Modeling of User Interfaces with State-Charts to Accelerate Test and Evaluation of different Gesture-based Multimodal Interactions

Sebastian Feuerstack, Mauro dos Santos Anjo, Jessica Colnago, Ednaldo Pizzolato

Universidade Federal de São Carlos, Departamento de Computacão,
Rod. Washington Luis, km 235 - SP, Brazil
{sfeu, mauro_anjo, jessica_colnago, ednaldo}@dc.ufscar.br

New forms of interactions such as the gesture-based control of interfaces could enable interaction in situations where hardware controls are missing and support impaired people where other controls fails. The rich spectrum of combining hand postures with movements offers great interaction possibilities but requires extensive user testing to figure out a user interface navigation control with a sufficient performance and a low error rate.
In this paper we describe a model-based interface design based on assembling interactors and multimodal mappings to design multimodal interfaces. We propose to use state charts for the rapid generation of different user interface controls to accelerate user testing. First we describe how our approach can be applied to design direct manipulation interfaces that rely on a mouse-based interface control. Then we present how we applied the interactor based design to quickly generate several variants of a gesture-based interface navigation control to demonstrate our approach. We compare the three most promising variants in a user test and report about the test results.

1. Introduction

Using gestures to control user interfaces could enable interaction in situations where hardware controls are missing like, for instance, wall-sized displays [6] and could support disabled people to interact with a computer when other controls fails.

Different to commonly used hardware like, for instance, mouse and keyboard setups or joysticks, gesture interaction does not suffer from a predefined and limited command setup. The amount of possible gestures is limited only by the users' creativity. Gesture recognition has been practiced since a long time driven by e.g. software that detects coloured gloves or even the bare hands using video cameras.

Recent frameworks such as Squidy [7] or the Open Interface Framework [8] support assembling multimodal controls including gestures out of components. These frameworks focus on connecting different interaction technologies by enabling the interconnection of device drivers and signal processing algorithms to form new kinds of multimodal interaction setups. But the specification of how an application is controlled with an assembled interaction setup is still done at the source code level.

The motivation of our work is to compare and evaluate different forms of interactions that can be performed with a multimodal interaction setup. We are

especially interested in understanding how changes of interactions within a certain combination of modes and media influence the interaction performance, and to learn about the interaction preferences of different user groups. Thus, we were confronted with the problem of quickly implementing different ways for interacting with the same application.

Recent approaches on model-driven development of user interfaces [2,9] have focused on defining processes and tools for the generation of interfaces for different devices but have been only applied them to generate multimodal interfaces to a very limited extend [13]. Further on, these approaches suffer from the fact that they introduce new languages and design processes through several abstract models that need to be learned by the designer. Additionally, they require anticipation skills to understand how manipulations in the abstract model design are reflected to the final generated interface.

Therefore, we decided to use state charts for interaction modelling, which have the advantage of being widely known and having a small basic vocabulary (states and transitions driven by events). Further on, there are already standards like e.g. SCXML and tools that support the graphical state machine design.

In this paper, we focus on presenting our approach of modelling interactions by assembling user interface interactors. We specify these interactors by state-charts. They can be directly executed to run a multimodal interface. The approach enables manipulation and extensive monitoring of the interaction, during user testing, since the state charts models are kept alive at runtime and the interaction can be logged and observed easily.

The paper is structured as follows. The next section introduces the interactor-based modelling of user interfaces and explains how their behavior is specified. Thereafter, in section 3, we describe how we can combine interactors to form different interactions by using declarative multimodal mappings to design basic direct-manipulation interfaces using a mouse as a running example.

In section 4 we apply our approach to design and test three variants of navigating through an interface using gestures and postures. We present the three most promising designs according to our modelling approach and report the results of some tests with users considering navigation efficiency and error rate. Section 5 presents related work and section 6, the conclusions.

2. Interactor-based Modeling

Using interactor models to describe interactive systems is an already well known and matured approach and has been extensively discussed, for instance, in [10]. They can be thought as an architectural abstraction similar to objects in object-oriented programming [10]. Several definitions of the interactor term have been proposed so far like for instance the PISA [11] or the YORK [3] interactor. Our approach relates to the latter one, which defines an interactor as:

“ a component in the description of an interactive system that encapsulates a state, the events that manipulate the state and the means by which the state is made perceivable to the user of the system.” [4]

Different to these early approaches that focused on the design of graphical interfaces, we assemble multimodal user interfaces by using interactors. An interactor can receive input from the user and send output from the system to the user. Each one is specified by a state machine that describes the interactor’s behaviour. A data structure is used to store and manipulate information that is received by each interactor or sent to others.
We use UML class diagrams to specify the data structure. Each interactor is associated with at least one class that needs to inherit from an abstract “Interactor”

