Building Multimodal Interfaces out of Executable, Model-based Interactors and Mappings

Sebastian Feuerstack, Ednaldo Pizzolato

Universidade Federal de São Carlos, Departamento de Computacão,
Rod. Washington Luis, km 235 - SP, Brazil
sfeu@cs.tu-berlin.de, ednaldo@dc.ufscar.br

Future interaction will be embedded into smart environments offering the user to choose and to combine a heterogeneous set of interaction devices and modalities based on his preferences realizing an ubiquitous and multimodal access. We propose a model-based runtime environment (the MINT Framework) that describes multimodal interaction by interactors and multimodal mappings. The interactors are modeled by using state machines and describe user interface elements for various modalities. Mappings combine these interactors with interaction devices and support the definition of multimodal relations. We prove our implementation by modeling a multimodal navigation supporting pointing and hand gestures. We additionally present the flexibility of our approach that supports modeling of common interaction paradigms such as drag-and-drop as well.

1 Introduction

Future interaction will be embedded into smart environments offering the user to choose and to combine a heterogeneous set of interaction devices and modalities based on his preferences realizing an ubiquitous and multimodal access.

Such a flexible multimodal interaction requires user interface dialogues that are adaptive regarding the utilized modalities and their alteration. But current multimodal interfaces are often implemented for a predefined interaction hardware setup. Like for instance for the common mouse and keyboard setup of a computer, for a controller of a game console that supports gesture detection, or for the control wheel with speech feedback for navigating through the menus of a cockpit control in cars.

To tackle the complexity of designing and implementing multimodal interfaces, recent research has been focused on three main aspects. First, by model-driven user interface development (MDDUI) that describes a process for the tool-driven design of multi-platform interfaces though several models. MDDUI projects, such as [1] demonstrated basic multimodal applications. Second, executable models have been introduced into MDDUI [2, 3]. They enable to adapt the interaction modalities to the actual context of a user in a smart environment. Finally, the characteristics of different modalities and their relations have been investigated in depth [4] and platforms have been developed that support building multimodal interfaces out of components [5, 6]. To our knowledge these approaches of the first two categories support the design of multimodal interfaces to a limited extent only. They enable modeling and adaptation between equivalent modalities and do not address multimodal fusion. The approaches in the third category are focusing on multimodal fusion between several modalities

but restrict their findings to a particular use case for a fixed setup of modalities (like a cockpit for instance) or on command and control-interfaces.

In this paper we present our approach on modeling user interface dialogues with connected executable interactors. Each interactor consists of a set of declarative models that describe both, the structure as well as the behavior. Interactors are connected to form dialogues to support different multimodal hardware setups by multimodal mappings.

2 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 based on pre-modeled interactors. Model-driven development has resulted in several connected design models that have been summed up by the Camelon Reference Framework [7] and in user interface languages such as USIXML [8]. But MDDUI has been applied to develop interfaces for pre-defined platforms only such as for example for XHTML+Voice browsers [1] or game consoles. Multimodal systems have been addressed by these approaches only to a very limited extend [1] and without addressing building interfaces out of complementary modes. Different to these approaches our interactor-based interfaces can be flexibly extended to new modes and media just by adding new interactors and mappings to a running system. Our approach is inspired by the findings of the iCARE platform [6] 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.

Further on, we considered earlier work that we have implemented in the MASP platform [2] to support executing interfaces based on the designed models without the need for a further implementation phase. Different to the MASP that currently supports modeling equivalent multimodal relations only and is bound to the Eclipse Modeling Framework, we rely on state machines. State machines have been widely used in Case-Tools and are already standardized as part of UML and the W3C multimodal framework and therefore reduce the entry barrier for developers.

Finally, different to earlier work that applied state machines for the actual design of the interface dialogues, we use them to describe interactors. The interactor abstraction is mature [9] and has been for instance recently used to specify HTML and Java [10]. By using interactors, the interface design can be done in the same manner as it is actually done by UI builders to compose interfaces based on a toolkit.

