Designing and Executing Multimodal Interfaces for the Web based on State Chart XML

Sebastian Feuerstack, Jessica H. Colnago, César R. de Souza, Ednaldo B. Pizzolato
Universidade Federal de São Carlos
Rodovia Washington Luís, km 235
São Carlos - São Paulo – Brasil
jessica_colnago@comp.ufscar.br, {cesar.souza, sfeu, ednaldo}@dc.ufscar.br

ABSTRACT
The design and implementation of multimodal interfaces that support a variety of modes to enable natural interaction is still limited. We propose a multimodal interaction web framework that considers on the one hand the current W3C standardization activities, such as Start Chart XML and the Model-Based UI Working Group. But on the other hand it implements recent research results enabling the direct execution of multimodal interfaces based on the information of the design models. This paper presents the basic concepts of modeling and executing multimodal interfaces for the web and illustrates our approach by presenting two case studies that have been implemented so far.

Keywords
State Chart XML (SCXML), Multimodal Interfaces, Specification, W3C.

1. INTRODUCTION
Nowadays the web is ubiquitous available. But the access is bound to a personal computer, a smart phone or a tablet pc. The web is controlled by using a mouse, a keyboard or by touching the display. Further modes of control such as using gestures or voice still remain restricted to certain applications. Although promising attempts have been made to bring multimodal interfaces to the web, such as the XHTML+Voice initiative of the EMMA and SMIL languages, there are still frameworks missing that support the design and generation of multimodal interfaces for the web.

In this paper we present the Multimodal Interaction framework (MINT) that enables the model-based design of multimodal interface components for the web and offers a web server-sided platform to execute these models to run a multimodal web application. Our approach differs in the following two aspects from earlier work on the model-driven development of user interfaces (MDDUI) as well as from current approaches to implement multimodal interfaces for the web:

- We focus on the design of a language, which consists of several models that are used to design multimodal interactors and capture specific interaction characteristics of different control modes. Earlier work on MDDUI has been performed to describe semi-automated generation processes to generate entire user interfaces for different platforms whereas we end up with multimodal interactors that can be assembled to form an interface. Therefore, the assembly of an interface by interactor can be performed using the well know user interface builder paradigm instead of abstract-to-concrete model refinements.
- Different to approaches like XHTML+Voice that are executed inside the browser, we synchronize all the modes on the server side by utilizing real-time web technologies such as Web Sockets. This enables us to even synchronize a multimodal application that spans across several devices, as well as to flexibly add further control modes or media later on.

In the next section we discuss the related work regarding these two points. Thereafter, in section 3, we first present the MINT language model that is based on state charts and enable the design of multimodal interactors. Then we describe the MINT platform that enables the execution of the interactors as a component of a web server to serve multimodal interfaces and a tool that a developer can use to design the interactors. In section 4 we present as examples two case studies, a gesture based drag-and-drop enabling to virtually locate furniture of an online shop into an augmented reality scenario, and an interactive music sheet that can be controlled just by head movements. Finally, section 5 states future work.

2. RELATED WORK
Model-driven development of user interfaces (MDDUI) has been applied for a long time to reduce the development costs of user interfaces that should run on different platforms. Languages, such as USIXML [7] and tools such as [11, 9] have been proposed to form a structured development process. These processes typical start from a very abstract description of the user interface tasks, to a more concrete one considering the characteristics of certain modalities (a concrete user interface model) and end up with a final user interface to be executed on a specific platform [4]. Although a structured process has the advantage that it represents an engineering method that can guide a developer from a problem to the solution, the diversity of design models involved requires a substantial learning effort. Additionally, this process through several abstractions requires anticipation skills to understand how changes on an abstract level will be reflected in the final user interfaces [13].