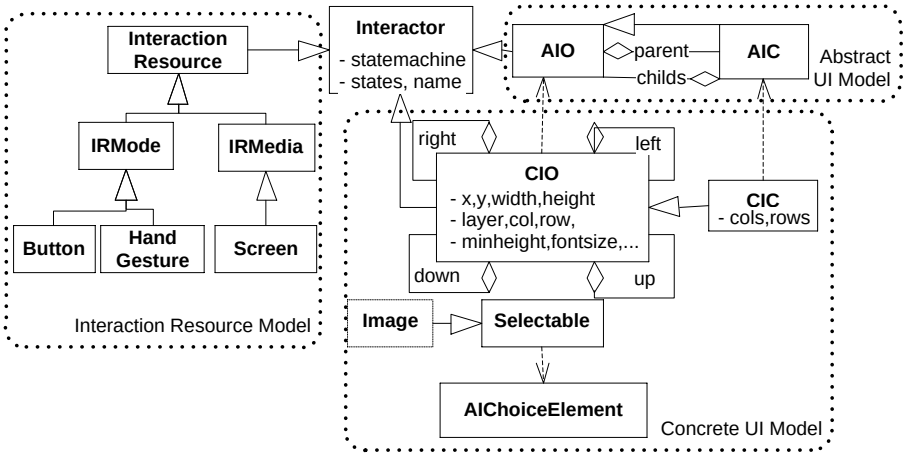

Fig. 1. Interactor relationships of the user interface models.

class. This class contains the basic data structure to support interactor persistence such as storing the current states of an interactor and encapsulates the functionality to load and execute the state machines.

Figure 1 shows the interactor class and its relations of the three basic interactors: the Abstract Interactor Object (AIO), the graphical Concrete Interactor Object (CIO), and Interaction Resource interactors, which describe the interaction capabilities of physical devices.

The Concrete User Interface (CUI) model is used to design a user interface for a certain mode or media. The Abstract User Interface (AUI) model describes the part of the interface that contains the behaviour and data specification shared among all modes or media. Interactors share the same database (which is currently a tuple space) to store and manage their data. Each interactor (that e.g. represents a button on a concrete graphical interface) is initiated several times to reflect all occurrences of e.g. a button in an application. This interactor is further on initiated at the AUI model level, and additionally for each supported modality at the CUI level. Additionally, at system run-time, interaction resource (IR) interactors are instantiated once for each

device that is connected to the system. IRs are, for instance, a gesture recognition that supports a certain set of gestures or postures or a WII remote control that supports a set of pre-defined gestures pointing to objects, and additionally has a joypad control.

2.1. Concrete Interface Interactors

Modeling a graphical interface requires the composition of screens by selecting appropriate interactors like buttons, combo-boxes or a menu for instance. Our basic interactor for all graphical CUI elements is depicted by the state machine in figure 2a.

The basic life-cycle includes a positioning phase in which the interactor calculates its screen coordinates, a presentation phase in that the element navigation is defined, as well as a disabled and hidden phase. During the presentation of a CIO interactor, a user can navigate to the interactor that then becomes "highlighted". This navigation is processed by receiving events (specified in square brackets). After the first CIO interactor has been highlighted, "up", "down", "left", and "right" events can be processed to navigate to the next interactor. Each of those events "unhighlights" the currently highlighted interactor, lets it search the interactor that should be navigated to (see "find" method of figure 2a) and sends to this interactor a "highlight" event.

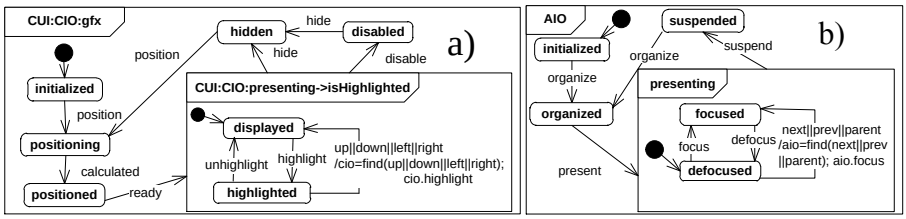

Fig. 2. The state machines of (a) the CIO and (b) the Abstract Interaction Object (AIO).

Every CUI interactor is connected to an abstract interactor of the AUI model that sums up all its mode and media independent behavior and data. Differently to USIXML, in our approach there is no redundancy of knowledge between AUI and CUI. This is also different to the CAMELEON-based MDDUI process [2] that specifies a continuous refinement from more abstract to more concrete models (and therefore the subsequent model contains the knowledge of the preceding ones) and ends up with a transformation to a final user interface (source or executable code). Instead of executing just the final user interface we require all user interface models during system run-time. The advantage of it is that we can define relations between modes and media even on the abstract level and can also add further ones during run-time, which is not possible without re-engineering final user interfaces generated by transformational processes.

