Model-based Design of Multimodal Interaction for Augmented Reality Web Applications

Sebastian Feuerstack[*] Állan C. M. de Oliveira[+] Mauro dos Santos Anjo[+] Regina B.Araujo[+] Ednaldo B. Pizzolato[+]
[*]OFFIS – Institute for Information Technology, Oldenburg, Germany; e-mail: feuerstack@offis.de
[+]Universidade Federal de São Carlos, São Carlos, Brazil; e-mail: {allan_oliveira, mauro_anjo, regina, ednaldo@dc.ufscar.br}

Abstract

Despite the increasing use of Augmented Reality (AR) in many different application areas, implementation support is limited and still driven by development at source-code level. Although efforts have been made to overcome these limitations, there is a clear gap between authoring environments and source code level framework approaches for creating AR interfaces for the web with multimodal control. Model-based design for interaction can offer support to fill this gap between authoring environments and frameworks. However, to the best of our knowledge, a declarative and model-driven design (MDD) has not yet been applied to model AR interfaces for a wide spectrum of modes. Thus, this paper presents an extension of the model-driven design to cope with interactors, whose novelty lies on the introduction of a modeling approach targeted at AR developers and designers in their task to design new forms of interactions that can be later used in authoring environments. To validate our approach, we demonstrate how a reality spanning Drag-and-Drop interaction can be modeled for an online furniture shop. And we implemented a gesture based control to show how new control modes can be added to an existing MDD-based design to extend the interaction capabilities.

CR Categories: H.5.2 [Information Interfaces and Presentation]: User Interfaces - Input devices and strategies, Interaction styles, Prototyping; D.2.2 [Software Engineering]: Design Tools and Techniques – User Interfaces.

Keywords: Augmented Reality, Model-based User Interface Design, Web Interfaces, Multimodal Interaction, Human Computer Interaction.

1 Introduction

Augmented reality (AR) promises to enhance the real, physical world as we see it with additional information. Thus, AR has been increasingly applied across different application areas, from military to education, from entertainment to advertising, and many more. Recent research activities are targeting on evolving standards to support authoring, distribution and consumption of AR content [Hill et al. 2010b]. However, the support for implementing AR applications is still limited - it occurs typically either at source code level, by offering re-usable basic components for marker detection, or at AR scene rendering, by frameworks, such as Junaio, and libraries, such as ARToolkit [Kato et al. 1999].

End user support to AR development can also be based on pre-defined widgets, such as BuildAR [BuildAR. 2015] and KHARMA [Hill et al. 2010a] that relies on HTML.

There is still a gap between a very low level support on the source code level, and end user authoring environments that limit the users` possibilities to create new interfaces to the widgets and components that have been made available in the environment. New features can only be introduced at the source code level of the tool or the framework and require extensive knowledge about their internal structures. Further on, the overarching principle of most of the available tools and frameworks is that they offer components or widgets as black-boxes that could be glued together either on the source code level or by “drawing lines” to connect their public interfaces. More complex mechanisms of component combination to create multimodal interfaces are not supported, such as the ones that require explicit timings to support the fusion of data from different modes. Also, the explicit design of interaction techniques, such as a drag-and-drop, requires a tight connection between the components. Moreover, processing their internals can only be done at source code level.

The model-based design (MBD) of interaction can offer support to fill this gap between authoring environments and frameworks (components can be connected on the source code level). By replacing source code with models, MBD introduces an abstract layer that eases the understanding and the design of interactions. Moreover, costs and time can be reduced by offering structured processes that enable the design of general (abstract) ways of interaction with the user and systematically derive more specific interfaces for different platforms.

In the last two decades the model-driven development of user interfaces (MDDUI) has been successfully applied to the design of multi-platform user interfaces and has been proved to generate, for instance, voice [Stanciulescu et al. 2005], web [Berti et al. 2004] and 3D interfaces [Gonzalez-Calleros et al. 2009]. But, to the best of our knowledge, it has not yet been considered for AR application development.

In this paper we propose the model-driven design of interactors to bridge the gap between authoring environments and source code level frameworks approaches for creating AR interfaces. The novelty of our solution lies on the introduction of a modelling approach based on state charts, to abstract from the code level.

New interaction modes can be easily added and changed by manipulating the declarative models instead of introducing changes at the code level. This is demonstrated by adding a hand gesture and posture control as well as sound feedback to our prototype.

This paper is structured as follows: The next section reviews the related work. Thereafter we introduce our approach for the model-based design of mixed-reality interaction. Then we present an exemplary application, an online furniture shop, that we use in the following section to explain our approach in detail. We present initial tool support for designing new interactors. Experienced problems and future work are described in pre-final section, followed by our final comments and conclusions.

2 Related Work

There are various approaches to support the creation of user interfaces. They can be distinguished by their targeted audience: authoring environments are targeted to high level or end-user development and offer a set of predefined widgets or components that can be assembled by using a tool; One can say that authoring enables a developer to visually develop applications with a tool based on a set of pre-defined components by selecting and connecting components.

Frameworks, libraries or toolkits focus on developing support and organizing a set of components that can be connected and re-used within a programming language. Figure 1 illustrates the different levels of abstractions of current tools and frameworks for AR and Multimodal interface development from the source code level up to end-user authoring tools.

On the lowest abstraction level, libraries, toolkits and frameworks assist developers to create their applications. They offer a set of components that can be re-used in a certain programming language. Common components for AR applications are features like marker tracking, displaying the video output and rendering 3D interfaces or hardware abstractions by drivers e.g. to access a head-mounted display or to access proprietary software like a speech recognition system. Examples include Studierstube [Schmalstieg et al. 2002] or AR/FlarToolkit [Kato et al. 1999], which offer components that can be re-used in C++ or Flash respectively.

On a higher abstraction level, model-based approaches abstract from a concrete programming language and focus on describing the interaction. The basic idea of model-based approaches is to reduce complexity (by the abstraction from the source code view) but still offering enough semantics to enable code generation or direct execution of the design models to form the interface.

