Beyond the Form Interpretation Algorithm

Towards flexible dialog management for conversational voice and multi-modal applications

Rahul P. Akolkar
IBM Watson Research Center
P.O. Box 704
Yorktown Heights, N.Y. 10598
Tel.: 1 914 784 6174
akolkar@us.ibm.com

ABSTRACT

The dialog or voice user interface choices available to application developers in the present version of VoiceXML are largely limited to the capabilities of the Form Interpretation Algorithm (FIA) combined with dynamic server-side generation of VoiceXML. This position paper discusses several improvements aimed at providing flexible dialog management in VoiceXML. The notion of recursive transition controllers at varying dialog granularity enables the presentation of flexible interfaces for conversational applications. As an example, the use of State Chart XML (SCXML) to declaratively describe such transition controllers is illustrated. These improvements are demonstrated to be backwards compatible with the latest version of VoiceXML (v2.1). Finally, the paper introduces the use of DOM events to manipulate the voice user interface, enabling compound namespace usecases where voice may be used with other modalities in the same document. Many of these improvements are under consideration for the next version of VoiceXML (v3.0).

Categories and Subject Descriptors

I.7.2 [**Document and Text Processing**]: Document Preparation—*Markup languages, Standards*; H.5.2 [**Information Interfaces and Presentation (e.g., HCI)**]: User Interfaces—*Interaction styles (e.g., commands, menus, forms, direct manipulation)*; H.5.3 [**Information Interfaces and Presentation (e.g., HCI)**]: Group and Organization Interfaces—*Web-based Interaction*

General Terms

Languages, Standardization

Keywords

VoiceXML, SCXML, web application, speech application, voice user interfaces

World Wide Web Consortium - First Workshop on Conversational Applications
Use Cases and Requirements for New Models of Human Language to Support Mobile Conversational Systems
18-19 June 2010
Somerset, NJ, US

1. INTRODUCTION

The creation of voice user interfaces for conversational applications provides a number of challenges, such as the development of sophisticated grammars and semantics, language models for natural language processing and dynamic generation of the voice user interface. Beyond server-side processing for generating dynamic VoiceXML, the present version of VoiceXML provides limited capabilities to describe dialogs for conversational applications. The Form Interpretaion Algorithm (FIA), specified as Appendix C of the VoiceXML 2.0 specification [5] provides the default sequence of processing fields within a form, with some level of control via guard conditions and flow control statements.

In order to provide dialog flexibility beyond what the FIA offers, a recursive Model-View-Controller (MVC) pattern may be employed. The basis of the resulting dialog management pattern is the existence of transition controllers at varying levels of granularity. In particular, such transition controllers may exist at all the levels of granularity or scope, viz. application, session, document and form. These transition controllers may be authored using a number of languages or notations. The subsequent section illustrates this using State Chart XML (abbreviated as SCXML) [3], though transition controllers may use other notations as well.

2. FLEXIBLE DIALOG MANAGEMENT

Being a general purpose controller notation, SCXML is suitable for describing the controllers for managing the dialog in VoiceXML 3.0 applications. Such SCXML controllers may be placed at application, session, document and form levels. This is illustrated in this section using examples of resulting compound documents containing VoiceXML 3.0 (abbreviated as V3) and SCXML namespaced elements.

For the examples in this section, we use a travel itinerary modification scenario. The first order of business is to retrieve an itinerary, which may be identified using either a record locator or traveler's lastname and other information. The use of an SCXML-based form level transition controller is illustrated using two flavors, viz. a system-driven and a user-driven approach. Both use similar set of fields in the form but different dialog management styles. While in this simple example, the voice user interface (VUI) may appear similar, but has distinct system vs. user driven flavors.

The communication between the controller and the V3 fields is discussed in detail in Section 5.

2.1 System-driven Dialog

```
<v3:form>
  <scxml:scxml initial="choose">

    <scxml:state id="choose">
      <scxml:onentry>
        <scxml:send target="#choice"
          type="DOM" event="DOMActivate"/>
      </scxml:onentry>
      <scxml:transition event="filled.choice"
        cond="choice" target="record"/>
      <scxml:transition event="filled.choice"
        cond="!choice" target="name"/>
    </scxml:state>

    <scxml:state id="record">
      <scxml:onentry>
        <scxml:send target="#locator"
          type="DOM" event="DOMActivate"/>
      </scxml:onentry>
<!-- Retrieve record, transition to app menu -->
    </scxml:state>

    <scxml:state id="name">
      <scxml:onentry>
        <scxml:send target="#lastname"
          type="DOM" event="DOMActivate"/>
      </scxml:onentry>
<!-- Collect other information needed to get
  record, then retrieve record, go to menu -->
    </scxml:state>

<!-- Remaining dialog control logic omitted -->
  </scxml:scxml>

  <v3:field id="choice">
    <v3:grammar src="boolean.grxml"
      type="application/srgs+xml"/>
    <v3:prompt>
      Welcome. Do you have the record locator
      for your itinerary?
    <v3:prompt>
<!-- Implicit filled.choice event on filled -->
  </v3:field>

  <v3:field id="locator">
    <v3:grammar src="locator.grxml"
      type="application/srgs+xml"/>
<v3:prompt>What is the record locator?<v3:prompt>
  </v3:field>

  <v3:field id="lastname">
    <v3:grammar src="lastname.grxml"
      type="application/srgs+xml"/>
<v3:prompt>Please say your last name.<v3:prompt>
  </v3:field>

<!-- Other form items, such as the subsequent
  application menu omitted -->
</v3:form>
```