We are using the same model abstractions as initially summed up by the CAMELEON framework [4], but using a development process that conforms to classical user interface construction: Using an user interface builder that offers a palette of widgets (that we call interactors) that can be directly assembled to form a multimodal interface. This is an approach that has been followed

by Gummy [8] as well, but was focused to create different graphical interface and has to our knowledge no support for extending the existing widget set. User interface builders ease the development of interfaces by offering predefined widgets but lack the possibility to add new ones or require the developer to enhance the tool itself on the source code level. Our approach tackles this issue. Therefore, we propose to construct interactors based on State Chart XML (SCXML). SCXML [1] is a general-purpose event-based state machine language that is based on Call Control eXtensible Markup Language and Harel State Tables [6]. It is an easily understandable language with only basic concepts (such as states, actions, transitions and events). This enables a structured design approach to introduce new interactors and to directly execute these interactors, since SCXML can be directly transformed to state machines.

The current most applied language to describe multimodal interfaces for the web is XHTML+Voice [2], which is already supported by tools like the IBM Websphere Multimodal Toolkit. Additionally, initial approaches from the MDDUI community to use the CAMELEON model abstractions to create XHTML+Voice interfaces have been proposed [12]. There are two main disadvantages of these efforts: first, they are currently limited to describe only one multimodal combination (graphical and voice output media with speech control), second, they implement fusion of modes inside the browser. Different to server-sided fusion this limits the distribution of modes and media to include several devices in one interaction.

Our approach is driven by server-side processing. Therefore we utilize recent technologies like Web Sockets as well as CSS3 manipulation to synchronize interfaces running inside the web browser between several devices and for the fusion and fission of different modes and media. This enables us to connect new modes such a gesture or head movement recognition to control the web interface.

3. MINT FRAMEWORK

To tackle the challenge to ease the creation of new multimodal ways of interacting with the web, we propose the Multimodal INTeraction Framework (MINT). It is divided into two parts: First, the language based on designing start-charts to describe the behavior of interactors and multimodal mapping to specify the multimodal fusion and fission mechanisms of them. Second, the runtime environment that runs inside the webserver to synchronize the modes and media utilized in an interaction by the user and follows the basic structure of the W3C multimodal framework architecture [3].

3.1 Language

The recent years a lot of languages for MDDUI have been proposed and their principle model abstractions have been summed up in the CAMELEON framework. But most approaches proposed proprietary languages and focus on the design of multi-platform interfaces. Multimodal interfaces have been considered only to a very limited extend so far. Therefore, we decided to use state charts and class diagrams as the basic design notation, since they are already widely known. Therefore we only add proprietary language extensions to describe the multimodal fusion and fission processes respectively the connections between modes and media.

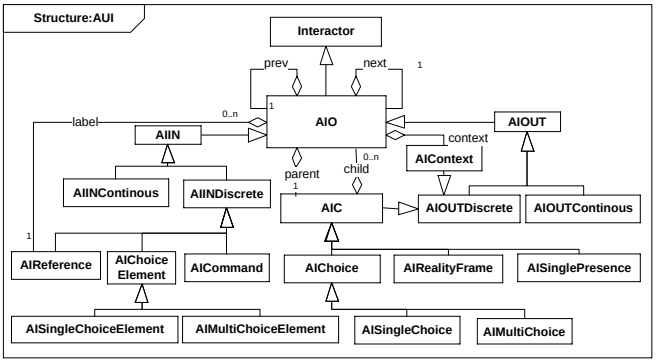

Fig.1. The interactor relations on the abstract level.

Each interactor, that could represent one interface element, such as a button, a list or a menu entry for instance, consists of two model abstractions: An abstract one that captures its behavior independent of mode and media – and several concrete ones to match its representation and the specific capabilities of a certain platform.

The class diagram of figure 1 shows the inheritance relations between the different interactors on the abstract level. An abstract interaction object (AIO) class is the basic abstract class for all other interactors. An AIO could either represent an input related (mode) control (AIIN) or an output related media (AIOUT). Both could represent either a continuously ongoing (such as a progress bar or a slider) or a discrete, static information or action (such as a button or a textual output).

The behavior of an interactor is specified by a state chart and inherited through the interactor's hierarchy shown in figure 1. In the following example we describe the behavior of the AISinglePresence interactor that manages a list of other interactors (it is derived from an abstract interaction container – AIC) and ensures that only one of its element is presented at a certain time). Additionally this interactor is output related and discrete and therefore only presents information to the user (where each of its information could be well distinguished). An example would be for instance a traffic light widget that either presents a green, yellow, or red light to the user. Different to common interactors like a single choice list (AISingeChoice) or a command (AICommand - which might be a button for instance), the AISingePresence interactor has been specifically designed for an application that implements a new form of interaction (which we present later on) and therefore represents the basic idea of our approach to ease the design of new interactors based on models instead of code.