Among some interesting approaches, we highlight Chasm [Wingrave et al. 2009]. It is a comprehensive approach to modeling 3D user interfaces based on Concept-Oriented Design

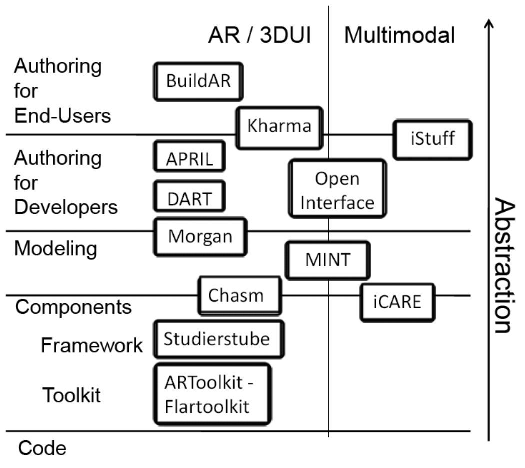

Fig. 1. Abstraction level of tools and frameworks for the development of AR and Multimodal interfaces

(COD). With Chasm, components can be designed and directly executed based on the tiered user interface description language. One of the advantages of this approach is it`s consideration of the practitioner`s side enabling to add a textual description of the desired behavior of components as part of the language. Our approach shares the general idea of multi-tier approach like Chasm, as we support high level and low level abstractions.

Further on, we share the concept of using states, events, transitions and actions for modeling. However, Chasm focus on 3D user interface development for which most of its case studies have been done so far, whereas our approach concentrates on the design and execution of multimodal interfaces (which results in the multimodal mapping definition to design the connection between different modes and media as we will describe in the next section).

Another platform worth mentioning is iCARE [Bouchet et al. 2005]. It supports building multimodal interaction out of components based on the CARE properties. These properties describe the relations between different modes, such as their complementary, redundant or equivalent combination and are inspired by the findings of multimodal theory [Bernsen 2008].

The Morgan framework [Broll et al. 2005], uses a visual tool for modeling interfaces and interactions by assembling interactions and behaviors of objects from pre-defined components. Recent work on this framework included the identification of mechanisms for supporting reusability of interactions and behaviors by using concepts like instantiation, templates, modules and inheritance. Morgan, in this way, can combine modeling and authoring features.

DART [MacIntyre et al. 2004] is an example of AR authoring that is built upon Adobe Director and supports the assembly of pre-determined behaviors defined in a scripting language. Another tool is APRIL [Ledermann and Schmalstieg 2005], for creating AR presentations based on an XML language. With APRIL it is possible to define hardware, content, temporal structure, behaviors and interactions. The Open Interface Framework [Lawson et al. 2009] supports prototyping of multimodal interactions out of components that can be assembled with the help of an authoring environment. A similar approach to prototype new interactions by connecting sensing with output devices has been proposed by the iStuff project [Ballagas et al. 2007].

Finally, on the highest level of abstraction, tools for end-user authoring have been proposed to create AR applications without programming like illustrated by figure 1. One commercial tool is BuildAR [BuildAR. 2015] for instance, that supports creation of AR applications by end-users. KHARMA [Hill et al. 2010a] implements a similar approach, but requires basic web programming knowledge to be used.

Authoring environments and frameworks offer various components to create a user interface, but the implementation of new components or the extension of existing ones is tedious. Often, it has to happen on the source code level using the specific programming language of the tool. It also requires in-depth knowledge about the organizational structure of the framework (to consistently include changes or additions). Documentation for these approaches typically focuses on the API. The overall structures as well as the components' descriptions of their internal structures are often handled as second-class citizens´ tasks.

This contradicts current advances in the interaction design for multimodal and augmented reality applications. New forms of interactions, such as different forms of Drag-and-Drop are continuously proposed and evaluated. For instance Rekimoto's pick-and-drop [Rekimoto 1997], which is grabbing an object in one display and putting it in another, or Rekimoto and Saitoh hyperdragging [Rekimoto and Saitoh 1999], which is dragging 2d objects through displays. A tangible drag-and-drop to configure music streaming to different devices is proposed in [Hopmann et al. 2011]. But these kinds of techniques are complex to add to existing AR toolkits. This is one reason for the great variety of AR toolkits and authoring environment. Another one is the diversity of programming. The variety of VR toolkits and the still missing common API of them has been recently discussed and identified as a limiting factor for the specification of interoperable 3D interaction techniques(3DIT) [Ray and Bowman 2007].

Therefore, an interoperating layer has been proposed to make 3DITs independent from the underlying VRToolkit layer, in order to encourage reusability [Ray and Bowman 2007]. But this approach is focused on the source code-level by standardized callback API and black-boxed components that can be configured based on XML descriptions.

To the best of our knowledge an approach that bridges the gap between source-code level frameworks and authoring tools is still missing. With MINT, the Multimodal Interaction Framework [Blinded] that will be explained in more details in the next section, we propose a model-driven design approach to fill in this gap.

3 Model-based Design of Mixed Reality Interaction

For the design and implementation of mixed-reality interaction we apply a model-driven design of user interfaces (MDDUI) process. This has the advantage that part of the interface semantics (the actual widgets and corresponding interaction), such as for describing commands, lists or selections need to be modelled only once and can be subsequently reused in interfaces for different media (like web interfaces, speech or augmented reality scenes). Actual MDDUI approaches [Calvary et al. 2003] describe these semantics that can be shared between several media by Abstract User Interface (AUI) models. Media specific designs are captured within Concrete User Interface (CUI) models.

MDDUI approaches are typically driven by model-to-model transformations and therefore implement a design process that first describes the interaction on a very abstract level (e.g. by task models) and then continuously refines these models by applying transformations to more concrete ones until they end up with final user interfaces to address several devices.

