, you select
  – one document to be the application root document, and
  – the rest to be application leaf documents.
• Each leaf document names the root document in its <vxml> element using the "application" attribute.
Executing a Multi-Document Application

• During interpretation one of the following conditions always hold:
  • Condition 1:
    The application root document is loaded and the caller is executing in it: there is no leaf document.
  • Condition 2:
    The application root document and a single leaf document are both loaded and the caller is executing in the leaf document.
Executing a Multi-Document Application

• Application root document (app-root.vxml)

```xml
<?xml version = "1.0" encoding="UTF-8"?>
<vxml version = "2.0">
  ...
</vxml>
```

• Leaf document (leaf.vxml)

```xml
<?xml version = "1.0" encoding="UTF-8"?>
<vxml version = "2.0" application = "app-root.vxml">
  ...
</vxml>
```
Example Dialog

Computer: Would you like to say Ciao?
Caller: Si. <"yes" or "no" is expected here>
Computer: I did not understand what you said.
Caller: Ciao. <"yes" or "no" is expected here>
Computer: Shall we say Ciao?
Caller: Operator.
Computer: <goes to operator.vxml, which transfers the call>
<?xml version = "1.0" encoding="UTF-8"?>
<vxml version = "2.0" application = "app-root.vxml">
  <form id = "say_goodbye">
    <field name = "answer">
      <grammar type = "application/srgs+xml"
        src = "/grammars/boolean.grxml"/>
      <prompt count = "1">
        Would you like to say <value expr = "application.bye"/>?
      </prompt>
      <prompt count = "2">
        Shall we say <value expr = "application.bye"/>?
      </prompt>
  </field>
</form>

Leaf Document (leaf.vxml)

```xml
<filled>
  <if cond = "answer">
    <exit/>
  </if>
  <clear namelist = "answer"/>
</filled>
</field>
</form>
</vxml>
```
<?xml version = "1.0" encoding="UTF-8"?>
<vxml version = "2.0">
  <var name = "bye" expr = "'Ciao'"/>
  <link next = "operator.vxml">
    <grammar type = "application/srgs+xml"
      root = "root"
      version = "1.0">
      <rule id = "root" scope = "public">
        operator
      </rule>
    </grammar>
  </link>
</vxml>
Comments

• In our example:
  1. The "leaf.vxml" document is loaded first.
  2. The "app-root.vxml" document is loaded.
  3. The variable "bye" is created in "app-root.vxml".
  4. The link "operator.vxml" is defined in "app-root.vxml".
  5. The dialog starts in the "say_goodbye" form of "leaf.vxml".
Benefits to Multi-Document Applications

- Root variables `<var>` are available for use by the leaf documents.
- Root document `<property>` elements can be used to specify default values for properties used in the leaf documents.
- Property values affect platform behavior, for example:
  - recognition properties (confidencelevel, sensitivity, etc)
  - prompt and collect properties (bargein, timeout, etc)
  - fetching properties (fetchaudio, fetchtimeout).
Example: Property

```xml
<?xml version = "1.0" encoding="UTF-8"?>

<!DOCTYPE vxml PUBLIC "-//Nuance/DTD VoiceXML 2.0//EN"
 "http://voicexml.nuance.com/dtd/nuancevoicexml-2-0.dtd">

<vxml version="2.0">
   <meta name = "Generator" content = "V-Builder 2.0.0"/>
   <property name = "speedvsaccuracy" value = "1.0"/>
   <property name = "maxspeechtimeout" value = "10s"/>
   <property name = "confidencelevel" value = ".50"/>
   <property name = "timeout" value = "8s"/>
   ...
```
Benefits to Multi-Document Applications

- ECMAScript code can be defined in root document element `<script>` and used in the leaf documents.
- Root document `<catch>` elements define default event handling for the leaf documents.
- If a root document has a document-level link `<link>`, its grammars are active when the user is in a leaf document.
Example: Script

```xml
<?xml version = "1.0" encoding="UTF-8"?>
<vxml version = "2.0">
  <form>
    <var name = "hours"/>
    <var name = "minutes"/>
    <var name = "seconds"/>
  </form>
  <block>
    <script>
      var d   = new Date();
      hours   = d.getHours();
      minutes = d.getMinutes();
      seconds = d.getSeconds();
    </script>
  </block>
</vxml>
```
Example: Script

<field name = "hear_another" type = "boolean">
    <prompt>
        The time is <value expr = "hours"/> hours,
        <value expr = "minutes"/> minutes, and
        <value expr = "seconds"/> seconds.
    </prompt>
    <prompt>
        Do you want to hear another time?
    </prompt>
</field>
Example: Script

```
<filled>
  <if cond = "hear_another">
    <clear/>
  </if>
</filled>
</field>
</form>
</vxml>
```
Form Items

