Chapter 9
Developing Portable Context-Aware Multimodal Applications for Connected Devices Using the W3C Multimodal Architecture

Raj Tumuluri and Nagesh Kharidi

Abstract Cue-me™ is one of the reference implementations of the W3C's multimodal interaction (MMI) architecture and is a context-aware multimodal authoring and run-time platform that securely houses various modality components and facilitates cross-platform development of multimodal applications. It features several multimodal elements such as Face Recognition, Speech Recognition (ASR) and Synthesis (TTS), Digital annotations/gestures (Ink), Motion Sensing and EEG-headset based interactions that were developed using W3C MMI Architecture and Markup Languages. The MMI architecture described elsewhere in this volume facilitates single-authoring of multimodal applications and shields the developers from the nuances of the implementation of individual modality components or their distribution.

9.1 Introduction

Given the proliferation of mobile devices/operating systems, developing and maintaining applications is a challenge for authors of applications. Ensuring usability, which largely depends on the availability of modes-of-interaction, while being mobile is even a bigger challenge, with all the complexities of dealing with various modality components such as Speech Recognizers, Synthesizers, Sensory interfaces, etc. Built as a reference implementation of W3C's Multimodal Architecture [1], Cue-me Multimodal Platform ("Cue-me") provides a multiplatform framework for the development of applications across mobile devices. Applications developed using the Cue-me architecture can run on desktops, laptops, tablets and a host of smartphone, wearable, and robotic devices.

R. Tumuluri (✉) • N. Kharidi
Openstream Inc, Somerset, NJ, USA
e-mail: raj@openstream.com

D.A. Dahl (ed.), *Multimodal Interaction with W3C Standards*,
DOI 10.1007/978-3-319-42816-1_9

9.2 Architecture Overview

The Cue-me architecture is based on the W3C MMI Framework [2]. The goal of the design is to provide a general and flexible framework providing interoperability among remote and local components on the device. The framework is based on the MVC Design Pattern widely used in languages such as Java and C++. The design pattern proposes three main parts: *a Data Model* that represents the underlying logical structure of the data and associated integrity constraints, one or more *Views* which correspond to the objects that the user directly interacts with, and *a Controller* which sits between the data model and the views. The separation between data and user interface provides considerable flexibility in how the data is presented and how the user interacts with that data. While the MVC paradigm has been traditionally applied to graphical user interfaces, it lends itself to the broader context of multimodal interaction where the user is able to use a combination of visual, aural, and tactile modalities.

9.2.1 Run-Time Framework

At the core, the Cue-me run-time framework consists of the Interaction Manager, interacting with several components in the framework. The Core components are HTML-GUI and Voice. Other components can be added based on the application interaction requirements. This approach provides the following flexibilities (Fig. 9.1):

- Granular Application Footprint Control for different devices and applications
- Platform extensibility using the component architecture
- Component level update/control

Figure 9.1 depicts the Cue-me platform with component interaction between the GUI, Voice, and other modalities. The application itself (Application Server and Database) can be local or remote.

9.2.2 Core Run-time Framework Components

The core run-time framework components consist of:

- GUI Modality (HTML)
- Voice Modality (providing speech interaction capability).

The core components provide the framework for building a multimodal application across platforms. The Modality Components can support local and remote interaction based on the platform choice and application configuration.

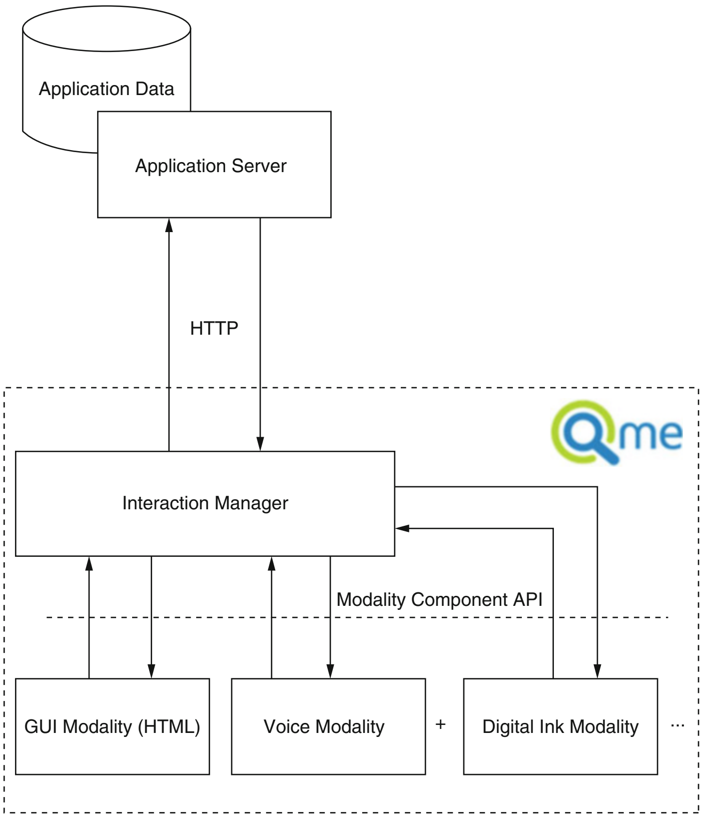


Fig. 9.1 Cue-me MMI architectural overview

For example, voice processing can be done remotely on a server or locally based on the choice of platform and the type of voice processing required by the application.

9.2.3 Other Components for Multimodal Applications

Mobility Applications require platform level component support. Cue-me achieves application portability by implementing mobile functionality using the component framework via the Interaction Manager. Some examples of other components are

- Ink Component using InkML [3]
- Signature Component
- DB Sync Component

. . ..

Because the Cue-me architecture uses an open framework for component integration (W3C MMI framework), components can be built by third parties and plugged in.

Hence, Cue-me is an *extensible* platform—components specific to the target platform can be plugged into the framework architecture for interaction via the Interaction Manager.

9.2.4 Anatomy of a Multimodal Application

A multimodal application developer must provide XML and text application files for each of the following components:

- GUI: The visual application delivers HTML or XHTML and the various supporting images and files including JavaScript and CSS to the web browser.

  It is very important that the GUI application is tailored to look and feel and run on a wide variety of mobile devices and wearables. The visual application should therefore be tested on as many different devices as possible before it is deployed.

  Two web sites which perform an analysis on the performance and quality of the visual mobile application are dev.mobi (http://ready.mobi) and the W3C mobile web initiative (http://validator.w3.org/mobile/).
- Voice: The voice application consists of a number of grammar and TTS files delivered to a speech engine for processing. The formats of the grammar and TTS files depend on what is supported by the speech engine. Grammar file formats include JSGF, ABNF, and SRGS. TTS file formats are either raw text or SSML. A developer should know what file formats the speech engine supports.

Other XML file formats may be used according to various requirements such as support for a legacy IVR application ported to run on Cue-me. This application may run one or more VoiceXML and/or CCXML files.

- Interaction Manager: The Cue-me Interaction Manager runs a variant of State Chart XML (SCXML) [4, 5], known as X-SCXML on the mobile phone or PDA. X-SCXML which implements SCXML, (with the exception of parallel states for want of resource optimization on mobile devices), can be used to integrate the interaction between the visual, voice, ink, and other modalities.

9.2.5 Basic Application Development Steps

Here are the basic application development steps, as shown by Fig. 9.3, below:

1. Create a Dynamic web project within a development environment such as Eclipse. Eclipse should be version 3.4.2 or higher and at a minimum the Web Standard Tools (WST) must be installed.

2. Create the visual web application, or import a legacy application into the project as required. The web application should be adapted for browsers running on mobile devices with limited memory and processing resources.
3. Identify the various states in the application. Each state may be associated with a separate web page or a web page may have more than one state, if the page can be associated with more than one set of active grammars. Generally, a unique set of active grammars represents a separate state.
4. Create the voice grammars, or modify for Cue-me if already available. There should be one or more grammars representing all controls and fields for each application web page. Usually there are grammars which remain active across several or all web pages.
5. Each grammar should also have a corresponding text file for displaying for the user what the user can say when the grammar is active.
6. The X-SCXML document is developed to integrate the voice and visual user interactions. When complete, a cookie is added to the application's home web page specifying the URL location of the X-SCXML document:

```
<%
Cookie
cookie = new Cookie ("IM-Loc", "http://www.ex.com/im/im.scxml");
cookie.setMaxAge (10);
response.addCookie (cookie);
 %>
```

   - A web page can set the Interaction Manager to another state using a cookie to specify the new state. For example, a JSP may have:

```
<%
Cookie cookie = new Cookie ("IM-State", "trade");
cookie.setMaxAge (10); // Do NOT set Max Age more than 10-20 s
response.addCookie (cookie);
 %>
```

   - The X-SCXML may direct the GUI component to execute a JavaScript function (e.g., in response to a voiceResult event). This function must of course be defined in the active web page.

7. The application and the voice proxy (one of the Cue-me deliverables) must both be deployed to a web server. After configuring the voice and proxy, the application is tested on a mobile phone running the Cue-me multimodal platform.

   - The web application and voice proxy may be deployed on different application servers (Figs. 9.2 and 9.3).

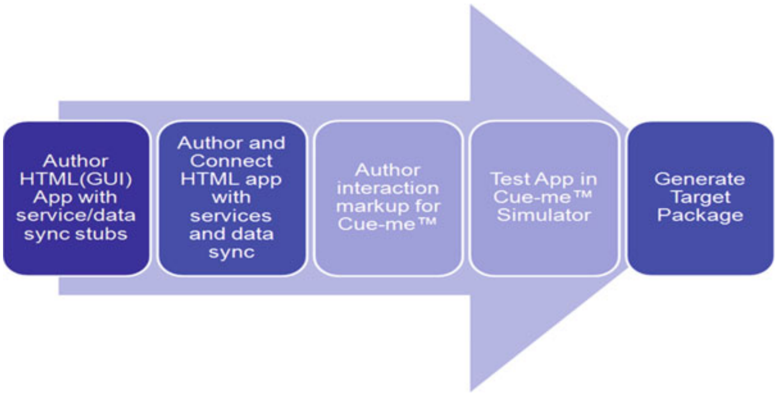

Author HTML(GUI) App with service/data sync stubs
Author and Connect HTML app with services and data sync
Author interaction markup for Cue-me™
Test App in Cue-me™ Simulator
Generate Target Package

Fig. 9.2 Overall application development flow

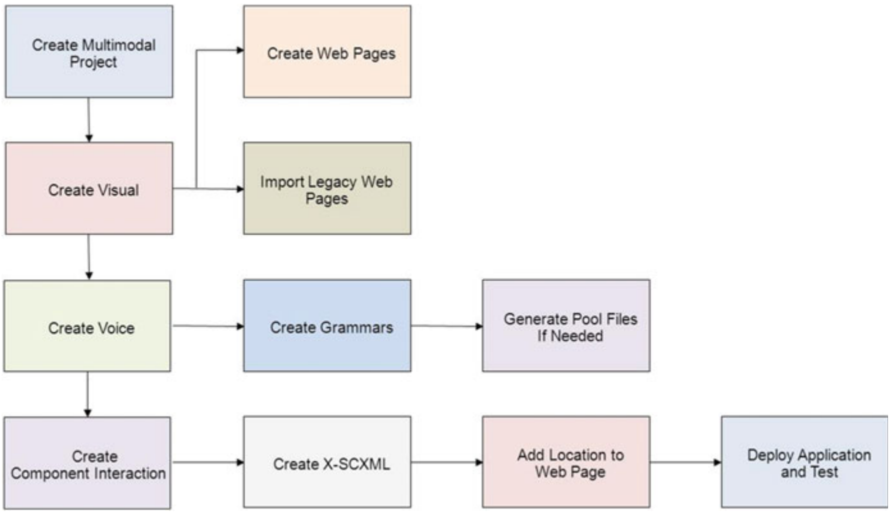


Fig. 9.3 Basic multimodal application development steps

9.2.6 Application Components Overview

A basic multimodal web application should have markup and software for processing by the following components:

- Application Server and Database (for business logic)
- Interaction Manager (for multimodal interaction integration)
- Voice Client Modality
- GUI Modality
- Access to a Cue-me Multimodal Server to provide voice interactions among other multimodal features.

