# Comparing the Multimodal Interaction Technique Design of MINT with NiMMiT

Sebastian Feuerstack
OFFIS – Institute for Information Technology, Escherweg 2
26121 Oldenburg, Germany
feuerstack@offis.de

Ednaldo Brigante Pizzolato
Universidade Federal de São Carlos
Rodovia Washington Luís, km 235
São Carlos - São Paulo – Brasil
ednaldo@dc.ufscar.br

## ABSTRACT
With new sensors that can capture hand and body movements in 3D, novel interaction techniques gain importance. But development of new forms of interaction is highly iterative, depends on extensive user testing and therefore is expensive. We propose a model-based notation using statecharts and mappings to ease multimodal interaction technique design. This model-based specification can be used to communicate designs, for evaluation and to enable re-use. Our contribution continues previous research on model-based interaction technique design considers multimodal interaction and addresses problems like the state explosion, error management and consideration of output modalities mentioned by earlier research. We evaluate our notation by comparing it with NiMMiT referring to the same use case to identify similarities, strength and problems.

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

## General Terms
Design, Human Factors, Languages.

## Keywords
Interaction Techniques, Interaction Metaphors, Multimodal Interaction, Model-Based Design.

## 1. INTRODUCTION
The great variety of different devices to access and control services in smart environments can be used to improve human-computer interaction to be more natural by considering and combining several modes of access in one interaction. Decades of research have been performed to figure out suitable interaction techniques for certain control modes. After the desktop and WIMP interaction metaphor have been introduced, the emerging variety of devices continuously substitutes classic interaction techniques that were optimized for the desktop PC. The Post-WIMP term sums up the trend to design interaction techniques specifically to a certain combination of an application and one or more interaction devices and their control modes.

Non-traditional interfaces that consider modes like speech or gestures and media such as augmented and hyper-reality offer a high degree of freedom in interaction design but make the design process cumbersome since extensive user testing is usually required to figure out an efficient and accessible way of interaction.

High-level description languages have been developed to ease the design of multimodal interaction techniques by providing means to design, prototype, communicate and store interaction techniques to be re-usable. Model-based user interface design is a widely adopted practice to shorten development-cycles [12]. The Cameleon Reference Framework [4] revealed several shared models between different proposals. Most of them start with a task model, and then follow an incremental abstract-to-concrete transformational process through several transitional models, such as an abstract one, a modality independent one and several concrete ones to consider specific platform capabilities.

Recent research proposes to execute models instead of transforming them into executable code [11]. This has the advantage that models can be still edited while already in execution, facilitates experimental prototyping and model observation at runtime to analyze user behavior.

In this paper we present our approach to design multimodal interaction techniques that we present and discuss in line with preceding research about interaction techniques descriptions, such as in InTml [9] and ICO [14] but focus on a comparison with NiMMiT [1], which is to the best of our knowledge the most recent proposal.

Like NiMMiT, our approach combines a data flow with a state-chart description with the overall goals to allow designers communicate about the functionality of an interaction technique and to offer a platform that can directly execute the design models to enable rapid prototyping and comparison of different interaction technique variants for user testing.

The intention of our model-based MINT design notation [8] is to consider the problems and drawbacks that have been identified by the authors of NiMMiT in [16]:

1. The state explosion for complex interaction technique designs that we tackle by separating the design into two types of models: statecharts for mode and media interactor design and mappings to define the data flow.

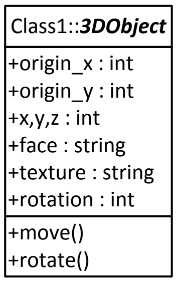

**Fig. 1.** Static model of the 3DObject.

2. The missing consideration of undo steps. These are required to recover from failed action that the interaction technique depends on while it is applied; we include the specification of undo steps as a notation feature in the data flow-based multimodal mappings.
3. The missing incorporation of output modalities as part of the model abstraction. In NiMMiT output modalities are added by custom tasks, which need to be coded or scripted. We separate the modality design from the interaction technique design and therefore make both, output media formats as well as the control modes characteristics re-usable and referable for the interaction technique design.

The paper is structured as follows: In the next section we present the basic idea of our approach: First, the interactor-based modeling of interfaces and interaction resources followed by an introduction about the multimodal mapping concept. They are used to synchronize models and combine interactors. We use the same interaction technique use case as it was presented for describing and evaluating NiMMiT: the Object-in-Hand metaphor to explain our approach and to ease the comparison between NiMMiT and MINT. Further notations are only discussed briefly, since they have been already compared to NiMMiT in [3]. Then, in the next section we compare both approaches in detail and list downsides and problems of both proposals. The comparison considers: the design models, the capabilities to model multimodal interactions, the capability to reuse existing models, error handling and iterative prototyping features. Finally, in the last section we conclude our approach.

