The W3C Multimodal Architecture, Part 2: The XML specification stack

Multimodal authoring with SCXML, XHTML, REX, and more

Gerald McCobb

May 31, 2007

Gerald McCobb continues his introduction to the forthcoming W3C Multimodal Architecture with a survey of the many XML languages that you can use to author multimodal applications. He then shows how several specifications -- SCXML, XHTML, REX, and XML Events -- could work together in a complete multimodal application.

In the first article in this series introducing the W3C Multimodal Architecture, I presented an overview of the architecture, which is still under development by the W3C Multimodal Interaction Working Group. I also discussed some of the limitations of the current proposed architecture, particularly with regard to performance, and explained how these factors could impact developers using the architecture to author multimodal applications.

In that article I touched on the importance of a standard for combining the XML representing various modalities (such as VoiceXML and XHTML) into a single document when all modalities are rendered on the client. In this article I will show how multimodal interaction could be supported in cases where the XML documents are distributed (that is, some documents are rendered on the client and some are rendered on one or more remote servers). Currently, none of the W3C XML languages directly support multimodal authoring, though in the near future the Multimodal Interaction Working Group may specify how to add that support.

The objective of this article is to introduce some of the XML languages in the W3C specification stack and show how they could be combined to create a complete multimodal application. I briefly introduce each language and explain the role (or roles) it could play in a multimodal application. I then offer a speculative example that shows how several of the specifications could be combined.

Related initiatives and activities

By its very nature, distributed multimodal authoring cannot be captured by one standard W3C recommendation. The W3C specifications for multimodal authoring range across numerous W3C initiatives and activities, and the review process also spans multiple working groups. Initiatives and activities concerned with the Multimodal Architecture and Interfaces...
The specification stack

As I explained in Part 1, the proposed Multimodal Architecture consists of a runtime framework and one or more distributed modality components that communicate with the runtime framework through a life-cycle events API. In the listing that follows I've categorized the XML specifications that could contribute to a Multimodal Architecture implementation, first in terms of the architecture's main components, and then in terms of components that support the life-cycle API. In some cases I've listed a specification more than once because it could play more than one role in the architecture. For example, it would be possible to use VoiceXML as both a dialog manager and a voice modality component. Because the selection is speculative I may have omitted relevant specifications unintentionally; the actual value of these specifications will not be tested until they are put to use authoring Multimodal applications. See Resources to learn more about the specifications and working groups listed.

Interaction management

The interaction manager sends and receives all messages between the Multimodal Architecture's runtime framework and modality components, and queries and updates the data component as needed. Any of the following XML languages could be used to maintain dialog flow, current state, and public data, and essentially contain the multimodal application.

SCXML (State Chart XML)

State Machine Notation for Control Abstraction is a working draft on track to become a W3C recommendation. SCXML is an XML implementation of David Harel's Statecharts. It can be used as an application's dialog or interaction manager, and has been chosen as the language to handle dialog management in the next version of VoiceXML, known as V3. SCXML is owned by the W3C Voice Browser Working Group.

CCXML

Voice Browser Call Control 1.0 is a last call working draft on track to become a W3C recommendation. Its main responsibility is to manage the call control for one or more VoiceXML dialogs. For example, CCXML manages telephony operations such as connecting, bridging, and conferencing. It has an event loop and conditional elements to allow it to manage multimodal dialog systems. CCXML is owned by the Voice Browser Working Group.

SMIL

Synchronized Multimedia Integration Language 2.1 became a W3C recommendation in December 2005. SMIL is an XML language for authoring real-time choreography of audio, video, and text presentations. A multimodal application could use SMIL to integrate synthesized speech (also known as text-to-speech) with streaming audio and a timed sequence of images. SMIL is owned by the Synchronized Multimedia Working Group.

VoiceXML

Voice Extensible Markup Language 2.0 became a W3C recommendation in March 2004. VoiceXML is an XML language for authoring interactive speech dialogs. As a dialog language, it could be used as the interaction manager in a W3C Multimodal Architecture. The next
version of VoiceXML, V3, will explicitly support integration into Multimodal Applications. VoiceXML is owned by the Voice Browser Working Group.

