Spatial Audio with the W3C Architecture for Multimodal Interfaces

Stefan Radomski
TU Darmstadt - Telecooperation Group
Hochschulstr. 10
64289 Darmstadt, Germany
radomski@tk.informatik.tu-darmstadt.de

Dirk Schnelle-Walka
TU Darmstadt - Telecooperation Group
Hochschulstr. 10
64289 Darmstadt, Germany
dirk@tk.informatik.tu-darmstadt.de

ABSTRACT
The development of multimodal applications is still hampered by the necessity to integrate various technologies and frameworks into a coherent application. In 2012, the W3C proposed a multimodal architecture, standardizing the overall structure and events passed between the constituting components in a multimodal application. In this paper, we present our experiences with implementing a multimodal application employing spatial audio, text-to-speech and XHTML.

General Terms
Standardization; Languages

Keywords
Multimodality, Dialog Managment

1. INTRODUCTION
In 2003, the W3C realized the necessity to standardize application development of multimodal applications by introducing a common conception for such applications as the W3C Multimodal Interaction (MMI) *Framework* [5]. By 2012, the W3C finally published the W3C MMI *Architecture* recommendation [3], defining the constituting components in a multimodal application with their responsibilities and events passed between them.

In an earlier publication we already outlined our experiences with providing a component in the W3C MMI architecture [10] and compared the approach to the state of the art with regard to multimodal interfaces in general. In this paper, we describe our experiences while implementing such an application with a focus on speech and spatial audio.

In the W3C MMI architecture, a multimodal application is decomposed into a nested structure of Interaction Managers (IM) coordinating Modality Components (MC), with their respective behavior described by various markup languages (see figure 1). The IM's responsibility is to model dialog management at different levels of granularity and to control the MCs employed by a multimodal application. The MCs offer access to a given modality or set thereof by accepting and processing events or complete modality specific documents. MC's them self can again be nested IM's, modeling more fine-granular aspects of dialog management like error-correction, form-filling or even sensor fusion and fission.

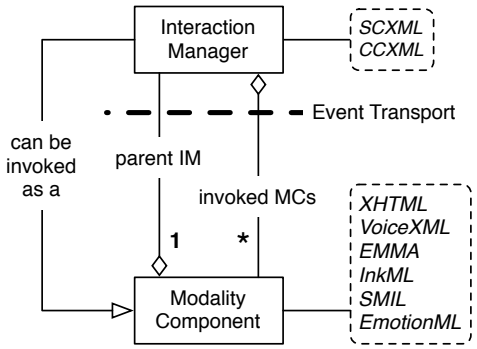

Figure 1: Collaboration diagram of MCs and IMs with their respective markup languages (adapted from [10]).

A multimodal application is described as a set of XML documents, with the topmost *root controller document* describing the global dialog structure. When such a document is processed, its interpreter invokes other MCs, passing modality specific markup as *presentation documents*.

One recommendation to express IMs is to employ State Chart XML (SCXML) [2] documents as an implementation of Harel state-charts [4] with nested and parallel machine configurations. An SCXML interpreter will enter an initial configuration, optionally invoke modality components and then waits for events. These events are usually passed by the invoked components, but can also be raised by the interpreter itself. The SCXML interpreter also features a datamodel as an embedded scripting language to maintain a state apart form the current configuration, guard transitions and perform processing on events sent and received. The SCXML standard itself does not talk about the W3C MMI architecture at any point, but its concepts lend themselves naturally to perform the responsibilities of an interaction manager.

The set of events described by the MMI architecture is given in table 1. An interactive session starts either by an MC sending a new context request to its parent IM, which will ultimately be answered by a new context response or by an IM directly sending an optional prepare request with a subsequent start request to one or more modality components.

Every modality component in the MMI architecture has to process and respond to the defined life-cycle events for instantiation, pause/resume, keep-alive and termination. Actual application specific events as e.g. user input or system output can either be sent directly in the prepare/start request or in subsequent extension notifications. The set of defined data fields per event is given in figure 2 and explained in table 2.

The overall approach of the W3C MMI architecture is to be distinguished from earlier / other attempts with e.g. XHTML+Voice [1] or the continuing trend for XHTML [6] to take on ever more responsibilities, by starting with a modality-agnostic control language and employ the respective markup languages to express modality specific interaction.