Although these MDDUI approaches have been proven to work well to generate interfaces for different devices and modes, like speech [Stanciulescu et al. 2005] or 3D interfaces [Gonzalez-Calleros et al. 2009] for instance, they end up with isolated interfaces for each targeted combination of mode and media which has several disadvantages like:

1. A interaction techniques like a Drag-and-Drop that spans different media or can be controlled by different modes cannot be implemented;
2. Multimodal interfaces that combine several modes and media for a more natural interaction can only be implemented to a limited extent: Changes of devices or addition of modes require processing all design models and their transformation again to generate new interfaces;
3. Starting a design process with a very high level of abstraction (such as by task models) requires the developer to have extensive anticipation skills and a deep knowledge of the transformational system to imagine how the final interfaces will behave and look like. This is often mentioned as a reason why MDDUI has not been adapted by the industry so far [Vanderdonckt 2008].

To address these challenges we propose the Multimodal Interaction Framework (MINT) [Feuerstack and Pizzolato 2011]. With the MINT framework, multimodal user interfaces are assembled by interactors. Different from the transformational development, the assemblage of interfaces by predefined elements is a common user interface development approach that is often supported by graphical user interface editors.

4 The Furniture Shop[1]

The furniture shop is a prototype of a web application that allows customers to buy furniture online. Using the furniture shop, a customer can choose among different objects and fill up a shopping cart. The furniture shop is targeted to address a common problem with buying furniture online: Does it really fit well in the room? Object sizes can be manually measured even if this is a

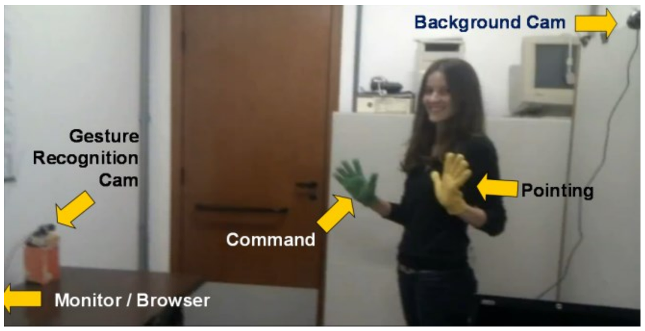

Fig. 2. The complete setup to control the web application using gestures.

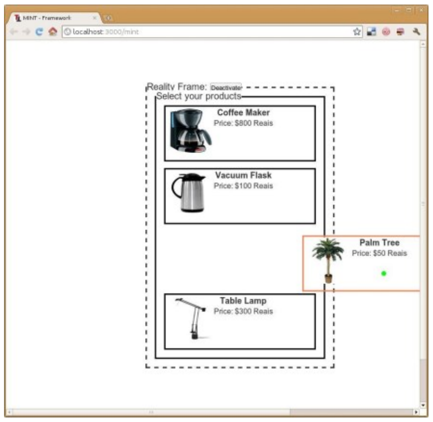

Fig. 3. After the reality frame has been activated, the online shop presentation is faded out and the shopping cart is moved to reflect the detected position of the monitor.

[1] A video documentation of the furniture shop prototype can be found online at http://multi-access.de/64

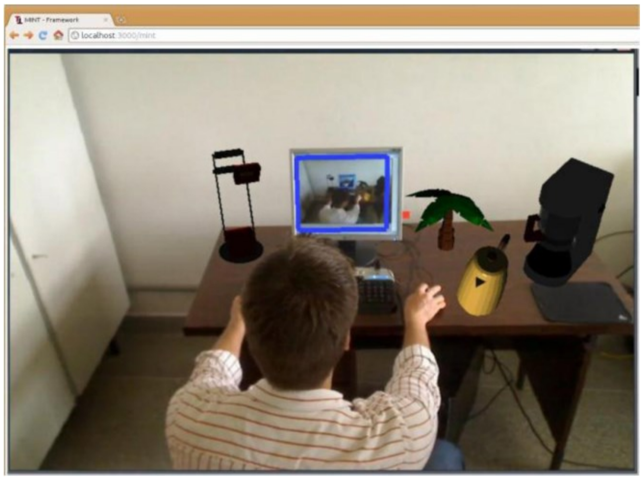

Fig.4. The final scene after dropping some objects out of the web browser. The blue frame displays the reality frame and the red dot the mouse pointer.

cumbersome and error-prone task. But deciding about the right colour or the object size in relation to other, already existing objects requires experience and good imagination skills. Therefore the online shop prototype includes an augmented reality viewer that depends on a background webcam. This webcam is connected to the user's PC running the browser and is directed to the place where the furniture should be positioned. For the interaction to work seamlessly the background cam also needs to captures the user's PC. Figure 2 illustrates this setup.

All interaction with the furniture shop prototype happens inside the browser. To switch between the online shop and the augmented 3D scene of the user's room we introduce a reality frame that needs to be crossed by a pointer. If an item is dragged out of the shopping cart, the reality frame gets activated and changes to display a dashed border. As soon as the user picks up an item of the shopping list and crosses the dashed line of the shopping cart (figure 3), the online shop website site is removed from the web interface and automatically replaced by a video stream of the background cam that shows the actual AR environment. The currently dragged item is toggled to a 3D VRML representation in the AR scene and the user can additionally move the object on the Z-axis (by using the mouse wheel) to position it exactly in the AR environment.

Figure 4 depicts the result of several objects that were successfully dropped to the AR scene. Since the AR scene shows the reality frame as well (the border around the monitor in figure 4) the user can easily navigate back to the web site by crossing the reality frame around the monitor with the pointer again. In the AR scene the pointer is shown as a red dot (see figure 4) and raises its size if it is moved closer.

The prototype supports two control modes: First, a basic one that uses the mouse to drag and drop items from the shopping cart into the augmented scene. Second, by using a second webcam that tracks the user's hands and recognizes basic gestures. The former one is targeted to be used with no extra hardware, but with the limitation that z-axis movements need to be performed with the mouse wheel. The latter one requires coloured gloves, a second webcam and good lightning conditions but implements a more natural interaction: One hand is used for pointing to objects and the other one from grabbing and releasing objects (figure 2).