**Delivery context interface**

The Multimodal Architecture's delivery context stores static and dynamic device properties, environmental conditions, and user preferences so that they can be queried and dynamically updated.

**DCI**

*Delivery Context: Interfaces* became a W3C candidate recommendation in October 2006. DCI defines an XML interface to dynamic and static device properties for Web applications. DCI is owned by the Ubiquitous Web Applications Working Group.

**Data storage**

The data component contains the public data model for a multimodal application.

**XForms**

*XForms 1.0 (second edition)* became a W3C recommendation in March 2006. Work has begun to develop an XML data model with XForms binding rules as part of an application backplane generally available to XML markup languages. As a result of a W3C working group coordination activity, the Forms Working Group published a *Rich Web Application Backplane* working group note in July 2006.

**DOM**

*Document Object Model Level 3 Core* became a W3C recommendation in April 2004. The DOM generally represents an XML document rendered by a client browser. It is also possible to implement a centralized DOM that sends remote events for XML messages between the DOM and its distributed components.

**Modality components**

Modality components interact directly with the user when performing tasks such as querying, navigating, and updating the application. Modality components therefore render the XML languages that represent the user interface, such as XHTML and VoiceXML.

**XHTML**

XHTML represents a visual modality, or GUI. The latest version, *XHTML 2.0*, was published by the XHTML 2 Working Group as a working draft in July 2006. XHTML comes in several other flavors, including XHTML Basic 1.1 and the XHTML Mobile Profile, both of which are intended for small devices.

**SVG**

*Scalable Vector Graphics 1.1* became a W3C recommendation in January 2003. SVG is an XML language for authoring scalable 2D vector graphics that are rendered as images (such as maps) by an SVG visual modality component. Like XHTML, SVG has a DOM.

**Timed Text**

The *Timed Text Authoring Format 1.0 -- Distribution Format Exchange Profile (DFXP)* became a candidate recommendation in November 2006. Timed Text represents a visual modality that combines timed presentation of text with audio and/or video for captioning or subtitling. It is owned by the Timed Text Working Group.
VoiceXML

VoiceXML represents a speech modality. Besides language elements for handling flow control and executable content, it combines two XML languages, *Speech Recognition Grammar Specification (SRGS) 1.0* and *Speech Synthesis Markup Language (SSML) 1.0*, for authoring speech input recognition and speech synthesis output, respectively. VoiceXML adds a third language, *Semantic Interpretation for Speech Recognition (SISR) 1.0*, for authoring an interpretation of a recognized speech input. One possible output of SISR is an EMMA document (see below).

SMIL

SMIL could be used to represent a visual modality rendering timed combinations of audio, video, and text.

Eventing

Scripting languages can perform multiple roles on either the client side or server side of a Multimodal Architecture. For example, a scripting language could act as an interaction manager, data storage component, and device configuration repository and interface. On the client, scripting languages such as EcmaScript, JavaScript (a superset of EcmaScript), and WMLScript are capable of managing complex dialogs. Client-side scripting can manage interaction locally but the W3C Multimodal Architecture specifies centralized management of a set of distributed "loosely coupled" modality components. Client-side scripting is necessary if a generic `<data>` event sent from the interaction manager is passed as-is to the authoring layer. In this case, scripting must be used to inspect the `<data>` event and process its contents.

Scripting also has an important role in asynchronous data transfers between the interaction manager and the client. Using Ajax to communicate with an interaction manager is outside the scope of this discussion, although a standard DOM interface to the `XMLHttpRequest` object is under development. In Part 3 I will explain how to use JavaScript as an interface between SOAP calls from a Mozilla browser and a Web service implementation of the Multimodal Architecture.

The Multimodal Architecture specifies a set of *life-cycle events*. The following XML languages and interfaces specify event representation, listening and handling, and transport. These languages don't directly support the life-cycle events but they could be used as content of the `<data>` event, or through an adapter interface to the `<data>` event.

