Publication III

Mikko Honkala and Mikko Pohja. 2006. Multimodal interaction with XForms. In: David W. Wolber, Neil Calder, Christopher H. Brooks, and Athula Ginige (editors). Proceedings of the 6th International Conference on Web Engineering (ICWE 2006). Palo Alto, California, USA. 11-14 July 2006. New York, NY, USA. ACM. Pages 201-208. ISBN 1-59593-352-2.

Multimodal Interaction with XForms

Mikko Honkala
Helsinki University of Technology, TML
P. O. Box 5400, FI-02015 HUT, Finland
mikko.honkala@tml.hut.fi

Mikko Pohja
Helsinki University of Technology, TML
P. O. Box 5400, FI-02015 HUT, Finland
mikko.pohja@tml.hut.fi

ABSTRACT

The increase in connected mobile computing devices has created the need for ubiquitous Web access. In many usage scenarios, it would be beneficial to interact multimodally. Current Web user interface description languages, such as HTML and VoiceXML, concentrate only on one modality. Some languages, such as SALT and X+V, allow combining aural and visual modalities, but they lack ease-of-authoring, since both modalities have to be authored separately. Thus, for ease-of-authoring and maintainability, it is necessary to provide a cross-modal user interface language, whose semantic level is higher. We propose a novel model, called XFormsMM, which includes XForms 1.0 combined with modality-dependent stylesheets and a multimodal interaction manager. The model separates modality-independent parts from the modality-dependent parts, thus automatically providing most of the user interface to all modalities. The model allows flexible modality changes, so that the user can decide, which modalities to use and when.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Voice I/O; Graphical user interfaces (GUI)*; I.7.2 [**Document and Text Processing**]: Document Preparation—*Markup languages*

General Terms

Design, Experimentation, Standardization, Languages

Keywords

Multimodal Interaction, UI Description Language, XForms

1. INTRODUCTION

The diversity of Web applications, and devices, which are used to access them, is increasing. Additionally, new usage contexts bring new kinds of requirements for Web applications. Using multiple modalities (e.g., aural and visual input and output) simultaneously can help to fulfill these requirements. For instance, Web applications could be used with voice input and output while driving a car, and a keyboardless device could use voice input to enhance a graphical application's input capabilities. Studies have also shown that new and complex tasks are solved faster with the use of multiple modalities [4]. Additionally, several countries have enacted legislation, which require public Web services to be accessible to people with disabilities [9][1].

The problem is that current approaches to authoring multimodal Web applications lack ease-of-authoring. They concentrate on one modality at the time. For instance, HTML+ECMAScript approach focuses on the visual UI, and it has well-known accessibility issues when used with speech tools, while VoiceXML is only intended for voice usage. Two main multimodal Web UI approaches, SALT [22] and XHTML+Voice (X+V) [1], allow to author and deliver the same user interface for both speech and GUI, but require each modality to be authored separately, and thus they limit ease-of-authoring. The solution is to raise the semantic level of the user interface description.

The main research problem was efficient authoring of multimodal user interfaces in the Web context. The research was conducted in two phases. First, a literature study was done, in order to derive requirements for a multimodal UI language. Second, the requirements were converted into an design, which was then implemented as a proof-of-concept. This approach was then compared with SALT and X+V based on the requirements.

As a result, we propose to use XForms as the user interface description language, because of its modality-independent nature. We define a multimodal UI model called XForms Multimodal (*XFormsMM*), which includes XForms 1.0 and modality-dependent stylesheets as the UI description language and a definition of a multimodal interaction manager. This solution allows filling and submitting any XForms compliant form simultaneously with aural and visual input and output. A proof-of-concept implementation was created and integrated into X-Smiles XML browser. Two use cases were developed to demonstrate the functionality of the implementation, including input and output of several datatypes, as well as navigation within a form. One of the use cases was also compared to an equivalent application realized with X+V.

The paper is organized as follows. The next two Sections give background information about the topic and introduces the related work. The research problem is discussed in Section 4. Section 5 introduces the design of XFormsMM, while Section 6 describes our implementation of it. Use case implementations are shown in Section 7. Section 8 compares XFormsMM to other approaches, discussion is in Section 9 and, finally, Section 10 concludes the paper.

[1]W3C WAI Policies, http://www.w3.org/WAI/Policy/

2. BACKGROUND

In a multimodal system, the user communicates with an application using different modalities (e.g., hearing, sight, speech, mouse). The input can be sequential, simultaneous, or composite between modalities. Moreover, uses of multimodality can be either *supplementary* or *complementary*. Supplementary means that every interaction can be carried out in each modality as if it was the only available modality, while complementary means that the interactions available to the user differ per modality. Finally, one of the key concepts is the interaction manager, which handles the dialog between the user and the application. [10]

As a modality, speech is quite different from graphical UI. The main difference is the navigation within the UI. In the visual modality, more information and navigation options can be represented simultaneously, while speech interface has to be serialized. On the other hand, spoken command input allows intuitive shortcuts. It has been shown an efficiency increase of 20-40 per cent using speech systems compared with other interface technologies, such as keyboard input [11]. Although speech interaction seems to be promising, the potential technical problems with speech interfaces may irritate the user and reduce task performance [18]. That is why speech is often used by combining it with other modalities to offset the weaknesses of them [5]. Especially, GUIs are often combined with speech modality.