Figure 2b depicts the AIO interactor that is linked to the CIO interactor. It has a similar structure like the CIO interactor but a different semantic. Since the AIO is used for all modalities it cannot consider a coordinate system where it can be positioned. It, therefore, supports only a basic navigation to its preceding and following AIO.

2.2. Abstract Interface Interactors

A multimodal setup combines at least one media with several modes. Thus parts of the interface interactors are output-related (e.g. sound or a graphical interface for instance), whereas others serve to address the modes used by the user to control the interface (like a joystick, speech, or a mouse, for instance). At the abstract, mode and media independent model level this has two implications. First,: a) input needs to be separated from output; and b) a continuous input or output (such as moving the mouse or a displaying a graph) needs to be distinguished from a discrete one. The following two sections describe how these implications could be addressed in an abstract user interface model. First by organizing all AUI interactors based on these findings in a static structure (a class diagram) and second, by specifying their behavior (by state charts).

2.2.1. Static AUI Structure

Figure 3 shows the class diagram of the AUI. The AIO class that is the parent class of all other classes of the AUI is derived from the "Interactor" class, which is the base class of all elements from all user interface models. It defines a name field that is a unique identifier (inside one model) and a state field that stores its current states

The AUI distinguishes, in general, between an abstract interactor input (AIIN) and output (AIOUT) elements, which could be either of a continuous or a discrete nature. Continuous outputs are a graphical progress bar or a chart, a tone that changes its pitch, or a light which can be dimmed, for instance. Discrete output could be implemented in different ways, such as written text, sound or a spoken note.

Grouping of interactors is handled by abstract interactor containers (AIC). AICs can be specialized to realize single (AISingleChoice) or multiple choices (AIMultiChoice) and are derived from the discrete output class. A discrete output class can be explicitly associated with every AIO, which we call AIContext. The AIContext is used to add contextual information that helps the user to control the interface or to understand the interface interactor. This could be, for instance, a tooltip, a picture or a sound file.

User input could be performed in a continuous manner as well. Some examples are: moving a slider or showing a distance with two hands. Discrete user inputs (AIINDiscrete) could be performed by commands that are issued by voice, or by pressing a physical or virtual button, for instance. We consider choosing one or several things from a list as user input as well (AISingleChoose and AIMultiChoose respectively).

In order to support distributing input and output to different devices, we strictly separate user input from output in the user interface models. There are some implications of this strict separation: For instance we require separating user choices to two different elements: Whereas a list of elements to choose from is considered as output (since it just signalizes the information that a user can choose together with a grouping of interactors), the current interactors to choose from are modeled separately as AIIN (by AISingleChoiceElement and AIMultiChoiceElement).

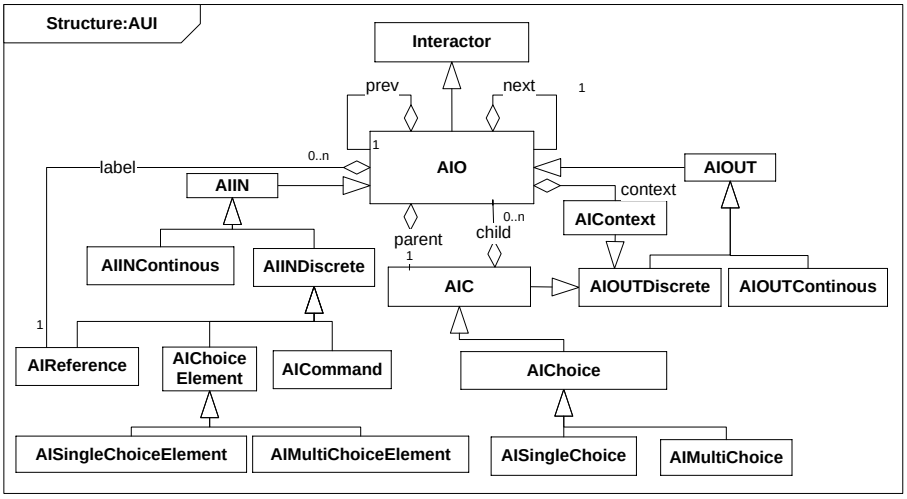

Fig. 3. Static AUI class diagram.