REX

*Remote Events for XML (REX) 1.0* was published as a working draft in October 2006. It specifies how to author messages to remotely update DOM node content and attribute values. It does not currently specify how to author a message containing a DOM event type. REX is owned by the Web API Working Group.

XML Events

*XML Events 2* was published by the HTML Working Group as a working draft in February 2007. XML Events allows authors to associate event handlers to listeners and attach to DOM Level 2 nodes. The XML Events module consists of a `<listener>` element and a set of attributes, which can be added to XML language elements that support the DOM Level 2 event model (and in the near future Dom Level 3).
DOM Level 2 and Level 3 Events

*Document Object Model Level 2 Events* became a W3C recommendation in November 2000. *Document Object Model (DOM) Level 3 Events* was published by the Web API Working Group as a working draft in April 2006. Most XHTML browsers today, including Mozilla Firefox, Internet Explorer, and Opera, currently support DOM Level 2 events. VoiceXML Version 3 should support DOM Level 3 events.

**Intent-based Events**

*Intent-Based Events* was published as a working draft in November 2003. It is not currently on track to become a W3C recommendation. This specification is relevant to the W3C Multimodal Architecture because it specifies a set of events that represent the "underlying intent" of user interaction. These intent-based events are therefore independent of modality. Interfaces such as XHTML, Voice, Pen, Haptic, and so on, could all emit and process the same set of events.

**CSS Events**

*CSS Events Module 1.0* was published as a working draft in January 2005. It is not currently on track to become a W3C recommendation. This specification implements XML Events event handling as a CSS event property. This allows an author to put the event handling in a separate style sheet and add to selective legacy application documents.

**Sandboxing**

Sandboxing is an important security concern for distributed applications, which must be allowed access to remote resources only when the author explicitly allows it.

**Enabling read access for Web resources**

*Enabling Read Access for Web Resources* was published by the Web Application Formats Working Group as a working draft in February 2007. It is on track to become a W3C recommendation. This specification defines a means by which a resource can specify the hosts or domains that have read access. As a result Ajax and SOAP calls from a Web browser may access resources residing outside of the Web application's domain. A multimodal interaction manager is one example of a resource that may reside on a separate domain.

**Data interchange**

The following XML languages can be used as a standard format for data interchange between modality components and an interaction manager.

**EMMA**

*Extensible MultiModal Annotation markup language* was published by the Multimodal Interaction Working Group as a last call working draft in April 2007. EMMA is an XML data format for representing an interpretation of a user input from a modality along with metadata annotations of the input (for example, timestamp, source, input tokens, and so on) captured by the input recognizer. An EMMA input result could be sent from a modality component to an EMMA processor; an EMMA processor on a server could also be a multimodal interaction manager.

**InkML**

*Ink Markup Language* was published by the Multimodal Interaction Working Group as a last call working draft in October 2006. InkML is an XML data format for representing the data.
entered by a digital pen or stylus. InkML could be sent from a device that accepts digital pen input to a handwriting or gesture recognizer (in turn the output of a handwriting recognizer could be an EMMA result document).

The multimodal coffee maker

Having been reminded of the breadth and diversity of the W3C XML specifications, you may be wondering how any subset of them could be combined to form a multimodal application. In this section I offer a speculative application that may answer that question. Keep in mind that I’m not concerned with showing how a real distributed multimodal application will be authored in the near future. Instead, I want to provide an example where all parts of the application are declarative (in other words, no scripting). I also want to demonstrate that it may not be necessary to modify any of the W3C XML specifications to support multimodal interaction.

The resulting application is a multimodal online coffee maker. This futuristic virtual appliance is connected to an actual coffee maker that has its own Web server, enabling a user to remotely start or stop the coffee maker. The state model for the coffee maker is very simple: If it is filled with water, and the coffee pot is in place, it is in the Coffee Making state. Otherwise it is in the Warming state. Both Coffee Making and Warming are sub-states of the On state, as shown in Figure 1.