Basically, there are three different approach to realize speech interface: command, dialog, and natural language based. Human-computer interaction through speech is discussed in [2]. The basis of the paper is that perfect performance of a speech recognizer is not possible, since the system is used, e.g., in noisy environments with different speakers. Therefore, dialog and feedback features are important to a recognizer because both human and computer may misunderstand each other. In conclusion, a speech interface should provide adaptive feedback and imitate human conversation. Natural language speech user interfaces are not necessary better than dialog and command based, and it has been shown 80 % of the users prefer a mixture of command and dialog based user interface over a natural language user interface. [13]

XForms 1.0 Recommendation [6] is the next-generation Web forms language, designed by the W3C. It solves some of the problems found in the HTML forms by separating the purpose from the presentation and using declarative markup to describe the most common operations in form-based applications [3]. It is possible to create complex forms with XForms using declarative markup, thus not resorting to scripting.

XForms form controls, grouping constructs and event types are abstract and generic. Therefore, it suits describing multimodal user interfaces. Raman [17] lists several features in XForms, which are of use when creating multimodal UIs. A recent study compared XForms with other abstract user interface description languages, and found it, along with the AIAP language, to best suit their universal access requirements [20].

3. RELATED WORK

Simon et al. have researched the same area of authoring multimodal Web applications [19]. They focus on creating an authoring environment, but they also summarize that a cross-modal UI language, which provides *single-authoring* for multiple modalities is the best approach. They conclude that XForms is well suited to act as this language. There are also other proposals to realize multimodal interaction.

Raggett and Froumentin propose extensions to Cascading Stylesheets (CSS) to allow simple multimodal interaction tasks [15]. The paper defines new CSS properties, which add aural modality to current HTML applications. In contrast to our proposal, the aural UI must be created separately from visual UI through the CSS extension.

XHTML+Voice profile [1] is based on W3C standards and is used to create multimodal dialogs with visual and speech modalities. In addition to XHTML Basic and modularized subset of VoiceXML 2.0 [12], it consists of XML Events module, the XHTML scripting module, and a module containing a small number of attribute extensions to both XHTML and VoiceXML 2.0. XHTML is used to define layout of the graphical content of a document, whereas VoiceXML adds voice modality to the document. XML Events provides event listeners and associated event handlers for the profile. Finally, the attribute module facilitates the sharing of multimodal input data between the VoiceXML dialog and XHTML input and text elements. VoiceXML is integrated into XHTML using DOM events. When an element is activated, a corresponding event handler is called. The event handler synthesizes the text obtained from the element that activated the handler. Also, a speech input to an XHTML form field is assigned through VoiceXML elements. XHTML+Voice documents can be styled both by the visual and aural style sheets.

Speech Application Language Tags (SALT) is an XML-based language for incorporating Speech input and output to other languages [22]. It can be embedded, for instance, in XHTML documents. The speech interaction model is dialog-based. SALT uses a the notion of subdialogs, which can be added to a page. SALT does not have explicit data model, but it binds to external data models, such XHTML forms, using ids. It has quite low-level programming model, thus allowing authors to fully control the speech dialog interface.

4. RESEARCH PROBLEM AND SCOPE

The research problem was to study efficient authoring of multimodal interfaces for the Web. The scope of this work is Web applications in general, with special focus on aural and visual modalities. The application domain contains medium-interaction applications, which would benefit from flexible user-defined modality changes.

4.1 Use Cases

Example applications include a *route planner* (public transportation and car routes), a *text message sending service*, and a *car rental service*. A common denominator in these services is the requirement of mobility and different usage contexts. For example, the applications could be used in the following contexts:

1. At the office. The user uses the car rental service to book a car for a holiday trip. Then she wants to know how to get to the car rental pick-up using public transportation. She uses mainly visual modality to interact with the route planner. The route planner saves the query.

2. Walking. While walking to the bus, the user realizes that they need a bigger car, so that the holiday skiing equip-

ment would fit in. She logs in to the car rental service using her mobile phone. Using speech input combined with visual output, she changes the car preference and notices that the pick-up point changes. She logs in to the route planner, and inputs the new pickup address using voice input. Finally, she uses the text message service using speech input to notify her husband that she will be picking up the rental car.
3. Driving a car. The user plans the route while driving, utilizing the onboard multimodal internet browser. She interacts purely with voice to select the holiday destination (the startpoint is filled in by her GPS receiver). She uses visual output for the map of the route and aural output for the driving instructions.

4.2 Requirements

We used the W3C Multimodal Interaction Requirements [10] to design our system. The requirements are summarized below:

General Requirements. The integration of modalities should be seamless. Easiness to author, use, and implement a multimodal language are required. Finally, requirements include also accessibility, security, and flexible delivery context related issues.

