Model-based Design of Interactions that can bridge Realities

The Augmented "Drag-and-Drop"

Sebastian Feuerstack, Allan C. M. de Oliveira, Regina B. Araujo
Departamento de Computacão, Universidade Federal de São Carlos,
Rod. Washington Luis, km 235 - SP, Brazil
{sfeu, allan_oliveira, regina@dc.ufscar.br}

***Abstract*—The development of Augmented Reality (AR) applications is still driven by development at source-code level. Although recent approaches focus to standardize AR functionality, to our knowledge a declarative and model-driven design (MDD) has not been applied for AR development so far. MDD approaches have been successfully applied to model user interfaces for a wide spectrum of modes (such as speech-command interfaces, or remote controls) and media such as HTML, 3D, and smart phone interfaces for instance. In this paper we propose a MDD approach for modeling seamless interaction between Web and AR interfaces. Therefore we implemented a prototype of a web furniture shop that supports arranging furniture in an augmented reality to prove our approach. We then present our approach of modeling a reality spanning Drag-and-Drop interaction between a 2D browser and an AR scene. Finally, we discuss the issues that we were confronted with to support reality spanning interaction like switching interaction modes and coordinate systems and present limitations that we have experienced with using other control modes, like a Wii-Controller or glove-based gesture detection.**

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

I. INTRODUCTION

Augmented reality (AR) promises to enhance the real, physical world as we see it with additional information. Recent research activities target to evolve standards to support the authoring and distributing of AR applications [12].

Although a lot of AR applications (e.g. Yelp and Google Sky) have been built, the support for implementing AR applications is still limited to happen at source code level, such as, frameworks (e.g. Junaio) and libraries (e.g. ARToolkit). Declarative modeling that has been successfully applied to design multi-platform user interfaces and has been proven to generate voice [21], web [3] and 3D interfaces [11] for instance, has not been considered for AR application development so far.

A declarative modeling of user interfaces promises to reduce development costs and time by offering structured processes that enable to design the general (abstract) way of interaction with the user first and then systematically derives more specific interfaces for different platforms thereafter.

In this paper we propose such a declarative modeling approach to design interaction that can span the realities between a web interface and the AR. To describe our approach we focus on a furniture online shop prototype that we have declaratively modeled and that support seamless reality spanning interaction between the 2 dimensional web interface and the 3 dimensional AR scene by an augmented Drag-and-Drop.

Different from other approaches that tightly couple the Drag-and-Drop implementation to a particular application or to basic operation system functionality, such as copying files, or ejecting CDs, we propose a declarative design. Therefore every element of a user interface, that we call an interactor, consists of an abstract mode and media independent user interface model (AUI model) and several concrete interface models (CUI)s to address the specific interaction behavior of different modes and media. Our approach is driven by the idea to enable the design of a seamless interaction between different modes and media. To explain our approach we focus on a mouse-based mode to control the interaction in both, the web and the AR. For the mouse to work seamlessly while crossing realities, we introduce the Reality Frame concept. The declarative modeling enables to add further modes easily later on. New interaction modes can be easily added and changed by manipulating the declarative models instead of introducing changes at the code level.

The paper is structured as follows: The next section introduces the general "Drag-and-Drop" paradigm. Thereafter in section 3, we review related work. Section 4 introduces our prototype, a web application to buy furniture that enables a user to interactively experience the different furniture and virtually arrange it in an augmented environment. Then, in section 5 we explain our approach of declaratively modeling the augmented Drag-and-Drop and our concept of a Reality Frame. Thereafter in section 6 we discuss the experienced problems like for instance the required merge of different coordinate systems and the advantages and downsides of different modes of interaction and state future work. Finally, we give a conclusion in section 7.

II. THE DRAG-AND-DROP

With the first graphical interfaces the introduction of the mouse significantly enhanced the interaction that was previously limited to keyboard inputs. The mouse enabled direct manipulation and implementing a desktop metaphor (files and folders). Drag-and-Drop that was initially

developed for the Apple Lisa[1] was already used in the early 80ies for supporting file operations like to drag files to a certain folder or the desktop. At the same time Bolt introduced the "put that there" scenario [4], which proposed a multimodal way of moving objects around by using gestures and speech commands on a graphical interface.

The basic sequence involved in Drag-and-Drop is:

- Press, and hold down, the button on the mouse or any other pointing device, to "grab" the object,
- "Drag" the object/cursor/pointing device to the desired location,
- "Drop" the object by releasing the button.

Even through the Drag-and-Drop is well known, it has been established only for very basic kind of actions like:

- Dragging a data file onto a program icon
- Moving or copying files to a new location
- Rearranging widgets in a graphical user interface to customize their layout

On the one hand it is often not obvious where and in which cases the Drag-and-Drop works, which is a usability problem. On the other hand the Drag-and-Drop is hard to implement since Drag-and-Drop modeling approaches are still missing [14]. Looking at popular programming languages, like Java for instance, the Drag-and-Drop has to be implemented manually and can be tedious and error-prone as there is no generic way to implement it once and use it for several components. Thus, for each Swing GUI component it includes implementing various interfaces and consists of the following steps to create a drag-able element:

Implementing a *DragGestureListener*, a *DragSource*, call the *createDefaultDragGestureRecognizer*, and creating a *DefaultDragGestureRecognizer*. To receive a drop, a component has to implement the *DropTargetListener* interface, an instance of *DropTarget* needs to be created and a *drop method* has to be implemented to add the dragged data.