## 2. MODELING INTERACTION TECHNIQUES

In [1] two different approaches for interaction modeling have been identified: State-driven notations based on formal mechanisms of finite state machines, such as the Harel State Tables [10] and data-driven approaches that define activities and their connections using data input and output ports. Following [1] in the interaction design domain state-driven notations have been used in ICO [14] and the Interact Objects Graph [5], whereas data flows have been applied in inTml [8], and Icon [6]. A data flow notation is also used in iStuff [17] to interconnect devices and in the Open Interface Framework [18] to compose multimodal interactions by combining device drivers, with filters and fusion algorithms. In data flow notations the control flow is managed by the data and therefore the major part of the control is defined inside the interconnected components. State-driven notations explicitly define the control by transitions and conditions. In [1] the authors

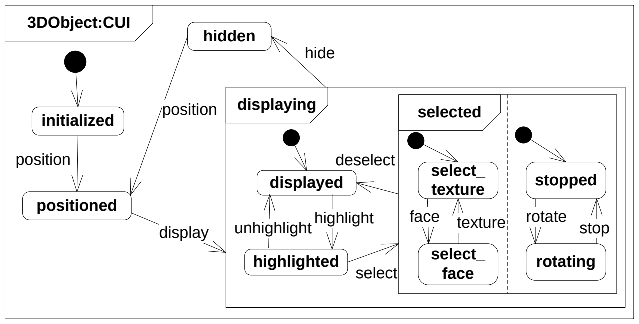

**Fig. 2.** Statechart of a 3D object to describe the behavior of its graphical presentation.

mention a study that revealed that the lack of data handling is a restricting aspect for interaction technique modeling. Further on, they state that with data flow notations the enabling and disabling of parts of an interaction technique can only be designed by complex structures. These motivated them to propose NiMMiT that merges both types of notations into “easy-to-learn and easy-to-read diagrams” [1].

## 3. MINT MULTIMODAL INTERACTION MODELING

Our approach on modeling multimodal interaction in based on three types of diagrams: UML class diagrams, SCXML-based statechart models, and a custom flow chart-based mapping notation that interconnects statecharts and is based on the multimodal relationships defined by the CARE properties [15].

As a running example to explain our approach we refer to the same interaction technique that has been used as a case study in NiMMiT, the Object-In-Hand metaphor that we briefly present in the next section. A detailed description and evaluation of this metaphor can be found in [2].

### 3.1 The Object-In-Hand-Metaphor

The Object-In-Hand metaphor is a two-handed interaction technique that is utilized in 3D worlds where common tasks are navigation, object selection and object manipulation.

The technique eases object manipulation and implements a sequential process: First the user points to an object, which then gets highlighted. The user confirms the selection by a click using the pointing device button and then uses the non-dominant hand

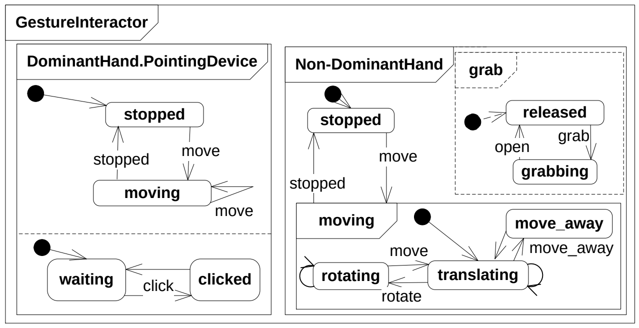

**Fig. 3.** Statechart of the control modality.

to move the object to the center of the 3D scene by a grab gesture (a closed hand) towards the dominant hand. As long as the closed hand remains near the dominant hand holding the pointer, the object can be manipulated. In the NiMMiT case study changes of object faces and textures can be manipulated with the dominant pointing hand and object rotation is supported by rotating the non-dominant while it remains closed. Finally, with a throw away gesture the centralized object can be moved back to its original position in the 3D scene.

## 3.2 Interactor Modeling

With the MINT framework, multimodal user interfaces are assembled by interactors. 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. Each interactor is specified by a statechart that describes the interactor's behavior and uses a data structure to store and manipulate data that it receives from or sends to other interactors. We use UML class diagrams to specify the data structure. Figure 1 depicts the class definition of the 3DObject that contains the attributes and member functions that can be accessed by the mappings.