Input Modality Requirements. It is required that modality related information can be authored. The input should be able to be given sequentially, simultaneously, and combined from several modalities. Also, temporal positioning of input events should be available.

Output Media Requirements. Output must be able to be represented both sequentially and simultaneously in different modalities.

Architecture, Integration, and Synchronization points. Compound documents with already existing languages should be preferred. Also, the specification should be modular. Data model, presentation layer, and application logic should be separated. The language should provide means to detect or prescribe the available modalities.

Nichols et al., derive another set of requirements for automatically generating multimodal remote control user interfaces for home appliances and services [13]. Those requirements differ from the W3C requirements, which are used in this paper. The most notable difference is whether the possibility to author modality-dependent information is explicitly required (W3C [10]) or disallowed (Nichols et al. [13]). Our and W3C's focus is enabling multimodal usage of Web applications, which are typically expected to be visually appealing. The other difference is that W3C do not have the requirement for two-way communications, since that is not typical for Web applications, and works against the request-response model of the Web.

Trewin, et.al. present another set of requirement from the point of view of universal access [20]. Their requirements are compatible with the W3C requirements, albeit on higher level and less detailed.

5. XFORMS AS THE MULTIMODAL UI LANGUAGE: XFORMSMM

We propose to use XForms as the abstract UI description language for multimodal applications. Our approach, XFormsMM, combines XForms as a *Multimodal Application Authoring Format* with *Modality-Dependent Stylesheets* and a description of an multimodal *Interaction Manager*.

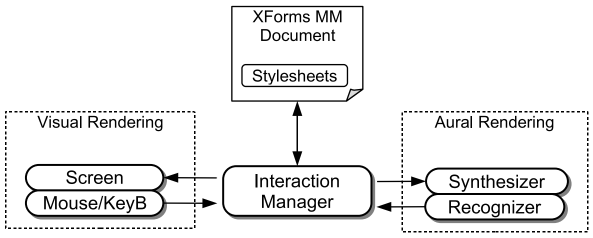


Figure 1: XFormsMM Execution Environment.

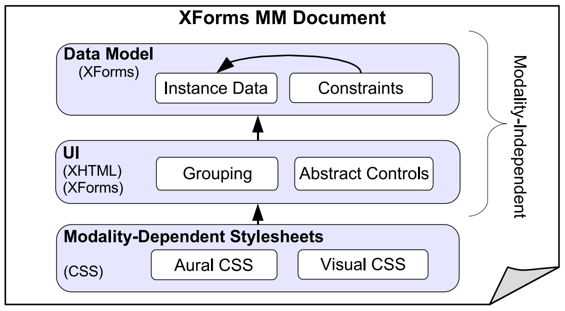


Figure 2: The XFormsMM Authoring Format.

The execution environment is depicted in Figure 1, where the interaction manager interacts with the XFormsMM document (i.e., the live DOM representation of it) and synchronizes the Visual and Aural Renderings so that simultaneous multimodal input and output and flexible modality switching are possible.

5.1 Authoring Format

XFormsMM user interfaces are authored in the XForms 1.0 language. Hence, non-speech-enabled user agents will be able to present the user interface in the visual modality. This is depicted in Figure 2, where the document is separated into modality-independent and dependent portions. The arrows represent the direction of dependencies. The modality-independent portion includes the XForms Data Model and the abstract UI. The data model is further separated into instance data and constraints, while the UI contains grouping and abstract form controls. XHTML 1.0 is used as the host language. The main modules used from XHTML are: Structure, Stylesheet, Hypertext, and Image Modules, while the Forms module is replaced by XForms. The modality-dependent portion contains stylesheets.

There are some limitations on authoring XFormsMM. First, using hierarchical grouping when defining the user interface is essential. Similar result can be found in [13]. If the grouping is only based on coordinates and font styles, the interaction manager will not be able to create a decent speech-only navigation. Second, free-text input fields and textareas should be avoided because of the limitations of the current automatic speech recognition (ASR) engines. Third, the number of simultaneously available items should be reduced to the minimum. This can be achieved with a multi-page layout (e.g., a wizard or tabs), instead of presenting all items simultaneously.

5.2 Modality-Dependent Stylesheets

Supplementary use of modalities [10] means that every interaction can be carried out in each modality as if it was the only available modality. This is provided in XFormsMM by default. Every user interaction is defined using the XForms User Interface Module [6], and can be interacted with either speech or GUI.

On the other hand, *complementary use of modalities* [10] means that the interactions available to the user differ per modality. In XFormsMM, this is provided by the use of modality-dependent CSS (cf. Figure 2). For the visual modality, CSS 2.1 is used, so for instance, setting a CSS value of the property *"display"* to *"none"*, effectively removes the interaction from the visual modality. For the aural modality, CSS Voice [16] is used, and e.g., setting the property of *"speak"* to *"none"*, removes the interaction from the aural modality.