Nowadays there are a lot of different kinds of devices and controls available requiring the Drag-and-Drop to be implemented very differently. For instance, for touch interfaces on common smart phone operating systems, like Android and iPhone, a long-press-then drag substitutes the click-and drag of a Desktop PC.

The general idea of our approach is to enable a reality spanning interaction between different modes and media by supporting declarative modeling of interaction that enables to adapt and personalize device-spanning interaction based on a graphical notation that ease the interaction design as it abstracts from the tedious code-level development.

We demonstrate the approach by a furniture shop prototype that we have implemented and that supports Web and AR media as well as a mouse and a Wii-remote mode of control.

III. RELATED WORK

Our work is based on the idea of combining earlier works about multimodal interaction and model-driven development of user interfaces to enable a developer to assemble multimodal interfaces. Model-driven development research has resulted in several connected design models that have been summarized by the Cameleon Reference Framework [8] and by user interface languages such as USIXML [17]. But it has been applied to develop interfaces for pre-defined platforms only, such as to design interfaces for small screens of cell phones, for speech interfaces or to develop television and 3D interfaces for instance. To our knowledge AR applications haven't been modeled by MDD approaches so far and Drag-and-Drop interactions have only be addressed by modeling to a very limited extend focusing on graphical desktop interfaces [14].

Different to these approaches our interactor-based interfaces can be flexibly extended to new modes and media by adding new interactors and mappings to a running system. Our approach is inspired by the findings of multimodal theory [2] and the iCARE platform [5] that supports building multimodal interaction out of components that are connected based on the CARE properties. These properties describe the relations between different modes, such as their complementary, redundant or equivalent combination. The Open Interface Framework [15] that supports prototyping of multimodal interactions out of components similar to our approach does not consider model-driven development for the design of interfaces based on abstract and concrete models.

Although the main feature of our work is the Drag-and-Drop transition of virtual objects from a web browser to reality (through augmented reality), the simple task of dragging in a 3d user interfaces is already a challenge. Therefore, in AR, much of the work done with Drag-and-Drop is based on moving objects between displays/surfaces, just dealing with 2d, like Rekimoto's pick-and-drop [19], which is grabbing an object in one display and putting it in another, or like Rekimoto and Saitoh hyperdragging [20], which is dragging 2d objects through displays.

For moving objects in 3d, it is necessary different techniques or equipments, which must deal with the Z axis. Some of the input techniques for 3d are: gestures [1], 3d pointer devices [24] (3d mouse, Wii-controller) and using markers (similar to gestures but simpler) [16, 6].

One of the first systems to use Drag-and-Drop across dimensionalities, transitioning from 2d to 3d and vice versa, was Hollerer and MacIntyre system's EMMIE [7]. A system designed for collaboration (with even the possibility of remote users). It used several displays like PCs, tablets, a projector, HMD (for augmented reality), so that the user could choose the best to work with.

A more recent work is Petersen and Stricker attempt to make a continuous natural user interface [18], which is a

[1] Bruce Horn, On Xerox, Apple and Progress, http://www.folklore.org/StoryView.py?project=Macintosh&story=On_Xerox,_Apple_and_Progress.txt, last checked 20/04/11

system with a more intuitive interface that uses gestures to control AR objects. The user can, for instance, point to select an object, and use his hand to drag it and drop in a different display. The focus is the simplified transition between devices.

Another project is the hybrid interface from Carvalho et al. [9], in which users can do transition between 2d WIMP interfaces, augmented reality and virtual reality (VR). This way the user experience with WIMP interfaces is exploited, and AR and VR can be used to better visualize and comprehend the 3d scene. The project is continued in [22], where the Open Interface framework is used to transform a desktop application into a hybrid interface (WIMP, AR and VR), and it is illustrated how Open Interface framework can facilitate the development of applications with several techniques of user interaction.

Albeit our work is similar to projects like EMMIE or the continuous natural user interface, our approach is focused on the Drag-and-Drop modeling and applying to work between the web browser and the AR.

IV. THE FURNITURE SHOP

To test our approach we implemented a prototype of a web shop that allows customers to buy furniture. Therefore a customer can choose between different furniture and fill up a shopping cart. Although the shopping cart is filled in a traditional way by browsing the web site and clicking on "buy" buttons, it is encapsulated by the reality frame and therefore extended with a reality crossing Drag-and-Drop support.

Figure 1 shows the actual shopping cart that is part of a typical online shop and has been enhanced by a Reality Frame. To enable a reality crossing Drag-and-Drop, the Reality Frame needs to be activated, which triggers the calibration of the system. For this the system displays a visual marker on the monitor to figure out the relative position of the monitor in its environment (figure 2). The calculation involves two steps: first, the position of the marker is calculated using ARToolkit [13], which is illustrated by the red border. Second, 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 frame around the screen of the monitor.