• Form items are visited by a form interpretation algorithm (FIA).
• There are two types of form items:
  – input items (e.g. <field>)
  – control items (e.g. <block>)
• Input items direct the FIA to gather a result for a specific element.
• Control items tell the FIA to execute code or to initialize a specific behaviour.
Input Items

- An **input item** specifies a form item variable (e.g. name = "answer ").
- Input items consists of:
  - `<filed>` declares an input field in a form
  - `<record>` records an audio sample
  - `<transfer>` transfers a caller to another telephone number
  - `<object>` interacts with a custom extension
  - `<subdialog>` invokes another dialog as a subdialog
Control Items

- There are two types of control items:
  - `<block>` executes a sequence of procedural statements
  - `<initial>` controls the initial interaction in mixed initiative.
- The `<block>` control item has an implicit form item variable:
  - it is set to true just before the block is interpreted.
- The `<initial>` control item has an explicit form item variable:
  - it is set to true, if at least one input item variable is filled.
Form Interpretation Algorithm

- Forms are interpreted by an implicit form interpretation algorithm.
- The form interpretation algorithm has a main loop that
  - selects a form item
  - and then visits it.
- The selected form item is
  - the first in document order
  - whose guard condition is not satisfied.
Guard Condition

• Each form item has a guard condition.
• The guard condition governs whether or not a form item can be selected by the form interpretation algorithm.
• The default guard condition just tests to see
  – if the form item variable has a value
  – if it does, then the form item will not be visited.
Form Interpretation Algorithm

• Interpreting a form item generally involves:
  – selecting and playing one or more prompts
  – collecting a user input or throwing of some event
  – interpreting any \(<\text{filled}\>)\ actions.
Form Interpretation Algorithm

• The form interpretation algorithm ends
  – when it interprets a transfer of control statement
    (e.g. <goto> or <submit>)
  – when no form item remains to select
    (implied <exit>).
Customizing the FIA

- The FIA can be customized in several ways:
  - assigning a value to a form item variable (<assign>)
  - setting a form item variable to undefined (<clear>)
  - specifying the next form item to visit (<goto>).
Examples

- `<assign name = "flavor" expr = "'chocolate'"/>
- `<clear namelist = "city state zip"/>
- Transition to another form item:
  - `<goto nextitem = "confirm"/>
- Transition to another dialog in the same document:
  - `<goto next = "#another_dialog"/>
- Transition to another document:
Subdialog Element

• A `<subdialog>` element can be used for decomposing complex sequences of dialogs to better structure them, or to create reusable components.
Customer Service Application (app.vxml)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2001/vxml
    http://www.w3.org/TR/voicexml20/vxml.xsd"
    version="2.0">
    <form id="billing_adjustment">
        <var name="account_number"/>
        <var name="home_phone"/>
        <subdialog name="accountinfo" src="acct_info.vxml#basic">
            <filled>
                <!-- Note the variable defined by "accountinfo" is
                returned as an ECMAScript object and it contains two
                properties defined by the variables specified in the
                "return" element of the subdialog. -->

                <assign name="account_number" expr="accountinfo.acctnum"/>
                <assign name="home_phone" expr="accountinfo.acctphone"/>
            </filled>
        </subdialog>
    </form>
</vxml>
```
<field name="adjustment_amount">
  <grammar type="application/srgs+xml" src="/grammars/currency.grxml"/>
  <prompt>
    What is the value of your account adjustment?
  </prompt>
  <filled>
    <submit next="/cgi-bin/updateaccount"/>
  </filled>
</field>
</form>
</vxml>
Account Information Subdialog (acct_info.vxml)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<voxml xmlns="http://www.w3.org/2001/vxml"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.w3.org/2001/vxml
                        http://www.w3.org/TR/voicexml20/vxml.xsd"
       version="2.0">
  <form id="basic">
    <field name="acctnum">
      <grammar type="application/srgs+xml" src="/grammars/digits.grxml"/>
      <prompt> What is your account number? </prompt>
    </field>
    <field name="acctphone">
      <grammar type="application/srgs+xml" src="/grammars/phone_numbers.grxml"/>
      <prompt> What is your home telephone number? </prompt>
      <filled>
        <!-- The values obtained by the two fields are supplied to the calling dialog by the "return" element. -->
        <return namelist="acctnum acctphone"/>
      </filled>
    </field>
  </form>
</voxml>
```
Executing Subdialogs

• Returning from the last subdialog document:
Executing Subdialogs

• Returning from the last subdialog document:
Take-Home Messages

- Each VoiceXML application has an application root document.
- Each leaf document names the application root document.
- Forms are interpreted by a form interpretation algorithm.
- Guard conditions define whether a form item can be selected or not.
- The form interpretation algorithm can be customized.
- Subdialogs can be used to structure an application.