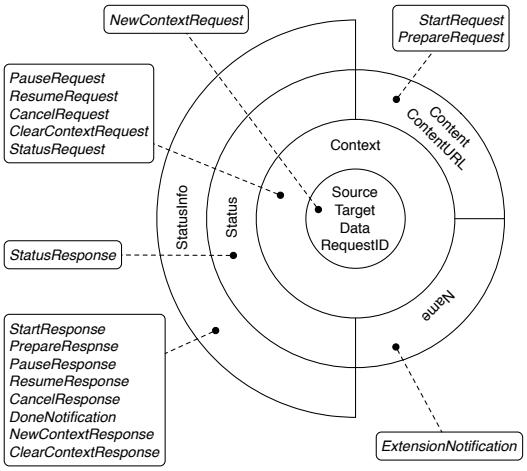

Figure 2: Data fields of the various W3C MMI events (adapted from [10]).

2. DEMO APPLICATION

For the demo application, we realized a use-case from an industrial system to monitor construction equipment deployed throughout an area. The equipment features various sensors to indicate its overall state and some critical measurements like oil-pressure or current velocity. In a series of preprocessing steps, these measurements are refined to geo-referenced messages indicating various failure or warning conditions. We assume the availability of these messages and for our application they are mocked-up using a publish/subscribe middleware.

In this monitoring application, the geo-referenced ticker messages are displayed on a map as they are received. Upon reception, a simple clicking sound is rendered as an audio notification from the direction relative to the current center of the map with its volume as a function of the distance.

Table 1: Life-cycle events between IMs and MCs (adapted from [10]).

Event	Origin	Description
Requests		
Prepare	IM	Initialize and preload data. Can be sent multiple times prior to starting.
Start	IM	Initiate processing of the document given as part of the request or per URL.
Pause	IM	Suspend processing of the current start request.
Resume	IM	Resume processing of the current start request.
Cancel	IM	Cancel processing of the current start request.
ClearContext	IM	Context no longer needed, free ressources and terminate if appropriate.
Status	IM	Keep-alive request.
NewContext	MC	Request for a new context from the interaction manager.
Responses		
Prepare	MC	If successful, the MC must respond with minimal delay to start requests.
Start	MC	Acknowledgement of success or failure.
Pause	MC	Acknowledge suspension.
Resume	MC	Acknowledgement of success or failure.
Cancel	MC	Acknowledgement of cancellation.
ClearContext	MC	Acknowledge end of context.
Status	MC	Keep-alive response if context is known, undefined otherwise.
Done	MC	End of processing reached.
NewContext	IM	Acknowledgement of success or failure for a NewContext Request.
Any		
Extension	Any	Application specific extensions with arbitrary data.

Furthermore, the messages' content is spoken via text-to-speech synthesis (see figure 3). Whenever the map is moved, the spatial audio component's listener position is updated to give the impression of hearing the audio notifications relative to the center of the map.

To model such an application in the W3C MMI architecture, we decomposed it into four modality components (i) the map display described in XHTML, (ii) the speech output via VoiceXML [7], (iii) a simple custom format for spatial audio notifications, (iv) a component to subscribe to the messages

Table 2: Life-cycle events between IMs and MCs (adapted from [10]).

Field	Type	Description
Source	URI	The origin of the event.
Target	URI	The destination of the event.
RequestID	UUID	A unique identifier for the request to be included in the response.
Data	Unspecified	Application and event specific data.
Context	UUID	A unique identifier for the overall interaction context.
Content	Unspecified	Encoded control or presentation document.
ContentURL	URL	The URL where to get the control or presentation document.
Status	Enumeration	`Success` or `Failure`
StatusInfo	Unspecified	Additional status information.

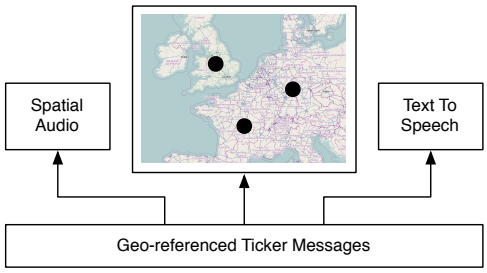

Figure 3: Demonstrator for an application of the W3C MMI architecture.

and deliver them as events and a central root interaction manager to coordinate the modalities (see figure 4).

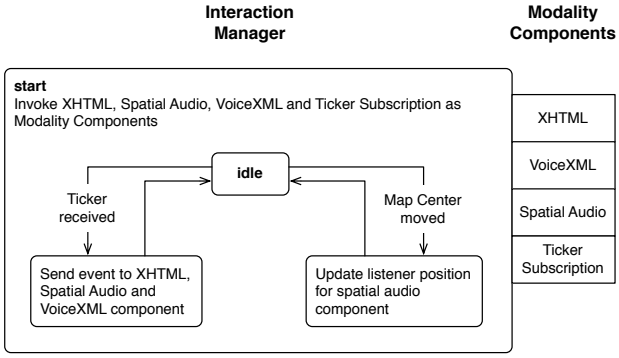

Figure 4: Single root IM and its MCs in the demo.