The calibration works best, if the monitor brightness and contrast are reduced and the camera is directly facing the monitor. The marker size could be relatively small – we got satisfying detection rates with 150pixel sized tags in 2 meter distance – but the viewing angle of the monitor limits the angle from which the camera can capture the tag. With a common 17 inch monitor we got good detection rates if the angle is varied maximal 15 degrees horizontally and vertically. The actual marker detection happens in milliseconds. Since the tag gets only displayed until it could be recognized the first time, the user recognizes the automatic calibration only by a short flickering of the web interface.

Thereafter the position of the monitor in the camera view is returned to the web browser, and the Reality Frame with

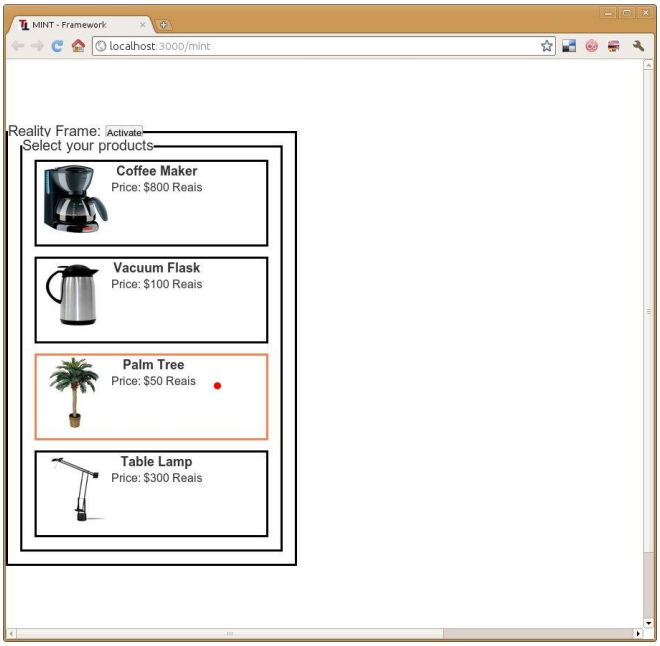

Figure 1. The shopping cart that is part of a common web shop while dragging an object across the reality frame that is illustrated by the dashed line. Unfortunatly the application we used to take the sceenshot has hidden the mouse pointer.

the included shopping card is repositioned so that it matches the relative x and y positions of the monitor in the AR scene. This new position can be seen in figure 3. The repositioning of the Reality Frame in the browser extends the freedom of movement of the mouse pointer in the actual AR representation. The original shopping cart position (figure 1) prevents dragging objects out of the browser to the left.

The activated Reality Frame is changed to display a dashed border and enables the shopping card items to be dragged out of the frame. As soon as the user picks up an

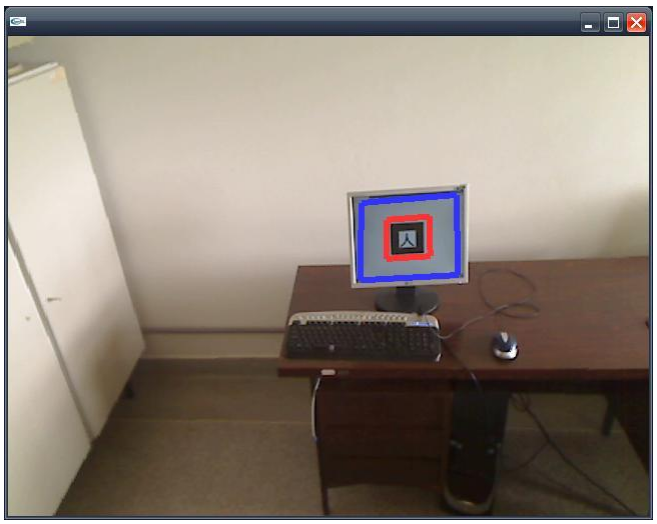

Figure 2. If the Reality Frame is activated, the Autocalibration detects the monitor position by displaying a visual marker and switches back to the web site after the marker has been detected.

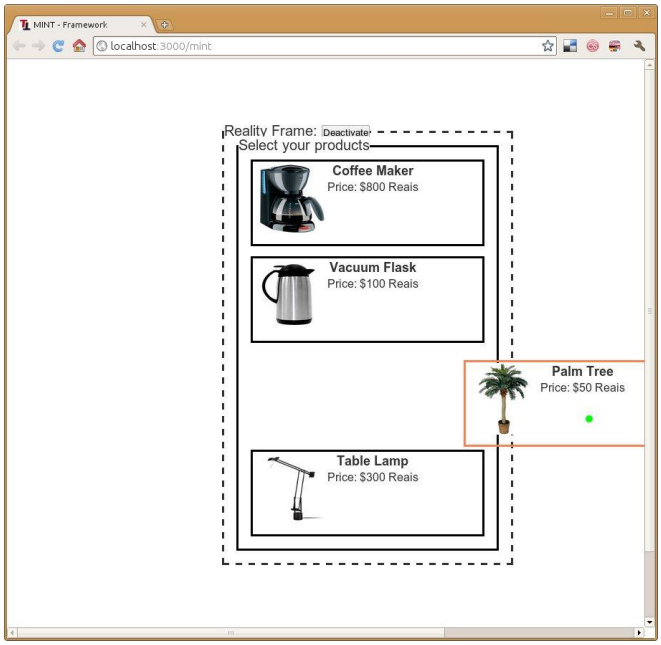

Figure 3. After the Reality Frame has been activated, the online shop presentation is faded out and the shopping cart is moved to reflect the detected position of the monitor .

item of the shopping list and crosses the dashed line of the shopping cart, the online shop website site is removed from the web interface and automatically replaced by a video stream 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 like shown by figure 4.