Notice that it is possible to dynamically change these CSS properties by using the XForms pseudo-classes in the CSS selectors. For instance, the following CSS code will remove read-only address from the voice interaction and navigation, while still leaving it to GUI:

```
#address:read-only  {display:block; speak:none;}
```

The read-only property itself is defined using an XPath statement, and it can change during the form-filling, based on the users input.

5.3 Multimodal Interaction Manager

The main roles of the XFormsMM interaction manager are synchronization of the modalities, flexible switching between the modalities, and navigation. It also receives events from the modality-dependent rendering subsystems.

The interaction manager for XFormsMM supports mixed-initiative dialog capabilities. The main idea behind the interaction manager is that aural and visual states are synchronized automatically. Setting the focus in one modality automatically transfers the focus also in the other modality. Since the modalities share a single XForms data model, all data items and their constraints are also automatically shared.

For the aural modality, it is necessary that the interaction manager analyzes the document for available focusable objects, and use that information to create feasible speech-only navigation of the user interface. The interaction manager distinguishes between four main types of speech-focusable objects: branch containers, toggles, input fields, and output fields. Branch containers include elements such as *xforms:group*, and *xhtml:div* which are logical container containing other elements. Toggles are interactors, which have only simple activation function, such as *xhtml:a* link and *xforms:trigger*. Input fields include all XForms form controls, while output fields have many manifestations, such as *xforms:output*, *xforms:label*, *xforms:help*, and *xforms:alert*. Note, that an input field can become an output field dynamically, based on XForms model item properties (i.e., constraints), such as *readonly*.

The interaction manager searches the focus points of a document and holds the current focus point. Elements, which do not have a label, have to be analyzed heuristically for possible label. Since the documents can change dynamically (e.g., via XForms constraints or dynamic UI bindings), the interaction manager must at each point re-evaluate the document for speech-focusable objects.

The interaction manager has two main procedures, MainLoop, which waits for events, and SearchFocusPoints, which searches for the next available aural focus points in a document. Both are described below:

Procedure MainLoop waits for any user or document events, and synchronizes the different modalities based on the current focus point type and the event.
Operation: For each document or user event E:

```
1. If E = document.load:
    a) Initialize G to general options ("help", "browser reload", "browser back", "back", "next", etc.).
    b) set FP=searchFocusPoints(document.documentElement).
       set FA = first aurally focusable in FP.
       set FV = first visually focusable in FP.
2. If E = value changed in widget W from modality M:
    a) Set new value through the XForms processor.
    b) Notify renderers in all modalities that new value has been set for W.
3. If E = focus in for widget W from modality M:
    a) if W is aurally available, set FA = W.
       if W is visually available, set FV = W.
    b) set FP = searchFocusPoints(FA).
4. If E = document change event:
    a) if a new message or alert element became available, present it in all modalities.
    b) if FA is no longer aurally available, set FA to next aurally available focusable element and set FP = searchFocusPoints(FA).
    c) if FV is no longer visually available, set FV to next visually available focusable element.
5. If E is a general command in G: do the requested navigation function and move both FA and FV accordingly to the new node N. Set FP = searchFocusPoints(N).
6. VISUAL: show document with the current focus point FV visualized.
7. AURAL: if FA not widget, present current options: FP+G. If some options are outputs, present also current value. Otherwise present the widget FAs prompt.
```

Procedure SearchFocusPoints(Node root) Gets as input a node in the current DOM, and returns the list of possible speech-focusable objects from that node.
DEFINITION: Node N is speech-focusable only if it is of focusable type (input, output, branch, toggle) and the *speak* CSS property of N is set to other than *none*.
Operation:

```
1. Initialize NodeSet F to root
2. Do a depth first search starting at root. For each node N:
    (a) If N is not speech-focusable or contains only one speech-focusable branch child node, ignore that node, and continue depth-first
    (b) Otherwise include N in F and terminate that branch of depth-first search
3. return F
```

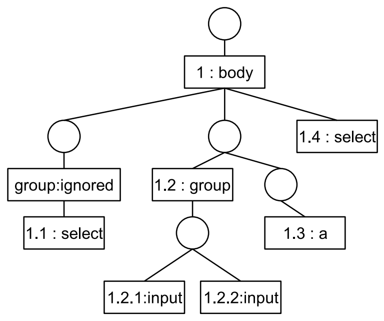


Figure 3: Focus points of a document.

A notable difference between the visual and aural navigation and focusable objects is that in aural modality, branch elements are considered explicitly focusable.

The forms are usually hierarchically grouped, so the navigation has to be possible both to siblings and to parent and children. To imitate a real dialog, user agent must be able to ask user a question, which relates to a current focus point, and repeat user's answer to ensure that the answer was correctly recognized. After repeating the answer, the interaction manager moves the focus to next logical position and waits for the user's action.

An example is given in Figure 3, which depicts a small document, which has been analyzed for focusable objects. In the figure, the possible focus points from point *1* are *1.1*, *1.2*, *1.3*, and *1.4*. Note that the branch items with only one choice are ignored (e.g., ancestor of *1.1*).