Figure 2 shows the state chart of this interactor. It implements the general life cycle of every abstract interactor that starts in the state initialized, gets organized when it knows about its neighbors (to support navigation between interactors), then presented to the user and finally suspended. What makes the behavior of this interactor specific is that it initially presents just his first child (presenting, on_entry) and enables to change the actual presented child by switching to the next or previous child respectively, but only if the interactor is currently "focused" by the user as well as "entered".

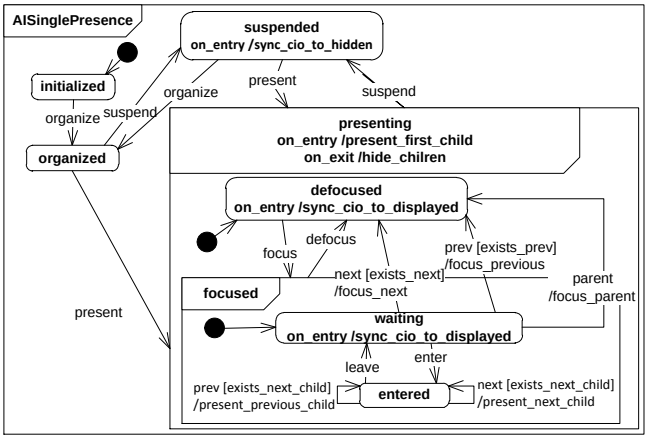

Fig.2. Behavior specification of the AISinglePresence interactor.

The specification of an interactor might look initially complex but it captures the entire behavior so that is can be stored in SCXML and directly executed (see section 3.2), which substitutes writing source code. Further on, the design of interactors is only required when new forms of interactor are designed, whereas a user interface designer can just instantiate these interactors and arrange them using a classical user interface builder tool (which is out of scope for this paper).

For the user to control an interactor, devices such as a keyboard or a mouse are required. They are designed by using state charts in the same way as the interactor above. Since we are interested in modeling new forms of interactions, figure 3 presents a basic interactor that observes head movements for controlling an interface.

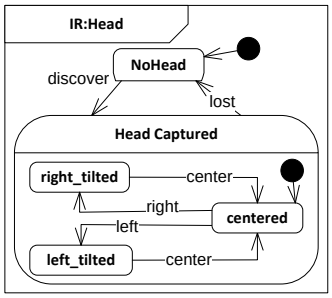

Fig.3. Interactor that describes the head movements.

As soon as the user's head is detected, it distinguishes the head from being centered, left or right tilted. It is stored in SCXML and executed like the AISinglePresence interactor, but is instantiated only once to represent the API of the device that captures the head movements and throws the corresponding events to trigger a state change based on the head movements of the user.

Finally, both the device interactor and the AISinglePresence interactor are connected by multimodal mappings. Figure 4 shows such a mapping that connects both interactors. Mappings consist of boxes with sharp and soft edges. The former ones are used to define function calls (e.g. *right_tilt_cmd*) or events (e.g. *dest.next*), the latter ones to define reactions on state changes of an interactor (e.g. *Head.centered*). In our case the mapping gets triggered as soon as the Head interactor enters the *right_tilted* state. The "S" operator specifies a sequential mapping ensuring

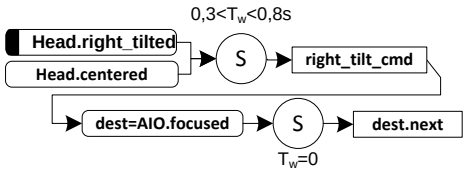

Fig.4. Basic Mapping to combine interactors.

that the commands will occur in the specified order and the T_w condition defines a temporal window in which the operator has to succeed. This means that after the head is found in the *right_tilted* position it must be *centered* during an interval of 300ms and 800ms for the mapping to continue, otherwise the mapping fails and is restarted. Immediately following the head gestured the focused object is selected (the *AISinglePresence* interactor in our use case) and it receives the *next* event.