The single interaction manager is described by a SCXML document with a single compound state `start`, in which all the external modality components are invoked before the nested state `idle` is entered. Within the `idle` state, two transitions can be taken, both leading back into the `idle` state with executable content attached. Whenever the map in the XHTML component is moved, the respective MC will return an extension notification event, causing the SCXML interpreter to send a new position to the spatial audio component to update its listener position. Whenever the ticker subscription component receives some geo-referenced messages, all other modality components are notified to, respectively, (i) place a marker on the map in the XHTML browser, (ii) play an audio notification in the spatial audio component and (iii) speak the message's text content via the VoiceXML browser.

2.1 SCXML Interaction Manager

SCXML as a W3C recommendation is still in the draft status and there are only a few implementations available. Most of the existing implementations take advantage of the existing XML and DOM facilities of modern XHTML browsers and are modeled as documents embedded in XHTML. This is insufficient to deploy SCXML as an Interaction Manager in the W3C MMI architecture as it limits the capabilities of such an implementation. For instance, it is not possible to invoke browsers for other presentation documents as Modality Components or receive requests from these.

In light of these limitations, we implemented our own SCXML interpreter in C++ and made it available as open source[1]. It is standard compliant as far as the SCXML draft requires, runs on desktop as well as mobile devices and features (among others) the ECMAScript datamodel. Different types of invokable (modality) components can be provided as plugins and we implemented the Interaction Manager for the demo application using this interpreter.

The SCXML draft does not specify what it means to actually instantiate a modality component in its role as an Interaction Manager. In fact, the SCXML draft does never reference the W3C MMI architecture directly - it is the W3C MMI recommendation that proposes SCXML as a description language for Interaction Managers. The SCXML draft does specify an interpreters ability to *invoke* components with e.g. VoiceXML as an example, suggesting that the language feature of *invoke* is to be used to instantiate modality components. Nevertheless, the approach to actually send MMI events is unspecified. They could be created by embedding the respective XML markup constituting the individual events or by providing a convenience layer on top of the events native to SCXML. For the demo application, we choose the latter approach in a rather pragmatic fashion, which will ultimately lead to interoperability issues as the transformation from native SCXML to MMI events is unspecified.

To start the application is to run the SCXML interpreter with the root controller document. As our SCXML implementation runs on desktops, as well as mobiles, this enables us to deploy the applications (or parts thereof) on a mobile device. The exception is the VoiceXML modality component, which has to run on a desktop system as we could not find an implementation available for mobile devices.

2.2 XHTML Modality Component

[1] `https://github.com/tklab-tud/uscxml`

There is a plethora of XHTML compliant browsers available and the standard, along with its ECMAScript extensions received a lot of attention in the last years. XHTML is the de-facto standard to deliver platform independent graphical user interfaces. In the past there were a few approaches to extend its graphical capabilities to include other modalities such as speech, voice recognition and even some limited support for spatial audio [11]. While we do think that the approach to model multimodal applications in SCXML and embed XHTML for graphical user interfaces is ultimately a better solution, it is nevertheless, very important to reuse the technological investments that went into XHTML browsers.

As such, every Interaction Manager needs a close integration with XHTML modality components. The foremost problem when employing an off-the-shelf XHTML browser as a modality component is their apparent inability to receive external events once a document has been loaded. By using asynchronous XML HTTP requests (XHR) as part of the standardized ECMAScript implementation in such a browser it becomes, nevertheless, possible to deliver external events via an idiom that came to be known as *comet* [9]: An XHR is issued towards the interaction manager by the XHTML browser as part of the delivered document but not replied to unless there is some data to be send. Whenever data arrived, the same XHR is re-issued, keeping the XHTML constantly asking for data from the Interaction Manager without polling. Which, in essence, achieves the same effect as delivering data into a running XHTML session.

This technique is employed in our demo application to receive ticker messages in the XHTML modality component to be placed on a map and to send the updated map position once the user moved the displayed map.

2.3 VoiceXML Modality Component

For the VoiceXML Modality Component, we employ the JVoiceXML implementation. Its latest version already accepts a subset of the MMI events as defined in the W3C MMI architecture recommendation, making it rather easy to send and receive such events. Our SCXML interpreter features a `vxml` type for the *invoke* element, which will send VoiceXML documents embedded in the SCXML document or referenced via URL in a StartRequest to an implementation running on a desktop.

