Two New Modalities for ARIA

Martin Franke, Jens Voegler, and Gerhard Weber

Technische Universität Dresden
Institut für Angewandte Informatik
D-01062 Dresden

Martin.Franke4@mailbox.tu-dresden.de
{jens.voegler,gerhard.weber}@tu-dresden.de

http://www.inf.tu-dresden.de

Abstract. Accessibility of rich internet applications is ensured by ARIA and assistive technology supporting this additional mark-up. We propose MM-ARIA for the integration of multiple types of assistive technology while supporting multimodal interaction. The Dojo toolkit's calendar widget is an example of a complex, non-standard widget requiring ARIA mark-up. MM-ARIA generates grammars for speech recognition and integrates gestural input for navigation in the calendar or selecting an item. MM-ARIA relies on multimodal mark-up language EMMA and an interpreter for parallel input. Our proof-of-concept application demonstrates the extensibility of widgets described by ARIA.

Keywords: Browser, speech, gesture, assistive technology

1 Introduction

Rich Internet Applications become more important for new websites. Their advantage over plain forms is that dynamic content can be presented to the user without loading a new web page, such as explaining the missing information in a partially completed contact form, or refreshing a news ticker automatically. It is also possible to use novel types of interaction, for example gestures for navigation, or moving an item directly within a map. Additional modalities beyond mouse and keyboard such as speech input are on the horizon as new browser support access to the microphone.

But often these advantages are not accessible for disabled users or create additional barriers. For example assistive technologies like screenreaders just represents a snapshot of a website and recognizes changes if the website is refreshed or a new site is loaded. Gestures and interaction like drag and drop requires a visual feedback and are time-dependent which may be a barrier for blind and motor-disabled users. Another disadvantage is that gestures are not consistent among different websites and browsers.

Accessible Rich Internet Applications (ARIA) is a markup language for developing accessible website for assistive technologies. The main goals of ARIA are [1]:

– expanding the accessibility information that may be supplied by the author,
– requiring that supporting host languages provide full keyboard support that may be implemented in a device-independent way, for example, by telephones, handheld devices, e-book readers, and televisions,
– improving the accessibility of dynamic content generated by scripts, and
– providing interoperability with assistive technologies.

With ARIA, websites such as Facebook become by and large accessible, but navigation for blind users, to name one user group, is currently only possible by keyboard. This paper will show how ARIA can be used to develop multi-modal websites which can be controlled by speech, keyboard and mouse jointly. Solving this problem requires to interpret ARIA attributes as a semantic enhancement of HTML elements while ensuring accessibility for people who benefit from the availability of one or multiple of these modalities.

2 State Of The Art

The main goal of ARIA is improvement of accessibility for dynamic web content. Dynamic changes can have inappropriate effects for users, which cannot use the mouse or have limited access to the keyboard. The worst case for these users is an inoperable state in a web application. ARIA defines states and properties of HTML-elements and landmarks for navigation of the website [1]. These new attributes for HTML elements may be accessed by Assistive Technology (AT), such as a screen reader. Each AT implements a separate but accessible user interface based on the extra mark-up such as:

– Roles for types: *button, dialog, gridcell, menuitem, menu, scrollbar, tree, tab, tabpanel*
– Roles for structure: *banner, navigation, main, search*
– Properties for showing the state of one object like pressed, or its value

Additional markup requires developers to manually enhance their code. Existing HTML and JavaScript will hardly by modified, but new JavaScript frameworks and toolkits may be developed with ARIA support from the beginning on. They are deployed with a set of widgets, already tested against ARIA specifications and often with AT or even by users relying on AT. One of this frameworks is the Dojo toolkit.

2.1 Dojo Toolkit

The Dojo Toolkit is a JavaScript framework, developed by the Dojo Foundation since 2004 and is currently in version 1.6. It is class-oriented and modularized in widgets. Each widget consists of HTML, CSS and JavaScript files. There are a wide variety of widgets available for professional internet applications. Most widgets of them are evaluated to be accessible, these include basic widgets like buttons, sliders or validating input fields, and also complex widgets such as: *Calendar, Dialog, Editor, InlineEditBox, Menu, MenuBar, ProgressBar, TabPanel, Toolbar, Tooltip and Tree* [2]. Within the Dojo toolkit, a life cycle manager controls the widgets and a data connector simplifies exchange of data between the widgets and an application.