3.2 Platform

The platform complements the framework by implementing the run-time part enabling the execution of the interactors that have been designed as state charts and instantiated to represent widgets of an application's user interface. Following the idea and the definitions of the W3C Multimodal Architecture and Interfaces working draft (W3C-MMI) [3], the platform runs on the server side as part of the web-server and implements an interaction manager that is connected to several modality components.

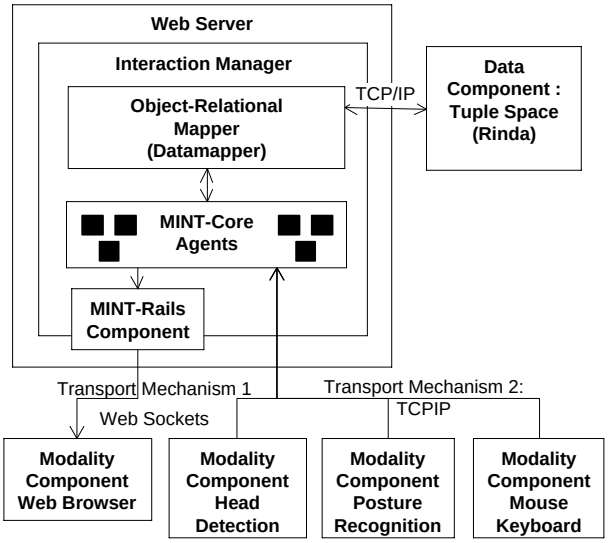

Fig.5. Basic Mapping to combine interactors.

Figure 5 illustrates our approach in relation to the W3C MMI specification. Our overall architecture is identical to the one proposed by W3C MMI. We implemented the optional Data Component as a tuple space and the interaction manager as a three layered architecture. Different to the W3C approach we strictly distinguish between mode and media. Thus media components, such as a web page are pure output components whereas modes implement components that handle user inputs. Therefore, even mouse and keyboard events are synchronized inside the interaction manager instead of the browser. The instantiated interactors and multimodal mappings are managed by a set of reactive software agents inside the interaction manager. Each state

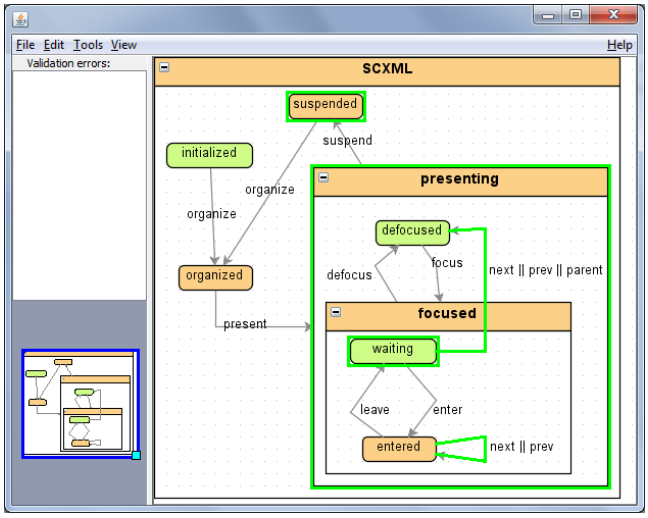

Fig. 6. Interactor design using scxmlgui

change of an interactor is written in the tuple space. The multimodal mappings subscribe for relevant state changes with the tuple space and send events to the interactors to initiate a state change. The agents are written in ruby and contain the functional core of an interactor that can be called with an action defined by a transition or an on_entry or on_exit events such as shown in the interactor of figure 2. Since the software agents communicate only with the tuple space and run in separate threads they can be distributed to tackle performance bottle-necks for cpu intensive calculations, such as a constraint solver for instance that we are using to calculate the user interface layout [5].

3.3 Tool

Even though the interactor design is not a regular task required for designing a multimodal application, the design of new interactors can get complex – especially for interactors that implement parallel running and history states. One tools that supports the state chart modeling with SCXML is the scxmlgui tool [10] shown in figure 6. It additionally supports animation of state changes so that state changes of individual interactors could be easily observed.