Finally, each AIO can be associated to an AIReference, which could be a label, a voice command, pressing a hotkey, pointing to it or by a categorizing color, for instance. AIReferences can be used for user interface navigation, e.g. to jump to a certain element or to relate elements by dragging and dropping them. A famous example is "put that there" where "that" and "there" are related to and finally resolved by AIReferences during multimodal fusion.

2.2.2. AUI Behavior Description

We complement each class of the static class structure with a state chart that describes the complete life cycle of a user interface element.
Looking at the AUI model, the basic AIO state machine that is used by all other AUI elements has already been described by figure 2b. For the sake of brevity we will discuss the most interesting state machines of the AUI only. But a comprehensive specification of the entire project is available online[1].

Figure 4a illustrates the more complex state machine of the AISingleChoiceElement that represents an element of a single choice list (AISingleChoice). It is specified to substitute the AIO state machine's "presenting" super state and allows being "chosen", "dragged" and "dropped". A single choice only permits one element to be chosen at a time. This behavior is specified in figure 4a by the "choosing" state that takes care of that as soon element is chosen and all other elements of the same AISingleChoice list get un-chosen. By the history symbol

[1] MINT AUI model specification:http:// <blinded for review>

"H" we define that the choice is persistent, even if the SingleChoiceElement gets deactivated and hidden from the user interface.

Figure 4b depicts the container for the AISingleChoiceElement that offers the capability to receive "drop" events if it is in the focus of the user (see "in(focused)"

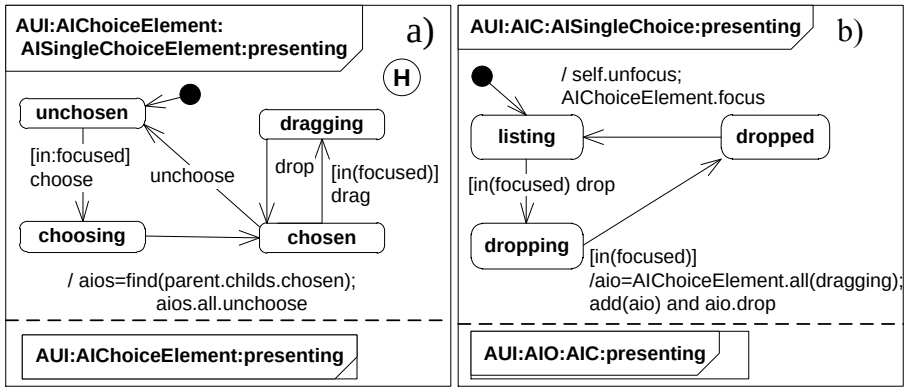

Fig. 4. The (a) AISingeChoiceElement and the (b) SingleChoice List statemachines.

condition). While in "dropping" state it retrieves all elements that are in "dragging" states and add them as children to it and notifies these dropped interactors that they have been successfully dropped.

2.3 Interaction Resource Interactors

For navigating through the interface several modes can be considered as appropriate. For the sake of simplicity, we intend to focus at the mouse. We design it as a composed interaction resource (IR) interactor and consists of a set of more basic interactors: a wheel, two buttons and a pointing interactor like shown in figure 5. The behavior part of the mouse resource interactors could be defined straight-forward like e.g. by the two state machines characterizing the pointer and a button. The pointer

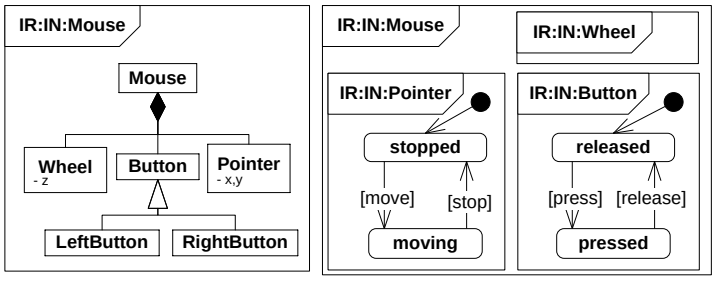

Fig. 5. The class diagram and state chart of a mouse.

could be “moving” or “stopped”. While the pointer is in the “moving” state, it communicates it’s x and y coordinates. The pointer is considered to be “stopped” if there is no movement for a particular time span. In the same manner, a mouse button can be described by a state machine to communicate its’ two states: “pressed” and “released”.

3. Model Synchronization and Multimodal Mappings

We use mappings as the glue to combine the AUI, CUI and interaction resource specifications. The mappings rely on the features of the state charts that can receive

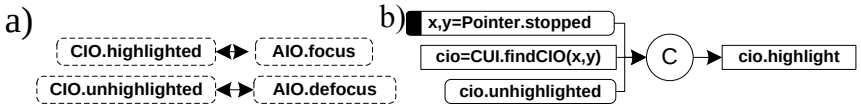

Fig. 6. (a) CUI with AUI synchronization. (b) Basic complementary pointer mapping.

and process events and have an observable state. Thus, each mapping can observe state changes and trigger events.