Building a flexible, accessible rich internet application based on multiple widgets requires to establish well controllable input facilities while supporting dynamic changes of the status of widgets. As in any graphical user interface, is the flow of control in Dojo based on events. The next step towards a multimodal system is the collection of events to prepare the synchronization of new modalities for interaction with ARIA-based widgets.

2.2 Collecting Events

Most of the events cause by user input can easily be caught by using JavaScript. jQuery[1] is one of the most used frameworks for this purpose. With this framework one can easily catch keyboard inputs, such key pressed, or mouse input, such as clicks or the position the mouse cursor. Via an additional script [3], it is also possible to recognize mouse gestures (see chapter 3.2). A limitation of jQuery is the lack of handling microphone input and speech recognition. None of the actual browsers is able to natively catch the stream of data arising from microphone input. However, HTML5 specifications include such a feature and soon industry standard browsers will support audio input [4]. One approach for adding speech processing to browsers is the Web-Accessible Multimodal Applications framework (WAMI) [5]. WAMI applications are not built with accessibility in mind, there is also no use of WAI-ARIA. Still, a switch among modalities is feasible. City Browser, is a sample implementation using WAMI with actual multi-modal interaction. It combines mouse and speech input within a browser [9].

WAMI provides an implementation of a client-server concept catching microphone streams in the client and recognition of speech input in the server. The client is implemented as a Java-applet and therefore nearly platform- and browser-independent. The applet records microphone data as an MP3 file and sends it to a server. This server checks the recorded stream against a predefined grammar and sends the recognized words as an JavaScript object back to the client application. With this object functions are triggered on the client side. A WAMI server can be deployed on a Java Servlet Container, like Apache Tomcat[2] or jetty[3], and uses an underlying speech recognizer, for example CMU Sphinx[4] or MIT's own City Browser speech server.

To summarize, we have presented approaches for additional mark-up based on WAI-ARIA to get an accessible RIA, the recognition of the user standard

[1] http://www.jquery.org
[2] http://tomcat.apache.org/
[3] http://jetty.codehaus.org/jetty/
[4] http://cmusphinx.sourceforge.net/

input through events, and an approach for handling speech input in browsers. Multimodality may be embedded into widgets at this point, however we aim at a modular approach using mark-up to model multimodality independently of specific modalities while preserving accessibility features of widgets. One well-known approach to multimodality recommended by W3C is EMMA - the Extensible Multimodal Annotation Language.

2.3 EMMA

EMMA is W3C Recommendation since 2009 with the main goal to establish multimodal interactions in the world wide web. It is a descriptive XML-based language for processing multimodal input. An EMMA message consists of the underlying modality specific mark-up embedded within EMMA-specific mark-up. The main concepts are the interpretation- and the group-element. An interpretation element describes the input modality and consists of [6]:

- id, unique identification
- tokens, the whole input string
- process, reference to the processor, like speech recognizer
- no-input, true, if input string is empty
- uninterpreted, true, if error in processor occurred
- signal, reference to the interpreted file, like the MP3 file
- media-type, media-type of the signal
- confidence, confidence of the right interpretation from 0.0 to 1.0
- source, reference to the application, which invoked the processor
- start, end, duration, points of the start/end of the event and the duration
- medium, acoustic, tactile or visual
- mode, modality of the signal, like voice, dtmf, gui, keys, video,...
- function, function of the signal, like recording, transcription, dialog, verification,...
- verbal, true or false
- cost, cost of the interpretation, such as CPU time, and
- dialog-turn, placeholder for application-dependent implementation

EMMA thereby supports a wide range of modalities and enables flexible interpretations without imposing restrictions on the temporal granularity.

The group element is an container element and consists of interpretation elements belonging together in a temporal sense, typically they are considered as being parallel. With this element the application knows, which modalities should interpreted as a whole and which remain independently. Our multimodal time model for recognizing originates from [7] and can be summarized as follows. There are two types of synchronization of modalities possible, overlapped and non-overlapped events. Overlapped events are events that are risen at the same time. Non-overlapped events are sequential events , but still belong to one multimodal interaction. Overlapped events are synchronized automatically, non-overlapped events are processed according to the following synchronization rules. Thes rules are based on the assumption of some maximum duration of user input. We consider for the purpose of this work only intra-gesture, extra gestures may be added nevertheless.