4. CASE STUDIES

We have applied our approach to model and execute multimodal interactions in several prototypes and experimented with designing interactors for different medias, such as web interfaces, sound, and augmented reality and modes, like a hand gesture and posture recognition, the Wii controller or by detecting head movements to control an interface so to prove the feasibility of our approach.

4.1 Interactive Music Sheet

During plays, music sheets are used as a guide to perform a musical piece. However, since songs may span across several pages, an extra amount of coordination is necessary to turn pages without disrupting the play. Therefore musicians are required to learn the music by hearth.

For most instruments, moving a page involves leaving the instrument's playing area, which can be a long and cumbersome task if one is not sufficiently used to this action. Some existing

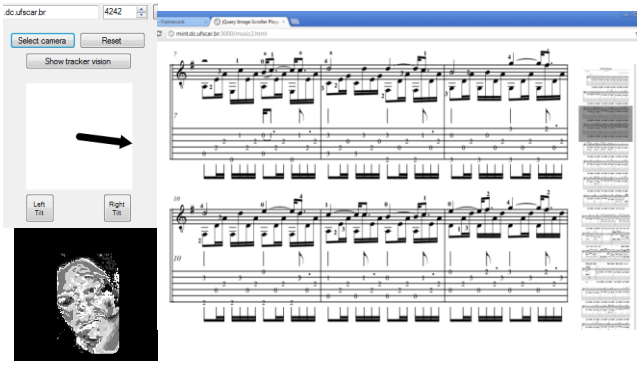

Fig.7. An excerpt of the head movement detection modality component (left) and a web browser displaying the music sheets (right)

solutions for this problem include the use of custom peripheral devices in order to turn pages with the feet. However, these solutions require additional hardware and, due to the extra hardware costs, are targeted to professional musicians.

The basic interactor that implements the music sheet is the AISinglePresence interactor of figure 2. It represents the entire sheet that is divided into several pages between that the user can navigate forward or backward.

While we tested this prototype, we realized that head movements are a suitable way to control the page turns, because as long as the musician has not memorized the music he is required to follow the notes and, therefore, his head remains quite stable while reading.

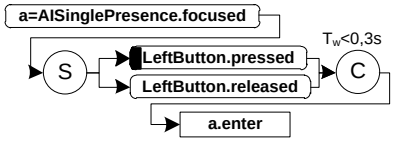

Fig.8. Mapping to start a control by head movements.

The head tracking needs to be started explicitly after the musician has seated to prevent unintended page turns in a noise environment. Figure 8 illustrates a mapping that waits for a button click on the music sheet to start the control by head movements. It uses the complementary operator, C, to define that a sequence of a button pressed and released has to happen in series (because of the sequence operator, S) and in a defined temporal window (Tw) to fire the enter event. This results in the AISinglePresence interactor to react on the head movement events.

4.2 Augmented Drag-and-Drop

For another use case we implemented a prototype of a web furniture shop that supports arranging furniture in an augmented reality environment. For this use case we were interested in figuring out how common interaction paradigms, for instance a drag-and-drop can be modeled with our approach for different modes and media. To arrange the furniture the user had to access a web site that enables to browse and choose between different furniture items and store them in a shopping cart (like depicted in figure 9). Then, the user can drag items out of the shopping cart and, while they are crossing the dashed border of the shopping cart, the web site switches to displaying an augmented scene that

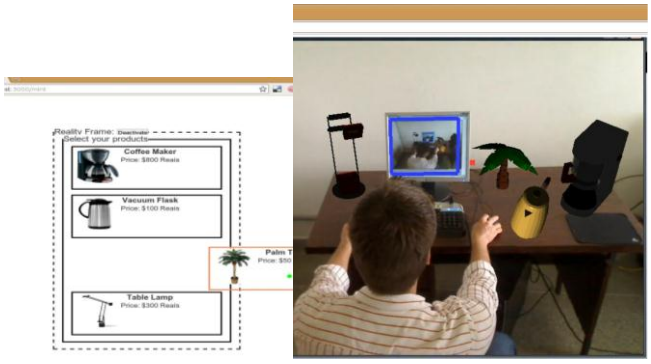

Fig. 9. Augmented Drag-and-Drop between a shopping cart within webpage (left) and the augmented scene (right)

is captured by a cam behind the user and films the surrounding environment of the user. Additionally the dragged item switches from 2D into a 3D representation and can be virtually positioned to see if it fits well in the environment, as shown in the right part of figure 9.