We distinguish between synchronization and multimodal mappings. The former ones synchronize the interactor’s state machines among different models of abstractions, like between task and AUI or AUI and CUI interactors. The latter ones are used to synchronize modes with media.

Synchronization mappings (figure 6a) are pre-defined together with the interactors and are automatically considered when the designer assembles a user interface by using these interactors. Multimodal mappings can be pre-defined as well (e.g. to support a certain form of interaction with a particular device or to implement an interaction paradigm like drag-and-drop) but are usually designed during application design (e.g. stating that a security critical command must be confirmed with a mouse click and a voice command). Figure 7b depicts an exemplary multimodal mapping specifying to change the highlighted graphical UI elements based on the current pointing position. Boxes with rounded edges stand for “observations” of state changes. Boxes with sharp edges are used to define backend function calls or the triggering of events. The mapping observes the state of the pointer interactor and gets triggered as soon as the pointer enters the state “stopped”. The “C” specifies a complementary relation, which requires all inputs of the C to be resolved for the mapping to get triggered. Therefore, as soon as the pointer has been stopped and coordinates of the stopped pointer could be retrieved, the findCIO function is called to check if there has been a CUI positioned on the coordinates, and if it is currently not highlighted. The complementary mapping is executed only if all 3 conditions can be evaluated. When that happens, then a highlight event is fired to the CIO.

Common interaction paradigms can be specified by combining basic interactors (introduced in the previous sections) with multimodal mappings.

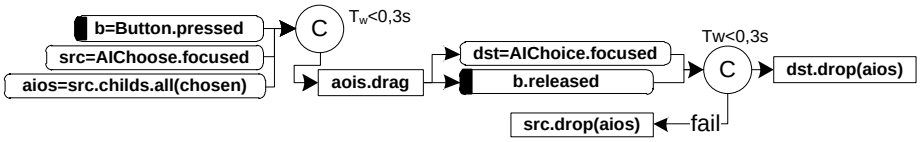

Fig. 7. Basic Drag-and-Drop mapping on the AUI model abstraction level.

Figure 7 depicts a mapping that specifies the drag-and-drop functionality for elements (AIChoiceElement) of an abstract list (AIChoice) on the AUI model level that we have presented earlier. The mapping gets triggered if (a) the mouse button is pressed (b) while a list (AIChoose) is "focused" (e.g. the mouse pointer is currently positioned over this list) and (c) at least one entry of the list is "chosen").

In the last sections we have introduced different interactors on the abstract and concrete model level, have synchronized them using synchronization mappings and connected them to an exemplary interaction resource interactor – a mouse - using multimodal mappings to explain our approach. The next section applies these concepts to design and evaluate several gesture and posture-based forms of controlling an interface. The rich spectrum of combining hand postures with movements offers great interaction possibilities but requires extensive user testing to figure out an optimal control with a sufficient control performance and a low error rate.

4. Modelling Multimodal Interaction with Gestures and Postures

Hand gestures are already widely used as a natural way of human-computer interaction [1]. But the definition of suitable gestures depends on various factors and extensive user testing. These factors include, for instance, the hand poses chosen, if one or both hands should be considered, the feedback of the interface when a gesture is recognized, delay on processing and communication, ergonomics, intuitiveness of the interaction, among other possible factors.

In our test cases we designed different gesture-based interactions and have considered four hand postures and one gesture that can trigger interface actions in two ways:

- A fixed hand posture: When the system recognizes a static gesture it triggers a single action and will wait for the next different posture to trigger the subsequent action. A fixed posture can have a temporal component like a ticker for instance to trigger the same event ~~in~~ at fixed intervals.
- A motion-related gesture: When a certain gesture is recognized, the triggered action varies according to the movement detected.

4.1. Specification of a Gesture-and Posture Interaction Resource

We based the implementation of the gesture recognition on the project of finger spelling recognition of sign language [12]. The system is able to recognize gestures using coloured gloves by doing segmentation based on the HSV colour space. We used images of 25x25 pixels and trained an artificial neural network Multi-Layer Perceptron (MLP) with an architecture of 625x100x4 neurons in each layer. This network is able to classify the different postures.