Figure 2 presents a concrete interactor that specifies the behavior of a 3D object representation that can be manipulated using the Object-in-Hand metaphor. During the application and before its presentation in the 3D scene it is positioned based on the pre-set object coordinates. While it is displayed as part of the scene, it can be highlighted and then be selected for manipulation. During the object selection either the face or the texture can be set and the object can be rotated simultaneously to its manipulation.

A control mode, like the two-handed gesture interactor for performing the Object-In-Hand metaphor, depicted in figure 3, is specified by a statechart as well: It distinguishes between a dominant and a non-dominant hand that it processes in parallel. It assumes that the user interacts with a pointing device in the dominant hand that additionally offers a button. For the implementation of the interactors, the statecharts get instantiated as state machines and implement the API to the device driver to access a modality or to the final interface representation of the 3D object. Different to data flow-based implementations that black-box their components [18], [17] the statechart represents a model to the driver to reflect the component's relevant behavior. It can be used for the interaction by querying for active states or by subscribing to get notified about state changes. Further on, the data flow can be controlled, which we describe in the following section.

## 3.3 Multimodal Mappings

Multimodal mappings connect interactors. They are defined at three different levels of abstraction:

- *Application level*: A multimodal mapping that connects specific interactor instances; e.g. "If the 'confirm reservation' button is pressed then close the window."
- *Interactor level*: A mapping that is pre-defined together with the interactor designs ("the meta level") to be used later on for concrete application development. E.g. "Each time a button is pressed play a 'click' sound".
- *Metaphor level*: A mapping that specifies an interaction paradigm, such as e.g. a "drag-and-drop" or a "double-click" that was prepared to work with the designed interactor set.

Figure 4 depicts the relevant mappings to specify the Object-In-Hand interaction technique to manipulate the 3D object interactors using the gesture interactor. We use a custom notation: Boxes with rounded edges stand for "observations" of state changes. Boxes with sharp edges define "actions", which are backend function calls (FC), event triggers or the activation of another mapping. Observations and actions are connected by an operator that specifies a relation between the observations. The default relation is a sequential one „S", which defines a top-down sequential processing of the observations. Actions are always processed sequentially. Further operators are based on the CARE properties [17]: A complementary "C" relation set observations that complement each other and need to be retrieved in a temporal window Tw. An assignment "A" requires a set of observations of a specific mode, redundant "R" relations require at least two observations from different modes, to execute the actions, and equivalent "E" relations define alternative observations.

The Object-In-Hand interaction technique is composed by five multimodal mappings: Two interactor level mappings ("Highlighting", "Selection") and three mappings at the metaphor level.

Both interactor level mappings define the way that the pointing device is used to control the concrete media representation of the interface. Interface highlighting happens if the pointing device is directed to a 3D object. Thus, if the pointing device remains for a certain amount of time in the same position, it is assumed to be "stopped", which evaluates the first observation of the highlighting mapping to true and sets the coordinates of the

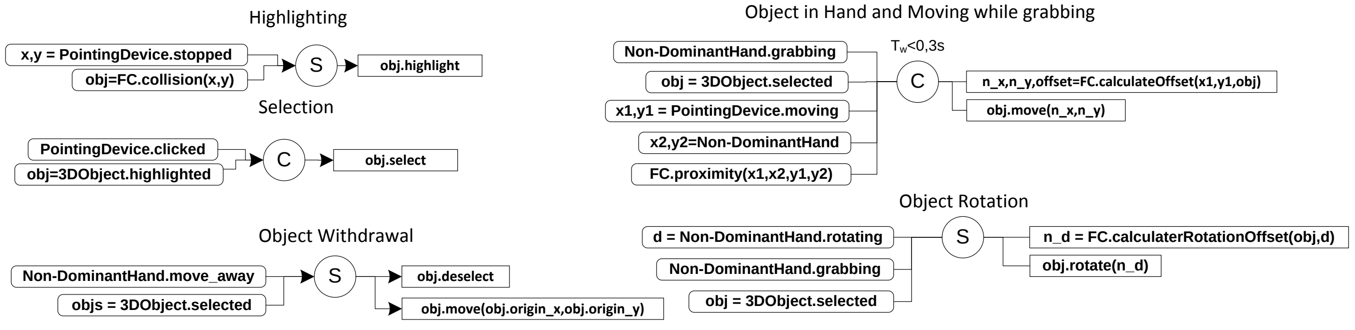
Highlighting
x,y = PointingDevice.stopped
obj=FC.collision(x,y)
S
obj.highlight
Selection
PointingDevice.clicked
obj=3DObject.highlighted
C
obj.select
Object Withdrawal
Non-DominantHand.move_away
objs = 3DObject.selected
S
obj.deselect
obj.move(obj.origin_x,obj.origin_y)
Object in Hand and Moving while grabbing
Tw<0,3s
Non-DominantHand.grabbing
obj = 3DObject.selected
x1,y1 = PointingDevice.moving
x2,y2=Non-DominantHand
FC.proximity(x1,x2,y1,y2)
C
n_x,n_y,offset=FC.calculateOffset(x1,y1,obj)
obj.move(n_x,n_y)
Object Rotation
d = Non-DominantHand.rotating
Non-DominantHand.grabbing
obj = 3DObject.selected
S
n_d = FC.calculaterRotationOffset(obj,d)
obj.rotate(n_d)