1-3s	SGI, speech command + intra-gesture
	GIGI, two intra-gestures
5s	SS, 2 speech commands
	GIS, intra-gesture + speech command

Fig. 1. Temporal model based on [7]

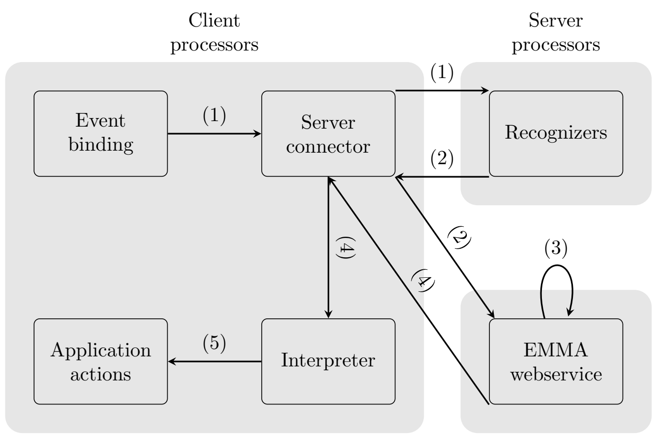
Intra-gestures, are described as any non-verbal tactile event, which belongs to the previous or next event. Every event outside of these time intervals are extra-gestures, and therefore interpreted independently. The interaction sketched in table Fig. 1 are multimodal and integrate two or more modalities based on the temporal model. An example following the rules in Fig. 1 may start with a speech command. If the next event (such as a intra-gesture) starts within 3s after the end of the spoken input, it will be grouped as two emma:interpretions in one emma:group element.

This approach to a temporal model independent of modalities and widgets lays the foundations for Multimodal Accessible Rich Internet Applications (MM-ARIA). We consider an application to be an MM-ARIA, if multiple and alternative modalities can be utilized when interacting with a web application, while assistive technologies are supported.

3 Implementing MM-ARIA

MM-ARIA is based on the modular extension of a client-server architecture. The client application consists of JavaScript files binding events and connecting with the server. The server is application independent and consists of recognizers and a web service to generate EMMA documents. Fig. 2 gives an overview of the individual parts and modules as well as the flow of control.

An event together with its event data, such as a keyboard event, or a spoken command is sent to the server connector (1). The Server connector routes the event data to the intended Recognizer. A Recognizer converts the event data to a json-object and sends it back to the Server connector (2). Only the Server connector needs to know the endpoint of the EMMA web service, prepares the json-object and sends it to the EMMA web service. This web service waits for upcoming events according to the temporal model (3), combines them to an EMMA document and sends it back to the server connector (4). The connector routes the EMMA document to the interpreter, which interprets the document and triggers the appropriate application actions with the parameters determined during processing the event (5).



(1) event data
(2) json object
(3) wait
(4) emma document
(5) operation name and parameters

Fig. 2. MM-ARIA application

We apply the following ‘algorithmic’ procedure for developing the actual mapping between modalities and widgets described by ARIA mark-up:

1. search for roles
2. determine for typical interactions
3. make a mapping from roles to interactions, like enhancement of gestures and speech grammar
4. determine for contextual interactions and attributes, like buttons and aria-labeledby
5. make a possible enhancement in gestures and grammar
6. develop multimodal approaches to simplify the interactions

A more elaborated example is described in the following discussion of dijit-calendar. This example demonstrates the procedure and also the shortcomings of using WAI-ARIA.

3.1 Multimodal Interaction with Calendar Widget

Dijit-calendar consists of one table with the *role='grid'*. Calendar days are tagged with *role='gridcell'* and the weekly header with *role='columnheader'*. As there is no WAI-ARIA role for a calendar, the calendar must be interpreted as a special kind of a table. Usual interactions on a table are selection of one item to insert, edit or delete it. Selection may be performed by navigating forward or backward once from one table item to another item. We need to develop algorithmically a mapping between WAI-ARIA roles and such an interaction. For dijit-calendar, we developed a mapping between *role='grid'* and gestures such as swipe left and right as well as spoken commands. This mapping matches well with the effect of selecting next and previous items. Similarly we map *role='grid'* with the possibility to mark an item and insert a new one, edit the actual one or delete it. Appropriate spoken commands are *edit this* or *delete this.* In a multi-modal system users may want to refer to an element 'this' also by the mouse or keyboard while saying 'this'.

Multi-modal operation on dijit-calendar links swipe gestures and spoken commands. The spoken command *year* followed by a swipe gesture will increase or decrease the year respectively. Similar *month* followed by the the same swipe gesture will change the month shown.

More complex mappings are generated from WAI-ARIA properties of buttons or drop down lists by identifying the elements and their corresponding *aria-labelledby* element. Fig. 3 shows the result of the analysis of other sample Dojo widgets.

