Designing Context-Aware Multimodal Virtual Environments

Lode Vanacken Joan De Boeck Chris Raymaekers Karin Coninx
Hasselt University - tUL - IBBT
Expertise Centre for Digital Media
Wetenschapspark 2
B-3590 Diepenbeek (Belgium)
{lode.vanacken,joan.deboeck,chris.raymaekers,karin.coninx}@uhasselt.be

ABSTRACT

Despite of decades of research, creating intuitive and easy to learn interfaces for 3D virtual environments (VE) is still not obvious, requiring VE specialists to define, implement and evaluate solutions in an iterative way, often using low-level programming code. Moreover, quite frequently the interaction with the virtual environment may also vary dependent on the context in which it is applied, such as the available hardware setup, user experience, or the pose of the user (e.g. sitting or standing). Lacking other tools, the context-awareness of an application is usually implemented in an ad-hoc manner, using low-level programming, as well. This may result in code that is difficult and expensive to maintain. One possible approach to facilitate the process of creating these highly interactive user interfaces is by adopting a model-based user interface design. This lifts the creation of a user interface to a higher level allowing the designer to reason more in terms of high-level concepts, rather than writing programming code. In this paper, we adopt a model-based user interface design (MBUID) process for the creation of VEs, and explain how a context system using an Event-Condition-Action paradigm is added. We illustrate our approach by means of a case study.

Categories and Subject Descriptors

D2.2 [**Design Tools and Techniques**]: User Interfaces; I3.6 [**Methodology and Techniques**]: Interaction Techniques

General Terms

Design, Human Factors

Keywords

Context-Awareness, Model-Based User Interface Design, Multimodal Interaction Techniques

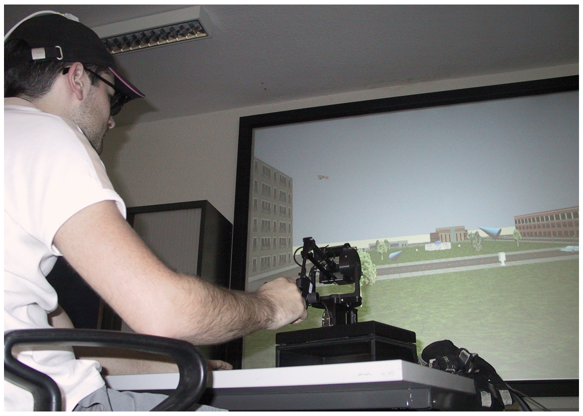


Figure 1: The setup used in the case study.

1. INTRODUCTION

The design and implementation of an intuitive user interface in a 3D interactive virtual environment (VE) is a time consuming and hence expensive process. Especially the process of finding and creating suitable interaction paradigms is not straightforward. Interaction in an interactive VE is often required to be multimodal by nature, supporting gestures, direct manipulation, speech input, etc. More often than not, the creation of such a rich user interface results in an iterative process in which a solution is defined, implemented and evaluated multiple times.

One possible approach to facilitate this process is adopting a model-based user interface design (MBUID). This lifts the creation of a user interface to a more abstract level allowing the designer to reason more in terms of high-level concepts, rather than writing programming code. In most MBUID processes different models are applied through gradual progression, converting one model into the next one. For instance a typical process may start at the level of a task model, moving over several other models, towards the final user interface.

Dependent on the domain in which the MBUID process is applied, the used models may vary. Specifically for the design of an interactive VE and the modelling of the rich multimodal interaction, several high level notations exist, defining the 'Interaction Description Model' [9, 15, 31]. In the research presented in this paper, we will use NiMMiT (Notation for Multimodal Interaction Techniques) [9] for describing the interaction.

In some situations, the interaction with an application may vary depending on the context in which it is applied. This is especially true for mobile applications that can be used on different platforms or in a variety of situations causing some features that may or may not be available.

Although no consent definition of *'Context'* exists [10], in this work we consider context as an artefact influenced by different factors [25] such as the user, the environment, platform, etc.

A context is then a multidimensional vector of these factors (sometimes also called observables [7]) (**definition 1**).

For convenience in the user interface of the proposed tool support, we call these factors '*Context Units*' and the values available for each unit '*Unit Values*'.

In the domain of interactive VEs, different contexts may also have an influence on the available interaction. The context in a VE is often defined by the available input and output devices, external parameters such as the experience level of the user, whether or not there are collaborative partners in the environment, or even the pose of the user (sitting or standing). Without the support for different contexts directly from within the MBUID process, the integration may often result in a lot of ad-hoc code, which is difficult and expensive to maintain.

In this paper, we present an approach to integrate the use of context-awareness in a MBUID process for the generation of an interactive VE using an 'Event-Condition-Action' paradigm [2, 13]. We will apply the VR-DeMo model-based process [26] and the accompanying tool support CoGenIVE [8].

2. RELATED WORK

In this section, we discuss several relevant topics within the scope of this paper. First, we discuss current frameworks for the creation of multimodal user interfaces in general and multimodal interfaces for virtual environments in particular. We also briefly look into their support with regard to the handling of context information. Next, we also will discuss how context can be modelled in general. Finally, we discuss some frameworks for virtual environments that, at least to some extend, provide support for contextual information.