Figure 5 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 blue-colored border around the monitor in figures 3 and 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 3 and 4) and raises its size if it is moved closer.

V. 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 the interaction with them) such as for describing commands, lists or selections need to be modeled only once and can be subsequently re-used in interfaces for different media (like web interfaces, speech or augmented reality scenes). Actual MDDUI approaches [8, 17] 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.

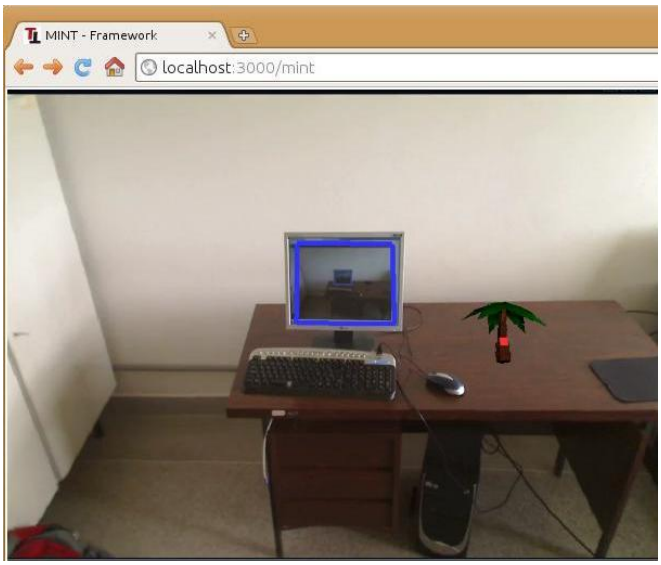

Figure 4. For the augmented scene, we are relying on a fixed camera that is focused into the actual part of the room where after crossing the reality frame the browser continues the dragging action, but changes to the augmented scene and a 3D representation of the object.

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 a final user interfaces for several devices.

Although these MDDUI approaches have been proven to work well to generate interfaces for different devices and modes, like speech or 3D interfaces for instance, they end up with isolated interfaces for each targeted combination of mode and media which has several general disadvantages:

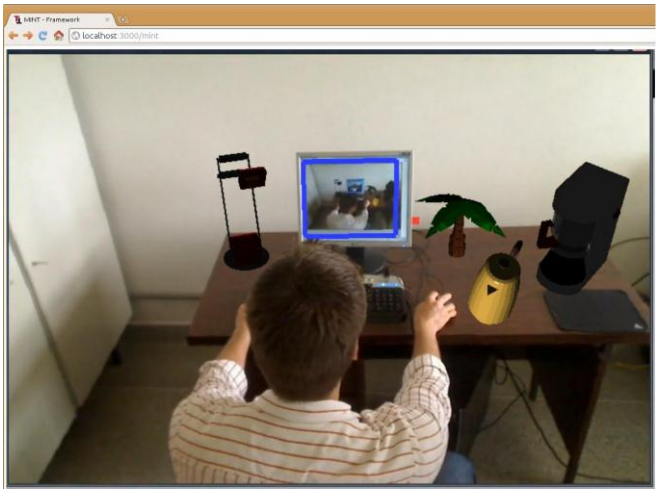

Figure 5. The final scene after droping some objects out of the web browser. The blue frame displays the reality frame and the red dot the mouse pointer.

1. A form of interaction like a Drag-and-Drop that spans mode and media could not be implemented.
2. Multimodal interfaces that combine several modes and media for a more natural interaction can only be implemented to a limited extend: 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 (like by task models) requires a 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 reason why MDDUI has not been adapted by the industry so far [23].

To address these challenges and to implement the augmented Drag-and-Drop of our prototype we enhanced the Multimodal Interaction Framework (MINT) [10]. With the MINT framework, multimodal user interfaces are assembled by interactors. Different to the transformational development, the assembling of interfaces by predefined elements is a common user interface development approach that is often supported by graphical user interface editors. An interactor mediates information between a user and an interactive system. It can receive input from the user to the system and send output from the system to the user.

For modelling the furniture prototype we equipped each interactor with three different models: One AUI model that specifies the general behaviour and two CUI models: one describes the graphical interface presentation behaviour in the web browser and a second CUI model that describes its appearance and behaviour in the AR scene. Every model of MINT consists of a static description that describes the data of all interactor as well as a description of each interactor's behaviour. We describe the former one by class models and the latter ones by state machines.

Figure 6 shows the static AUI model structure: 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 basic abstract class the central *Abstract Interactor Object* (AIO) class is derived that describes all abstract interactions, for that an interactor can be offered and includes already basic interface navigation capabilities (to the previous and next AIO respectively). The static AUI model distinguishes between interactors that are primarily designed for capture user input - these are derived from the Abstract Interactor Input (AIIN) class and Abstract Interactor Output (AIOUT) interactors that are used to resturn output to the user. This distinction conforms to the separation between mode and media and therefore helps to distinguish which parts of an interface can be handled by what kind of device.