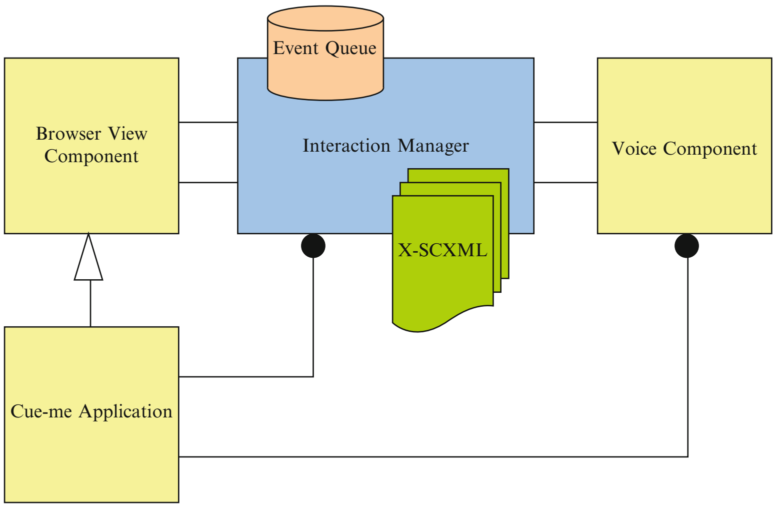


Fig. 9.4 Cue-me components object model

Figure 9.4 shows the object model of the Cue-me components on the client.

The components which reside on the client device are the Interaction Manager, the GUI (or HTML) modality, and the Voice client. GUI modality interactions, as represented by HTML, are deployed by an Application Server such as Apache Tomcat. The Cue-me Multimodal Server runs on a remote server and provides support for voice interactions.

When the voice recognition and TTS are rendered by a Cue-me Multimodal Server, the voice modality spans both client and server with a voice component on the client communicating with the server. However, voice recognition and TTS may both be rendered on the device, or recognition may be rendered by the server and TTS rendered on the device, or vice versa.

Both the GUI and Voice components on the device communicate with an Interaction Manager, which performs the modality integration by processing an X-SCXML document. X-SCXML is covered in a later section in this document.

9.2.6.1 Application Server

The Application Server deploys the application for the GUI modality represented by HTML and the X-SCXML for processing by the Interaction Manager. For an Application Server such as IBM WebSphere, the application is comprised of JSP's, Servlets, and static HTML pages.

Typically the application requires one or more databases for storing user information.

The application server returns some flavor of HTML or XHTML to the GUI component of Cue-me in response to an HTTP GET or POST request. This response may contain the URL location of the X-SCXML document required by Cue-me's Interaction Manager Component. The URL location is returned to the client in the response header as a Cookie:

Set-Cookie:

IM-Loc=http://www.example.com/ChineseFood/im/im.scxml;

The response may also contain a cookie for setting the ID of one of the states contained in the X-SCXML document. In response to this directive the Interaction Manager will move to this state. The cookie is specified as follows:

Set-Cookie:

IM-State=home;

The application server may specify the location of the X-SCXML document and set the state at one time by appending a "#" followed by the state ID to the IM-Loc cookie:

Set-Cookie:

IM-Loc=http://www.example.com/ChineseFood/im/im.xscxml#changeorder;

If the application sets a cookie for IM-Loc with the initial state appended, as shown as above, and also sets a cookie for IM-State, the cookie for IM-State is ignored. This is because the contents of the IM-Loc cookie always have priority. Best practice is to set a separate IM-Loc and IM-State cookies and not append the state to the IM-Loc URL.

The application server may return an X-SCXML document in response to the HTTP request. Support for processing X-SCXML should be specified in the HTTP Accept header as follows:

Accept: application/xscxml+xml, text/html, text/plain, image/*

9.2.6.2 Interaction Manager

The Interaction Manager (IM) processes one or more X-SCXML documents for a single multimodal application, where X-SCXML is an XML language derived from SCXML. The X-SCXML directs the IM to retrieve resources, add event listeners to specified events, and to process the events generated by each multimodal component.

While X-SCXML is similar to SCXML, a number of small changes were made to facilitate processing on the client. This "SCXML-like" language is processed by the IM as follows (examples included):

(1) Set each modality component's base URI for documents retrieved from the server:

```
<base id="x-voice" url="http://A.B.C.D/sb/voice"/>
```

or, preferably:

```
<send event="base" url="$APP_BASE/voice"/>
```

As shown above, the Cue-me Platform supports an $APP_BASE macro containing the application's host address and application context. According to the example above,

$APP_BASE = http://A.B.C.D/sb

$HOST is the other supported macro and contains the value of the application's host address. According to the example above,

$HOST = http://A.B.C.D

(2) Send initial (e.g., initialize or prepare) commands to the modality components:

```
<send event="prepare" to="x-voice" url="http://.../voice"/>
```

(3) On entry to a state send one or more commands to the modality components:

```
<send event="addGrammar" to="x-voice" url="login.jsgf"/>
<send event="addHelp" to="x-html" url="login.txt"/>
```

(4) Add an event listener to listen for a click event and in response send a command to the voice modality to response with TTS (e.g., "you said hello"):

```
<go on="click" node="hello_id">
<send event="playUrl" to="x-voice" url="voice/sayHello.txt"/>
</go>
```

(5) Go to the menu state if the event data contains the "order" string:

```
<go on="location" to="menu" if="event.name=='order'"/>
```

The Interaction Manager is composed of the following components.

9.2.7 *XML Language Parser*

The parser is a SAX parser which parses the X-SCXML into a state machine object. While it parses the X-SCXML it also generates events for initial processing and adds event handlers to the modality components.

9.2.8 *State Machine*

The state machine maintains the current state and retrieves the actions for an event which triggers an event handler. It also returns the actions contained by the *onentry* and *onexit* tags for each state, and the final tag for the document.

9.2.9 *Event Queue*

All events are placed on an internal event queue to be processed one-by-one by the Interaction Manager.

9.2.10 Data Model

The current application may have a data model specified by the X-SCXML document. The data model is a set of EcmaScript variables, which may be assigned in the X-SCXML with the <assign> tag and accessed by the <send> tag's "*expr*" attribute. The <script> tag may also be used to declare, assign, and access variables and functions.

9.2.11 X-SCXML Markup Language

The X-SCXML Markup language defines the actions and event listeners for a set of states enabling multimodal interactions between the user of a mobile device and a web application running on the device.

X-SCXML is a simplified version of State Chart XML (SCXML) [4, 5], an XML language that represents the execution environment of a state machine based on Harel state charts.

A state machine as defined by X-SCXML is a set of:

- States: A state embodies information about the current situation in the system.
- Events: An event is an input message to the state machine.
- Transitions: A transition is either a change from one state to another, usually triggered by an event, or it is a set of actions to be performed when triggered by an event.
- Data model variables.
- Actions: An action is an activity to be performed by the state machine at a given point in time.
- Primitives: to express guard conditions on transitions.

X-SCXML is processed by an Interaction Manager (IM) residing on a limited resource mobile client. The IM's role on the client is generally to enable multimodal interaction between a web application and the user, as defined by the W3C Multimodal Interaction Working Group's architecture.

According to this architecture:

- A state is the set of currently active grammars, files, web pages, etc., in the multimodal system.
- A life-cycle event is a message sent between a modality component (Speech, Visual, SMS, Ink, etc.) and the IM.
- An action is an activity the IM requests the modality component to perform.
- A transition moves the system either to another multimodal state or triggers a set of actions to be performed by one or more modality component. For the latter case, a transition defines an event handler.

X-SCXML is a mobile profile of SCXML (https://www.w3.org/TR/scxml/), in that it currently excludes the SCXML <parallel> tag which can capture concurrent behavior within a state machine, as it is not a requirement for most mobile applications.

Here is an example X-SCXML document representing a "Food Order" application. Its outermost state is identified as "home." The "home" state has one "order" state. For any voice result event received by the IM while within the nested states, the view component calls a JavaScript function "handleCommand" with a parameter string contained in the "event.value" property.

The data model declares an EcmaScript variable, "_data.welcome." Upon entering the "home" state, the voice component is instructed to play, "Welcome to Chinese Food Order!."

```
<?xml version="1.0" encoding="UTF-8"?>
<xscxml initialstate="home" version="1.0">
  <initial>
<send event="base" to="x-html" url="$APP_BASE/gram/"/>
           <send event="base" to="x-voice" url="$APP_BASE/gram/"/>
    <send event="prepare" to="x-voice" url="$HOST/mmivoice"/>
  </initial>
  <datamodel>
    <data id="welcome" expr="'Welcome to Chinese Food Order!'">
  </datamodal>
  <state id="home">
           <onentry>
     <send event="addGrammar" to="x-voice" url="chineseord.gram"/>
     <send event="addHelp" to="x-html" url="chineseord.txt"/>
     <send event="playText" to="x-html" expr="_data.welcome"/>
           </onentry>
           <go on="voiceResult" from="x-voice">
            <send event="execute" data="event.value" to="x-html"
          target="handleCommand" />
           </go>
           <state id="order">
     <onentry>
       <send event="addGrammar" to="x-voice" url="foods.gram"/>
   <send event="addHelp" to="x-html" url="foods.txt"/>
     </onentry>
     <onexit>
       <send event="removeGrammar" to="x-voice" url="foods.gram"/>
   <send event="removeHelp" to="x-html" url="foods.txt"/>
     </onexit>
</state>
  </state>
  <final>
    <send event="unPrepare" to="x-voice" url="$HOST/mmivoice"/>
  </final>
</xscxml>
```

9.2.12 Cue-me Multimodal Server

The Multimodal Server is required for a voice component configured for remote speech recognition and synthesis. The Multimodal Server handles communication and data transfers between the Voice Component running on the client.

The Multimodal Server processes speech input against one or more provided grammars, returns results, and generates TTS audio to be rendered by the device. This is the remote speech processing model and no application resources need to be stored on the Server.