In earlier work, we already described an approach to enable the output of JVoiceXML to be rendered on any device found an environment [8]. This is not utilized for now, but we do plan to integrate the text-to-speech output with a mobile device.

2.4 Ticker Subscription Modality Component

The ticker subscription component is a plugin of our SCXML interpreter and employes the uMundo publish/subscribe middleware[2] to receive geo-referenced messages with in a custom format. The messages are published by an external component as a mockup of the processing pipeline as outlined above.

[2] `https://github.com/tklab-tud/umundo`

2.5 Spatial Audio Modality Component

The spatial audio component is, again, realized as a plugin in our SCXML interpreter. It directly accepts SCXML events, to update the listener position and render audio via OpenAL. At the moment, this component will only render a simple clicking noise for every incoming messages from its geo-referenced position relative to the displayed map's center. Ultimately, this component should distinguish between the various classes of messages and render different notifications accordingly.

3. CONCLUSION

Until support for the W3C MMI architecture and its respective life-cycle events is more common, application developers still have to rely on kludges and work-arounds to realize and integrate modality components. Employing the architecture as defined in the W3C MMI architecture in an actual application still leaves many unspecified gaps, e.g. how to map SCXML events onto MMI events or, when embedding the markup in the SCXML document, how to fill the dynamic parts with regard to the SCXML datamodel.

Nevertheless, the overall conception and composition of an application as a set of interaction managers and modality components seems more suited to model multimodal applications than the continued overloading of XHTML to take on ever more responsibilities.

4. REFERENCES

[1] J. Axelsson, C. Cross, H. W. Lie, G. McCobb, T. V. Raman, and L. Wilson. XHTML+Voice Profile 1.0, W3C Note. http://www.w3.org/TR/xhtml+voice/, Dec. 2001.

[2] J. Barnett, R. Akolkar, R. Auburn, M. Bodell, D. C. Burnett, J. Carter, S. McGlashan, T. Lager, M. Helbing, R. Hosn, T. Raman, K. Reifenrath, and N. Rosenthal. State chart XML (SCXML): State machine notation for control abstraction. W3C working draft, W3C, Feb. 2012. http://www.w3.org/TR/2012/WD-scxml-20120216/.

[3] M. Bondell, D. Dahl, I. Kliche, J. Larson, B. Porter, D. Raggett, T. Raman, B. H. Rodriguez, M. Selvari, R. Tumuluri, A. Wahbe, P. Wiechno, and M. Yudkowsky. Multimodal Architecture and Interfaces. W3C recommendation, W3C, Oct. 2012. http://www.w3.org/TR/2012/REC-mmi-arch-20121025/.

[4] D. Harel and M. Politi. *Modeling Reactive Systems with Statecharts: The Statemate Approach.* McGraw-Hill, Inc., Aug. 1998.

[5] J. A. Larson, T. Raman, D. Raggett, M. Bodell, M. Johnston, S. Kumar, S. Potter, and K. Waters. Multimodal Interaction Framework, W3C Note. http://www.w3.org/TR/2003/NOTE-mmi-framework-20030506/, May 2003.

[6] S. McCarron, M. Ishikawa, and M. Altheim. XHTMLâĎć 1.1 - Module-based XHTML - Second Edition, W3C Recommendation. http://www.w3.org/TR/2010/REC-xhtml11-20101123/, Nov. 2011.

[7] M. Oshry, R. Auburn, P. Baggia, M. Bodell, D. Burke, D. C. Burnett, E. Candell, J. Carter, S. McGlashan, A. Lee, B. Porter, and K. Rehor. Voice Extensible Markup Language (VoiceXML) Version 2.1, W3C Recommendation. http://www.w3.org/TR/voicexml21/, June 2007.

[8] S. Radomski and D. Schnelle-Walka. VoiceXML for Pervasive Environments. *International Journal of Mobile Human Computer Interaction*, 4(2):18–36, 2012.

[9] I. Russell. Toward server-sent data w/o iframes. `http://infrequently.org/2005/08/toward-server-sent-data-wo-iframes/`, Aug. 2005. [Online; accessed 02-April-2013].

[10] D. Schnelle-Walka, S. Radomski, and M. Mühlhäuser. Jvoicexml as a modality component in the w3c multimodal architecture. *Journal on Multimodal User Interfaces*, pages 1–12, 2013.

[11] G. Shires and H. Wennborg. Web Speech API Specification, W3C Community Group Final Report. `https://dvcs.w3.org/hg/speech-api/raw-file/tip/speechapi.html`, Oct. 2012.