Figure 1. The coffee maker's simple state model

SCXML

It isn’t necessary to send out state transitions within the coffee maker’s state machine. Instead, the SCXML interpreter keeps track of the application state for the interaction manager. The state diagram shown in Figure 1 is represented in SCXML as shown in Listing 1. You can also download the code for this listing and others in the article if you like.

Listing 1. An SCXML representation of the application state

```xml
<?xml version="1.0"?>
<scxml xmlns="http://www.w3.org/2005/07/scxml"
version="1.0">
  ...
</scxml>
```
In Listing 2, a separate SCXML document represents the application's interaction manager. Keeping the interaction manager SCXML separate from the application SCXML follows the principle of separation of concerns. When separately developed, the application and interaction manager SCXML documents are cleaner and easier to maintain.
Listing 2. An SCXML representation of the interaction manager

```xml
<?xml version="1.0" encoding="us-ascii" ?>
<scxml xmlns="http://www.w3.org/2005/SCXML"
   id="coffeeDialog"
   version="1.0" initialstate="generateContext">
  <datamodel>
    <data name="context"/>
  </datamodel>
  <state id="generateContext" <!-- create context id -->
    <onentry>
      <assign name="context" expr="new Context()"/>
    </onentry>
    <transition target="startup"/>
  </state>
  <state id="startup" <!-- prepare and start the modalities -->
    <parallel>
      <state id="startupVxml"
        initial>
        <transition target="prepareVxml"/>
      </initial>
      <state id="prepareVxml">
        <send event="prepare" target="/vxml/coffeemaker.vxml"
          targettype="vxml" context="context"/>
        <transition target="startVxml" event="PrepareResponse"
          cond="status=='success' && data.eventType=='vxml'"/>
        <transition target="failureVxml" event="PrepareResponse"
          cond="status=='failure' && data.eventType=='vxml'"/>
      </state>
      <state id="startVxml">
        <send event="start" target="/vxml/coffeemaker.vxml"
          targettype="vxml" context="context"/>
        <transition target="started" event="StartResponse"
          cond="status=='success' && data.eventType=='vxml'"/>
        <transition target="failureVxml" event="StartResponse"
          cond="status=='failure' && data.eventType=='vxml'"/>
      </state>
      <state id="startupXhtml"
        initial>
        <transition target="prepareXhtml"/>
      </initial>
      <state id="prepareXhtml">
        <send event="prepare" target="/html/coffeemaker.xhtml"
          targettype="xhtml" context="context"/>
        <transition target="startXhtml" event="PrepareResponse"
          cond="status=='success' && data.eventType=='xhtml'"/>
        <transition target="failureXhtml" event="PrepareResponse"
          cond="status=='failure' && data.eventType=='xhtml'"/>
      </state>
      <state id="startXhtml">
        <send event="start" target="/html/coffeemaker.xhtml"
          targettype="xhtml" context="context"/>
        <transition target="started" event="StartResponse"
          cond="status=='success' && data.eventType=='xhtml'"/>
        <transition target="failureXhtml" event="StartResponse"
          cond="status=='failure' && data.eventType=='xhtml'"/>
      </state>
    </parallel>
  </state>
</scxml>
```
In this example the interaction manager manages the Multimodal Architecture's life-cycle events and forwards the content of <data> events from either a visual or speech modality component to the application's SCXML. In turn, transitions received from the application's SCXML are forwarded to both the visual and speech components.
XHTML

The online coffee maker's XHTML is shown in Listing 3. It has two radio buttons for turning the coffee maker on or off and a text area to report the current status of the coffee maker.