For the sake of brevity we focus in the following on an explaining of the two central interactors that we rely on to model the Drag-and-Drop: The overall choice container element (*AISingleChoice)* and its individual list elements that can be chosen (*AISingeChoiceElement).* Although the distinction of the choice into two elements seems quite unfamiliar in the first moment, it is the result of the strict

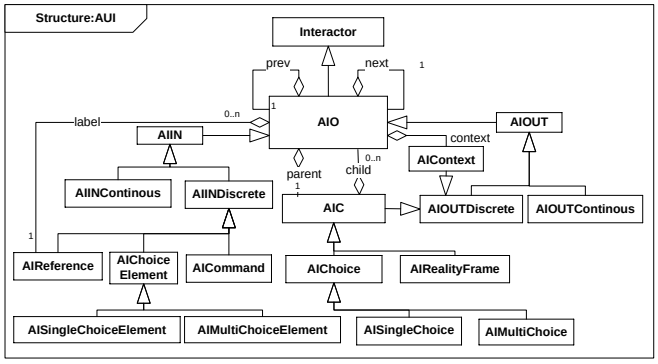

Figure 6. Abstract User Interface Model (MINT-ML V1.1)

separation between input and output elements in the AUI as well as 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 sections we describe both, the overall AUI model and then the graphical web and the augmented scene CUI model implementation. Finally, based on these findings we report how the Drag-and-Drop is modelled and implemented for the choice interactor.

A. AUI Choice Interactor

Figure 7 shows the behaviour model of an element of a choice. It implements the basic lifecycle of all AIOs. After it's initialization it can be organized, presented and suspended. Each AIO that is part of an interface presentation is organized to enable navigation between it and its neighbours. During an AIO is presented it can be in the user's focus, while the user navigates using previous and next commands. Finally an AIO element is suspended if it is no longer part of the interface presentation.

An AIChoiceElement supports to be dragged to another AIChoice and to be 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 droped 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 machine-based element specification supports calling functions and sending events to other elements' state machines. We use this functionality to move the user's focus with the element navigation to the next, previous or parent one. And we ensure that only one list element is chosen any time for a SingleChoice (see figure 7).

B. Connecting Modes to the Interface

In the last section 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 although be flexible combined to offer a multimodal control of interfaces. The combination of modes with media is specified using mappings. Figure 7 presents the mouse mode

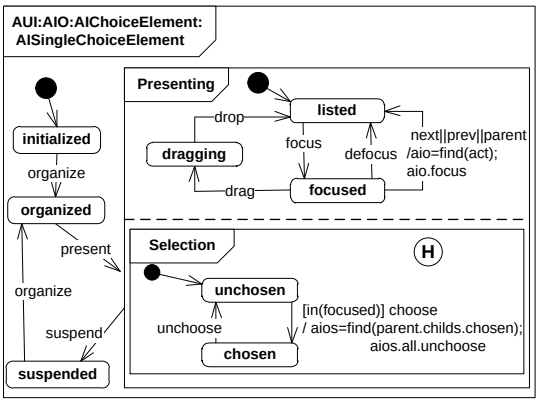

Figure 7. Choice element behavior description.

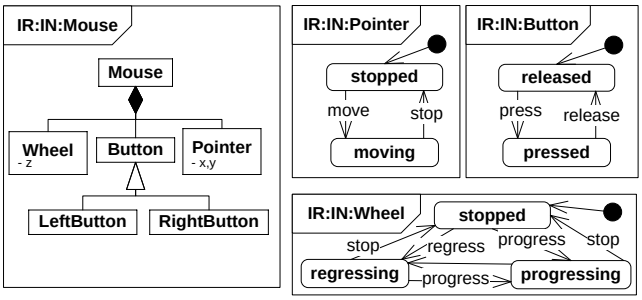

Figure 8. The composition of a mouse (left) and the behaviour models of the mouse components (right).

specification that we use beneath the Wii-Remote to control our prototype.

The behavior specification of the components of the mouse is straight forward as illustrated by the state machines of figure 8. We distinguish two different types of states: states that describe a continuously ongoing action – like moving the mouse and state an action that has already happened. The former continuously update data – like the x and y coordinates of the pointer when it is in the "moving" state. It communicates its x, y, and z coordinates. The class diagram is used to identify the data structure as well as to enable device composition. Thus, a mouse is composed of components like buttons, a pointer and a wheel for instance. A Wii controller description can reuse parts of these components like the pointer or the button components.

C. The Augmented Drag-and-Drop Design

Now that we have modelled a mouse device, 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 paradigm, 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 9 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 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) that 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 7) 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 the actual focus the complementary operator fails and the elements are dropped back to their origin.

The corresponding behavior of a list that contains the Choice elements is depicted in figure 10. It is initialized in state listing and is able to receive drop events (e.g. from the Drag-and-Drop mapping). All elements that are dropped onto the choice are added as new elements and receive a drop event thereafter to inform them that they have been successfully dropped (this returns all dragged elements to state listed as depicted in figure 7).

D. Differences in Web- and Augmented Scene selection Interactors for CUI Modeling

Different from the AUI model that describes general semantics of the interaction, which is independent of mode and media, the concrete user interface (CUI) model captures mode and media dependent information that is required for presenting and controlling the interface on a specific device.