Figure 8a depicts a state chart that specifies our first variant of a gesture-based navigation control. It supports a basic movement to the next or the previous user interface element with the two gestures in top of figure 8b). The navigation speed is defined by a ticker that throws a "tick" event. For the first variant, we use a static ticker that throws a tick every second. Thus, if the user shows a "previous" gesture for instance, the state machine enters the super state "Predecessor" and starts with the initial "start_p" state. With every tick and as long the user remains showing the previous gesture, the "previous" state is entered with every tick event again and navigation step is performed by the interface. By showing the "select" gesture (figure 8b) the state machine switches to the "Command" mode, stops the navigation, and selects the current element. The user can, then, confirm the selection by performing the confirmation gesture.

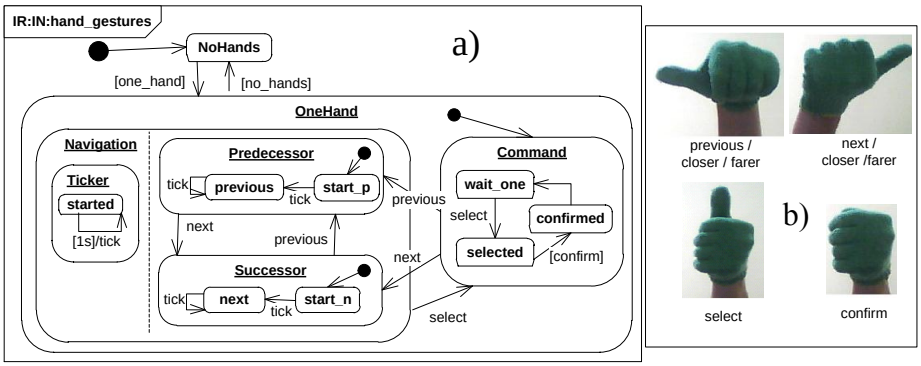

Fig. 8. (a) The first variant of the gesture-based navigation control.
(b) The four basic gestures we selected to navigate through the interface.

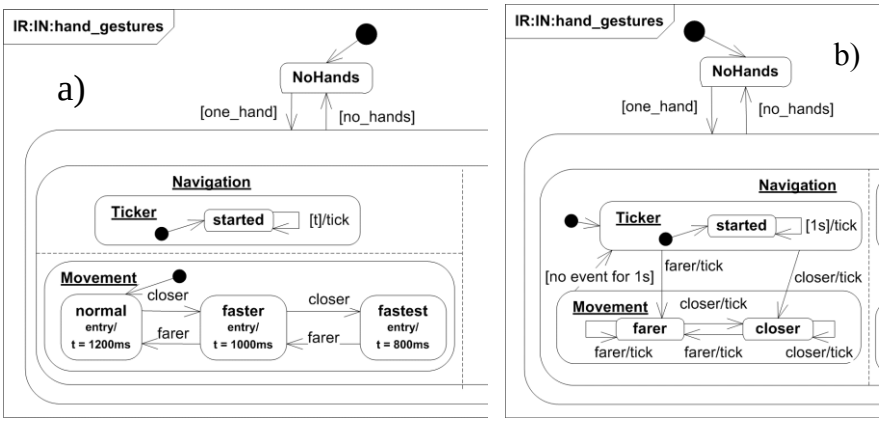

Fig. 9. (a) Second Variant: To manipulate the navigation speed the user can move his hand closer to or farer away from the camera.
(b) Third Variant: Instead of a timed navigating step every second, by every hand movement closer or farer away from the cam a navigation step is done.

The two state machines of figure 9 show two navigation alternatives. For the sake of brevity only the differences to the one of figure 8 are illustrated. Thus, the "Predecessor", "Successor", and "Command" states remain the same as already depicted in figure 8. The state chart of figure 9a) specifies the second variant and introduces an adjustable ticker. The time "t" between each "tick" could be adjusted by moving the hand that shows the previous or next gesture closer to or farer from the camera. Moving the hand closer to the camera results in a smaller ticker value (from 1200ms up to a speed of 800ms between the ticks). The variant 3 (figure 9b) additionally enables the user to switch between the timed ticker and by explicitly issuing ticks by moving the hand closer to or farer away from the camera. These movements temporary disable the ticker (since it is not modelled as part of a parallel state) and enable a quicker navigation just by moving the hand.

To define how the interface should react on the gestures we defined two main multimodal mappings that are depicted in figure 10.

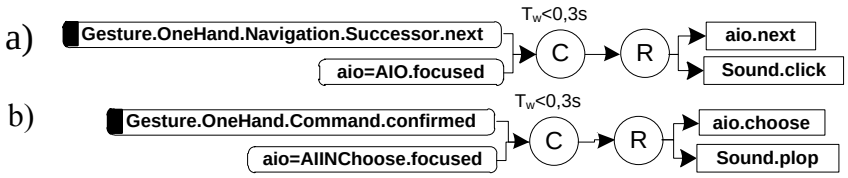

Fig. 10. Mappings to connect sound media and a gesture-driven control to the interface.