Listing 3. XHTML for the application

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>My Coffee Maker</title>
    <style>
      h2 { background: #ffffff;
          color : #0000a0;
          font-weight: bold;
          font-size : 18pt;
          font-family : Arial }
      body { color : #000000;
          background: #ffffff;
          font-size : 14pt;
          margin: 30px 0px 0px 30px;
          font-family : Comic Sans MS }
      textarea { width: 400px;
                  height: 300px;
                  border: 1px solid #0000a0;
                  padding: 20px; }
    </style>
    <link rel="stylesheet" href="css/events.css" type="text/css"/>
  </head>
  <body>
    <h2>My Coffee Maker</h2>
    <form action=".">
      <input class="onoff" name="OnOff" type="radio" value="On"/> On<br/>
      <input class="onoff" name="OnOff" type="radio" value="Off"/> Off<br/>
      <p>Status Messages<br/>
      <textarea id="box" name="box"></textarea><br/>
    </form>
  </body>
</html>
```

Ideally, the application would also include a speech modality, represented by VoiceXML. I haven't included VoiceXML here because the first working draft for VoiceXML V3 has not yet been published, and VoiceXML V2 does not implicitly support sending and receiving either multimodal life-cycle events or DOM events through REX. The grammar for the speech modality is very simple, however: "turn on," "turn off," and "show me status." The speech modality would also speak the current status of the coffee maker displayed in the status area.

REX

Next, an adapter on the server intercepts the `<data>` event sent from the interaction manager to the XHTML browser and transforms it into a REX document -- REX is the `<data>` content. The REX document, shown in Listing 4, is received by a REX session started by the XHTML browser. The REX session updates the XHTML document running in the XHTML browser as specified by the incoming REX document. In this example there are two REX events. The first event sets the On radio button. The second event inserts character data into the text area.
Listing 4. Remote events for CoffeeMaker.xhtml

```xml
<?xml version="1.0"?>
<rex xmlns="http://www.w3.org/ns/rex#"
    target-document="coffeemaker.xhtml">
    <!-- Event messages for the CoffeeMaker application -->
    <event target="id('on')" attrName="checked" name="DOMAttrModified" newValue="true"/>
    <event target="id('box')" name="DOMCharacterDataModified"
        newValue="Coffee Maker is turned on...
        Coffee Maker is warming...
        Coffee pot is replaced...
        Water is Filled...
        Coffee Maker is making coffee..."/>
</rex>
```

The multimodal coffee maker application is shown running in Firefox in Figure 2. In this example Firefox is the visual modality component. The figure shows the text area updated after Firefox has received and processed the REX messages.

Figure 2. The coffee maker running in Firefox

XML Events

One way to send REX events from a modality component to the interaction manager is to use the XMLHttpRequest object, but this would require adding JavaScript to the application’s XHTML. An alternative is to add one or more XML Events listeners to XHTML elements using CSS. As shown in Listing 5, I used a CSS Events event property to add an XML Events listener to the body element and to the two radio button input elements, which both belong to the OnOff CSS class.
The XHTML document references the external style sheet, containing the event using the XHTML <link> element. This completes the application.

Listing 5. CSS events for CoffeeMaker.html (events.css)

```html
body {
  event: load url(http://example.com/scxml/mycoffee.xml#coffeeDialog);
}
input.onoff {
  event: select url(http://example.com/scxml/mycoffee.xml#coffeeDialog);
}
```

In conclusion

In the first article in this series I wrote about the challenges currently facing developers who plan to use the W3C Multimodal Architecture to author multimodal applications. In this article I've offered an example of how some of those challenges could be addressed in a distributed application.

One of the major factors currently limiting the Multimodal Architecture is its dependence on XML specifications, none of which yet actually support the architecture. In my example application I was able to get around that limitation, but not by modifying any of the languages leveraged in the application. Instead, I placed an adapter interface between the XML Events event-handling mechanism and the Multimodal Architecture's native <data> event. I used another adapter interface between the <data> event and the REX message output. Note that adapter interfaces must also be specified because they are authoring interfaces.

In the final article in this series I'll offer another speculative example in getting around the current limitations of the Multimodal Architecture. The example will be a Web services application that uses JavaScript as an interface to SOAP calls from a Web browser.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample code from this article</td>
<td>coffeemaker.zip</td>
<td>3KB</td>
</tr>
</tbody>
</table>

© Copyright IBM Corporation 2007

**Trademarks**
(www.ibm.com/developerworks/ibm/trademarks/)