Widget	Roles	Gestures	Speech Commands
calendar	grid, gridcell, column-header, row	swipe	delete this, edit this, this row, this cell, this column, sort by, button
dialog	dialog, button	-	minimize, maximize, close
inlineeditbox	button	-	-
menu	menu, menubar, menuitem	-	-
tree	tree, tree-group, treeitem	hitch	collapse all / this, expand all / this
landmark roles	navigation, search	-	-

Fig. 3. Outcomes of wigdet analysis

This findings are examples and the most practical used roles of WAI-ARIA. The table Fig. 3 can simply been extended with roles, gestures and grammars in case of the generic system.

3.2 Building the grammar

One goal of our approach is to enhance ARIA with speech control and inparticular WAMI WAMI requires a grammar, modeled in JSGF syntax. Our analysis shows that control elements are defined in two different ways. On the one hand WAI-ARIA roles are set (see Fig. 3). And on the another hand elements like images can be used as buttons. These buttons have a predefined function on the click and also a descriptive name in the *aria-labelledby* attribute. Therefore the grammar is generated in two ways, a static and a dynamic one. For the static approach a script extracts the role-attribute and adds the results to the grammar. Possible commands are e.g. *this cell* or *this row* or *this column* for setting the focus and *delete this* or *edit this* for the selection of a an item. The dynamic method searches for special words in the role-attribute like button or navigation in the DOM of the page. In this way menuitems, navigation lists and many more can be analyzed and added to the grammar. Combining both approaches within a context grammar increases the accuracy of speech input recognition up to nearly 100% [8]. Every time the user focuses on an element, the grammar changes. Only some elements, such as items belonging to navigation commands are part of the grammar at any time.

From the user's point of view, it becomes possible to *click* the button in the focused element without moving the mouse or using some keyboard controls and easily enhance the application with speech recognition.

3.3 Enabling gesture recognition

Gesture recognition is included by only one JavaScript file. ARIA roles as listed in Fig. 3 will be interpreted either internally by this gesture recognition engine or externally on a recognition server. A gesture starts by holding the right mouse key and is followed by drawing some shape. We aim at gestures which may be mapped to keyboard input and avoid direct manipulation. Our gestures are concatenations of *up, right, left, down.* Thereby it becomes possible to simulate the gestures by pressing the control-key and use the arrow keys.

Based on speech and gesture recognition, as well as mouse and keyboard events, MM-ARIA converts all events to the EMMA format. Based on this format it is possible to trigger and execute multimodal operations on the client application.

3.4 Generating EMMA

The EMMA generator is implemented as a RESTful web service, which takes the events and combines them according to the EMMA-XML format. The data are stored in an INMEMORY database for the calculation the response. This is efficient since after an event has been received and the operation has been executed, the event becomes invalid and never needs to be reused. The structure of the database is similar to the EMMA overview from chapter 2.3 with the following changes:

– process: url of the client website invoking the web service
– signal: url of the interpreter
– source: session_id of the user
– dialog-turn: id of the DOM-element, involved in the interaction

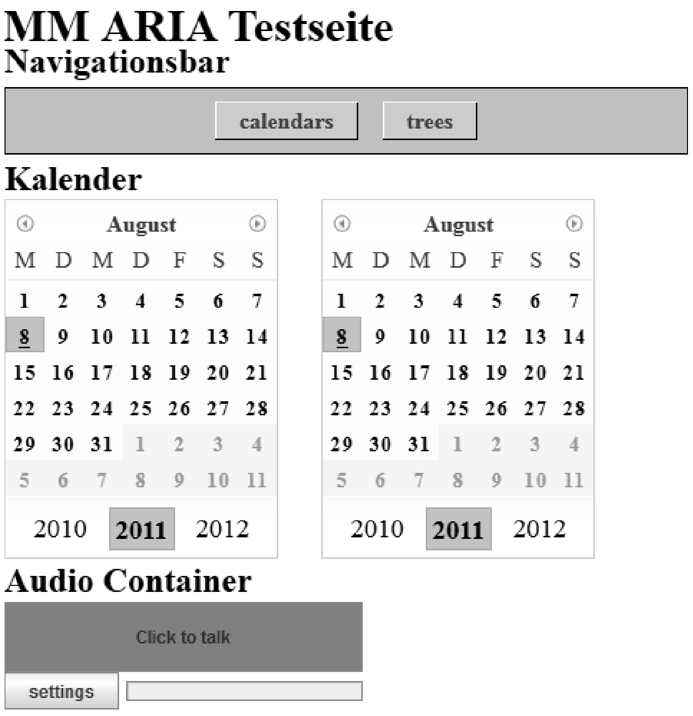
The receiving process of the web service is divided into two stages. This is necessary to measure the time accurately. In the first stage, the initial data describing the kind of event will be sent from the client to the server. The callback at this stage is the Id of the database table entry and the start time as deterined by the server. During the user input the recognizers interpret the data and send the interpreted data to the server with the id of the callback for further aggregation and interpretation. The integration of events according to the time model (see Fig. 1) generates some trigger preparing control of the widget.