5.4 Rendering

Each modality has it's own rendering subsystem, whose main roles are outputting the current state to the user and receiving modality-dependent events from the user. The rendering takes care of detailed events (such as mouse movements in visual modality or partial speech recognition events in aural modality), and provide higher-level events (such as value changed and focus changed) to the interaction manager. This is depicted in Figure 1.

6. IMPLEMENTATION

XFormsMM and the multimodal interaction manager were implemented on top of an open-source XHTML+CSS+XForms implementation, X-Smiles [21]. The components of the implementation are depicted in Figure 4.

6.1 Architecture

The XForms engine [8] provides the state of the document. Interaction manager is a thin layer, which communicates between the XForms engine and the GUI and speech rendering subsystems (cf. Figure 4). It implements the Main Loop algorithm described in Section 5.3. The GUI rendering is provided by the CSS Layout Engine [14]. Speech rendering was implemented in this study, and it consist of a *dialog handler*, *speech widgets*, a *focus provider*, and a *Speech API*. The dialog handler uses speech widgets and focus provider to provide a speech UI navigation and a form field filling. Speech widgets are created for each element according to the datatypes, and they store the input data to a form in correct format. The Speech API is a low-level interface for

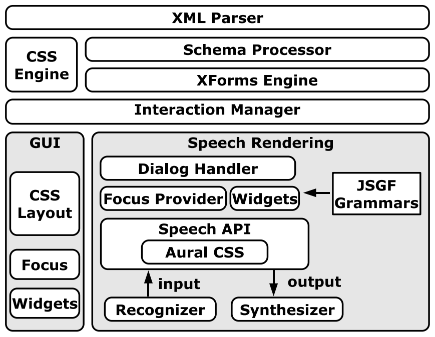


Figure 4: The components of the XFormsMM implementation in X-Smiles.

speech recognition and synthesis. Sphinx-4[2] was used as the speech recognition engine and FreeTTS[3] as a speech synthesis engine, while the Speech API allows an easy way to plug in different ASR and TTS engines.

6.2 Dialog Handler

The Dialog Handler resides on the topmost level of the Speech Rendering. It generates aural rendering for the current focus point. It also receives the voice input and determines which object should handle it. Additionally, it implements XForms alerts and navigation within repeating structures.

When requested by the interaction manager, the dialog handler creates a question or command list and delivers it to the speech synthesizer. The question consists of label of an element and possibly the selections that can be made. The selections can be, for instance, navigable focus points or items of a selection input. For all the questions or command lists, the dialog handler must create a new grammar for the recognizer. JSGF[4] grammar format was used, since that is what Sphinx-4, and Java-based tools in general, support. The grammar consists of possible replies user can give and, in addition, navigational commands (e.g., back). The reply part is received from a corresponding speech widget and navigational commands are added by the dialog handler.

6.3 Focus Provider

The visual focus management is already provided by the graphical XForms implementation [8]. Hence, the interaction manager registers itself as a listener for graphical focus events. In addition, a speech focus provider was created. The focus provider implements the algorithm SearchFocusPoints for traversing the DOM tree and searching for possible focus points (this algorithm was described in detail in Section 5.3). Additionally, it provides functionality to find the parent focus point, and to store the current focus points.

6.4 Speech Widgets

The speech widgets are the aural counterpart of the visual form controls. Their responsibilities include providing a grammar about possible replies for dialog handler and

[2]Sphinx-4, http://cmusphinx.sourceforge.net/sphinx4/
[3]FreeTTS, http://freetts.sourceforge.net/docs/index.php
[4]Java Speech Grammar Format, http://java.sun.com/products/javamedia/speech/forDevelopers/JSGF/

parsing the user's reply and passing it to a corresponding element in correct format. Creation of a grammar depends on a form control. Certain types of control have a constant grammar (e.g., date and numbers). In that case, the grammar is fetched from a file. If possible selections are defined in a form, the grammar must be formed according to those. When the user replies, the dialog handler receives the reply from the recognizer and delivers it to a corresponding widget. The user's answer is returned to a widget as a string, and the widget parses it and interprets to a corresponding element in correct format. Finally, the widget informs the interaction manager, that the focus should be moved.

7. USE CASES

Two use cases from Section 4 were selected and implemented as a proof-of-concept of the model. The first one, a multimodal short message service, was compared with a similar use case implemented in X+V. The second one is a more realistic Web car rental user interface with a wizard-type of visual interface.

7.1 Multimodal Short Message Service

We implemented a multimodal SMS sending example, which already had implementation in X+V[5]. We were able to realize same features in 72 lines of code (2.3 KB), whereas the X+V implementation takes 261 lines including grammars (9.3 KB), which are generated automatically in XFormsMM. In X+V, the UI has to be partially defined twice, i.e., for both modalities separately. In addition, the X+V implementation requires a lot of scripts.