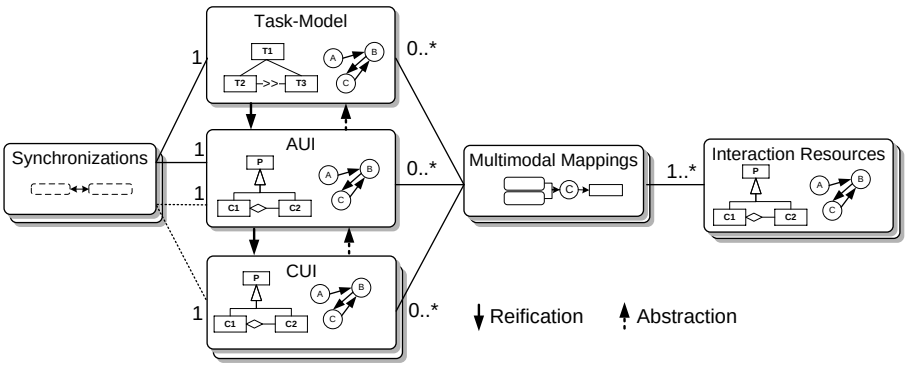

Fig. 1. The relations between the principal models.

3 Approach

Figure 1 illustrates the basic conceptual components of our approach as well as the relations between the models that we use to generate multimodal interfaces. Our approach supports the general MDDUI process that starts with a task analysis- and design to separate user from system tasks. An abstract user interface (AUI) model is used to specify the (modality-independent) interface elements that need to be realized for every interaction setup. The specific aspects of each concrete modality are captured by a concrete user interface (CUI) model. Additionally, and different to other approaches like CAMELEON[7] and USIXML[8], we describe the capabilities of the interaction resources (devices and modalities) in declarative models as well and target our approach to the design of multimodal interfaces that can be adapted to different modalities at system runtime. Further on we define mappings in a separate model and use them to glue all the models together at runtime [2].

Each design model is referring to interactors to describe an interface. Interactors are pre-defined components that describe the behavior of each model's atomic elements. We are using self-executable, reactive software agents to encapsulate interactors and specify their behavior by state machines. In the task model design for example, the developer assembles application and interaction task interactors that consist of an own lifecycle at runtime (initiated, executing, stopped, inactive, and done) like described earlier for instance in [11].

At design time, state machines (which are specified as declarative models) complement the user interface markup language with a behavior specification, which is currently missing in actual approaches like e.g. USIXML [8]. State machines are a well-known and a straight-forward way of describing and running reactive systems. They have been applied in several case tools that can generate source code from UML state charts[1] and will be standardized by the W3C to support speech-application design soon[2].

In the following sections we describe how multimodal applications can be designed and implemented with our Multimodal Interaction (MINT) Framework[3] and subsequently enhanced to support different multimodal setups. As a running example we focus on a multimodal dialogue navigation prototype. To proof our approach we applied several common interaction patterns to evaluate if they can be implemented with our approach. One of them, the drag-and-drop paradigm, will be presented in the end of this paper. The MINT framework does not require starting with a particular model. Thus, we start our example by modeling a concrete (graphical) interface and add the AUI layer for integrating further modalities thereafter.

3.1 Modeling Navigation for Graphical Interfaces

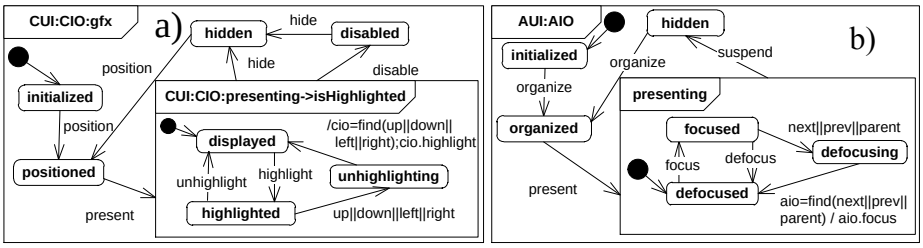

Fig. 2. a) Graphical CIO state machine. b) AIO state machine.