The server triggers the client fully asynchronously through the use of COMET. In our implementation of COMET, client and server open a channel for sharing asynchronous messages without beeing forced to adhere to the plain HTTP request-response-pattern. Through this channel the server can be exact, based on the time model, when sending synthesized EMMA document to the client. Finally, the EMMA document must be interpreted by the client and triggers operations.

3.5 MM-ARIA prototype

Our MM-ARIA prototype demonstrator consists of a client and a server part. The server part consists of recognizers and the EMMA web service. The client part contains ARIA mark-up generated by Dojo. It is automatically inspected and enhanced by JavaScript files for the connector and one for interpreter. These connectors are the implementing the multi-modal design as described in previous sections, and consist of grammars for speech processing, gestures and WAI-ARIA event binding.

An interpreter has been developed taking all user input into account and which identifies the proper feedback within dijit-calendar. In other words, it triggers the appropriate operations for various combinations of events. Below the calendars a button named *Click to talk* is placed. The user can start the speech recognition by pressing ⟨shift⟩+⟨ctrl⟩, future implementations of speech recognition engines may avoid such explicit triggering. Button *settings* opens the configuration menu for the microphone and the volume of the audio input. In the textfield to the left of the button the command recognized is shown. The visual functionality of our MM-ARIA demonstrator is presented in Fig. 4 by two calendars, two trees and one navigation bar.



August						
M	D	M	D	F	S	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4
5	6	7	8	9	10	11
2010	2011	2012				

August						
M	D	M	D	F	S	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4
5	6	7	8	9	10	11
2010	2011	2012				

Fig. 4. MM-ARIA Prototype

4 Conclusion

Our work shows that ARIA may be expanded with additional modalities like speech and gestures by using JavaScript, WAMI and EMMA. Existing ARIA-elements are interpreted and a grammar for speech recognition is generated by JavaScript based on ARIA and HTML tags. The two new modalities are described by EMMA for multimodal interpretation. We analyzed some Dojo widgets and developed sample multimodal operations.

A major advantage of this approach is that only JavaScript is required on the user's computer. The result is a prototype of the calendar and tree which can be controlled by standard interaction like keyboard and mouse, but also with speech commands and combined commands based on speech and gesture, such as saying delete this while pointing on a element.

Future works should analyze the requirements of integrating multiple types of AT. For example it is unclear if the mapping between keyboard input and gesture controlled interaction is appropriate for motor impaired users. It is also important to consider how temporal dependencies within multimodal interaction, e.g. speech commands and gesture, can be represented to the users in order to allow for sufficient time (and in accordance with WCAG). One possibility is to further the standardization of spoken and gestural interaction in RIA.

References

1. W3C: Accessible Rich Internet Applications (WAI-ARIA) 1.0, `http://www.w3.org/TR/wai-aria/`, (2011)
2. The Dojo Foundation: Dijit Widgets, `http://dojotoolkit.org/reference-guide/dijit/index.html`, (2010)
3. Adrien Friggeri: jQuery Gesture Plugin, `http://random.friggeri.net/jquery-gestures/`, (2010)
4. W3C: HTML, `http://www.w3.org/TR/html5/`, (2010)
5. Alexander Gruenstein, Ian Mcgraw and Ibrahim Badr: The WAMI Toolkit for Developing, Deploying, and Evaluating Web-Accessible Multimodal Interfaces (2008)
6. W3C: EMMA: Extensible MultiModal Annotation markup language, `http://www.w3.org/TR/emma/#s3.1`, (2010)
7. Rainer Wasinger: Multimodal Interaction with Mobile Devices: Fusing a Broad Spectrum of Modality Combinations (2006)
8. Alexander Gruenstein, Chao Wang, and Stephanie Seneff, Context-Sensitive Statistical Language Modeling, Proc. Interspeech, 2005.
9. WAMI Toolkit: City Browser, `http://web.sls.csail.mit.edu/city/`, (2006)