To initial setup requires a calibration that calculates the relative position of the user's PC in the part of the environment that is captured by the background cam. For this, the system displays a visual marker full screen in the browser (figure 5). The calculation involves two steps: 1) the position of the marker is calculated using ARToolkit [Kato et al. 1999]; 2) the monitor size is calculated by retrieving the browser screen resolution and the actual DPI setting. Both are detected by using the Javascript browser API and the result is presented as a blue reality frame around the screen of the monitor (figure 5).

The furniture web application is used in the following section to explain our approach since is a good example for the challenges we address with our approach: It describes (1) a well-known interaction technique (a drag-and-drop), that (2) needs to be adapted to function media-spanning (between the 2D web and the 3D augmented reality), and (3) should consider different control modes (a mouse and gestures).

5 Modeling Multimodal Interaction

With the MINT framework interactors mediate information between a user and an interactive system. They can receive input from the user to the system and send output from the system to the user.

Each interactor is internally equipped with one AUI model that specifies the general, media independent representation and several CUI models to represent specific characteristics of a certain media. In the furniture shop prototype we used one CUI model to describe the graphical interface presentation in the web browser and a second CUI model to specify its appearance and behaviour in the AR scene. Each of these models consists of a static description that specifies the data as well as a description of each interactor's behaviour. We describe the former one by class models and the latter one by state charts.

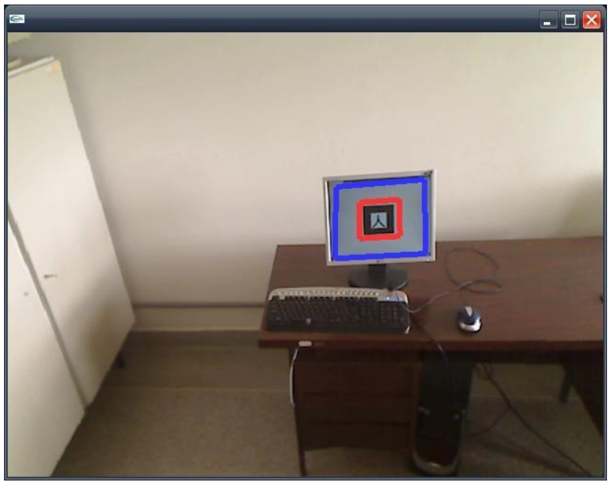

Fig. 5. If the reality frame is activated, the auto-calibration detects the monitor position by displaying a visual marker and switches back to the web site after the marker has been detected.

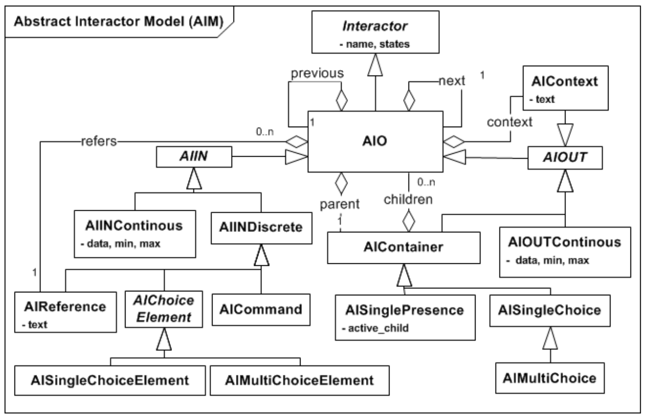

Fig.6. The static, abstract, media independent interactor model.

The modelling of interactors that we will present in the following sections might look initially complex but it enables the models to be transformed to code and to execute them as state machines. Further on, new interactors are only required to support new forms of interaction or include a new mode or media representation of an existing interface. Thus, interfaces are still created based on a pre-existing set of interactors that can be assembled without knowledge about their interns.

5.1 Media Modeling

Figure 6 shows the static AUI model structure, which represents all interactors that we have realized so far. All AUI interactors are derived from the basic Interactor class that sets a unique name and enables storing the actual states of an interactor. From the Interactor class the central Abstract Interactor Object (AIO) class is derived. It describes all abstract media independent interactions. The static AUI model distinguishes between interactors that are primarily designed for capturing user input (derived from the Abstract Interactor Input, AIIN class) and interactors that are used to return output to the user (Abstract Interactor Output, AIOUT). This distinction conforms to the separation between mode and media and therefore helps to understand, which parts of an interface can be handled by what kind of device.

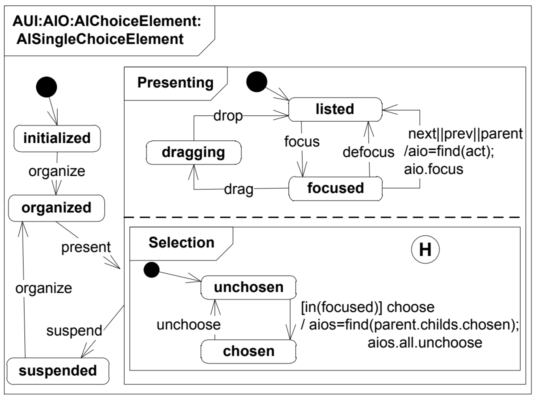

Fig.7. Choice element behavior description.