9.2.13 Voice Component

The Voice Component records the speech input and plays the output (TTS) as directed by the Interaction Manager. It will send the recorded input to either a remote or local speech engine for processing, depending on how the mobile device is configured.

Events and data from the Multimodal Server are forwarded to the Interaction Manager as events for processing.

9.2.14 GUI Modality

The GUI modality is the mobile phone's Web Browser, except that it also adds event listeners as directed by the IM, sends events to the IM, and handles events from the IM. Event listeners added to the GUI modality add listeners for DOM events to the current HTML page.

The GUI modality can listen for the any of the HTML DOM Events supported by HTML 5.

9.3 X-SCXML Processing

9.3.1 Introduction

The X-SCXML document is processed on a mobile device after being downloaded from a remote server. While it is being parsed all initial actions as well as on-entry actions for the initial state are placed on the internal event queue for processing by the IM.

For each state the parser encounters it also saves all go transitions to a state object identified by the state's identifier. If the state is the initial state it adds an "add event listener" event to the event queue for each go transition.

Once the state objects are constructed for all the states the state machine is created to store the states as well as the final section of the X-SCXML document. The state machine sets the current state to the state identified by the "initialstate" attribute on the <xscxml> tag.

The X-SCXML is comprised of the following:

9.3.2 Data Model

The <datamodel> tag declares the data model for the X-SCXML application. One or more <data> tags below <datamodel> declare a set of EcmaScript variables for storing and accessing information during X-SCXML document processing.

9.3.3 Initial and Final

The <initial> tag contains a set of actions, each denoted by the <send> tag, which one or more modality components are to perform with the initialization of the X-SCXML document.

The <final> tag complements the <initial> tag, containing the set of <send> actions which the modality components are to perform upon exiting the application.

9.3.4 States

At any time a multimodal application may be in one of a set of interaction states. Each active state has its own set of the files (e.g., grammars, web page, etc.) and event handlers. Some or all of these files and handlers may be replaced when there is a transition to another state.

States may be nested. There are several advantages of having nested states: (1) the outermost state can be the global state for the application. That is, event handlers declared in the outermost state apply to all the nested states below. (2) The X-SCXML can model applications which are more complex. (3) X-SCXML is more compliant with Harel State charts and SCXML. Transformations between SCXML and X-SCXML are thereby simpler.

9.3.4.1 State Transitions

A state transition is declared with the <go> tag and is usually triggered by an event from a modality component. The next state is declared by the <go> tag's "to" attribute.

Here is an example state transition:

```
<go on="voiceResult" from="x-voice" to="foodorder" node="myhome"/>
```

9.3.4.2 Event Handlers

An event handler is declared with the <go> tag and is usually triggered by an event from a modality component. For example, a "voiceResult" event from the voice modality might trigger an action to update a field within a web page with the data returned with the "voiceResult" event.

A <go> tag which is an event handler won't have the "to" attribute and has one or more <send> tags as children. If there are conditionals specified by <if>, <elseif>, and <else> tags, only <send> actions below the conditionals which evaluate to true upon receipt of the event are performed by the modality components.

Here is an example event handler:

```
<go on="voiceResult" from="x-voice">
            <send event="execute" data="event.data" to="x-html"
                        target="handleCommand" />
     </go>
```

9.3.4.3 On Entry and On Exit Processing

A state may contain one <onentry> and one <onexit> tag. When the state machine transitions to a new state the actions contained by the <onentry> tag below the new state are sent by the IM to the modality components.

The <onexit> tag is the complementary to <onentry>; the actions contained below <onexit> for the old state are performed when the state machine transitions to a new state.

9.3.5 Event Types

Following the W3C MMI Life-cycle API, the modality components support various API calls as appropriate.

9.3.5.1 Voice Client Event Types

The Voice Client event types are independent of the location of the speech rendering with the exception of the "prepare" and "unPrepare" events. The latter events are required by the Voice Proxy for setting up and tearing down resources on the remote Voice Server.

9.3.5.2 Visual (GUI) Modality Event Types

Event types emitted by visual modality

Event	Description	Detail
click	User clicks on a DOM node	MouseEventDetail
getFieldResponse	Returns content requested with the getField event	Target is the id of an HTML tag containing text
newPage	A new web page has been received	Title of the Web Page URL of the Web page
newContextRequest	A new X-SCXML document is available (discovered in HTML header or cookie)	URL of the X-SCXML document
nextState	Move to a new X-SCXML state	New state ID[a]

[a]The Interaction Manager ignores the nextState event if currently in the state specified by the event's new state ID.

9.3.6 Example getField, getFieldResponse, and playText

A typical use case for the getField and getFieldResponse event combination is to get the contents of a text area or paragraph and forward to the voice client to be played as TTS. For example, the X-SCXML can be programmed to request a paragraph identified by "art_text_id" when requested by the user. When the getFieldResponse event is received, the contents of the associated event.property can be included as data with the playText event sent to the voice component. The X-SCXML snippet is shown below:

```
<go on="voiceResult" from="x-voice">
  <if node="readId"/>
<send event="getField" to="x-html" target="art_text_id" />
  </if>
</go>
<go on="getFieldResponse" from="x-html">
  <send event="playText" data="event.value" to="x-voice" />
</go>
```

9.3.7 Example Noinput and Message

A possible use case for the noinput and message event combination is to display a message for the user when the voice client emits a noinput event. The message lets the user know that the voice client is waiting for speech input. The X-SCXML snippet is shown below:

```
<go on="noinput" from="x-voice">
<send event="message" to="x-html" data="Please begin \
        talking. Press help if you need help as to what to say." />
</go>
```

9.3.7.1 The Event Object

An event object is included with every event put on the internal queue by a modality component. This object is most useful in capturing the voice result when a user's speech input has matched against a grammar. The event object has two properties:

- Name—the node name or the name associated with a name/value pair of a semantic interpretation result.
- Value—the raw result of the value associated with a name/value pair of a semantic interpretation result.

The event object properties include confidence levels and other properties conforming to the W3C EMMA language [6–8] for annotating recognized user input.

9.3.7.2 URL Macros

The IM will process the following two macros when encountered in an X-SCXML document:

- $HOST: This macro represents the server host of the current X-SCXML document's URL. It consists of protocol + host name + [optional] port number. For example, $HOST would contain "http://example.com:5650" if the X-SCXML was retrieved from "http://example.com:5650/im/example.xml."
- $APP_BASE: This macro represents the host plus the application context root retrieved from the current X-SCXML document's URL. For the above example, $APP_BASE would contain "http://example.com:5650/im."

As already explained *IM is the linchpin for both modality and context*, and forms the control plane driving/adapting the behavior of the application for the given context/device or situation.

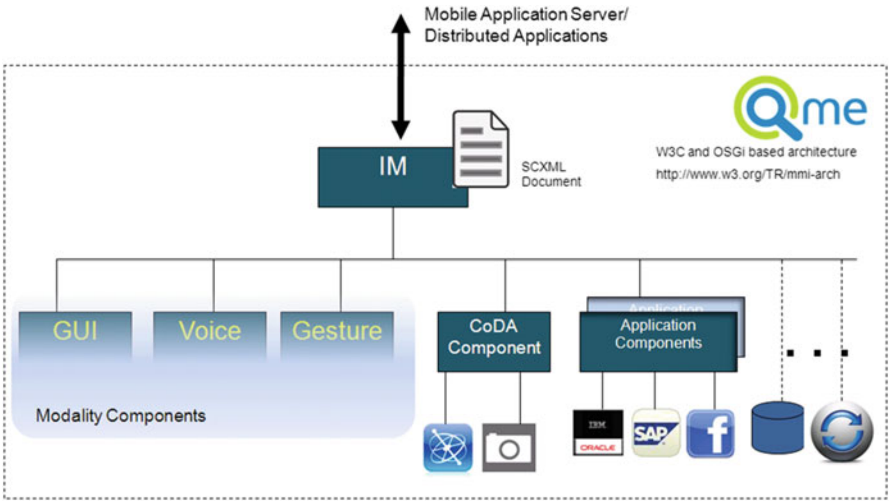


Fig. 9.5 Anatomy of a Cue-me™ application

The ContextDeliveryArchitecture *CoDA component (aka Delivery Context Component)* delivers device context information like location, camera, acceleration, etc. The application uses this context information and drives behavior via state transitions in the IM.

Several Enterprise Utility and Infrastructure components like *Database*, *Synchronization*, etc., are available and co-ordinated through the Interaction Manager (IM) for Enterprise application use.

For the purposes of on-device integration of both Enterprise and Cloud Application services, the *component architecture is extensible* to include application level components connecting to external services and creating services based mashups (Fig. 9.5).

Upon application launch, the *Interaction Manager* starts the Interaction and initializes one or more components based on the Application's SCXML document. The SCXML document itself may be present locally, or obtained from a URL resource. As part of the HTML Component initialization, HTML resources are either obtained from the Cue-me™ sandbox or via URL request to the appropriate server. Speech, Gesture, and other components are initialized and the Application is ready to execute.

The typical execution model is depicted in Fig. 9.6.

9.4 Application Development

Resources for the application are created and packaged using Open Web IDEs or an Eclipse IDE with Cue-me™ Studio Plugins.

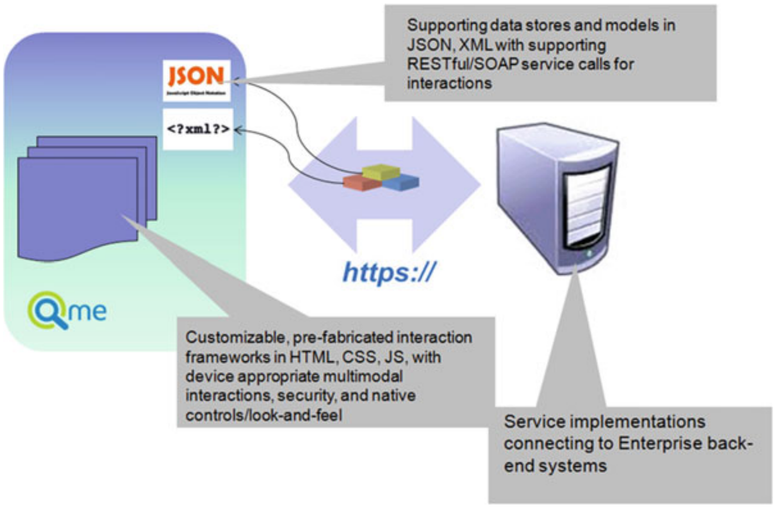

Supporting data stores and models in JSON, XML with supporting RESTful/SOAP service calls for interactions
JSON
<?xml?>
https://
Qme
Customizable, pre-fabricated interaction frameworks in HTML, CSS, JS, with device appropriate multimodal interactions, security, and native controls/look-and-feel
Service implementations connecting to Enterprise back-end systems

Fig. 9.6 MMI (cue-me) application execution model

9.4.1 HTML Editor with Interaction Context (SCXML) Palette

Interaction Context defines the interaction modes and device peripherals that a Cue-me app can interact with. Interaction Context is device-specific because devices capabilities and peripherals differ.

Cue-me Studio provides an HTML Editor with SCXML palette. The SCXML palette allows users to bind the SCXML components to the elements of HTML document. For example, a Text input field can have a Gesture and Scanner inputs in addition to the keyboard input.