**Fig. 4.** The multimodal mappings used to model the Object-in-Hand interaction technique.

pointer. Since the mapping specifies a sequential relation, only after the coordinates of the stopped pointer are known the mapping checks for a collision of the pointer with a 3D object. In case there is a collision, the object is saved in the "obj" variable and the mapping triggers its action to send a highlight event to this object.

In the same manner the selection mapping is processed and runs in parallel to the other mappings. But since it requires an object to be highlighted while the pointing device button is "clicked" (the complementary "C" relation), it is connected to the "highlighting" mapping.

The Object-In-Hand interaction technique is specified by three mappings. For the sake of brevity we will just describe the Object-In-Hand mapping, which is the most complex one. Since it specifies a complementary relation, all observations are processed at the same time, in a temporal window of Tw = 300ms. The arrowless lines define this mapping as a binding: As long as the observations remain true, updated variables of the observations are directly feeded into sequential processing of the actions. The actions calculate the object offset to the center of the scene and then moves the selected object.

## 4. COMPARISON with NiMMIT

In this section, we compare MINT with NiMMiT [1], which was to the best of our knowledge the first notation for modeling multimodal interaction techniques that combined a state-driven with a data flow-driven modeling. Other, data flow-driven notations have been already evaluated by the authors of NiMMiT in [3].

This section is structured into several subsections. First we briefly describe the NiMMiT notation using the Object-In-hand use case, then we compare both notations in detail by analyzing their design models, the capacity to consider multimodal interaction, identify mechanism to support reuse and error management, and finally discuss their processes to support iterative prototyping.

### 4.1 NiMMiT

Figure 5 depicts the NiMMiT diagram of the Object-In-Hand interaction technique as presented by the authors in [2]. NiMMiT diagrams can be read like state transition diagrams. Thus, an interaction has a start and end state, depicted as cycles and several intermediary states connected by transitions and task chains. Events, generated by user inputs trigger a task chain that is depicted as a shaded rectangle. Task chains describe a sequential process of interconnected tasks. Each task (e.g. "Calculate Offset") can exchange data using input and output ports (e.g. "offset", "origin") with other tasks of the same chain. The geometry of the black symbol defines the data type. Using labels, data can be shared over the entire diagram and between task chains. Labels are depicted beside a task chain (e.g. "selected") and are connected to input or output ports. After a task chain has been processed a transition to the next state is performed. NiMMiT models can be-reused in a hierarchical structure. For instance the "Select Object" task is specified in a different model.

### 4.2 Design Models

The NiMMiT notation implements the requirements to be event-driven, state-driven, data flow-driven, and the support for hierarchical reuse. The inclusion of all four aspects in one single diagram is an advantage of the notation.

MINT requires three different types of models: class diagrams, statecharts, and multimodal mappings, but captures the interaction technique specification by the multimodal mapping notation. Both approaches use a proprietary notation that needs to be learned.

The distribution into a set of multimodal mappings to compose an interaction technique specification of MINT solves the state explosion problems from NiMMiT with complex interaction techniques. Since all MINT mappings are processed in parallel by default, no explicit transitions need to be added. An example is the handling of the "gesture_moveaway" event that is used to finish the interaction technique use and moves the manipulated 3D object back to its original position. With MINT the "Object Withdrawel" mapping is always active by default (figure 4).

With NiMMIT (figure 5), explicit transitions have to be defined for each new state in that the "gesture_moveaway" event should be supported. MINT also supports sequential composition of mappings, to specify that one successful mapping can activate another mapping and therefore reflect the default behavior of NiMMiT. But in our experience this behavior is rarely needed. A strict prevention of a certain behavior that was available in a previous context irritates users. Often certain behavior also does not need to be restricted explicitly: For instance, it is not a problem that the "gesture_moveaway" gesture is also active in the case that no object has been selected. Much more frequently, we have experienced commands that have been predefined to control a certain interface, in that a specific interaction technique is embedded and therefore needs to consider predefined commands. An example are the interactor-level mappings that are common interface commands and are reused in the interaction technique and need to be considered to be used in all subsequent steps of the interaction technique. Using NiMMiT a command that always can be processed can only be modeled by adding transitions to all states of the interaction technique specification.