For the sake of brevity we focus on the explanation of the two central interactors that we rely on to model the Drag-and-Drop interaction technique later on: The overall choice container element (AISingleChoice) and the individual elements of the choice container that can be chosen (AISingleChoiceElement). Although the distinction of the choice into two elements seems quite unfamiliar at first, it is the result of the strict separation between input and output elements in the AUI. It is also driven by the idea of modelling self-executable interactors that can be moved between modes and media and have an individual behaviour model. In the following subsections we describe our main contribution, the model-based design of interactors. First we present exemplary two interactors that represent the shopping cart (which we call "choice") to explain the basic concepts of media interactor modelling in the first subsection. Thereafter, in the second subsection, we complement the choice interactors with a first very basic mode: a mouse. Then, in the third subsection, we describe our concept of multimodal interaction technique design with mappings that specify how the user can control an interactor through a mode. We focus on the drag-and-drop interaction technique of the furniture shop to explain such a connection. Finally, the last subsection briefly introduces initial tool support for the interactor design.

AUI Choice Interactor

Figure 7 shows the behaviour model of a choice list element. It implements the basic lifecycle of all AIOs. After its initialization (state "initialized"), it is "organized", "presenting" and "suspended". Within the "organized" state the AIO's neighbours are calculated. In state "presenting" an AIO is shown to the user and can be in the user's focus (state: "focused").

By navigation ("next", "previous", "parent" transitions) the focus can be moved to another AIO interactor. Finally, an AIO element is suspended if it is no longer part of the interface presentation.

An AIChoiceElement supports being dragged to another AIChoice and being chosen. The dragging process is part of the interface navigation and can only be issued if the element is in the user's focus. After an AIChoiceElement has been dropped to its destination it is set to the state "listed" again. The element selection is managed in parallel to the interface navigation but choosing an element is only possible if it is in the current user's focus.

The state chart-based element specification supports calling functions (actions) and sending events to other elements' state machines. Thus, to move the focus the next element, we call the

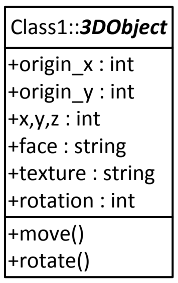

Fig. 8. Static model of the 3DObject.

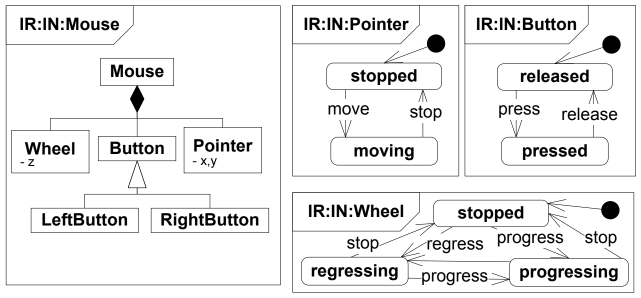

Fig.9. The composition of a mouse (left) and the behaviour models of the mouse components (right).

action "find(next)" and send a "focus" event to the next interactor. Furthermore, by using a condition, we ensure that only one list element is chosen at any time.

Each abstract interactor model can be complemented by a concrete model to consider media specific characteristics. In the case of the furniture shop, the shopping cart consists of 3DObjects, which consider media specific features, such as a 3D coordinate system and specific functionality like object rotation and movements like illustrated by the class structure of the 3dObject in figure 8. Further on, media specific behaviour can be added by a concrete model state chart.

5.2 Control Mode Modeling

In the last subsection we described how we model an interface element as an example for the media part of an interface. To control the interface we need modes. These can be described in the same way like the media, but modes can also be combined to offer a multimodal control of interfaces. Since the modes are declaratively modeled and directly executed by state machines, they can be easily manipulated at system runtime to prototype different ways of interacting with the same mode. Further on, new modes can be added by designing new mode models and using

mappings to connect them. Figure 9 presents the mouse mode specification that we use to control the prototype.

Mouse Mode

The behaviour specification of the components of the mouse is straightforward as illustrated by the state charts of figure 9. We distinguish two different types of states: states that describe a continuously ongoing action and states that describe an action that has just happened. An example for the former one is the moving state of the pointer that expresses continuously updating coordinates of the moving mouse, whereas the stopped state defines an action that just has happened. The class diagram is used to identify the data structure as well as to enable device composition. Thus, a mouse is composed by components like buttons, a pointer and a wheel for instance.

Gesture Mode

Hand gestures are already widely used as a natural way of human-computer interaction [Ballagas et al. 2007, Bolt 1980]. But the definition of suitable gestures depends on various factors and extensive user testing. These factors include for instance: the chosen hand poses, the number of hands considered, the feedback of the interface when a gesture is recognized, the delay on processing and communication, the ergonomics, the intuitiveness of the interaction, among other possible factors.

Our system works with only 4 simple gestures and it can reach accuracy close to 100%. It performs basic image processing techniques by using colour segmentation with coloured gloves that avoid environmental and hardware constraints on segmentation of images.

Figure 10a depicts the behaviour model of the gesture interaction resource that we use to control the furniture shop. It allows the