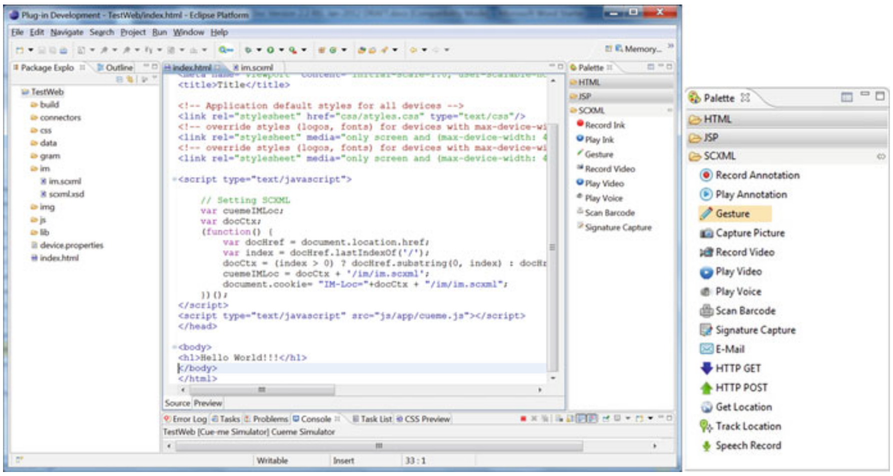
The following picture shows the HTML Editor with the SCXML Palette on its right (Fig. 9.7).

9.4.2 Multimodal Interactions (SCXML)

SCXML Editor is an XML editor with enhanced code assist for multimodal components and events. The editor can be used to edit a multimodal application's Interaction Manager (IM) document that was generated.

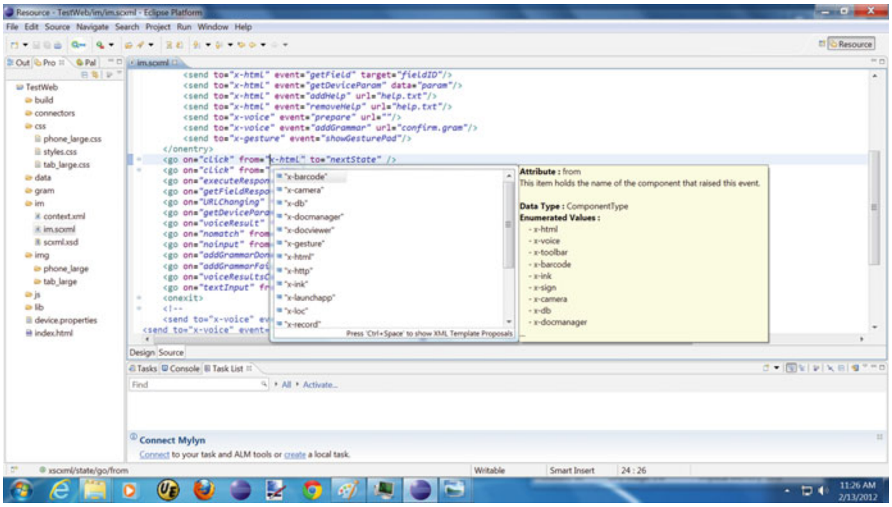
9.4.2.1 Components

Components like "x-html, x-voice, etc.," handle the action events from IM and perform requested actions. Components can also raise events after performing the



```
<title>Title</title>

<!-- Application default styles for all devices -->
<link rel="stylesheet" href="css/styles.css" type="text/css"/>
<!-- override styles (logos, fonts) for devices with max-device-wi
<link rel="stylesheet" media="only screen and (max-device-width: 4
<!-- override styles (logos, fonts) for devices with max-device-wi
<link rel="stylesheet" media="only screen and (max-device-width: 4

<script type="text/javascript">

    // Setting SCXML
    var cuemeIMLoc;
    var docCtx;
    (function() {
        var docHref = document.location.href;
        var index = docHref.lastIndexOf('/');
        docCtx = (index > 0) ? docHref.substring(0, index) : docHr
        cuemeIMLoc = docCtx + '/im/im.scxml';
        document.cookie= "IM-Loc="+docCtx + "/im/im.scxml";
    })();
</script>
<script type="text/javascript" src="js/app/cueme.js"></script>
</head>

<body>
<h1>Hello World!!!</h1>
</body>
</html>
```

Fig. 9.7 HTML editor with SCXML palette



```
        <send to="x-html" event="getField" target="fieldID"/>
        <send to="x-html" event="getDeviceParam" data="param"/>
        <send to="x-html" event="addHelp" url="help.txt"/>
        <send to="x-html" event="removeHelp" url="help.txt"/>
        <send to="x-voice" event="prepare" url=""/>
        <send to="x-voice" event="addGrammar" url="confirm.gram"/>
        <send to="x-gesture" event="showGesturePad"/>
    </onentry>
    <go on="click" from="x-html" to="nextState" />
    <go on="click" from="
    <go on="executeRespon
    <go on="getFieldRespo
    <go on="URLChanging"
    <go on="getDevicePara
    <go on="voiceResult"
    <go on="nomatch" from
    <go on="noinput" from
    <go on="addGrammarDon
    <go on="addGrammarFai
    <go on="voiceResultsC
    <go on="textInput" fr
    <onexit>
    <!--
    <send to="x-voice" ev
  <send to="x-voice" event=
```

```
"x-barcode"
"x-camera"
"x-db"
"x-docmanager"
"x-docviewer"
"x-gesture"
"x-html"
"x-http"
"x-ink"
"x-launchapp"
"x-loc"
"x-record"
Press 'Ctrl+Space' to show XML Template Proposals
```

Attribute : from
This item holds the name of the component that raised this event.

Data Type : ComponentType
Enumerated Values :
- x-html
- x-voice
- x-toolbar
- x-barcode
- x-ink
- x-sign
- x-camera
- x-db
- x-docmanager

Fig. 9.8 SCXML editor

requested actions. In the following picture, code assist after (ctrl+space bar), "to=" displays the list of components (Fig. 9.8).

9.4.2.2 Action Events

Action Events are the events that can be sent to Components like "x-html." In the following picture, code assist after (ctrl+space bar), "event=" displays the list of events that can be sent to components.

9.4.2.3 Raised Events

Raised Events are the events that can be raised on Actions like "go." In the following picture, code assist displays the list of raised events (Fig. 9.9a, b).

9.5 Example: The Chinese Food Order Application

This section will put all the previous sections together by describing an example multimodal application and how all the pieces, HTML, grammars, and X-SCXML, fit together.

The following two images show the main page of the Chinese Food Order application. In most cases developers will be adding multimodal interaction to a legacy web application so it makes sense to start with the HTML pages or JSPs.

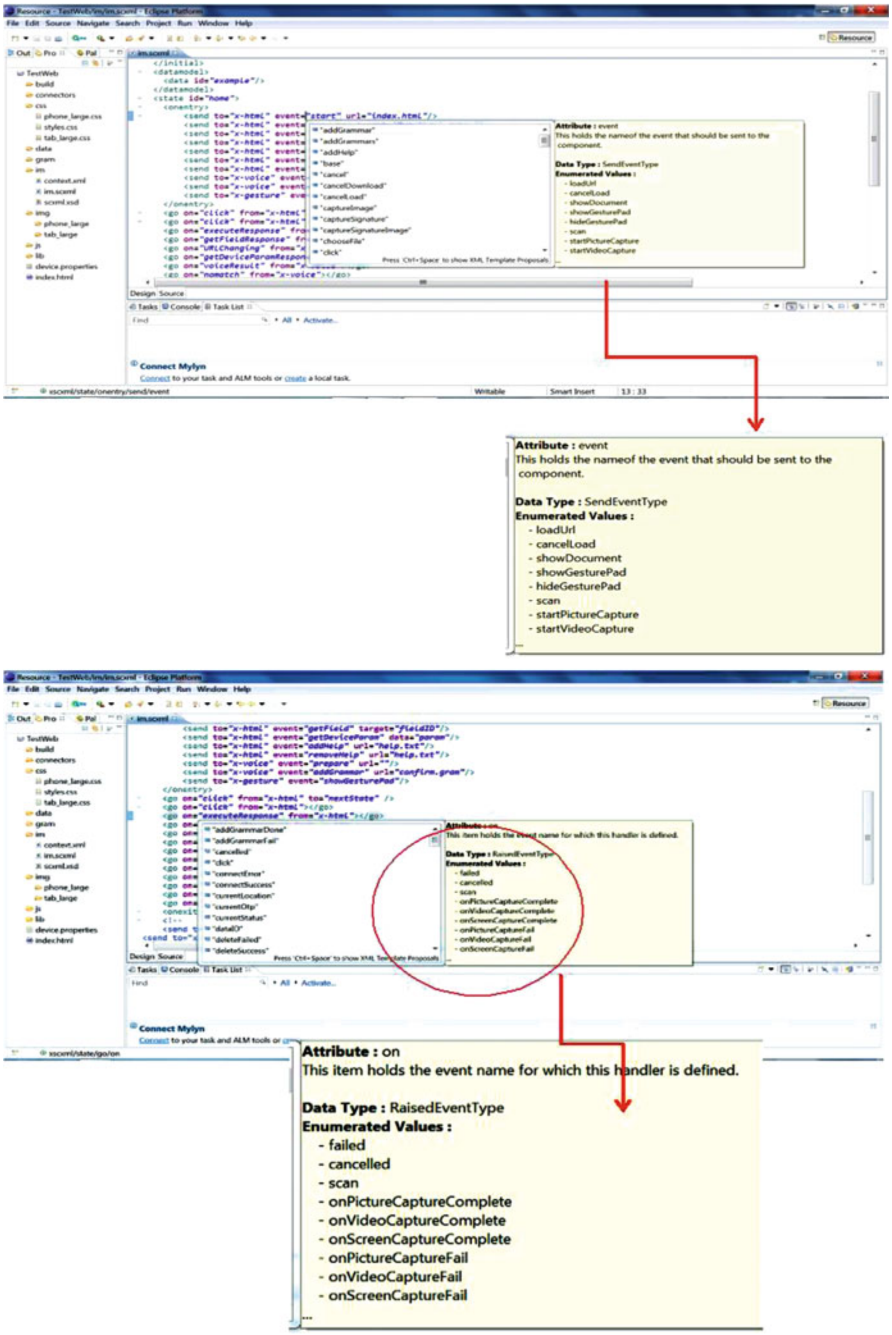
All pages will generally have a set of global controls for navigating the application while other fields and buttons can be said to be specific to each of the pages.

It is important that each web page has a unique title. Because multiple web pages may be served by the same servlet, the different responses (pages) can refer to same URL. Cue-me adjusts interaction state by associating each web page with a unique combination of URL and title. Requiring a unique title is a good practice in any case and <title> is a required XHTML tag (Fig. 9.10).

The next step is to add the voice interaction to this page. Adding voice interaction means adding the grammar and help files which will perform the same actions as clicking on the buttons and entering text into the fields with a keypad or stylus.