Since we focus on graphical media as the output format for presenting the interface, the management of a coordinate

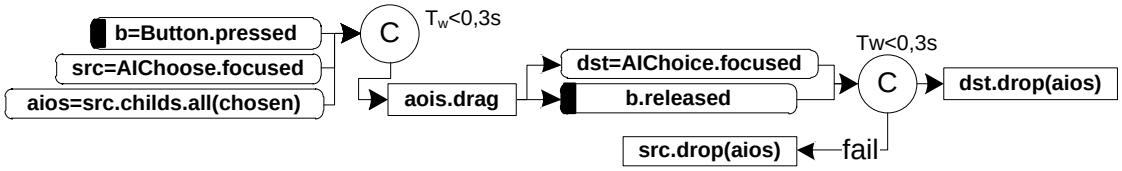

Figure 9. The Drag-and-Drop Mapping using a pointer and a button.

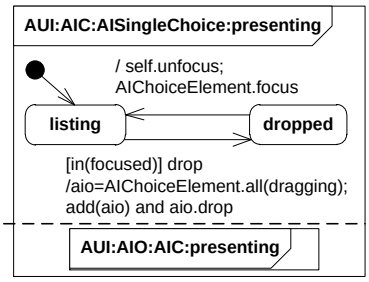

Figure 10. Choice element behavior description.

system is the basic functionality that has to be added on the CUI model level. The coordinate system needs to be able to position elements on the web interface as well as to retrieve elements based on the coordinates of a pointer.

Figure 11 illustrates the relations between the AUI models, the graphical CUI model and the final interface elements that we use to represent the furniture in the HTML interface (an image) as well as in the AR scene (a VRML object). Both are derived from the CUI Selectable class, enabling them to be selected and used for Drag-and-Drop interaction. They share the same properties of the basic concrete interactor object (CIO), which enables already 3D positioning. Thus, for the representation of the VRML elements in the augmented scene, we only need to add the depth as well as the object rotation information.

E. Crossing the Reality Frame

Since the same input device is used in our system to control both the 2d browser interface and the 3d AR interface, it was necessary to make transformations between the coordinate systems present in the desktop and in AR.

To make the system more usable we developed an auto-calibration feature that quickly displays a marker tag in the first time of use, tracks and saves this position. Thus, as long as the camera is fixed, the system does not need to display a marker during its use to recalibrate the AR.

However, if a similar scale system among different monitors and a precise size of the reality frame are both requirements, then information on the exact marker size in the monitor as well as the monitor's size is necessary. This information is obtained by finding out the screen size and DPI, so that the AR scene can be scaled accordingly. Therefore we can create a relative coordinate system that can work in any screen size automatically.

Since the mouse is initially being used for user interaction, a space in the screen is necessary from where the mouse can be used to control the AR (where the 2d mouse becomes a 3d pointer). This space is all but the "Select your products" box (the reality frame). However, as can be seen in figure 1, object moves are limited only to the right, above, and under the monitor. Thus the optimal screen placement for the reality frame would be the same position of the monitor in the camera view, giving the user freedom to place virtual objects anywhere around the monitor, but still inside the camera view.

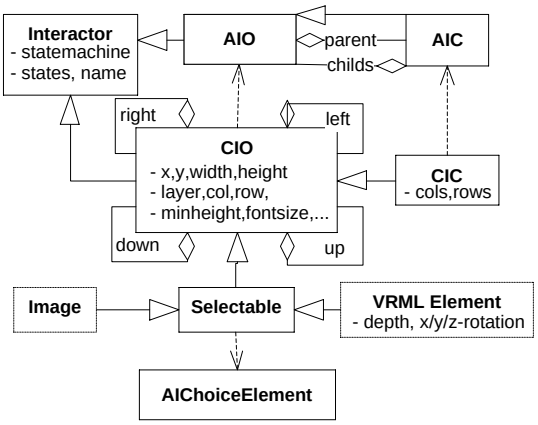

Figure 11. Relations between AUI, grahical CUI and final AR and HTML interface elements.

Therefore, the position of the monitor in the camera view must be the same position of the reality frame in the browser, so that the user can use the entire screen space around the frame to move the mouse and execute the drag and drop. This is done so that the mouse does not shift its location when coming from the browser to the augmented reality. If the reality frame position is similar to figure 1, and the monitor position is like figure 11, then when the user drags something out of the frame the mouse would shift its position in the AR scene to the left of the monitor, while it actually should be in the right side of the monitor.

Figure 12 shows some of the measurements necessary to adjust the coordinate system, which are explained below.

The initial steps to calibrate the system are as follows: at the beginning of execution, the web browser resolution and DPI settings are obtained, and then passed to the AR application. The expected marker size is 80 millimetres (red color in figure 12). If, for instance, the marker actual size is 72mm, the AR objects have to be rescaled to 90% of their original size. However, this does not correct the monitor overlay (the blue-colored border shown in figures 4, 5 and 12). For that, it is still necessary to make adjustments in this particular interface component, which are relative to the monitor size (gray color in figure 12).

The monitor position in the camera view is then passed back to the browser, which adjusts the Reality Frame to have a similar position.