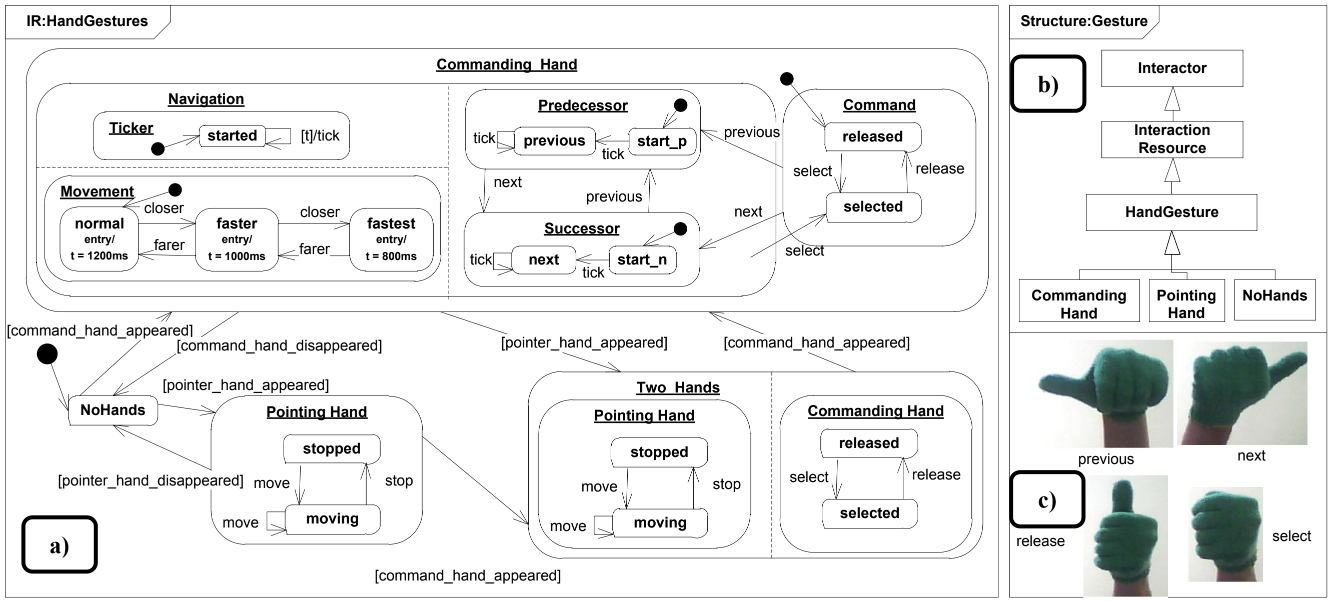

Fig. 10 (a) A behavior description of a one – and two handed gesture and posture control for the furniture shop. (b) The static interaction resource composition and (c) the four principal postures of the controlling hand.

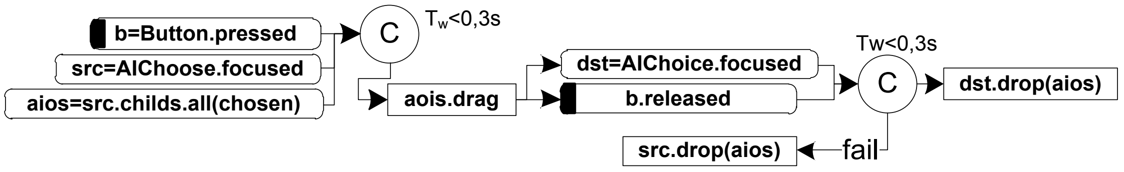

Fig.11. The Drag-and-Drop Mapping using a pointer and a button.

user to flexibly interact with one or two hands. The left hand of the user is used for pointing, a motion-related gesture, and the right hand for initiating the drag-and drop by a "button-press" kind of gesture, like illustrated in figure 10c. In the case that just the commanding hand is detected the user additionally has the option to navigate (e.g. to select one element of the shopping cart) by showing a next- or previous static posture to substitute the pointing. The navigation mode benefits from a ticker that triggers a navigation action in fixed intervals as long as the navigation posture is shown. The ticker can be manipulated by the user by moving the commanding hand closer or farther to the camera (see the "movement" super state in figure 10a).

5.3 Interaction Technique Modelling

Now that we have modelled a mouse and a gesture mode, we can connect its components to the user interface. We use mappings to specify these connections. Mappings rely on the features of the state charts that can receive and process events and have an observable state. Thus, each mapping can observe state changes and trigger events.

Mappings can be either pre-defined (e.g. to support a certain form of interaction with a particular device or to implement an interaction technique, such as the drag-and-drop that we describe in the following) or specified during the application design (e.g. stating that a security critical command must be confirmed with a mouse click and a voice command).

Figure 11 depicts the principal drag-and-drop mapping that we specify using a graphical notation similar to a flow chart. The mapping is bound to a (hardware) button and the AUI model part. Thus it can be applied for different media. Rounded boxes specify observations of state changes whereas boxes with sharp edges state events. With cycles, several different multimodal relations could be stated, such as redundant or sequential actions for instance. For the drag-and-drop mapping we just need the complementary operator, C, which is evaluated to true if all observations can be evaluated to true in a defined temporal window Tw.

The mapping is activated as soon as the Button is pressed and one AIChoice element (like the AISingleChoiceElement of figure 3) is in "focused" state. It then collects all selected list elements (aios) and sends them a drag event. As soon as the user releases the button while focusing on another list of the type AIChoice, the mapping sends a drop event to this list together with all elements that have been dragged. If the button is released without a list in focus, the complementary operator fails and the elements are dropped back to their origin.

To use the mapping with the hand gesture recognition only minor adjustments need to be made: the initial "b=Button.pressed" observation of the mapping has to be changed to "b=HandGestures.pressed" to be triggered in case the control hand shows the "button pressed".

Comparing to the mouse mode we presented earlier, the virtual "button pressed" posture misses a feedback to give the user the impression of a physical click. Figure 12 presents two exemplary multimodal mappings that add sound as a media to give feedback to the user. The first mapping shown in figure 12a implements this feedback: it waits for the "select" posture (figure 10c) that is shown while an interface element is in focus to play a sound. By using the redundancy operator, R, the interface navigation by the next and previous postures can be enriched by a sound feedback.

The redundancy operator outputs two events with a redundant meaning to different media. Thus, like illustrated in figure 12b, if the next posture is shown while an element is in focus, the focus will move to the next element and a specific sound (a click) is issued to additionally confirm the navigation.

5.4 Tool Support

The behaviour specification of interactors that can describe media, such as the graphical web interface that we are using in the furniture shop, as well as various modes, is supported by a design tool. We decided to base our approach on State Chart XML (SCXML), which is an upcoming W3C standard based on the Harel state chart definition [Harel 1987].

Different from other model-driven approaches that introduce new languages and design processes through several abstract models which need to be learned by the designer, MINT (using state charts for interaction modelling) has the advantage that state charts are already widely known and have a small sized basic vocabulary (mainly states and transitions driven by events).

For our interactor models we used the scxmlgui editor, which is written in Java and supports the basic SCXML vocabulary that our state machine designs are based on. Figure 13 shows a screenshot of the editor during the design process and the generated SCXML code that we parse to generate state machines.