Here is the Chinese Food Order grammar in SRGS or ABNF format:

```
#ABNF 1.0 iso-8859-1;
language en-US;

mode voice;
root $chineseorder;
tag-format <semantics/1.0>;

public $chineseorder = [I would like | I'd like] [to (order|get)]
[please] [give me] ((change [my | the] order {$.changeOrder=true})
       | (what is (my | the) order {$.what=true})
       | (submit [my | the] [order] {$.done=true})
| ([I'm] (done | finished | ready) [with] [my | the] [order] {$.done=true})
       | ([and] ([an] egg roll {$.appEggRoll=true}
     | [some] pork dumplings {$.appPkDump=true}
     | [some] crispy spring rolls {$.appCriSpr=true}
     | [with] [a|an|some] fried rice {$.riceFried=true}
     | roast pork noodle soup {$.spRoastPk=true}
     | hot and sour soup {$.spHotSour=true}
     | chicken with sweet corn soup {$.spChicken=true}
     | mixed vegetables in oyster sauce {$.entree="Mixed Vegetables"}
```



```
</initial>
<datamodel>
  <data id="example"/>
</datamodel>
<state id="home">
  <onentry>
      <send to="x-html" event="start" url="index.html"/>
      <send to="x-html" event
      <send to="x-html" event
      <send to="x-html" event
      <send to="x-html" event
      <send to="x-html" event
      <send to="x-html" event
      <send to="x-voice" event
      <send to="x-voice" event
      <send to="x-gesture" eve
  </onentry>
  <go on="click" from="x-html"
  <go on="click" from="x-html"
  <go on="executeResponse" fro
  <go on="getFieldResponse" fr
  <go on="URLChanging" from="x
  <go on="getDeviceParamRespon
  <go on="voiceResult" from="x
  <go on="nomatch" from="x-voice"></go>
```

"addGrammar"
"addGrammars"
"addHelp"
"base"
"cancel"
"cancelDownload"
"cancelLoad"
"captureImage"
"captureSignature"
"captureSignatureImage"
"chooseFile"
"click"

Press 'Ctrl+Space' to show XML Template Proposals

Attribute : event
This holds the nameof the event that should be sent to the component.

Data Type : SendEventType
Enumerated Values :
- loadUrl
- cancelLoad
- showDocument
- showGesturePad
- hideGesturePad
- scan
- startPictureCapture
- startVideoCapture

```
      <send to="x-html" event="getField" target="fieldID"/>
      <send to="x-html" event="getDeviceParam" data="param"/>
      <send to="x-html" event="addHelp" url="help.txt"/>
      <send to="x-html" event="removeHelp" url="help.txt"/>
      <send to="x-voice" event="prepare" url=""/>
      <send to="x-voice" event="addGrammar" url="confirm.gram"/>
      <send to="x-gesture" event="showGesturePad"/>
  </onentry>
  <go on="click" from="x-html" to="nextState" />
  <go on="click" from="x-html"></go>
  <go on="executeResponse" from="x-html"></go>
```

"addGrammarDone"
"addGrammarFail"
"cancelled"
"click"
"connectError"
"connectSuccess"
"currentLocation"
"currentOtp"
"currentStatus"
"dataID"
"deleteFailed"
"deleteSuccess"

Press 'Ctrl+Space' to show XML Template Proposals

Attribute : on
This item holds the event name for which this handler is defined.

Data Type : RaisedEventType
Enumerated Values :
- failed
- cancelled
- scan
- onPictureCaptureComplete
- onVideoCaptureComplete
- onScreenCaptureComplete
- onPictureCaptureFail
- onVideoCaptureFail
- onScreenCaptureFail

Fig. 9.9 (**a**) Action events, (**b**) Raised events

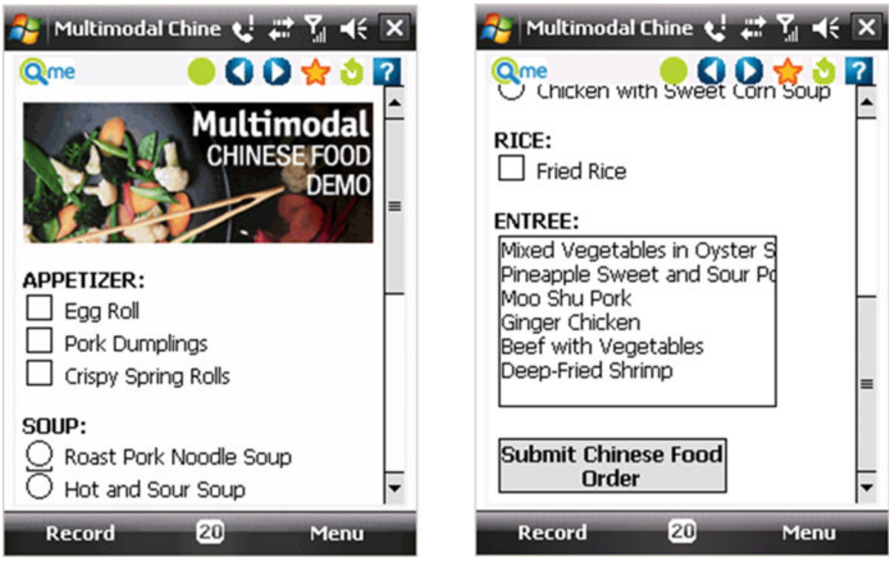


Fig. 9.10 Chinese food order demo

```
  | pineapple sweet and sour pork {$.entree="Sweet and Sour Pork"}
     | moo shoo pork {$.entree="Moo Shu Pork"}
     | ginger chicken {$.entree="Ginger Chicken"}
     | beef with vegetables {$.entree="Beef"}
     | deep fried shrimp {$.entree="Shrimp"}))<1->);
```

9.5.1 Chinese Food Order Global Grammar

Grammars for Cue-me should always be written in the above format using semantic interpretation to set the node and value. The node and value set with semantic interpretation correspond to the name and value properties of the event object associated with the voiceResult event processed by X-SCXML. For example, "$.entree="Shrimp" sets the node to "entree" and the value to "Shrimp."

Associated with each grammar should be a help file which tells the user what he or she can say on the page. This is a text file stored on a server along with the grammar file. Here is an example help file for the above Chinese Food Order grammar:

You may specify most or all of your order at one time, if you already know what you want.
For example, you could say:

I would like moo shoo pork, with
an eggroll, hot and sour soup,
and some fried rice.

To change your order you can say:
    I'd like to change my order.

When done with your order you can say:
    Submit my order.

To hear your order you can say:

    What is my order?

9.5.2 Chinese Food Order Grammar Help Text

9.5.2.1 Interaction Integration with X-SCXML

The Interaction Manager integrates the visual and voice interactions as directed by the X-SCXML document associated with the Chinese Food Order application. Each modality component has an identifier: "x-html" identifies the GUI and "x-voice" identifies the voice. The X-SCXML document's initial state is "home."

In the initial section of the X-SCXML, each component's base URL is set, the voice component is directed to prepare the voice modality for this client session, add the global Chinese Food Order grammar, and get the help text to be displayed by the GUI.

```
<?xml version="1.0" encoding="UTF-8"?>
<xscxml initialstate="home">
  <initial>
    <send event="base" to="x-html" url="$APP_BASE/gram/"/>
            <send event="base" to="x-voice" url="$APP_BASE/gram/"/>
    <send event="prepare" to="x-voice" url="$HOST/.../mmivoice"/>
  </initial>
```

9.5.3 Initial Section

The data model declares EcmaScript variables for use during the X-SCXML document processing. The Chinese Food Order application defines one variable, "welcome" which is accessed as "_data.welcome" within the application.

```
<datamodel>
    <data id="welcome" expr="'Welcome to Chinese Food Order!'"/>
  </datamodel>
```

9.5.4 Data Model Section

The Chinese Food X-SCXML has a home state and a nested state below the home state so to update the single page dynamically as if the user was engaged in a dialog.

On entry to the home state for the page the grammar and help text events for the page are added. The voice component is also sent the welcome message to play contained in the "_data.welcome" variable.

The X-SCXML <go> tag specifies an event handler. For example, when a voice result event is received by the Interaction Manager, the Interaction Manager will send a JavaScript execute event to the GUI component. The JavaScript function *submit_form* will be called if the event name is "submitButton," and in other words when the user clicked on the button with ID attribute set to "submitButton." If the global attribute is set to "true," this event handler is active for all the states in the grammar. This is the global event handler associated with the global Chinese Food Order grammar.

Here are the order form and nested dialog state sections:

```
<state id="orderform" initialstate="dialog">
          <onentry>
           <send event="addGrammar" to="x-voice" url="chineseorder.gram"/>
           <send event="addHelp" to="x-html" url="chineseorder.txt"/>
           <send event="playText" to="x-voice" expr="_data.order"/>
          </onentry>
          <state id="dialog">
      <go on="voiceResult" from="x-voice" to="dialog"
          cond="event.name!='changeOrder'&amp;&amp;event.name!='done'&amp;&amp;event.name!='what'">
            <send event="setField" data="event.value" to="x-html" target="event.name"/>
      </go>
     </state>
           <go on="click" from="x-html" node="submitButton" to="orderform">
         <send event="playText" to="x-voice" data="Thanks for your order!"/>
                 <send event="execute" to="x-html" target="submit_form"/>
           </go>

   <go on="voiceResult" from="x-voice" node="done" to="orderform">
        <send event="playText" to="x-voice" data="Thanks for your order!"/>
           <send event="execute" to="x-html" target="submit_form"/>
    </go>

   <go on="voiceResult" from="x-voice" to="changeorder" node="changeOrder"/>
   <go on="voiceResult" from="x-voice" node="what" global="true">
          <send event="execute" to="x-html" target="getOrder"/>
   </go>

        <go on="executeResponse" from="x-html" node="getOrder" global="true">
          <send event="playText" to="x-voice" expr="event.value"/>
        </go>
        <onexit>
   <send event="removeGrammar" to="x-voice" url="chineseorder.gram"/>
                 <send event="removeHelp" to="x-html" url="chineseorder.txt"/>
                 </onexit>
        </state> ...
```

9.5.5 Order Form State

On exit from the order form state the order form grammar and help files are removed from the speech engine.