After this a correctly scaled system is achieved and the remaining issue is about setting the x and y axis ranges so that the AR objects can be moved around in the scene and still be seen by the camera. For that the distance between the monitor and the camera is captured as the z axis relation returned from ARToolkit. As an example, with a z of 1000, the monitor is approximately 1,5 meters away from the camera. Since the center (0,0,0) of the coordinate system of ARToolkit is assumed to be the middle of the marker for the example shown in figure 12, objects x value can vary from -450 to 450, and y can go from -400 to 400.

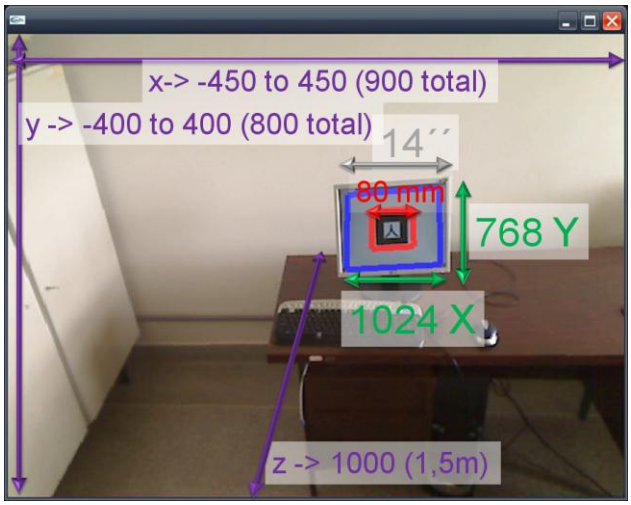

Figure 12. Calculating the coordinate system.

Since the user is now using the mouse to control the AR, and considering, for instance, Figure 12, when the monitor's resolution is 1024 x 768 (green color), this resolution has to be transformed, to 900, the AR width, and 800, the AR height, respectively. The proportion of 1,137 for x and 0.96 for y is obtained by dividing the resolution size by the AR size.

Now we can put together a transformation equation to be used by the AR application. Since there is no negative in the browser coordinate system, the equation is:

```
1. X_AR = X_RES/1.137-450

2. Y_AR= (Y_RES/0.96)*(-1)+400
```

X_AR and Y_AR and X_RES and Y_RES are the values of x and y for the AR and browser, respectively. The numbers that multiply X_RES and Y_RES are the browser/AR rate. In equation 1, 450 is subtracted (the smallest AR x axis' value), and in equation 2, 400 is added (after multiplying by -1, since the y value has to be inverted or else going up with the mouse would make the AR object goes down).

VI. Limitations and Ongoing Work

As seen in the last section, the reality frame's position has to be equalized with the monitor's position in the camera view for a better usage, and the mouse's position in the screen has to be transformed into an AR position in the space (therefore limited to the screen size). These limitations, resulting from the use of the mouse, could be overcome by using a Wii-Controller or a glove (for hand gestures) to control the augmented drag-and-drop, since these devices are not limited to the desktop system, as the mouse is.

The downside of using a Wii-Controller is its small angle of view. Since it was originally projected to just cover a normal television size (42 inches) as a pointer (while its main feature is the motion detection using inertial sensors), it is not possible to use it to cover a large area in a room. Thus our Furniture Shop application was limited to a small area around the infrared optical sensor (the device that projects infrared light to be captured by the camera in the controller).

This field of view limitation of the Wii-Controller is not present in the use of gloves, which 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 can not 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.

The Wii-Controller can be a better solution when its embedded gyroscope is used because then, the rotation of the controller could be the rotation of the objects. Nevertheless, this solution would also involve a two steps control from the user, the first for the drag and drop and the second to rotate the object.

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.

Regarding the implementation of the system another problem arises. For the augmented reality tracking and rendering of virtual graphics the original implementation of ARToolkit in C++ was used. Since the objective is to run everything inside the web browser, a decision was made to stream the result of the AR scene to an embedded flash player. With this approach the user can view and use the system with just a web browser. The downside of this solution is the added latency due to the stream transmission, which is noticed by the users. Also, this stream solution uses key frames to reduce the bandwidth needed. Thus, it buffers and waits for a key frame in the beginning of the execution, causing delay to the web browser stream visualization. The most appropriate solution in this case would be to switch the implementation of ARToolkit for a compatible with the web browser, like for instance a flash implementation (e.g., FLARToolkit).

VII. Conclusion

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

Different from other works reported in the literature, our work supports the behavior modeling of interface elements by state machines, which have been earlier applied to model graphical interfaces but not to span different modes and media. By means of mappings, modes and media could be combined in a declarative manner as well as design interaction paradigms like the augmented Drag-and-Drop.

By combining declarative modeling 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). The concept of a reality frame that was implemented as part of our furniture shop prototype proved that reality spanning interaction can be implemented and different modes can seamlessly adapt to different realities. This was possible by transforming the coordinate system and benefiting from the abstract modeling of interface elements that can transform themselves into different presentation while maintaining the interaction behavior that is specified in the abstract user interface model.

ACKNOWLEDGMENT

Sebastian Feuerstack is grateful to the Deutsche Forschungsgemeinschaft (DFG) for the financial support of his work. Allan Oliveira and Regina Araujo are grateful for CNPq and FAPESP for the funding to INCT-SEC, process 573963/2008-8 and 08/57870-9.

REFERENCES

[1] Hrvoje Benko, Edward W. Ishak, and Steven Feiner. Cross-dimensional gestural interaction techniques for hybrid immersive environments. In *Proceedings of the IEEE Virtual Reality. Conference*, pages 209–216, 2005.