### 4.3 Multimodal Interaction

Both notations consider multimodal interaction. In NiMMiT sequential, parallel and equivalent relations are implemented by event-based conditions with the transitions (and, or). With MINT, the operators that connect observations with actions are used to specify: sequential, complementary, assignment, equivalent and redundant relations. Additional, with MINT, a temporal window can be set to specify temporally related events.

Different to MINT, NiMMiT does not explicitly design the capabilities of modalities. Instead, it requires modalities to trigger events to that an interaction technique can react. Events can be grouped into "families" according to their originating modality or device. Whereas it preserves simplicity, which was one of the design goals of NiMMIT, this approach has two limitations:

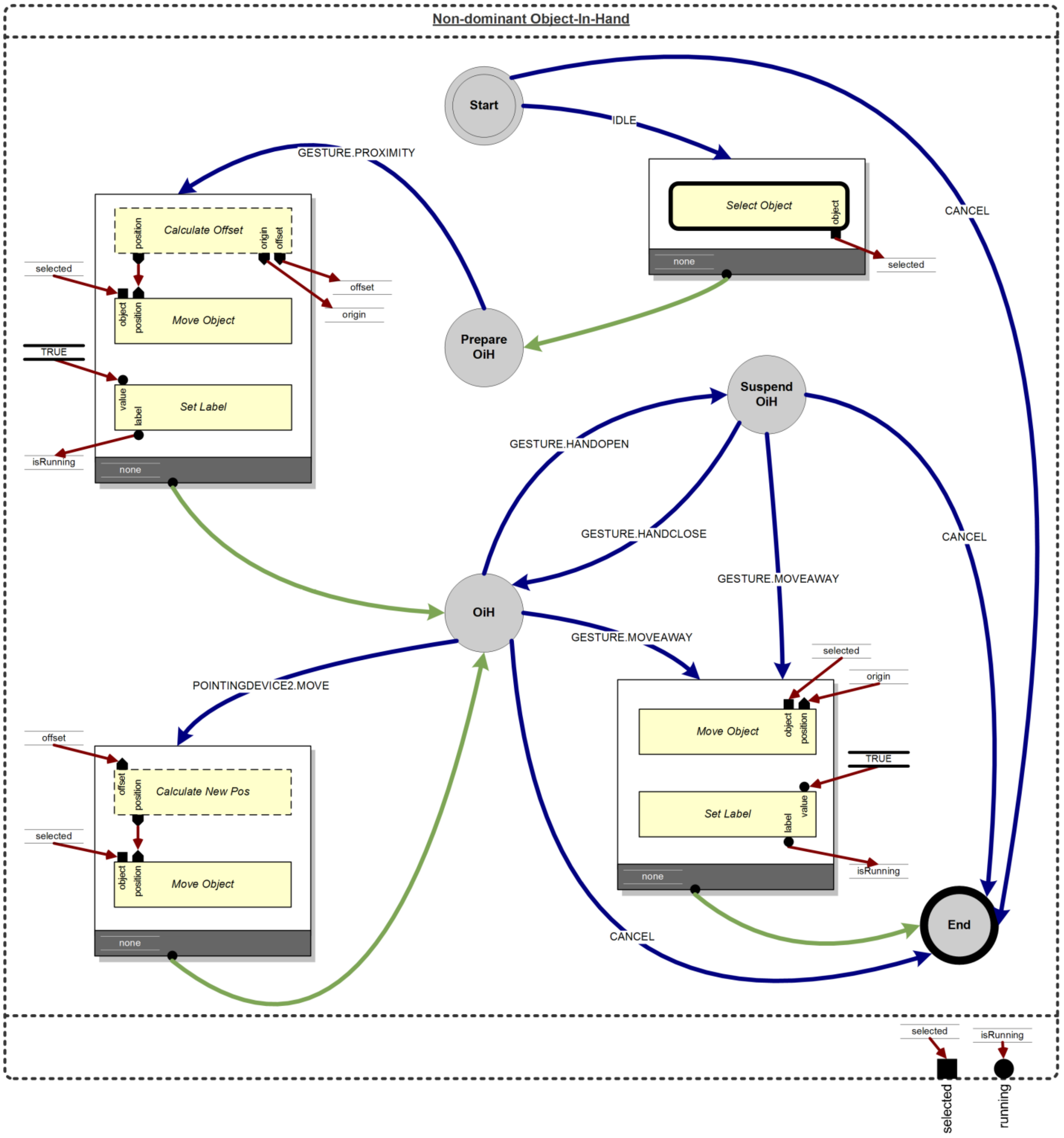

**Fig. 5.** The NiMMIT model for the dominant hand part of the Object-in-Hand interaction technique. Taken from [[1]]

First, output modalities need to be controlled by custom scripted tasks, as the authors mentioned in [1]. With MINT there is no difference in the design and usage of input (that we call mode) and output modalities (which we call media). Figure 6 illustrates an example that adds a second modality to an existing mapping that specifies object highlighting: a confirming “clicking” sound that is played each time an object gets highlighted. There are two ways to design this behavior with MINT. One option is to model the redundant output (visual highlighting and sound output) as an application level mapping like shown in figure 6a using the redundant operator. The other option is a more abstract interactor level mapping definition that defines that always, if an object is highlighted (independently if it was triggered by the pointing device), a sound should be played.

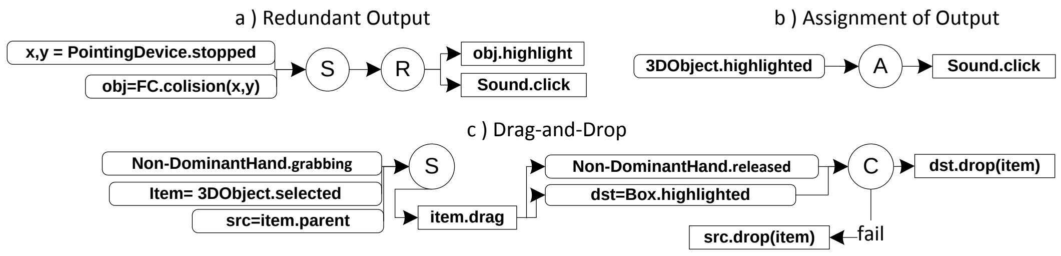

**Fig. 6.** Mappings with redundant output (a+b) and error handling (c).

Second, to our knowledge, with NiMMiT, a data flow can be specified only between tasks. Tasks of different task sets can exchange data using variables that are called labels. But data input or output from modalities is not considered explicitly in the design notation. An example is depicted in figure 5 by the "PointingDevice2.Move" event triggering the transition originating from the "OiH" state. The task set that is activated by this transition consumes data from the "offset" label that has been calculated by a preceding task set. But which modality is used to produce the output position data is hidden within the "Calculate New Pos" task. In MINT the data flow between a modality and the interaction technique is specified explicitly. Thus, for instance in the "Object In Hand" mapping of figure 4, the data flow of the moving pointing device is set to move the 3D object (as long as it remains in proximity to the non-dominant hand).