The Interaction Manager can direct the GUI component to update a field with the "setField" event. For example, when the voice result has an event node set to anything besides "changeOrder," "done," or "what", the GUI is directed to set the field with ID set to the value contained in the event object's name property, with a value contained in the value property.

```
<go on="voiceResult" from="x-voice" to="dialog"
      cond="event.name!='changeOrder'&amp;&amp;event.name!='done'&amp;&amp;event.name!='what'">
         <send event="setField" data="event.value" to="x-html" target="event.name"/>
</go>
```

9.5.6 Set Field with ID Contained in the Event Object's Name Property

In the final section of the X-SCXML document, the voice proxy is told to unprepare the current session.

```
<final>
 <send event="unPrepare" to="x-voice" url="/MMoic...Proxy/mmivoice"/>
</final>
```

9.5.7 Final Section

9.5.7.1 Add a Cookie to the Home Page

When development of the X-SCXML document is complete, a cookie must be added to the home page specifying its URL location. For example, for the Chinese Food Order application, the cookie is added to the market's summary JSP as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-html40/loose.dtd">
<html>
<%
Cookie cookie =
  new Cookie ("IM-Loc", "http://example.xml/Chinese/im/ch.xscxml");
cookie.setMaxAge(10); // Do NOT set Max Age more than 10-20 s
cookie.setVersion(1);
response.addCookie(cookie);
 %>
```

9.5.7.2 "IM-Loc" Cookie for the X-SCXML URL

The "IM-Loc" cookie should not have a max age set to more than 20 s. Otherwise, it may remain active after the user leaves the Chinese Food Order application.

9.6 Extending the MMI Architecture to Connected Devices

9.6.1 Development of a Prototype Multimodal Robot Using MMI

Let's call our robot EasyHealthAssistant EHA. EHA uses Text-To-Speech (TTS), Speech recognition, Face Recognition, Motion sensing and Mindwave (EEG) interactions to effectively engage patients and care-givers. The robot features bluetooth connectivity to third party health devices and wearables such as a Blood Pressure Monitor or electronic fitness trackers. EHA keeps care-givers informed by providing real time updates concerning the patient's adherence, vital readings, and any new developments in the condition of the patient. The robot facilitates video conference calls leveraging WebRTC with the physician or the care-giver on command (speech/touch) (Fig. 9.11).

9.6.2 Design of EHA

EHA is a static robot which has a revolving head and three passive Infrared Motion (PIR) sensors. The camera, display, microphone, and speakers from a phone are located on the revolving head. When motion is detected, the head turns in the corresponding direction and faces the user (Fig. 9.12).

One of the core functions of EHA is the dispensing of medication to the correct user at the prescribed time. While, the servos, motors, PIR sensors, and Wi-Fi connectivity are controlled using an Arduino microcontroller, the robot itself is controlled by EHA program implemented using the W3C Multimodal architecture, leveraging Openstream's Cue-me multimodal platform.

The control-logic (Interaction Manager) is implemented in SCXML, while the modality components are implemented in other languages with events flowing between Interaction Manager and modality components including the Arduino controller (Fig. 9.13).

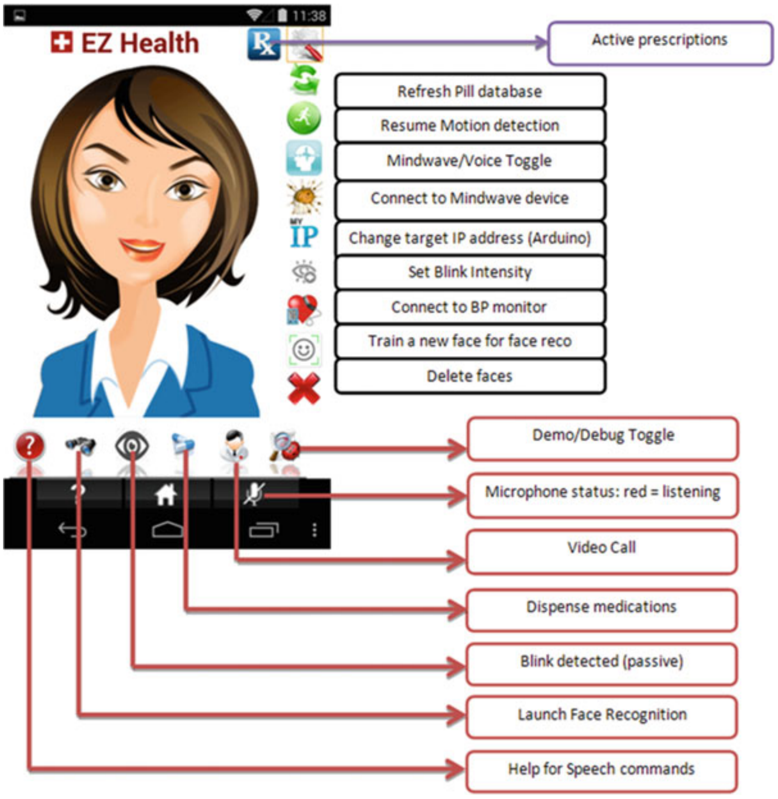


Fig. 9.11 Feature-overview

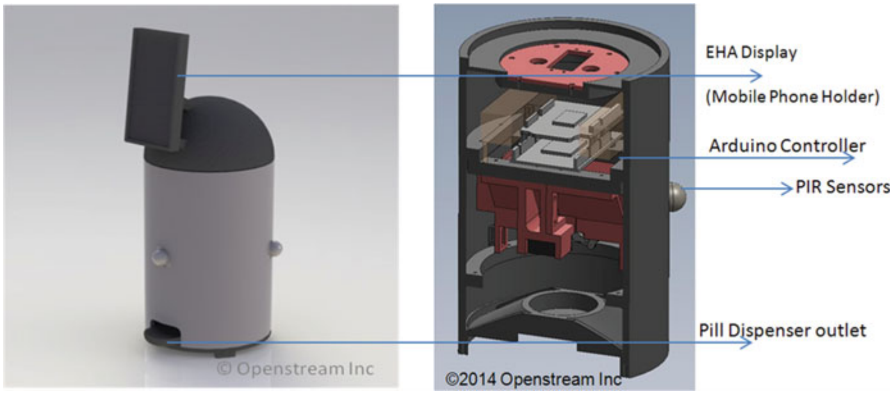


Fig. 9.12 Functional prototype (fully built and sectional views)

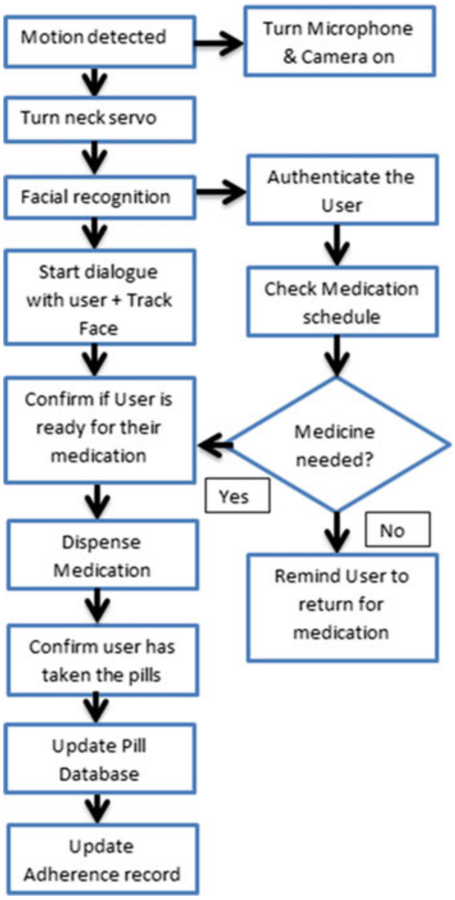

Motion detected
Turn Microphone & Camera on
Turn neck servo
Facial recognition
Authenticate the User
Start dialogue with user + Track Face
Check Medication schedule
Confirm if User is ready for their medication
Medicine needed?
Yes
No
Dispense Medication
Remind User to return for medication
Confirm user has taken the pills
Update Pill Database
Update Adherence record

Fig. 9.13 Sample flow and high-level events

9.6.3 Motion Detection

There are three PIR sensors located 120° apart to allow for a full range of motion sensing. The "motion detected" event is the trigger which begins the interaction between the user and EHA. When motion is detected, the robot turns to face the direction of motion and launches face recognition and face tracking.

9.6.4 Face Recognition

The face recognition engine serves the purpose of authentication and personalization for the user. The authentication of the user gives EHA context for any upcoming interaction, such as reminding the user when it is time to take medication

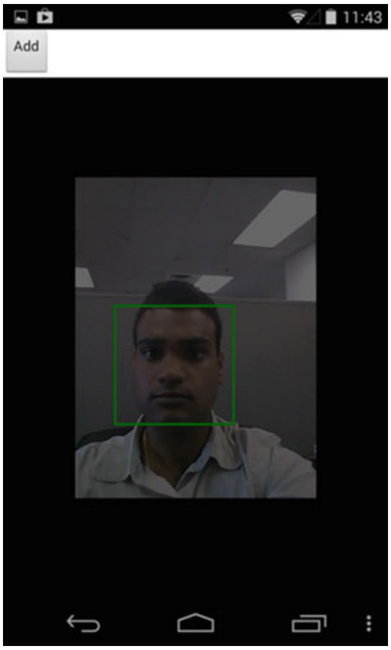


Fig. 9.14 Face-recognition (enhanced camera-component)

or if the care-giver has changed the regimen, while it improves the quality of the interaction by referring to the user with her first name (Fig. 9.14).

An EMMA 2.0-based representation of the face recognition result is given below. The result includes an ID and name corresponding to the recognized match. More than one match can be returned, in which case, the confidence score can be used to arrive at the most likely match.

```
<emma:emma version="2.0"
      xmlns:emma="http://www.w3.org/2003/04/emma"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.w3.org/2003/04/emma
      http://www.w3.org/TR/2009/REC-emma-20090210/emma.xsd"
      xmlns="http://www.example.com/example">
  <emma:one-of    id="r1"    emma:start="1087995961542"    emma:
  end="1087995963542"
      emma:medium="visual"    emma:mode="video"    "emma:device-
      type:"camera"  emma:source="http://example.com/camera/LFDJ-
      43U">
    <emma:interpretation id="int1" emma:confidence="0.85">
      <matchId>43879</matchId>
      <matchName>John Doe</matchName>
    </emma:interpretation>
    <emma:interpretation id="int2" emma:confidence="0.27">
      <matchId>90328</matchId>
      <matchName>Mark Johnson</matchName>
    </emma:interpretation>
  </emma:one-of>
</emma:emma>
```

9.6.5 Face Tracking

Face tracking allows EHA to respond to the users movements in a more human-like manner. Once motion is detected and the head turns in the corresponding direction, the camera locates the user and sends commands to the neck servo to face the user squarely. As the user moves around, EHA will continue to track the user's face and adjust its head-position accordingly.

An EMMA 2.0-based representation of the face tracking result is given below. The tracking information is provided in terms of the top-left and bottom-right co-ordinates of the "box" that contains the face in the picture captured by the video camera. It is assumed that there is a single face in the picture being interpreted. The incremental results feature of EMMA 2.0 is used here so that face tracking data can be continuously generated as the user's movements are tracked.