The first one (a) implements the navigation to the next element of the interfaces and plays a "tick" sound on each successful next movement. There are several other operators that can be used to define mappings that we have presented earlier [5]. In this exemplary mappings the complementary operator, (C), states observations that need to happen in a certain temporary windows (Tw) and the redundancy operator,

(R), publishes information to different media (such as changing to the next element and playing a "click" Sound at the same time. The Second mapping (figure 10b) shows a mapping that implements a selection of an element of a list that is in the current focus of the user.

Figure 11 shows a screenshot of the user interface that we used for testing the interface navigation. It shows an excerpt of a table of 5x5 cells, each consisting of a unique letter and a radio box that could be selected. It was the users' task to navigate from the top-left box "A" to a cell that is marked with a grey background and select the button of this cell. This navigation task was repeated 20 times for each variant with a fixed set of 20 different letters that were randomized on test start-up. There was a short-cut to navigate directly from the first "A" cell to the very last cell by issuing a "previous" gesture. All gestures were equipped with sound feedback to confirm a previous or next navigation step (a clicking sound), and a different one for the selection and confirmation gesture (figure 1b).

We tested this system with 16 persons (9 male / 7 female), most of them aged between 20 and 29 (2x30-39/1x40-59). During the user tests sometimes gestures have not been recognized or an error occurred. Users were not trained and just received a short explanation and demonstration of the supported gestures. Thus they generally used the first task in each test variant to get comfortable with the system. We removed these first step tasks from our test data set and ended up with a total set of 259 (variant 1), 262 (variant2), and 295 (variant 3) performed tasks.

The evaluation of a standardized questionnaire that was handed out after the test, revealed that all persons appreciated the sound support and 15 of the persons also

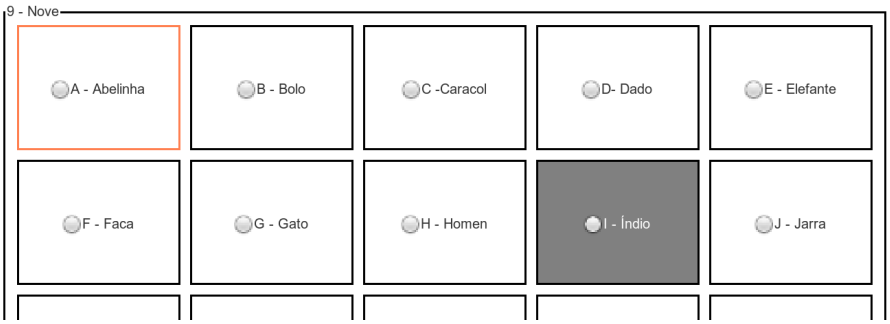

Fig. 11. Detail of a screenshot from the user interface for testing the gesture-based navigation.

appreciated the visual feedback of a moving orange box that showed the current position of the navigation in the interface (1 person was indifferent). With the first movement the navigation prototype recorded all movements and times to proceed to the final cell.

Cluster\ Variant	1	2	3
Navigation success without error	71,43% (187)	65,48% (170)	46,78% (138)
1-2 corrections	21,62% (56)	30,92% (81)	32,20% (95)
3-6 corrections	6,56% (17)	3,82% (10)	16,61% (49)
7 and more corrections	0,39% (1)	0,38% (1)	4,41% (13)
Success rate	97,50 %	98,57%	97,19%
Average time per step (ms)	1651	1382	1038

Table 1. Comparison of the three tested variants.

Although the overall success rate (stating that the user could navigate and select the correct cell) was similar (97%-98%) for all three tested variants, they differ in their navigation error rate and the overall navigation speed as illustrated in table 1.

We defined five different error classes:

1. The user was able to directly move to the final cell.
2. The user was able to navigate to the target cell but missed the target cell by one or two cells, which required the user to correct the navigation path.
3. The user missed the correct cell by 3 to 6 cells and needed to correct his navigation to end up on the correct cell.
4. The user missed the correct cell by more than six cells or tried several times until the user ended up selecting the correct one.
5. The user selected and confirmed the wrong cell.

As shown in table 1 with the first variant the user made less navigation errors, but the overall average navigation time to reach the correct cell was high. For the last variant it was the opposite: The users could benefit from a high navigation performance but it included a high navigation error rate that required re-navigating to end up at the correct cell.

In the questionnaire we asked the users which variant that was the easiest variant to use for them. The majority (9 persons) preferred the second variant which was seen as a compromise between adjusting the speed with a low error rate (4 persons preferred the third, 2 persons the first variant). The time for the entire test was around 12 minutes for each person. Therefore we were although interested at learning which variant was experienced as less exhausting by the users. The majority of the 16 people (11 of them) experienced the third variant as less exhausting (4 persons marked the second variant as less exhausting).