[2] Niels Ole Bernsen. Multimodality theory. In Dimitrios Tzovaras, editor, *Multimodal User Interfaces*, Signals and Communication Technology, pages 5–29. Springer Berlin Heidelberg, 2008. 10.1007/978-3-540-78345-9_2.

[3] Silvia Berti, Francesco Correani, Giulio Mori, Fabio Paterno, and Carmen Santoro. TERESA: A transformation-based environment for designing and developing multi-device interfaces. In *ACM CHI 2004, Extended Abstract*, volume II, pages 793–794, Vienna, Austria, April 2004. ACM Press.

[4] Richard A. Bolt. "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)*, pages 262–270, New York, NY, USA, 1980. ACM Press.

[5] Jullien Bouchet, Laurence Nigay, and Thierry Ganille. The icare component-based approach for multimodal input interaction: Application to real-time military aircraft cockpits. In *HCI International*, 2005.

[6] Volkert Buchmann, Stephen Violich, Mark Billinghurst, and Andy Cockburn. Fingartips: gesture based direct manipulation in augmented reality. In *Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, GRAPHITE '04, pages 212–221, New York, NY, USA, 2004. ACM.

[7] Andreas Butz, Tobias Haellerer, Clifford Beshers, Steven Feiner, and Blair MacIntyre. An experimental hybrid user interface for collaboration. Technical report, Columbia University New York, Department of Computer Science, 2005.

[8] Gaelle Calvary, Joelle 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.

[9] Felipe Carvalho, Alberto Raposo, and Marcelo Gattass. Uma interface hybrida para desktop integrando realidade virtual, realidade aumentada e 2d wimp. In *Proceedings of the IX Simposium on Virtual and Augmented Reality SVR 2007*, pages 152–161, 2007.

[10] Sebastian Feuerstack and Ednaldo Pizzolato. Building multimodal interfaces out of executable, model-based interactors and mappings. In Accepted for: 14th International Conference on Human-Computer Interaction, Orlando, Florida, USA., July 2011.

[11] Juan Manuel González-Calleros, Jean Vanderdonckt, and Jaime Muñoz Arteaga. A structured approach to support 3d user interface development. In *Proceedings of the 2009 Second International Conferences on Advances in Computer-Human Interactions*, ACHI '09, pages 75–81, Washington, DC, USA, 2009. IEEE Computer Society.

[12] Alex Hill, Blair MacIntyre, Maribeth Gandy, Brian Davidson, and Hafez Rouzati. Prerequisites for open ar standard. Technical report, GVU Center, Georgia Institute of Technology, International AR Standards Workshop-October 11-12, 2010, 2010.

[13] H. Kato, M. Billinghurst, B. Blanding, and R. May. Artoolkit. Technical report, Hiroshima City University, 1999.

[14] Jean-Francois Ladry, Philippe Palanque, David Navarre, and Eric Barboni. Model-based usability evaluation and analysis of interactive techniques. In *Proceedings of the 5th International Workshop on Model Driven Development of Advanced User Interfaces (MDDAUI 2010)*, pages 21–24, 2010.

[15] Jean-Yves Lionel Lawson, Ahmad-Amr Al-Akkad, Jean Vanderdonckt, and Benoit Macq. An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components. In *EICS '09: Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, pages 245–254, New York, NY, USA, 2009. ACM.

[16] Gun A. Lee, Claudia Nelles, Mark Billinghurst, and Gerard Jounghyun Kim. Immersive authoring of tangible augmented reality applications. In *Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '04, pages 172–181, Washington, DC, USA, 2004. IEEE Computer Society.

[17] Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, and Victor Lopez-Jaquero. USIXML: A language supporting multi-path development of user interfaces. In Remi Bastide, Philippe A. Palanque, and Joerg Roth, editors, *EHCI/DS-VIS*, volume 3425 of *Lecture Notes in Computer Science*, pages 200–220. Springer, 2004.

[18] Nils Petersen and Didier Stricker. Continuous natural user interface: Reducing the gap between real and digital world. In *Proceedings of the 2009 8th IEEE International Symposium on Mixed and Augmented Reality*, ISMAR '09, pages 23–26, Washington, DC, USA, 2009. IEEE Computer Society.

[19] Jun Rekimoto. Pick-and-drop: A direct manipulation technique for multiple computer environments. In *ACM Symposium on User Interface Software and Technology*, pages 31–39, 1997.

[20] Jun Rekimoto and Masanori Saitoh. Augmented surfaces: a spatially continuous work space for hybrid computing environments. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, CHI '99, pages 378–385, New York, NY, USA, 1999. ACM.

[21] 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.

[22] Daniela Trevisan, Felipe Carvalho, Alberto Raposo, Carla Freitas, and Luciana Nedel. Supporting the design of multimodal interactions: a case study in a 3d sculpture application. In *Proceedings of the XII Symposium on virtual and Augmented Reality*, pages 153–162, 2010.

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

[24] Joanne Zucco, Bruce Thomas, and Karen Grimmer. Evaluation of four wearable computer pointing devices for drag and drop tasks when stationary and walking. *Wearable Computers, IEEE International Symposium*, 0:29–36, 2006.