6 Limitations and Ongoing Work

Using the mouse, the reality frame's position has to be matched with the monitor's position in the camera view for seamlessly

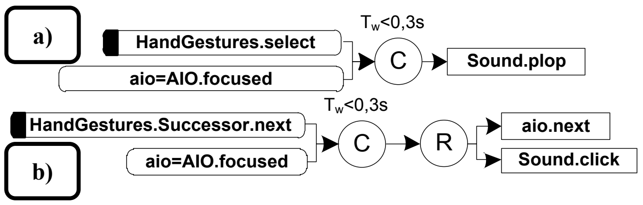

Fig. 12. Mappings to connect sound media and a gesture-driven control to the interface.

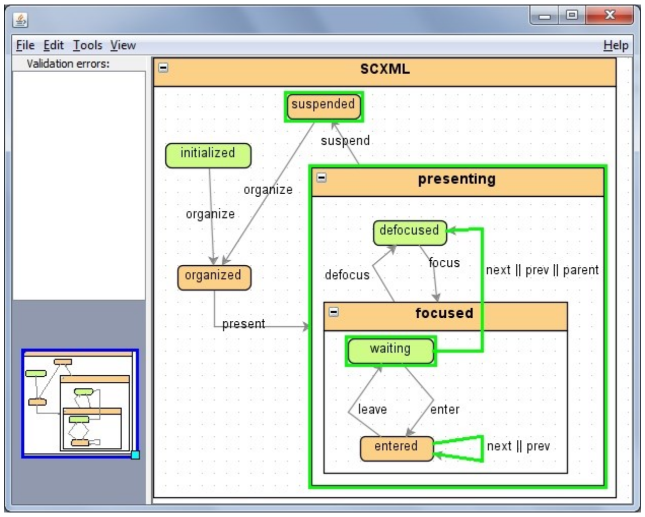

Fig.13. The scxml gui editor to design interactor behavior.

moving the pointer between the web interface and the AR scene. Although the mouse has the advantage of being the common device for web browsing, unfortunately, it was designed for a 2D coordinate system. We performed movements along the z axis in the AR scene by using the scroll wheel. But it turned out to be quite uncomfortable to drag objects having the button pressed and simultaneously using the wheel to arrange objects along the z axis.

These problems could be solved by the gesture-based drag-and-drop control since these modes are not so tightly connected to the desktop system, as the mouse. The gesture-based control can address a much larger space in the room and mainly depends on the viewing angle of the camera that captures the hand movements. But gloves have a different issue. Since they cannot be used to control the web browser interface, the user is bound to use two different controlling devices (mouse and hand), having to switch between them during the use of the application. Also, there is the need for two cameras, one for the AR and one for the gestures detection.

Considering the rotation of objects, it is possible to implement some commands with the mouse, such as right button click rotates in the x axis, and left button click rotates in the y axis, but this can be confusing and cumbersome for the user to make the desired rotation.

A solution using hand gestures has the same problems, requiring two different steps. It could also create a number of gestures too big for the user to remember.

The first prototype of the furniture shop was based on the ARToolkit for implementing the AR tracking and 3D object rendering. But to run everything inside the web browser, which is one of our main objectives, we had to stream the rendered AR scene from the ARToolkit implementation to an embedded flash player in the Web site. This solution was not working very well because of the latency due to the stream transmission, which is noticed by the users when they moved the pointer around.

Therefore we re-implemented the furniture shop prototype and based the implementation on FLARToolkit (a flash implementation of ARToolkit) for the marker tracking part and used X3DOM (an open source framework that integrates HTML5 and declarative 3d content) for the 3D object rendering. FLARToolkit feeds X3DOM with the tracking results and X3DOM renders and controls the 3D scene with Javascript, resulting in a Web based AR system working with no delays when moving the pointer or objects around.

We are currently extending the gesture control to experiment with different gestures to rotate objects and preparing our prototype to be published on our website for users to try out our approach. We hope to collect enough data to compare different ways of positioning the furniture using the mouse and different gestures and postures.

7 Conclusion

This paper presented a novel approach to design seamless interactions between different media and control modes. A model-based approach was used and a drag-and-drop application between different media (AR and web interfaces) and different modes (mouse and gesture-based control) was implemented.

Different from other works reported in the literature, our work supports the behaviour modelling of interface elements by state charts, which have been earlier applied to model graphical interfaces but not to span different media or consider multimodal control in AR. By means of mappings, modes and media could be combined in a declarative manner as well as design interaction techniques like the drag-and-drop. By combining declarative modelling with flow-chart oriented graphical notation, changes in interactions as well as support for new media and modes can be done on an abstract level instead of code-level (a tedious task even for a single device and computing language)

Acknowledgements

Ednaldo B. Pizzolato thanks FAPESP for supporting the presentation of this research at the conference. Sebastian Feuerstack is grateful to the Deutsche Forschungsgemeinschaft (DFG) and the EU ARTEMIS JU project HoliDes SP-8, GA No.: 332933 for the financial support of his research. Any contents herein reflect only the authors' views. The ARTEMIS JU is not liable for any use that may be made of the information contained herein.

References

[Ballagas et al. 2007] BALLAGAS, R., MEMON, F., REINERS, R., AND BORCHERS, J. O. 2007. istuff mobile: rapidly prototyping new mobile phone interfaces for ubiquitous computing. In *Computer Human Interaction*, 1107–1116.

[Bernsen 2008] BERNSEN, N. O. 2008. Multimodality theory. In *Multimodal User Interfaces*, D. Tzovaras, Ed., Signals and Communication Technology. Springer Berlin Heidelberg, 5–29. 10.1007/978-3-540-78345-9_2.

[Berti et al. 2004] BERTI, S., CORREANI, F., MORI, G., PATERNÃ², F., AND SANTORO, C. 2004. Teresa: A transformation-based environment for designing and developing multi-device interfaces. In *ACM CHI 2004*, ACM Press, Vienna, vol. II, 793–794.