Modeling a graphical interface requires the composition of screens by selecting appropriate interactors like buttons, combo-boxes or menus for instance, which we call Concrete Interaction Objects (CIO). Our basic interactor for graphical concrete interface elements is depicted by the state machine in figure 2a.

---

[1] OMG Unified Modeling Language (OMG UML), Superstructure Version 2.2".http://www.omg.org/spec/UML/2.2/Superstructure/PDF/, last accessed 20/12/2010

[2] State Chart XML (SCXML): W3C Working Draft 16 December 2010http://www.w3.org/TR/scxml/, last accessed 20/12/2010

[3] The MINT platform is available as open source at http://www.multi-access.de, last accessed 20/12/10

We support inheritance for interactor design and therefore the CIO interactor serves as the basis for all other, more specific interactors of a graphical interface. The basic life-cycle consists of a positioning, presentation, hidden and disabled phase. In the positioning phase, each CIO interactor calculates its absolute screen coordinates by identifying its neighbors that can be reached by interface navigation. During the presentation phase the interactor is part of the active user interface (UI) and reacts on navigation events. The user can therefore navigate to the interactor that gets then "highlighted".

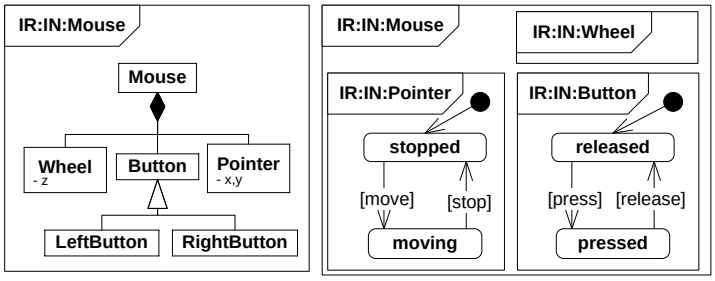

Fig. 3. The class diagram and state chart of a mouse.

After the first CIO interactor has been highlighted, up, down, left, and right events can be sent to an interactor that is highlighted to navigate to the one. Each of those events "unhighlight" the receiving interactor, let it search the event's target interactor to that a "highlight" event will be sent.

For navigation several modalities can be considered as appropriate. The most basic one is a mouse that we design as a composed interaction resource (IR), which consists of a set of more basic interactors: a wheel, two buttons and a pointing interactor like shown in figure 3. The behavior part of the mouse could be defined straight-forward by two state machines characterizing the pointer and a button. The pointer could be in the states "moving" or "stopped". While the pointer is in the "moving" state it communicates its 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".

3.2 Mapping Specification for UI Element highlighting while pointing

Like depicted in figure 1 we use mappings as the glue to combine the AUI, CUI and interaction resource specifications. The mappings rely on the features of the state machines that can receive and process events and have an observable state. Thus, each mapping can observe state changes and send events.

Fig. 4. a) Basic pointer mappings. b) CUI with AUI synchronization.

Figure 4a shows the principal mapping we are using to change the highlighted graphical UI elements based on the actual pointing position. A mapping consists of Boxes with rounded and sharp edges. The former one define reactions on state changes of an interactor, the latter ones define system function calls or events. The mapping of figure 4a observes the state of the Pointer IR and gets triggered as soon as the pointer enters the state "stopped". The "C" specifies a complementary mapping, which requires all inputs of the C to be resolved for the mapping to start. The second mapping "assigns" (A) the observation of a moving pointer to an event that ensures that no CIO is highlighted during pointer movements.
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 CIO positioned at these coordinates and if it is currently not highlighted. The complementary mapping only executes if all three conditions can be evaluated and in our example fires a "highlight" event to the corresponding CIO.