2.2 User-driven Dialog

```
<v3:form>
  <scxml:scxml initial="choose">

    <scxml:state id="choose">
      <scxml:onentry>
        <scxml:send target="#choice"
          type="DOM" event="DOMActivate"/>
      </scxml:onentry>
      <scxml:transition event="filled.choice"
            cond="choice == 'locator'"
            target="record"/>
      <scxml:transition event="filled.choice"
            cond="choice == 'lastname'"
            target="name"/>
    </scxml:state>

    <scxml:state id="record">
<!-- Same contents as system-driven dialog -->
    </scxml:state>

    <scxml:state id="name">
<!-- Same contents as system-driven dialog -->
    </scxml:state>

<!-- Remaining dialog control logic omitted -->
  </scxml:scxml>

  <v3:field id="choice">
    <v3:grammar src="choice.grxml"
        type="application/srgs+xml"/>
    <v3:prompt>
      Welcome. How would you like to
      look up your itinerary?
    <v3:prompt>
  </v3:field>

  <v3:field id="locator">
<!-- Same contents as system-driven dialog -->
  </v3:field>

  <v3:field id="lastname">
<!-- Same contents as system-driven dialog -->
  </v3:field>

<!-- Other form items, such as the subsequent
  application menu omitted -->
</v3:form>
```

The transition controller captures the complete model of the correspondng dialog rather than a more distributed definition using guards and flow control statements on presentation elements, and thereby, as illustrated in the examples above, allows for making localized changes to the controller logic to manipulate the voice user interface.

In the user driven dialog above, a more sophisticated grammar (`choice.grxml` for the `choice` field rather than the simpler `boolean.grxml` used in the system driven dialog) is used along with the corresponding new semantic interpretation in the guard conditions of the transitions which decide the next fields in the form, to illustrate a different dialog using essentially the same set of fields within the form.

To ensure backwards compatibility with VoiceXML 2.0 and 2.1 applications, a graceful degradation approach is suggested that causes behavior to fallback to the FIA in the absence of an explicit transition controller.

3. GRACEFUL DEGRADATION

The absence of an explicit transition controller (such as the `<scxml:scxml>` child element) a `<v3:form>` should revert behavior to be identical to a `<v2:form>` element where the V2 Form Interpretation Algorithm would be in charge. In the presence of a `<scxml:scxml>` child, the FIA should be suppressed and the more expressive SCXML controller used, which then allows application developers to design the form VUI in a flexible manner. In other words, the following `<v3:form>` below:

```
<v3:form>

  <!-- No transition controller child -->

  <!-- Various form items etc. -->

</v3:form>
```

should behave as would:

```
<v2:form>

  <!-- Various form items etc. -->

</v2:form>
```

4. BASIS FOR RECURSIVE MVC

The examples in Section 2 illustrated a form-level SCXML controller. Another transition controller may then be used as a document level controller, where it would be managing the interaction across `<v3:form>`s, rather than `<v3:field>`s. This is illustrated below.

```
<v3:vxml>

  <scxml:scxml ...>
    <!-- document level controller managing
   interaction across form1, form2 and form3 -->
  </scxml:scxml>

  <v3:form id="form1">
    <!-- form1 content, might also have a form
      level SCXML controller -->
  </v3:form>

  <v3:form id="form2">
    <!-- form2 content, might also have a form
      level SCXML controller -->
  </v3:form>

  <v3:form id="form3">
    <!-- form3 content, might also have a form
      level SCXML controller -->
  </v3:form>

</v3:vxml>
```

Further, similar transition controllers may be authored at the application and session levels to provide flexible dialog management across many levels of user interface granularity.

5. DOM EVENTS AS THE GLUE

The glue that enables the interaction between the transition controller namespace (such as SCXML) and the VoiceXML 3.0 presentation elements is DOM events [6]. The functionality provided by the voice user interface presentation elements such as `<v3:field>` is exposed via DOM interfaces, and the elemental function may be invoked by dispatching a `DOMActivate` event with the element as the target.

As seen in the examples in Section 2, this results in the following pattern to invoke the functionality of a VoiceXML field when a particular state in the transition controller state machine is entered.

```
<scxml:state id="state-id">

  <scxml:onentry>
    <scxml:send target="#field-id"
      type="DOM" event="DOMActivate"/>
  </scxml:onentry>

  <!-- Rest of state content, such as
    transitions -->

</scxml:state>
```

The DOM I/O processor defined by SCXML processes the `<scxml:send>` element and in turn dispatches the `DOMActivate` event on the V3 field with ID `field-id`.