## 4.4 Reuse

An interaction technique frequently includes reoccurring elements, such as tasks to select, remove or confirm. Thus, supporting reuse when modeling new interaction techniques prevents re-designs of already existing techniques.

NiMMiT supports encapsulation of tasks for a hierarchical reuse. Such a task encapsulation is used in the Object-in-Hand NiMMiT model depicted in figure 5. The "Select Object" task has been separately designed, since object selection can be considered a very common task, and therefore has been reused for the Object-in-Hand model.

Different to the explicit reference in NiMMiT, in MINT a set of mappings without the need for direct references defines an interaction technique. This has the advantage that if an interaction technique depends on several reusable tasks that should be available in all of the states of the new interaction technique there is no need to explicitly reference the included component.

From the ease-of-use perspective the implicit reference (by existence) is rather hard to use as it requires the designer to know about all active mappings. But often most of the mappings are defined at the interactor-level and are connected to a specific modality or device. Thus, like in our example in figure 4, mappings define actions or events for a specific device, like the pointing device, which should be always used for selecting and highlighting objects. In practice interactor level mappings are easy to memorize since they usually define the default actions for a specific modality or device.

## 4.5 Error Handling

NiMMiT does not include error handling to recover from a failure that happens when an interaction technique is used [16].

In MINT, multimodal mappings can consider error cases. Figure 6c illustrates how an error handling can be defined to undo an action that has been partly done using an interaction technique. The mapping defines a drag and drop interaction technique between 3D objects that are stored in two different boxes. By defining a fail-case for the second operator (the complementary relation) the mapping ensures that after a user has started to drag a 3D object, it is returned to its origin box if the user has failed to drop it to another box.