5. Related Work

Recent research has focused on evaluating different forms of gesture-based interaction from a distance to control interfaces displayed on wall sized displays [6].

Several frameworks have been proposed to ease the creation of multimodal interaction controls like the Open Interface Framework [8] or Squidy [7].

But these frameworks are limited to assemble the current multimodal control (like e.g. combining gestures with sound or choosing between different technologies for gesture recognition). Currently a developer is still required to connect the multimodal control to a specific application, which still need to be performed at the source code level. Further on, the current interaction inside the application that specifies what happens if the user issues certain gesture in a certain state of the application is still a programming task as well.

The model-driven development of user interfaces has been around for a long time to tackle this issue and resulted in several connected design models that have been summarized by the Cameleon Reference Framework [2] and by user interface languages such as USIXML [10]. 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. Multimodal systems have been addressed by these approaches only to a very limited extend [13].

State machines have been widely used in Case-Tools and are already standardized as part of UML and the W3C multimodal framework with the SCXML standard and therefore reduce the entry barrier for developers as various tools are already available to design state machines.

6. Conclusion

With our approach of specifying interaction based on state machines, different forms of interactions could be efficiently designed and generated since only the state machines have to be changed. The mappings to describe the overall multimodal interaction have to be changed only if more than one mode of interaction needs to be manipulated.

To demonstrate our approach we designed three variants of navigation through an interface by using hand gestures and postures. We demonstrated that just the declaratively modelled start machines had to be manipulated to change the way of interacting with our prototype on a multimodal manner.

The test results showed that people prefer a navigation variant that allows adjustments of navigation speed. Even if people experienced a high overall success rate, they prefer a navigation that requires less corrections of the navigation. Although the variant that requires more correction of navigation was overall faster to perform successfully than the variant 2 with less navigation errors, the users preferred the variant 2. The variant 3 requires the hand to move forward and backwards to move the cursor forward and was stated as less exhausting compared to the other two variants.

References

1. J.-L.Bernardes Jr., R.Nakamura, and R.Tori. Design and implementation of a flexible hand gesture command interface for games based on computer vision. In *Proceedings of the 2009 VIII Brazilian Symposium on Games and Digital Entertainment*, SBGAMES '09, pages 64–73, Washington, DC, USA, 2009.
2. G.Calvary, J.Coutaz, D.Thevenin, Q.Limbourg, L.Bouillon, and J.Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289–308, 2003.
3. D.Duke, G.Faconti, M.Harrison, and F.Paternó. Unifying views of interactors. In *AVI '94: Proceedings of the Workshop on Advanced Visual Interfaces*, pages 143–152, New York, NY, USA, 1994. ACM, ISBN:0-89791-733-2.
4. D.J. Duke and M.D. Harrison. Abstract interaction objects. *Computer Graphics Forum*, 12(3):25–36, 1993.
5. S. Feuerstack, E. Pizzolato; Building Multimodal Interfaces out of Executable, Model-based Interactors and Mappings; HCI International 2011; 14th International Conference on Human-Computer Interaction; J.A. Jacko (Ed.): Human-Computer Interaction, Part I, HCII 2011, LNCS 6761, pp. 221—228. Springer, Heidelberg (2011), 9-14 July 2011, Orlando, Florida, USA.
6. F. Fikkert. *Gesture Interaction at a Distance*. PhD thesis, Universiteit Twente, Centre for Telematics and Information Technology, 2010.
7. W. A. Kaenig, R. Raedle, and H. Reiterer. Interactive design of multimodal user interfaces - reducing technical and visual complexity. *Journal on Multimodal User Interfaces*, 3(3):197–213, Feb 2010.
8. J.-Y. L Lawson, A.-A. Al-Akkad, J.Vanderdonckt, and B.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.
9. Q. Limbourg, J.Vanderdonckt, B.Michotte, L.Bouillon, and V.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.
10. P. Markopoulos. *A compositional model for the formal specification of user interface software*. PhD thesis, Queen Mary and Westfield College, University of London., 1997.
11. F. Paterno. A theory of user-interaction objects. *Journal of Visual Languages & Computing*, 5(3):227 – 249, 1994.
12. E. B. Pizzolato, M. Santos Anjo, and G. C. Pedroso. Automatic recognition of finger spelling for libras based on a two-layer architecture. In Proceedings of the 2010 ACM Symposium on Applied Computing, pages 969–973, 2010.
13. A.Stanciulescu, Q.Limbourg, A.Vanderdonckt, B.Michotte, and F.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.