```
<emma:emma version="2.0"
  xmlns:emma="http://www.w3.org/2003/04/emma"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2003/04/emma
  http://www.w3.org/TR/2009/REC-emma-20090210/emma.xsd"
  xmlns="http://www.example.com/example">
  <emma:one-of   id="r1"   emma:start="1087995961542"   emma:
  end="1087995963542"
  emma:medium="visual" emma:mode="video" emma:device-type:″camera″
  emma:source="http://example.com/camera/LFDJ-43U">
    <emma:interpretation id="int1" emma:confidence="0.7"
      emma:tokens="{'top-left':{'x':323.219, 'y':643.980}, 'bottom-
      right':{'x':732.543, 'y':132.376}} ">
      emma:token-type="json"
      emma:stream-id="s1"
      emma:stream-seq-num="0"
      emma:stream-status="begin"
      emma:stream-token-span="0-1"
      emma:stream-full-result="true"
    </emma:interpretation>
    <emma:interpretation id="int2" emma:confidence="0.43"
      emma:tokens="{‘top-left’:{‘x’:196.088, ‘y’:778.692}, ‘bottom-
      right’:{‘x’:478.011, ‘y’:389.289}} ">
      emma:token-type="json"
      emma:stream-id="s1"
      emma:stream-seq-num="0"
      emma:stream-status="begin"
      emma:stream-token-span="0-1"
      emma:stream-full-result="true"
    </emma:interpretation>
  </emma:one-of>
</emma:emma>
```

9.6.6 TTS and Speech Recognition

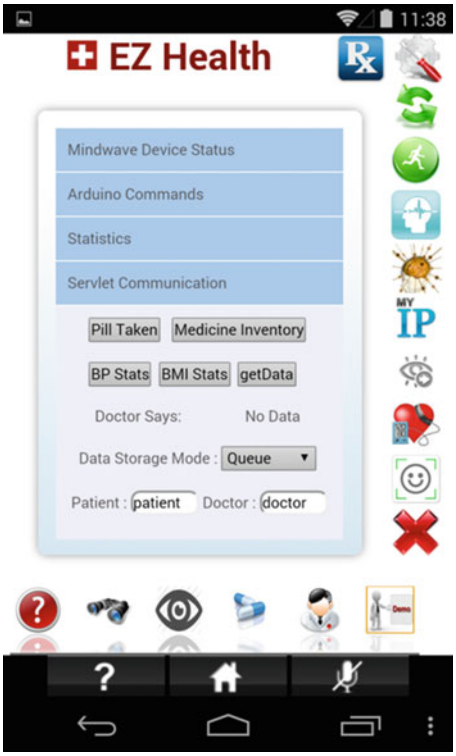
EHA is equipped with TTS and local speech recognition to make the interaction with the user more natural. The speech recognizer uses a lead word to reduce false positive recognition and end of speech recognition to simulate a normal conversation (Fig. 9.15).

9.6.7 EEG Headset

There are two modes for EHA; a passive mode, where the user provides commands and the robots executes them, and an active mode where EHA initiates and drives the interaction. Once paired with the NeuroSky Mindwave headset, EHA asks binary questions in a logical progression and a redundant manner. The user has to voluntarily blink for an affirmative response. This method of interaction is capable of communicating with individuals that are incapable of speech or are disabled in a way that prevents from using normal interaction (Fig. 9.16).

9.6.8 Bluetooth Integration

EHA leverages Cue-me™'s Bluetooth Low Energy (BLE) component and can read directly from Bluetooth Health-care Device Profile (HDP) and other compatible health-monitoring devices and medical instrumentation obviating the need for user to enter the readings manually off the instrument-displays.



Doctor Says: No Data

Data Storage Mode : Queue

Patient : patient Doctor : doctor

Fig. 9.15 Monitoring vitals

Fig. 9.16 Mindwave headset

9.6.9 Ink and Speech Annotation

EHA facilitates capture of images using camera and high-lighting the picture with Ink and Voice annotations. The annotations are exchanged as W3C Synchronized Multimedia Integration Language (SMIL) documents for maintaining the isochronous nature of the multimodal annotations as shown below:

Sample SMIL document:

```
<smil>
  <head>
    <layout>
      <root-layout width="240" height="299" />
    </layout>
  </head>
  <body>
  <par>
            <audio  src="audio_10_0.wav"  begin="500"  dur="6486"
            region="main" />
            <audio src="audio_10_1.wav" begin="15029" dur="10696"
            region="main" />
            <img src="image_10.jpg" region="main" />
            <ref  src="inkml_10.xml"  type="application/inkml+xml"
            region="main" />
  </par>
  </body>
</smil>
```

The corresponding InkML document referenced in the SMIL document is as given below (truncated for brevity) :

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<inkml:ink xmlns:inkml="http://www.w3.org/2003/InkML">
  <inkml:definitions>
    <inkml:context xml:id="ct_0" >
      <inkml:inkSource xml:id="inksrc_0" >
        <inkml:traceFormat>
                <inkml:channel name="X" type="integer" max="480"
                units="dev"/>
                <inkml:channel name="Y" type="integer" max="598"
                units="dev"/>
        </inkml:traceFormat>
        <inkml:channelProperties>
             <inkml:channelProperty channel="X" name="resolution"
             value="0" units="1/dev"/>
```

```
          <inkml:channelProperty channel="Y" name="resolution"
          value="0" units="1/dev"/>
      </inkml:channelProperties>
    </inkml:inkSource>
  </inkml:context>
  <inkml:brush xml:id="br_0">
    <inkml:brushProperty name="color" value="#FF0000" />
    <inkml:brushProperty name="width" value="3" units="dev" />
    <inkml:brushProperty name="style" value="1" />
  </inkml:brush>
  <inkml:brush xml:id="br_1">
    <inkml:brushProperty name="color" value="#FF0000" />
    <inkml:brushProperty name="width" value="3" units="dev" />
    <inkml:brushProperty name="style" value="1" />
  </inkml:brush>
  <inkml:brush xml:id="br_2">
    <inkml:brushProperty name="color" value="#FF0000" />
    <inkml:brushProperty name="width" value="3" units="dev" />
    <inkml:brushProperty name="style" value="1" />
  </inkml:brush>
  <inkml:brush xml:id="br_3">
    <inkml:brushProperty name="color" value="#FF0000" />
    <inkml:brushProperty name="width" value="3" units="dev" />
    <inkml:brushProperty name="style" value="1" />
  </inkml:brush>
  <inkml:brush xml:id="br_4">
    <inkml:brushProperty name="color" value="#FF0000" />
    <inkml:brushProperty name="width" value="3" units="dev" />
    <inkml:brushProperty name="style" value="1" />
  </inkml:brush>
  <inkml:brush xml:id="br_5">
    <inkml:brushProperty name="color" value="#FF0000" />
    <inkml:brushProperty name="width" value="3" units="dev" />
    <inkml:brushProperty name="style" value="1" />
  </inkml:brush>
  <inkml:brush xml:id="br_6">
    <inkml:brushProperty name="color" value="#FF0000" />
    <inkml:brushProperty name="width" value="3" units="dev" />
    <inkml:brushProperty name="style" value="1" />
  </inkml:brush>
  <inkml:brush xml:id="br_7">
    <inkml:brushProperty name="color" value="#FF0000" />
    <inkml:brushProperty name="width" value="3" units="dev" />
    <inkml:brushProperty name="style" value="1" />
  </inkml:brush>
```

```
    <inkml:brush xml:id="br_8">
      <inkml:brushProperty name="color" value="#FF0000" />
      <inkml:brushProperty name="width" value="3" units="dev" />
      <inkml:brushProperty name="style" value="1" />
    </inkml:brush>
    <inkml:brush xml:id="br_9">
      <inkml:brushProperty name="color" value="#FF0000" />
      <inkml:brushProperty name="width" value="3" units="dev" />
      <inkml:brushProperty name="style" value="1" />
    </inkml:brush>
    <inkml:brush xml:id="br_10">
      <inkml:brushProperty name="color" value="#FF0000" />
      <inkml:brushProperty name="width" value="3" units="dev" />
      <inkml:brushProperty name="style" value="1" />
    </inkml:brush>
  </inkml:definitions>
                <inkml:trace contextRef="#ct_0" brushRef="#br_0"
duration="1284" timeOffset="1001" >281 48,281 48,277 48,277 48,273
48,273 48,269 50,269 50,264 51,264 51,261 51,261 51,255 53,255 53,247
55,247 55,237 57,237 57,229 59,229 59,225 60,225 60,221 62,221 62,212
64,212 64,208 65,208 65,201 69,201 69,197 70,197 70,196 71,196 71,189
75,189 75,187 77,187 77,181 83,181 83,177 87,177 87,175 92,175 92,173
96,173 96,171 103,171 103,171 106,171 106,169 114,169 114,171 117,171
117,172 124,172 124,175 128,175 128,176 133,176 133,180 139,180
139,184 144,184 144,189 149,189 149,193 152,193 152,199 155,199
155,203 157,203 157,208 160,208 160,216 162,216 162,221 163,221
163,233 164,233 164,248 165,248 165,257 164,257 164,263 164,263
164,268 164,268 164,277 163,277 163,283 162,283 162,292 161,292
161,296 160,296 160,301 159,301 159,311 157,311 157,316 157,316
157,327 154,327 154,332 153,332 153,343 150,343 150,352 148,352
148,356 146,356 146,359 145,359 145,363 143,363 143,369 139,369
139,372 137,372 137,375 135,375 135,377 132,377 132,393 109,393
109,395 107,395 107,395 103,395 103,395 101,395 101,393 97,393 97,393
96,393 96,392 90,392 90,392 86,392 86,391 84,391 84,391 83,391 83,389
79,389 79,387 76,387 76,384 74,384 74,381 72,381 72,379 70,379 70,376
68,376 68,373 66,373 66,371 64,371 64,367 63,367 63,364 61,364 61,360
59,360 59,357 57,357 57,353 55,353 55,348 54,348 54,345 53,345 53,340
51,340 51,337 50,337 50,332 47,332 47,325 45,325 45,321 44,321 44,319
43,319 43,315 43,315 43,308 43,308 43,304 43,304 43,296 44,296 44,291
45,291 45,281 47,281 47,276 48,276 48,276 48,276 48</inkml:trace>
  <inkml:trace contextRef="#ct_0" brushRef="#br_1" duration="955"
timeOffset="3537" >545 69,545 69,541 70,541 70,537 70,537 70,533
70,533 70,529 71,529 71,525 72,525 72,521 74,521 74,517 74,517 74,512
76,512 76,507 78,507 78,504 79,504 79,501 81,501 81,499 83,499 83,493
86,493 86,492 87,492 87,489 89,489 89,485 93,485 93,481 96,481 96,480
```