We have also extended the original example. First, we added possibility to edit messages in visual modality through keyboard. Second, the application automatically calculates and represents the number of characters left for a message. It also alerts the user when the number of characters exceeds the maximum. Screenshot of the SMS sending example is shown in Figure 5.

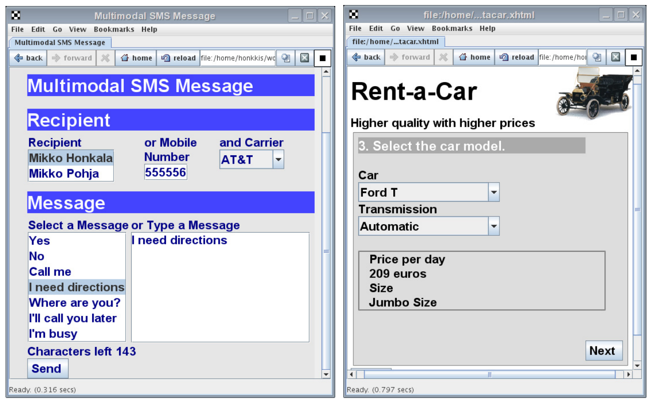
7.2 Multimodal Car Rental

The second use case is a more realistic car rental Web user interface. The graphical user interface of the system is shown in Figure 5. The form is divided to several pages through *xforms:switch* element. There are different input types in the document. Pick up location, car model, and transmission can be selected from a predefined list, dates are calendar type inputs, and number of child seats must be input as integers. Finally, there are triggers to submit and navigate the document. A possible voice-only dialog between the system and the user is shown below. Note that at any point, the user might use the pointing device to focus or interact in the visual modality, which would also affect the speech dialog flow accordingly.

System: 1. When to pickup and return the car. Pickup Date. Return Date. Number of days. Next. Back.
User: return date
System: Return Date October 12, 2005. Select a date.
User: august nineteen
System: Alert: The return date must be after the pickup date! Return Date : August 19, 2005. Select a date.
User: october nineteen

[5]X+V-based SMS sending, http://www-128.ibm.com/developerworks/wireless/library/wi-send.html

System: Return Date October 19, 2005. Select a date.
User: back
System: Back. 1. When to pickup and return the car. Pickup Date. Return Date. Number of days. Next. Back.
User: next
System: Next. 2. Location of pickup and return.. City. Next. Back.
User: next
System: Next. 3. Select the car model.. Car. Transmission. Next. Back.
User: transmission
System: Transmission .Automatic. Select from: "Automatic", "Manual", "Semi-automatic"
User: manual
System: Transmission . You selected: Manual
3. Select the car model.. Car. Transmission. Next. Back.
User: next
System: Next. 4. Optional extra selections.. Child seats. Luggage Racks. Next. Back.
User: next
System: Next. 5. Check your reservation.
Pickup Date October 12, 2005, Return Date October 19, 2005
Number of days 6
The car model Ford T, Transmission Manual
Rate 1194 euros
Confirm. Cancel. Back
User: confirm
System: Confirm. Thank for your reservation!



Multimodal SMS Message
Recipient
Recipient: Mikko Honkala, Mikko Pohja
or Mobile Number: 555556
and Carrier: AT&T
Message
Select a Message or Type a Message
Yes
No
Call me
I need directions
Where are you?
I'll call you later
I'm busy
I need directions
Characters left 143
Ready. (0.316 secs)

Rent-a-Car
Higher quality with higher prices
3. Select the car model.
Car: Ford T
Transmission: Automatic
Price per day
209 euros
Size
Jumbo Size
Ready. (0.797 secs)

Figure 5: Screenshots of the use cases.

8. COMPARING XFORMSMM TO OTHER APPROACHES

We compared our approach, XFormsMM, with two other approaches, X+V and SALT. The W3C Multimodal Interaction Requirements are used as the criteria [10]. This section lists only those requirements, which are fullfilled differently by the approaches.

8.1 General Requirements

Supplementary and complementary use of different modalities: XFormsMM is clearly better at *supplementary interaction*, since all interaction is always automatically available in both modalities, and the user can, at any point, switch between the modalities. On the other hand, *complementary interaction* is possible with all three.

Seamless synchronization of modalities: XFormsMM enabled seamless synchronization between the modalities, but only at the field-level. Both SALT and X+V allow synchronization at all levels, but they require scripting.

In both *Multilingual support* and *Accessibility*, XFormsMM has clear benefits over the others. Every label can be easily internationalized (@ref on labels), and accessibility is one of the benefits of XForms over XHTML (labels for all form controls, specific elements for help, hint, and alert).

In *Easy to implement*, there are big differences. *Ease-of-authoring* is best in XFormsMM, where only single UI definition is required, compared to authoring both aural and visual parts in SALT or X+V. On the other hand, *Ease-of-use*, when using only aural modality, is better in SALT and X+V, since their aural expressive power is greater.