[Bolt 1980] BOLT, R. A. 1980. "put that there": Voice and gesture at the graphics interface. In *Proceedings of the 7th Annual Conference on Computer Graphics and Interactive*

Techniques (SIGGRAPH '80), ACM Press, New York, NY, USA, 262–270.

[Bouchet et al. 2005] BOUCHET, J., NIGAY, L., AND GANILLE, T. 2005. The icare component-based approach for multimodal input interaction: Application to real-time military aircraft cockpits. In *HCI International*.

[Broll et al. 2005] BROLL, W., LINDT, I., OHLENBURG, J., HERBST, I., WITTKAMPER, M., AND NOVOTNY, T. 2005. An infrastructure for realizing custom-tailored augmented reality user interfaces. *IEEE Transactions on Visualization and Computer Graphics 11*, 722–733.

[BuildAR. 2015] BUILDAR., 2015. Buildar. Available at:http://www.buildar.co.nz/home/download/ last checked 09/01/15.

[Calvary et al. 2003] CALVARY, G., COUTAZ, J., THEVENIN, D., LIMBOURG, Q., BOUILLON, L., AND VANDERDONCKT, J. 2003. A unifying reference framework for multi-target user interfaces. *Interacting with Computers 15*, 3, 289–308.

[Feuerstack and Pizzolato 2011] FEUERSTACK, S., AND PIZZOLATO, E. 2011. Building multimodal interfaces out of executable, model-based interactors and mappings. In *Proceedings of the HCI International 2011; 14th International Conference on Human-Computer Interaction; Human-Computer Interaction, Part I*, Springer, Heidelberg (2011), Hilton Orlando Bonnet Creek, Orlando, Florida, USA., J. Jacko, Ed., no. LNCS 6761, pp. 221–228.

[Gonzalez-Calleros et al. 2009] GONZALEZ-CALLEROS, J., VANDERDONCKT, J., AND MUOZ-ARTEAGA, J. 2009. A structured approach to support 3d user interface development. In *ACHI '09. Second International Conferences on Advances in Computer-Human Interactions, 2009.*, 75 –81.

[Harel 1987] HAREL, D. 1987. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program. 8*, 3, 231–274.

[Hill et al. 2010a] HILL, A., MACINTYRE, B., ANDBRIAN DAVIDSON, M. G., AND ROUZATI, H. 2010. Kharma: An open kml/html architecture for mobile augmented reality applications. In *Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality, ISMAR*.

[Hill et al. 2010b] HILL, A., MACINTYRE, B., GANDY, M., DAVIDSON, B., AND ROUZATI, H. 2010. Prerequisites for open ar standard. In *Technical report, GVU Center, Georgia Institute of Technology, International AR Standards Workshop*.

[Hopmann et al. 2011] HOPMANN, M., GUTIERREZ, M., THALMANN, D., AND VEXO, F. 2011. Bridging digital and physical worlds using tangible drag-and-drop interfaces. *Transactions on Computational Science 12*, 1–18.

[Kato et al. 1999] KATO, H., BILLINGHURST, M., BLANDING, B., AND ARTOOLKIT., R. M. 1999. Artoolkit. Tech. rep., Hiroshima City University.

[Lawson et al. 2009] LAWSON, J.-Y. L., AL-AKKAD, A.-A., VANDERDONCKT, J., AND MACQ, B. 2009. An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components. In *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, ACM, New York, NY, USA, EICS '09, 245–254.

[Ledermann and Schmalstieg 2005] LEDERMANN, F., AND SCHMALSTIEG, D. 2005. April a high-level framework for creating augmented reality presentations. In *Proceedings of the 2005 IEEE Conference on Virtual Reality*.

[MacIntyre et al. 2004] MACINTYRE, B., GANDY, M., DOW, S., AND BOLTER, J. D. 2004. Dart: a toolkit for rapid design exploration of augmented reality experiences. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*.

[Ray and Bowman 2007] RAY, A., AND BOWMAN, D. A. 2007. Towards a system for reusable 3d interaction techniques. In *Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, ACM, New York, NY, USA, VRST '07, 187–190.

[Rekimoto and Saitoh 1999] REKIMOTO, J., AND SAITOH, M. 1999. Augmented surfaces: a spatially continuous work space for hybrid computing environments. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, New York, NY, USA, 378–385.

[Rekimoto 1997] REKIMOTO, J. 1997. Pick-and-drop: a direct manipulation technique for multiple computer environments. In *UIST '97: Proceedings of the 10th annual ACM symposium on User interface software and technology*, ACM Press, New York, NY, USA, 31–39.

[Schmalstieg et al. 2002] SCHMALSTIEG, D., FUHRMANN, A., HESINA, G., SZALAVARI, Z., ENCARNACAO, L. M., GERVAUTZ, M., AND PURGATHOFER, W. 2002. The studierstube augmented reality project. *Presence: Teleoperators and Virtual Environments, 11*, 33–54.

[Stanciulescu et al. 2005] STANCIULESCU, A., LIMBOURG, Q., VANDERDONCKT, J., MICHOTTE, B., AND MONTERO, F. 2005. A transformational approach for multimodal web user interfaces based on usixml. In *ICMI '05: Proceedings of the 7th international conference on Multimodal interfaces*, ACM Press, New York, NY, USA, 259–266.

[Vanderdonckt 2008] VANDERDONCKT, J. 2008. Model-driven engineering of user interfaces: Promises, successes, failures, and challenges. In *Proceedings of ROCHI 08*.

[Wingrave et al. 2009] WINGRAVE, C. A., LAVIOLA, JR., J. J., AND BOWMAN, D. A. 2009. A natural, tiered and executable uidl for 3d user interfaces based on concept-oriented design. *ACM Trans. Comput.-Hum. Interact. 16* (November), 21:1–21:36.