98,480 98,455 130,455 130,452 138,452 138,451 141,451 141,451 144,451
144,451 148,451 148,451 152,451 152,455 162,455 162,456 165,456
165,460 169,460 169,464 174,464 174,468 177,468 177,477 185,477
185,484 189,484 189,488 192,488 192,491 193,491 193,496 194,496
194,505 196,505 196,509 197,509 197,520 197,520 197,525 197,525
197,535 195,535 195,555 190,555 190,565 186,565 186,569 184,569
184,573 181,573 181,577 179,577 179,584 173,584 173,591 169,591
169,593 165,593 165,597 161,597 161,599 157,599 157,601 151,601
151,604 143,604 143,607 136,607 136,607 132,607 132,608 124,608
124,609 121,609 121,609 118,609 118,609 115,609 115,608 110,608
110,607 107,607 107,605 104,605 104,604 102,604 102,601 97,601 97,600
95,600 95,600 93,600 93,597 88,597 88,596 87,596 87,592 83,592 83,591
81,591 81,588 79,588 79,585 78,585 78,579 76,579 76,573 75,573 75,571
75,571 75,567 75,567 75,561 74,561 74,556 73,556 73,556 73,556 73</
inkml:trace>
<inkml:trace contextRef="#ct_0" brushRef="#br_2" duration="647"
timeOffset="8272" >59 220,59 220,60 222,60 222,64 229,64 229,67
233,67 233,68 237,68 237,71 242,71 242,73 249,73 249,75 253,75 253,76
256,76 256,79 262,79 262,80 266,80 266,80 267,80 267</inkml:trace>
<inkml:trace contextRef="#ct_0" brushRef="#br_3" duration="110"
timeOffset="8941" >76 241,76 241,79 239,79 239,81 237,81 237,85
234,85 234,88 232,88 232,93 229,93 229,93 229,93 229</inkml:trace>
<inkml:trace contextRef="#ct_0" brushRef="#br_4" duration="262"
timeOffset="9314" >91 214,91 214,91 217,91 217,92 218,92 218,95
226,95 226,97 232,97 232,99 235,99 235,107 250,107 250,111 251,111
251,111 251,111 251</inkml:trace>
<inkml:trace contextRef="#ct_0" brushRef="#br_5" duration="354"
timeOffset="9854" >121 245,121 245,123 243,123 243,125 242,125
242,127 239,127 239,132 234,132 234,133 230,133 230,136 225,136
225,135 220,135 220,133 218,133 218,132 215,132 215,124 213,124
213,117 217,117 217,113 222,113 222,112 225,112 225,112 234,112
234,113 238,113 238,121 240,121 240,125 241,125 241,128 240,128
240,132 240,132 240,139 238,139 238,141 236,141 236,144 234,144
234,147 232,147 232,147 230,147 230,147 230,147 230</inkml:trace>
<inkml:trace contextRef="#ct_0" brushRef="#br_6" duration="361"
timeOffset="10382" >151 201,151 201,155 203,155 203,156 205,156
205,159 207,159 207,160 210,160 210,163 214,163 214,165 218,165
218,167 221,167 221,167 225,167 225,164 222,164 222,164 220,164
220,164 218,164 218,164 216,164 216,163 214,163 214,163 211,163
211,163 209,163 209,161 202,161 202,163 200,163 200,164 198,164
198,165 197,165 197,168 195,168 195,168 195,168 195</inkml:trace>
<inkml:trace contextRef="#ct_0" brushRef="#br_7" duration="346"
timeOffset="10962" >200 210,200 210,203 208,203 208,204 206,204
206,204 204,204 204,205 202,205 202,205 200,205 200,205 194,205
194,204 189,204 189,203 187,203 187,200 185,200 185,191 189,191

189,188 192,188 192,185 197,185 197,184 201,184 201,184 204,184
204,184 209,184 209,185 211,185 211,192 213,192 213,195 213,195
213,199 212,199 212,203 212,203 212,209 209,209 209,216 207,216
207,219 205,219 205,223 202,223 202,223 202,223 202</inkml:trace>
 <inkml:trace contextRef="#ct_0" brushRef="#br_8" duration="1012"
timeOffset="16118" >397 214,397 214,395 214,395 214,389 215,389
215,383 215,383 215,373 216,373 216,369 217,369 217,361 218,361
218,353 221,353 221,349 222,349 222,344 223,344 223,336 226,336
226,331 229,331 229,327 230,327 230,323 233,323 233,315 238,315
238,311 240,311 240,301 248,301 248,299 252,299 252,297 255,297
255,293 262,293 262,293 266,293 266,292 269,292 269,292 273,292
273,293 281,293 281,295 285,295 285,295 289,295 289,296 293,296
293,299 297,299 297,300 301,300 301,309 312,309 312,316 319,316
319,320 323,320 323,324 326,324 326,333 332,333 332,337 335,337
335,344 336,344 336,360 342,360 342,367 343,367 343,373 344,373
344,379 345,379 345,404 345,404 345,411 344,411 344,417 344,417
344,424 343,424 343,437 340,437 340,443 340,443 340,449 338,449
338,457 336,457 336,465 333,465 333,475 331,475 331,491 325,491
325,511 319,511 319,516 316,516 316,524 311,524 311,528 310,528
310,531 307,531 307,536 302,536 302,537 299,537 299,541 292,541
292,543 288,543 288,543 286,543 286,544 279,544 279,543 271,543
271,540 270,540 270,539 268,539 268,533 265,533 265,531 263,531
263,528 262,528 262,523 257,523 257,520 254,520 254,515 249,515
249,512 246,512 246,508 243,508 243,501 239,501 239,497 238,497
238,493 237,493 237,484 233,484 233,480 231,480 231,475 230,475
230,465 226,465 226,461 223,461 223,409 214,409 214,403 214,403
214,397 214,397 214,383 214,383 214,379 214,379 214,379 214,379
214</inkml:trace>
 <inkml:trace contextRef="#ct_0" brushRef="#br_9" duration="792"
timeOffset="18293" >105 375,105 375,111 376,111 376,121 377,121
377,128 378,128 378,139 378,139 378,144 377,144 377,149 376,149
376,171 372,171 372,180 368,180 368,185 366,185 366,193 362,193
362,196 360,196 360,204 355,204 355,207 352,207 352,211 348,211
348,217 341,217 341,220 337,220 337,223 333,223 333,228 325,228
325,229 321,229 321,232 315,232 315,233 311,233 311,232 309,232
309,229 304,229 304,228 303,228 303,221 299,221 299,219 296,219
296,215 295,215 295,204 289,204 289,196 286,196 286,192 285,192
285,187 284,187 284,176 283,176 283,165 283,165 283,159 283,159
283,152 283,152 283,143 283,143 283,137 283,137 283,133 284,133
284,121 287,121 287,116 291,116 291,112 292,112 292,108 295,108
295,101 299,101 299,99 301,99 301,92 306,92 306,88 308,88 308,83
313,83 313,80 316,80 316,79 319,79 319,77 328,77 328,79 331,79 331,81
337,81 337,83 340,83 340,84 343,84 343,87 349,87 349,88 352,88 352,91
357,91 357,92 360,92 360,96 364,96 364,101 367,101 367,107 368,107
368,111 368,111 368,115 367,115 367,120 366,120 366,124 366,124
366,129 365,129 365,139 365,139 365,139 365,139 365</inkml:trace>

```
  <inkml:trace contextRef="#ct_0" brushRef="#br_10" duration="598"
timeOffset="20416" >608 218,608 218,604 220,604 220,600 222,600
222,596 224,596 224,592 226,592 226,587 230,587 230,584 233,584
233,583 234,583 234,580 238,580 238,579 240,579 240,577 242,577
242,577 245,577 245,575 250,575 250,575 252,575 252,572 255,572
255,572 260,572 260,572 265,572 265,573 269,573 269,577 272,577
272,583 274,583 274,587 275,587 275,592 276,592 276,596 276,596
276,601 275,601 275,607 272,607 272,609 271,609 271,611 269,611
269,616 265,616 265,617 263,617 263,620 260,620 260,620 256,620
256,620 251,620 251,620 249,620 249,620 245,620 245,620 242,620
242,621 238,621 238,621 236,621 236,621 234,621 234,619 230,619
230,617 229,617 229,601 222,601 222,601 222,601 222</inkml:trace>
</inkml:ink>
```

The captured image and audio recording play while redrawing the ink trace on the image, enabling sharing of rich annotations with care-givers.

9.7 Conclusion

Implementation of the platform using the W3C MMI architecture helped in streamlining the process of structured authoring for application developers. It mainly helped in the separation of concerns between application-design and interaction-design and helped them focus on the application functionality without unduly getting distracted by the differences in the features of underlying device/OS architecture. Further, it helped portability of their applications while shielding authors from nuances of the implementation of modality-components or getting locked-in to a particular vendor's speech-engine or location-service or other modality components. The soundness of the component-architecture of MMI enabled authors in extending modality components and thus the Cue-me platform to suit their needs. As of the date of writing this book, Cue-me applications are deployed on over 3.5 million devices around the world. The types of deployed applications range from banking and financial services, field-force automation, retail-store operations, health-care services, and corporate applications facilitating collaboration and enhancing human interaction, while increasing user-productivity. It helped support distributed deployment of applications across connected devices. Some of the latest features of the Cue-me platform include registration and discovery of modality components, so that authors can now write applications that can dynamically discover and bind to modality components/services.

Acknowledgments Thanks to all the team members of Openstream especially, R. Anthapu, R. Sripada, G. McCobb, K. Patel, N. Shah, T. Sindhaghatta, S. Anthapu, B. Narayana, and S. Khandavilli, who helped in the design and development and the guidance and help provided by D. Dahl, K. Ashimura at W3C MMI.

References

1. Barnett, J., Bodell, M., Dahl, D. A., Kliche, I., Tumuluri, R., Larson, J., Porter, B., et al. (2012). Multimodal architecture and interfaces. World Wide Web Consortium. http://www.w3.org/TR/mmi-arch/. Accessed 20 Nov 2012.
2. Larson, J. A., Raman, T. V., & Raggett, D. (2002). W3C Multimodal Interaction Framework. W3C. http://www.w3.org/TR/mmi-framework/.
3. Watt, S. M., Underhill, T., Chee, Y.-M., Franke, K., Froumentin, M., Madhvanath, S., et al. (2011). Ink Markup Language (InkML). World Wide Web Consortium. http://www.w3.org/TR/InkML. Accessed Nov 2012.
4. Barnett, J. (2016). Introduction to SCXML. In D. Dahl (Ed.), *Multimodal interaction with W3C standards: toward natural user interfaces to everything*. New York, NY: Springer.
5. Barnett, J., Akolkar, R., Auburn, R. J., Bodell, M., Burnett, D. C., Carter, J. et al. (2015) State Chart XML (SCXML): State Machine Notation for Control Abstraction. World Wide Web Consortium. http://www.w3.org/TR/scxml/. Accessed 20 Feb 2016.
6. Johnston, M. (2016). EMMA. In D. Dahl (Ed.), *Multimodal interaction with W3C standards: towards natural user interfaces to everything*. New York, NY: Springer.
7. Johnston, M., Baggia, P., Burnett, D., Carter, J., Dahl, D. A., McCobb, G., et al. (2009). EMMA: Extensible MultiModal Annotation markup language. W3C. http://www.w3.org/TR/emma/. Accessed 9 Nov 2012.
8. Johnston, M., Dahl, D. A., Denny, T., & Kharidi, N. (2015). EMMA: Extensible MultiModal Annotation markup language Version 2.0. World Wide Web Consortium. http://www.w3.org/TR/emma20/. Accessed 16 Dec 2015.

URL's

Easy Health Assistant use case video: http://youtu.be/x2Tte0QyTiA.
Openstream Easy Health Assistant Mindwave Demo: http://youtu.be/aEgdRU-yM2o.
Poor Medication adherence costs $290 Billion Annually: http://mobilehealthnews.com/3901/poor-medication-costs-290-billion-a-year/.
W3C Multimodal Architecture: http://www.w3.org/TR/mmi-arch/.
Cue-me™ Multimodal Authoring Platform: http://www.openstream.com/cueme.
SCXML – State Chart XML: (http://www.w3.org/TR/scxml/).
Neurosky Mindwave: http://neurosky.com/products-markets/eeg-biosensors/hardware/.