The *Delivery and context* requirement is not very well met by any of the languages. There might be additional tools on top of any of the languages, which would transform the document based on the context. For this approach to be successful, the abstraction level should be high. XFormsMM has the highest abstraction level, X+V the second highest, while SALT has the lowest abstraction level. For non-automatic (i.e., authored) approaches, a solution would be a set of DOM events to notify about context changes.

For *Navigation specification*, all specifications support navindex, or similar for the visual part. For the aural modality, SALT and X+V support very detailed navigation specification, while XFormsMM counts on single definition with navindex, grouping, and modality-dependent CSS.

8.2 Input Modality Requirements

In *Input processing*, X+V and SALT both rely on modality-dependent strategies, such as voice grammars. In XFormsMM, cross-modality definitions, such as data and input types are used. This means, that the author has more control on input processing in X+V and SALT, but XFormsMM is much easier to author and maintain. *Sequential multimodal input* is supported by all of the three languages. *Simultaneous multimodal input* and *Composite multimodal input* are also supported, but X+V and SALT provide more author control, while similar effects can be achieved with modality-dependent CSS in XFormsMM. *Semantics of input* is best provided by XFormsMM, with data- and input types. Also *Coordinated constraints* is best provided by XFormsMM, by sharing a single structured datamodel (along with declarative constraints and calculations) between modalities.

8.3 Output Media Requirements

All three languages meet output media requirements similarly, although SALT provides best synchronization of output events by the use of a special *Prompt Queue*.

8.4 Architecture, Integration and Synchronization Points

Since we focus on client-side implementation, some of the requirements, such as *Distributed processing* are not taken into account. From this set of requirements, we found only two, which differ between the languages. First, *Separation of data model, presentation layer and application logic*, is properly supported only by XFormsMM. *Synchronization granularities* is best supported by SALT and X+V, since XFormsMM only supports field- and event level synchronization.

9. DISCUSSION AND FUTURE WORK

During the study, it became obvious that XForms has clear advantages over XHTML forms in multimodal application authoring. The main advantages of XForms are strong datatyping, better accessibility, and less reliance on scripting. The better accessibility is due to the higher semantic level of the elements. For instance, alert, hint and message provide more information compared to generic script, which modifies the document through the DOM API, and mandatory label on groups and form controls is better than a heuristic search of possible label. Similarly, datatypes can be utilized very well to generate speech grammars. Also, XForms supports dynamic changes in the UI based on user input. This means that the same instance of the form can stay at the client longer, thus retaining the speech focus point.

We also noticed that creating a usable speech UI requires grouping of the UI elements. That is, the UI elements must be divided into the groups in order to reduce possible choices, from which user can select at a time. The finding is equivalent to the one represented by Nichols et al. [13].

The weakness of using pure XForms to describe multimodal interfaces, is that the exact behavior of voice dialogs is left to the interaction manager. More work is required to study the possibility of creating shortcuts in the navigation sequence. Also, some author control of speech input and output is needed. A promising approach is to use sXBL [7] to bind in VoiceXML constructs to create aural-only subdialogs. CSS Voice [16] might also need to be extended to be able to describe better the aural semantics of the UI grouping, and to dynamically control, which elements are included in the aurally focusable elements list at a given time.

The current implementation could be improved and extended. More widget types and a more complete datatype support should be added. More research is needed to verify the usability of the proposed speech navigation and compare it to different approaches. Also, currently the implementation handles only XForms, and support for XHTML should be added.

10. CONCLUSIONS

Enabling the Web user interface authors to write multimodal Web applications is a current problem. Today's web technologies make applications usable only with one modality at the time (e.g., HTML and VoiceXML). We have proposed a novel approach, called XFormsMM, for authoring multimodal applications. It uses XForms as the UI description language, and enables the multimodal use of the applications, focusing on the simultaneous use of GUI and speech.

There are several accessibility features in XForms, which are of use when creating multimodal UIs. The main advantages of XForms are strong datatyping, better accessibility, and less reliance on scripting.

W3C Multimodal Interaction Requirements were used to evaluate our proposal against two other approaches, X+V and SALT. The main difference is the semantic level, which is higher in XFormsMM. This results in better ease-of-authoring, since only one UI description needs to be written. Also, synchronization between the modalities is automatic in XFormsMM, and scripting is not required for most user interfaces.

XFormsMM has a built-in interaction manager, which provides navigation and form filling in different modalities. It also provides automatic synchronization between the modalities. The interaction manager suits most use cases and simplifies the authoring of the user interfaces drastically. In the other approaches, the interaction manager needs to be re-programmed by the author for each application.

On the other hand, our proposal has less control over the voice modality and synchronization granularity. Use of modality-dependent stylesheets in XFormsMM is good way of providing complementary use of different modalities.

As a conclusion, XFormsMM best fits usage scenarios, where the author does not know which modalities for input and output are present. The user can freely choose between any of them. On the other hand, in scenarios where the author needs a complete control on modality switches, the proposed solution is not optimal.