Furthermore, many of the voice user interface presentation elements communicate their execution status and progress by firing upon themselves DOM events that the rest of the document can listen to. Transition controllers register themselves as listeners to the appropriate voice user interface elements, for example, a form level transition controller registers itself as a listener for the `filled` events on all of the form items.

As seen in the examples in Section 2, this results in the following pattern where the transition controller makes an intelligent dialog decision on moving to either of the next two candidate fields based on a recognition result from a previous field.

```
<scxml:state id="current-state">

  <!-- Rest of state content, such as
    onentry -->

  <!-- Transition controller is a listener on
    all filled events from child form items of
    the associated form -->
  <scxml:transition  event="filled.field1"
        cond="..."  target="candidate-state-1"/>
  <scxml:transition  event="filled.field1"
        cond="..."  target="candidate-state-2"/>

</scxml:state>
```

6. COMPOUND DOCUMENTS, MULTIPLE MODALITIES

The DOM interfaces for the various voice user interface presentation elements also enable the development of conversational applications where more than the speech modality is in use. The following example demonstrates a rewrite of a simple XHTML + Voice Profile [2] document using the features mentioned in the preceding sections.

```
<html
 xmlns="http://www.w3.org/2002/xhtml"
 xmlns:ev="http://www.w3.org/2001/xml-events"
 xmlns:v3="http://www.w3.org/TR/voicexml30">
  <head/>

  <form action="myform.do" method="post">
    <input id="myInput" ev:event="onfocus"
           ev:handler="#startvoice"/>
    <!-- Rest of the XHTML form -->
  </form>

  <v3:field id="myField" ev:event="filled"
            ev:handler="#fillinput">
    <v3:grammar src="grammar.grxml"
      type="application/srgs+xml"/>
    <v3:prompt>
      What is the value of the input?
    </v3:prompt>
  </v3:field>

  <script xml:id="startvoice"
     type="application/ecmascript">
  document.myField.dispatchEvent('DOMActivate');
  </script>

  <script xml:id="fillinput"
     type="application/ecmascript">
document.myInput.value = document.myField.value;
  </script>

</html>
```

Here, the dominant modality is XHTML [1] and the graphical user interface consists of a simple web form with one input field. The graphical modality drives the voice user interface consisting of one V3 field at the appropriate moment, on focus i.e. when the user clicks or tabs into the input field.

The value to be entered may then be keyed in by the user or supplied as a speech response to the prompt played. In case the value is received from the speech modality, it is synchronized with the XHTML input before the form is submitted. This synchronization happens by listening to the `filled` event on the V3 field.

The XML Events [4] notation is used to define the necessary event listeners in this example, by attaching the XML Events syntactic attributes directly to the corresponding observer elements, which in this case are the XHTML form input for the `onfocus` and the V3 field for the `filled` events. The submission of the user supplied form data happens via the graphical modality with the XHTML form submission.

7. CONCLUSION

Increasing complexity in managing the dialog in conversational applications may be entrusted to the voice browser or client by affording more flexibility with the introduction of transition controllers at multiple levels of dialog granularity. Such introduction reduces the extent of server-side dependence on dynamic voice user interface generation and is entirely backwards compatible. These transition controllers allow the localized definition of the corresponding dialog and may be expressed using different notations including SCXML. The resulting uniform, loosely-coupled and event-driven programming model not only eases the development of conversational applications but also enables both well-defined and opportunistic compositions of speech with other modalities.

DOM events are used as the binding agent using which VoiceXML presentation elements interact with the transition controllers as well as other modalities in compound namespaced documents. The functionality of various voice user interface elements may be invoked by dispatching the appropriate DOM event onto them, while these elements communicate interesting aspects of their execution status and progress by firing upon themselves DOM events for which listeners may be registered in the rest of the document.

This paper recommends the adoption of the discussed features in VoiceXML 3.0.

8. REFERENCES

[1] Altheim, M. et al (eds.) *XHTML 1.1* W3C Recommendation (2001). http://www.w3.org/TR/2001/REC-xhtml11-20010531/

[2] Axelsson, J. et al (eds.) *XHTML+Voice Profile 1.0* W3C Note (2001). http://www.w3.org/TR/2001/NOTE-xhtml+voice-20011221/

[3] Barnett, J. et al (eds.) *State Chart XML* W3C Working Draft (2010). http://www.w3.org/TR/2010/WD-scxml-20100513/

[4] McCarron, S. et al (eds.) *XML Events* W3C Recommendation (2003). http://www.w3.org/TR/2003/REC-xml-events-20031014/

[5] McGlashan, S. et al (eds.) *VoiceXML 2.0.* W3C Recommendation (2004). http://www.w3.org/TR/2004/REC-voicexml20-20040316/

[6] Pixley, T. (ed.) *DOM Level 2 events* W3C Recommendation (2000). http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/