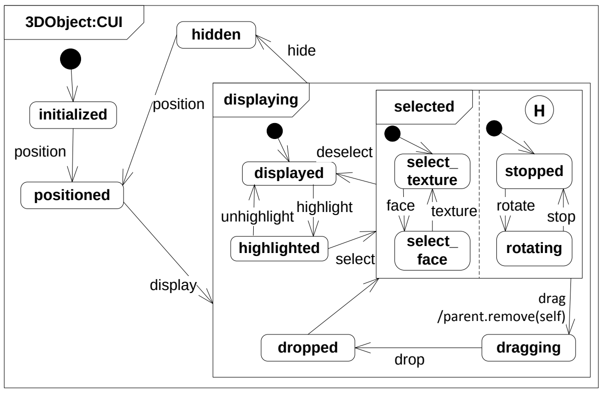

**Fig. 7.** Redesign of the 3DObject Statechart to consider drag-and-dropping.

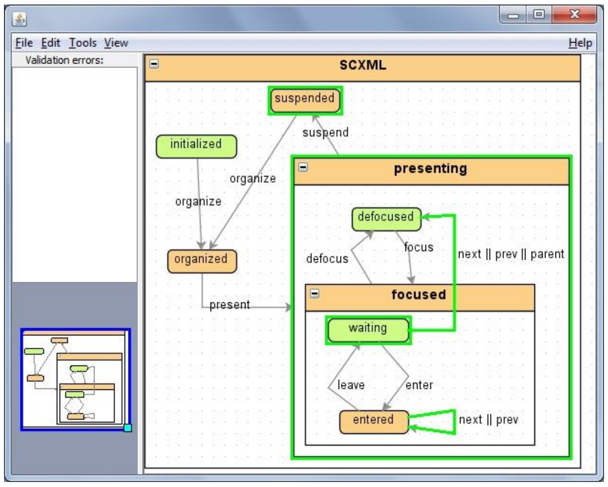

**Fig. 8.** The adopted scxmlgui editor to design SCXML-based statesharts.

### 4.6 Iterative Prototyping

Both, MiMMiT and MINT offer an application framework to interpret the diagrams at runtime to quickly test and compare design alternatives without much coding effort. To further support the designer creating new interaction techniques both notations are tool supported and save their models in XML. According to [1] the NiMMiT process is a sequential one: After a NiMMiT model has been designed with a tool, it is saved in XML, and the loaded and executed in the application framework.

In MINT the initial deployment of the interaction technique to the platform follows the same process like NiMMiT, but requires two design tools: One, an adopted version of the scxmlgui editor [13], is used to specify the statechart for describing the behavior of the widgets and the interaction resources, and a second one to design the multimodal mappings. Figure 8 shows a screenshot of the former one.

After the platform has been started with the initial deployment of a prototype, both design tools can be connected to the platform to observe the mappings and states of the statecharts being activated while the interaction technique is used. In MINT the iterative prototyping cycle length has been further reduced: statecharts can be manipulated when the application is running with the restriction that a state cannot be removed while it is active. An animation of the active states in the design tool supports the designer to identify these states. Manipulations at runtime ease to correct usability problems and to fix errors when they occur.

Figure 6c) depicts a further interaction technique, a drag-and-drop like technique that enables a user to grab 3D objects out of a box into another one. This mapping can be added as a new mapping at system runtime without the need to restart running applications. But for the mapping to function with the 3D object, the object's behavior needs to be adjusted by adding two new states: “dragging” and “dropped” to the statechart like depicted in figure 7.

We have evaluated the MINT framework against the requirements of the W3C multimodal initiative, which requires a multimodal toolkit to implement a structural mechanism for user interface composition, an explicit control structure, an extensible event definition mechanism, consider data modeling, and offer reusable components in [8]. Further on, the MINT framework has been classified by using the characteristics for multimodal frameworks proposed by Dumas et.al. [7]. An initial performance analysis has been performed [8].

## 5. CONCLUSION

This paper presents MINT, a model-based approach to design multimodal interaction techniques. The graphical notation combines statecharts to design modalities and media features, uses data flow based mapping, and is targeted to allow the explorative design of multimodal interactions to compare different variants without much coding effort.

Previous approaches can be classified in either statechart-driven or data flow-driven proposals. NiMMiT was the first graphical notation to combine both, but suffers from a set of limitations: State explosion for complex models, no explicit design support for output modalities, and missing error handling.

MINT addresses each of these aspects by separating statechart and data flow into two different complementary models and a notation for defining recovery steps inside the definition of the interaction techniques. We illustrate our approach by using the same use case that has been used for describing NiMMiT: The Object-In-Hand interaction technique. We demonstrate further advances of MINT that reduce the iterative prototyping cycles by supporting model manipulation at runtime to improve usability and inconsistencies.

## 6. ACKNOWLEDGMENTS

Sebastian Feuerstack is grateful to the Deutsche Forschungsgemeinschaft (DFG) for the financial support of this work.