We have presented an implementation of XFormsMM, which is now part of the open-source X-Smiles XML browser. Also, in the paper, two use cases were developed to demonstrate the functionality of the implementation. The first one is a multimodal short message service, which required less than third of the code lines, compared to a similar application written in X+V. The second use case is a wizard-type of car rental Web interface. Both demonstrate input and output of several datatypes and complex navigation within a form.

11. ACKNOWLEDGMENTS

The authors would like to thank Dr. Pablo Cesar for in-depth comments about this paper. Also, we would like to thank Alessandro Cogliati, the developer of the CSS engine, and Petri Vuorimaa, the leader of the research group. The author Mikko Pohja would like to thank Nokia Foundation for providing support during the research.

12. REFERENCES

[1] J. Axelsson, C. Cross, J. Ferrans, G. McCobb, T. V. Raman, and L. Wilson. XHTML+Voice Profile 1.2. Technical report, March 2004. Available online http://www.voicexml.org/specs/multimodal/x+v/12/.

[2] S. E. Brennan and E. A. Hulteen. Interaction and feedback in a spoken language system: a theoretical framework. *Knowledge-Based Systems*, 8(2):143–151, 1995.

[3] R. Cardone, D. Soroker, and A. Tiwari. Using XForms to simplify web programming. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 215–224, New York, NY, USA, 2005. ACM Press.

[4] P. Cohen, M. Johnston, D. McGee, and S. Oviatt. The efficiency of multimodal interaction: A case study. In *Proceedings of the International Conference on Spoken Language Processing*, pages 249–252. IEEE, 1998.

[5] P. R. Cohen. The role of natural language in a multimodal interface. In *UIST '92: Proceedings of the 5th annual ACM symposium on User interface software and technology*, pages 143–149, New York, NY, USA, 1992. ACM Press.

[6] M. Dubinko, L. L. Klotz, R. Merrick, and T. V. Raman. XForms 1.0. W3C Recommendation, 2003.

[7] J. Ferraiolo, I. Hickson, and D. Hyatt. SVG's XML binding language (sXBL). W3C working draft, W3C, September 2004. Available at http://www.w3.org/TR/sXBL/.

[8] M. Honkala and P. Vuorimaa. A configurable XForms implementation. In *Proceedings of the IEEE Sixth International Symposium on Multimedia Software Engineering (ISMSE'04)*. IEEE, 2004.

[9] H. Jahankhani, J. A. Lynch, and J. Stephenson. The Current Legislation Covering E-learning Provisions for the Visually Impaired in the EU. In *EurAsia-ICT '02: Proceedings of the First EurAsian Conference on Information and Communication Technology*, pages 552–559, London, UK, 2002. Springer-Verlag.

[10] S. H. Maes and V. S. (eds.). Multimodal interaction requirements. W3C NOTE, 2003.

[11] G. L. Martin. The utility of speech input in user-computer interfaces. *Int. J. Man-Mach. Stud.*, 30(4):355–375, 1989.

[12] S. McGlashan and et al. Voice Extensible Markup Language (VoiceXML) Version 2.0. W3C recommendation, W3C, March 2004.

[13] J. Nichols, B. Myers, T. Harris, R. Rosenfeld, S. Shriver, M. Higgins, and J. Hughes. Requirements for automatically generating multi-modal interfaces for complex appliances. In *ICMI '02: Proceedings of the 4th IEEE International Conference on Multimodal Interfaces*, page 377, Washington, DC, USA, 2002. IEEE Computer Society.

[14] M. Pohja and P. Vuorimaa. CSS Layout Engine for Compound Documents. In *Proceedings of the Third Latin American Web Congress*, pages 148–157, Buenos Aires, Argentina, October 2005. IEEE.

[15] D. Raggett and M. Froumentin. CSS Extensions for Multimodal Interaction. WWW page, 2004. Available online http://www.w3.org/2004/10/css-mmi/.

[16] D. Raggett and D. Glazman. CSS3 speech module. W3C working draft, W3C, July 2004.

[17] T. Raman. *XML Powered Web Forms*. Addison-Wesley, 1st edition, 2003.

[18] B. Shneiderman. *Designing the user interface: strategies for effective human-computer interaction*. Addison-Wesley, 2nd edition, 1992.

[19] R. Simon, F. Wegscheider, and K. Tolar. Tool-supported single authoring for device independence and multimodality. In *MobileHCI '05: Proceedings of the 7th international conference on Human computer interaction with mobile devices & services*, pages 91–98, New York, NY, USA, 2005. ACM Press.

[20] S. Trewin, G. Zimmermann, and G. Vanderheiden. Abstract user interface representations: how well do they support universal access? In *CUU '03: Proceedings of the 2003 conference on Universal usability*, pages 77–84. ACM Press, 2003.

[21] P. Vuorimaa, T. Ropponen, N. von Knorring, and M. Honkala. A Java based XML browser for consumer devices. In *The 17th ACM Symposium on Applied Computing*, Madrid, Spain, March 2002.

[22] K. Wang. Salt: A spoken language interface for web-based multimodal dialog systems. In *Proceedings of the 7th International Conference on Spoken Language Processing (ICSLP'02)*, 2002.