3.3 Connecting further Modalities

To add further, no-graphical media like sound or modes like gestures to control the interface we require the AUI model to keep them synchronized. Figure 2b depicts the basic abstract interactor object (AIO) that serves as a base for all other abstract interactors. Whereas it is quite similar to the basic CIO of figure 2a its semantics are different and independent of any modes or modalities. Thus, the AIO state machine processes the ordering of elements for a basic navigation. It supports browsing through all elements using "previous", "next" and "parent" commands (in contrast to the graphical-oriented navigation that supports directions like "up"/"down" based on their calculated coordinates). Further on, the highlighting feature of the CIO becomes a "focus" in the AIO describing that the actual interactor is in the focus of the user. Now, both interactors can be synchronized using the two bi-directional mappings of figure 4b.

Now that we are having the navigation synchronized between the graphical CIO and the AIO we have two options to connect further modes and media. First, we can attach them to the AUI interactors so that we can benefit from the already existing synchronizations between AUI and CUI. Second, we could attach them to another CUI and add further synchronizations between the new CUI model and the AUI. The mappings shown by figure 5a add gesture-based navigation and sound media and implement the former case and connect directly to the AUI. They require a gesture control interactor that supports two modes: When using two hands, like depicted in

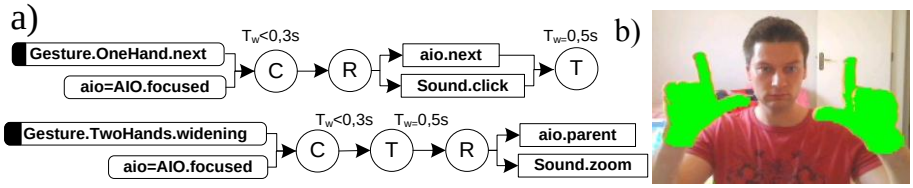

Fig. 5. a) Mappings to connect sound media and a gesture-driven control to the interface. b) Zoom gesture for widening and narrowing the user's focus.

figure 5b, the distance between the hands is measured and interpreted as "narrowing" or "widening". With only one hand detected, the interactor distinguishes two postures: one for issuing "next" and another one for issuing a "previous" command.

The first mapping of figure 5a waits for the next posture to appear while an interactor is in the state "focused". If this is the case the mapping gets executed and sends to events: A "next" event to the AIO (aio) that is in the focus of the user and a "click" event to a sound interactor that can generate a "click" sound. Both events are specified as redundant (R), which requires both by processed successfully. Alternatively they could be marked as equivalent (E), which requires only at least one of them (at least the aio.next event) to be processed successfully.

After both events have been fired the mapping waits (T) for half a second to re-initiate itself. Thus, if the user remains his hand in the "next" posture, the mapping gets fired every half second. The second mapping of figure 5a to widen the focus to the parent interactor works in the same manner but does not require a timeout. A threshold inside the gesture interactor defines at which sensitivity the "widening" event is fired (this depends on the camera resolution and the users ability to keep the distance between their hands stable).

3.4 Mappings to specify Interaction Paradigms

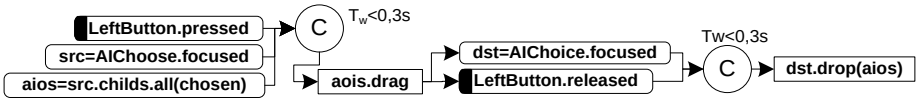

Fig. 6. Basic Drag-and-Drop mapping on the AUI model abstraction level.

Our approach of describing interactions of composed interactors based on state machines and mappings turned out to be very flexible. Not only multimodal relations can be addressed by the mappings but also interaction paradigms like a "double-click" or "drag-and-drop" as well. Figure 6 depicts a mapping that specifies the drag-and-drop functionality for elements (AIChoiceElement) of an abstract list (AIChoice) on the AUI model level that is bound to the left button of a mouse. The AISingleChoice interactor (figure 7a) introduces a parallel super state to the presenting state of the AIO interactor. Additionally to the capability of gaining the user's focus, this super state describes the ability of a list to receive list elements that are dropped to the list.