## 7. REFERENCES

[1] Boeck, J. D.; Vanacken, D.; Raymaekers, C. & Coninx, K. (2007), 'High-Level Modeling of Multimodal Interaction Techniques Using NiMMiT', *Journal of Virtual Reality and Broadcasting* **4**(2).

[2] Boeck, J. D. ; Cuppens, E.; De Weyer, T.; Raymaekers, C. & Coninx, K. (2004), Multisensory interaction metaphors with haptics and proprioception in virtual environments, *in* 'Proceedings of the third Nordic conference on Human-computer interaction', ACM, New York, NY, USA, pp. 189--197.

[3] Boeck, J. D.; Raymaekers, C. & Coninx, K. (2007), Comparing NiMMiT and data-driven notations for describing multimodal interaction, in 'Proceedings of the 5th international conference on Task models and diagrams for users interface design', Springer-Verlag, Berlin, Heidelberg, pp. 217--229.

[4] Calvary, G.; Coutaz, J.; Thevenin, D.; Limbourg, Q.; Bouillon, L. & Vanderdonckt, J. (2003), 'A Unifying Reference Framework for Multi-Target User Interfaces', *Interacting with Computers* **15**(3), 289--308.

[5] Carr, D. A. (1997), ‘Interaction Object Graphs: An Executable Graphical Notation for Specifying User Interfaces’, *Formal Methods in Human-Computer Interaction*, Springer, pp. 141--155.

[6] Dragicevic, P. & Fekete, J.-D. (2004), Support for input adaptability in the ICON toolkit, *in* 'Proceedings of the 6th international conference on Multimodal interfaces', ACM, New York, NY, USA, pp. 212--219.

[7] Dumas, B.; Lalanne, D. and Oviatt, S. (2009), 'Multimodal interfaces: A survey of principles, models and frameworks', Human Machine Interaction, pp. 3--26.

[8] Feuerstack, S. and Pizzolato, E. (2012 Engineering Device-spanning, Multimodal Web Applications using a Model-based Design Approach, WebMedia 2012, the 18th Brazilian Symposium on Multimedia and the Web, October 15-18, 2012, São Paulo/SP, Brazil

[9] Figueroa, P.; Green, M. & Hoover, H. J. (2002), InTml: a description language for VR applications, *in* 'Proceedings of the seventh international conference on 3D Web technology', ACM, New York, NY, USA, pp. 53--58.

[10] Harel, D. (1987), 'Statecharts: A visual formalism for complex systems', *Sci. Comput. Program.* **8**(3), pp. 231-274.

[11] Lehmann, G.; Blumendorf, M.; Feuerstack, S. & Albayrak, S. (2008), Utilizing Dynamic Executable Models for User Interface Development, *in* T. C. Nicholas Graham &

Philippe Palanque, ed., 'Interactive Systems - Design, Specification, and Verification', Springer-Verlag Gmbh.

[12] Meixner, G.; Paterno, F. & Vanderdonckt, J. (2011), 'Past, Present, and Future of Model-Based User Interface Development', *i-com* **10**(3), pp. 2-11.

[13] Morbini, F. (2011), 'Scxmlgui', http://code.google.com/p/scxmlgui/, last accessed 08/01/13

[14] Navarre, D.; Palanque, P.; Bastide, R.; Schyn, A.; Winckler, M.; Nedel, L. & Freitas, C. (2005), A Formal Description of Multimodal Interaction Techniques for Immersive Virtual Reality Applications, *in* Maria Francesca Costabile & Fabio Paternò, ed., 'Human-Computer Interaction - INTERACT 2005: IFIP TC13 International Conference, Rome, Italy', Springer-Verlag GmbH, , pp. 170.

[15] Nigay, L. & Coutaz, J. (1997), Multifeature Systems: The CARE Properties and Their Impact on Software Design'Intelligence and Multimodality in Multimedia Interfaces'.

[16] Raymaekers, C.; Vanacken, L.; Boeck, J. D. & Coninx, K. (2008), High-Level Descriptions for Multimodal Interaction in Virtual Environments, *in* 'Proceedings of CHI 2008'.

[17] Ringel, M.; Tyler, J.; Stone, M.; Ballagas, R. & Borchers, J. (2002), iStuff: A Scalable Architecture for Lightweight, Wireless Devices for Ubicomp User Interfaces, *in* 'Proceedings of UBICOMP 2002'.

[18] Serrano, M.; Nigay, L.; Lawson, J.-Y. L.; Ramsay, A.; Murray-Smith, R. & Denef, S. (2008), The openinterface framework: a tool for multimodal interaction., *in* 'CHI '08 Extended Abstracts on Human Factors in Computing Systems', ACM, New York, NY, USA, pp. 3501--3506