Different to the previous use case the principle interactor is a list (AISingleChoice) with contains elements (AISingleChoiceElement) that can be dragged and dropped. Figure 10 shows the drag-and-drop mapping to be used with a posture recognition modality component. It is initiated as soon as the user shows a (button) "pressed" posture while pointing to a list element (that is in "focus" while the user is pointing to it). If this is the case the focused list element receives a drag event (which enables it to be dragged around) and the mapping waits for the left hand showing a (button) "released" posture to drop the element to the destination that the user is pointing to while showing the "released" posture.

5. FUTURE WORK

We presented a framework to design and execute multimodal interfaces for the web that conforms to and implements actual W3C standards like SCXML and Web Sockets for instance. Furthermore it follows the basic W3C MMI architecture, has been implemented on the server side and has been made publically available for others to proof and enhance our work.

For the future we intend to offer a tool that supports the developer designing multimodal mappings, which currently still needs to be programmed in ruby (different to the state chart design that is already tool supported). Furthermore the current implementation of the platform is already structured as proposed in the W3C MMI specification but currently uses a proprietary TCP/IP communication instead of the proposed message protocol. Therefore we will investigate in how to implement a standardized communication between the interaction manager and the modality components while preserving the communication speed of our current implementation.

6. REFERENCES

[1] State chart xml (scxml): State machine notation for control abstraction, w3c working draft 26 april 2011, http://www.w3.org/tr/2011/wd-scxml-20110426/, last checked august 25, 2011.

[2] Xhtml+voice profile 1.0, W3C, http://www.w3.org/tr/2001/note-xhtml+voice-20011221, last checked august 25, 2011.

[3] Multimodal architecture and interfaces, W3C, http://www.w3.org/tr/2011/wd-mmi-arch-20110125/, last checked august 25, 2011, 2011.

[4] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289–308, 2003.

[5] Sebastian Feuerstack, Marco Blumendorf, Veit Schwartze, and Sahin Albayrak. Model-based layout generation. In Paolo Bottoni and Stefano Levialdi, editors, *Proceedings of the working conference on Advanced visual interfaces*. ACM, 2008. Proceedings of the working conference on Advanced visual interfaces 2008.

[6] David Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274, 1987.

[7] Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, Murielle Florins, and Daniela Trevisan. Usixml: A user interface description language for context-sensitive user interfaces. In *Proceedings of the ACM AVI'2004 Workshop" Developing User Interfaces with XML: Advances on User Interface Description Languages*, pages 55–62, 2004.

[8] Jan Meskens, Jo Vermeulen, Kris Luyten, and Karin Coninx. Gummy for multi-platform user interface designs: shape multiply fix use me. In *Proceedings of the working conference on Advanced visual interfaces*, AVI '08, pages 233–240, New York, NY, USA, 2008. ACM.

[9] Francisco Montero, Vctor López-Jaquero, Jean Vanderdonckt, Pascual González, Mará Dolores Lozano, and Quentin Limbourg. Solving the mapping problem in user interface design by seamless integration in idealxml. In *DSV-IS*, pages 161–172, 2005.

[10] Fabrizio Morbini. Scxmlgui. http://code.google.com/p/scxmlgui/, Last checked August 25, 2011. 2011.

[11] Giulio Mori, Fabio Paternò, and Carmen Santoro. Ctte: Support for developing and analyzing task models for interactive system design. *IEEE Trans. Software Eng.*, 28(8):797–813, 2002.

[12] Adrian Stanciulescu, Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, and Francisco Montero. A transformational approach for multimodal web user interfaces based on usixml. In *ICMI '05: Proceedings of the 7th international conference on Multimodal interfaces*, pages 259–266, New York, NY, USA, 2005. ACM Press.

[13] Jean Vanderdonckt and Pierre Berquin. Towards a very large model-based approach for user interface development. In *UIDIS*, pages 76–85, 1999.