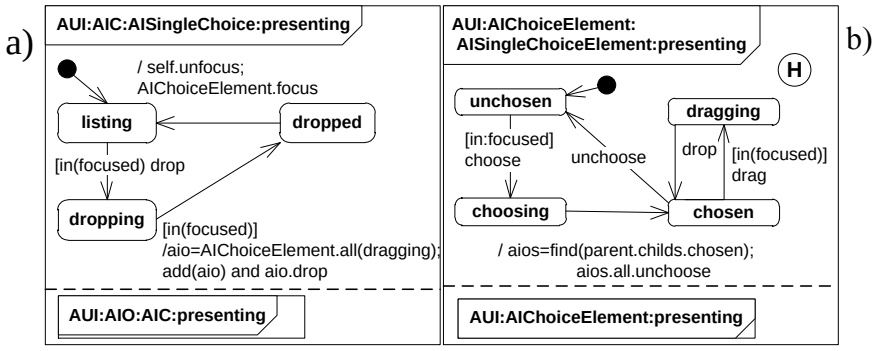

Fig. 7. a) AISingleChoice, the container for all SingleChoiceElements.
b) SingleChoiceElement AIO state machine.

Like shown in figure 7b, the dragging feature is not part of the list interactor but implemented by the interactor that describes an individual list element's behavior. There, we introduce a parallel super state as well that adds the capability of an element do be chosen and dragged. An element can only be chosen, if it is in the user's focus and it takes care, that all other list elements get "unchosen" if they are part of a single choice list. Given these interactor specifications, the drag-and-drop mapping of figure 6 is easy to understand: It waits for the left mouse button to be pressed and a list of AIChoiceElements that are in the state "chosen" to issue a "drag" event to them. As soon as the button is released it fires the drop event to the list that is currently in the user's focus.

Acknowledgments. The author is grateful to the Deutsche Forschungsgemeinschaft (DFG) for the financial support of his work.

References

1. 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.
2. Marco Blumendorf, Grzegorz Lehmann, Sebastian Feuerstack, and Sahin Albayrak. Executable models for human-computer interaction. In T. C. Nicholas Graham and Philippe Palanque, editors, *Interactive Systems. Design, Specification, and Verification: 15th International Workshop, DSV-IS 2008 Kingston, Canada, July 16-18, 2008 Revised Papers*, pages 238–251, Berlin, Heidelberg, 2008. Springer-Verlag.
3. Alexandre Demeure, Gaelle Calvary, and Karin Coninx. Comet(s), a software architecture style and an interactors toolkit for plastic user interfaces. pages 225–237, 2008. Design, Specification, and Verification, 15th International Workshop, DSV-IS 2008, T.C.N. Graham & P. Palanque (Eds), Lecture Notes in Computer Science 5136, Springer Berlin / Heidelberg, Kingston, Canada, July 16-18, 2008, pp 225-237.
4. 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.
5. 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.
6. Bouchet, J., Nigay, L., & Ganille, T. (2005). The ICARE Component-Based Approach for Multimodal Input Interaction: Application to real-time military aircraft cockpits. In Conference Proceedings of HCI International 2005, the 11th International Conference on Human-Computer Interaction, Las Vegas, Nevada, USA, July 2005. Lawrence Erlbaum Associates.
7. 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.
8. 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.
9. David Duke, Giorgio Faconti, Michael Harrison, and Fabio 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.
10. Hallvard Traetteberg. Dialog modelling with interactors and UML Statecharts - A hybrid approach,. *In Proceedings of 10th International Workshop*, DSV-IS 2003, Funchal, Madeira Island, Portugal. Lecture Notes in Computer Science, Springer-Verlag, 2003, pages 346–361, 2003.
11. Birgit Bomsdorf, The WebTaskModel approach to web process modelling, *In Proceedings of the 6th international conference on Task models and diagrams for user interface design (TAMODIA 2007)*, November 07-09, 2007, Toulouse, France