2.1 Multimodal User Interfaces

For the creation of general 2D multimodal user interfaces, Flippo et al. [16] present a framework for the rapid development of those interfaces. They focus on re-usable patterns to implement their interfaces. With regard to the use of context, the only aspect being present in this framework is the ability to perform reference resolution for a fusion mechanism of the input modalities. According to our definition of context, this type of context is not considered by our system because modality fusion is achieved by the interaction notation as described in section 3.1.

More applicable when creating virtual environments is the COTERIE framework by MacIntyre et al. [21]. This framework facilitates the design and separation of application semantics from low-level input handling. Alternatively, Irawati et al. [17] present the VARU framework, a rapid prototyping framework for virtual reality, augmented reality and ubiquitous computing integrated in one single platform which gives the possibility to explore different types of collaboration across the different spaces. Three other toolkits for the creation of tangible and multimodal interfaces were proposed by Dumas et al. [12]. The SCS system they discuss has several similarities with a MBUID process: it contains a mix of a task model, dialog model and interaction model in a single state machine. However, important within the scope of this paper, the toolkits described above, have no genuine support for context-awareness and the possible integration of context is usually not discussed.

Guitiérrez et al. [22] present a system accompanied with a tool which allows for real-time configuration of multimodal virtual environments. Devices and interaction itself are presented using xml-based descriptors and coupled using tool support. Here the designer has to define which device modality is activated by which interaction modality. An almost similar system, focusing on both the application's needs and the devices' capabilities was developed by Fernandes et al. [14]. Alternatively, the 'Input Configurator Toolkit' [11] is a toolkit which provides a visual programming interface for modifying the mapping between input devices and functionalities of an application. The system enables interactive applications to adapt to special interaction devices as well as user preferences and needs and thus defines the interaction between the user and the system. A similar framework is the OpenInterface framework [28], a component-based tool for rapidly developing multimodal input interfaces using generic or tailored components. Another way to abstract the device from the application is to use an 'Interaction Description Model'. Several solutions exist, each with their own features or design: Interaction Object Graphs [4], InTml [15], ICO [23].

In the scope of this paper, we adopt the VR-DeMo process [26], a model-based design process, which generates a working VE application by creating high level models (task model, dialog model, 'Interaction Description Model' (see section 3.1) and a presentation model). The models are created using the accompanying tool support CoGenIVE [8].

None of the above mentioned systems supports changes depending on the context information, either.

2.2 Context Modelling

In order to be able to apply context-awareness, a context-model is necessary to represent this kind of information. Many different models exist [1, 7, 20, 24]. A complete discussion of all these models falls out of the scope of this paper. Instead we will limit our discussion to some related work focusing on modelling context and in which tool support is present or in which the modelling is embedded in the model-based user interface design process.

ICap [29] supports rapid prototyping of context-aware applications. Their focus is to support end users in order to quickly design prototypes of a context-aware application without the production of source code.

Rousseau et al. [27] present a multimodal output specification and simulation platform. Both platforms are supported by tools to help the design of multimodal output. A design process is described in which interaction context is taken into account. This is implemented such that the designers know the different contexts that can occur and how these contexts might influence the multimodal output. The different contexts are described in a specification tool using election rules evaluated to define which output has to be presented depending on the context. MobiLife [18] aims at exploiting the synergetic use of multimodal user interface technology and contextual information processing. A 'Device and Modality Function' provides functionalities to abstract the interaction between the devices/modalities and the application. Depending on the context, other devices/modalities are chosen or combined. A 'Device Gateway Function' makes it easier to provide these functionalities.

In the domain of MBUID, the use of context also has been studied thoroughly during the past years [5]. Clerckx [5] presents DynaMo-AID which allows context-aware application development for mobile applications. The primary focus of this work is on the integration of context at the task level.

2.3 Context in Virtual Environments

Applying contextual knowledge to the domain of virtual environments, Lee et al. [19] present a framework to converge context-awareness and augmented reality, it is separated in three layers in which a context layer is used to communicate between augmented reality and the ubiquitous services using a broker system. The universal data model is used, in which nodes, representing a person, place, task or service, are associated with each other and events are fired to notify which tasks need to be executed when a context change occurs. In these frameworks, no tool support is present to aid developers or designers to use this framework, they mostly aid users during low level programming development.

In this paper, we describe the definition of a context system, detecting and switching contexts at run-time. The proposed solution is built upon the existing VR-DeMo MBUID process [26]. The contexts can be defined from within CoGenIVE, the accompanying tool support, and can have their influence on the 'Interaction Description Model'. In section 4, we will motivate our approach by means of a case study.

3. DEFINING A CONTEXT SYSTEM

Before discussing in detail the 'Event-Condition-Action' paradigm that has been adopted for this solution, we shortly describe the NiMMiT notation (Notation For MultiModal Interaction Techniques), which is applied for the definition of the 'Interaction Description', and which is also useful to describe the context detection and switching diagrams. In section 3.4, we finally show how our approach is suitable for the automatic generation of template diagrams for the runtime support of the context-awareness diagrams, and how this all is integrated in the existing tool CoGenIVE.

3.1 The Interaction Description Model

The 'Interaction Description Model' is a high level description that is specifically necessary for the description of the rich interaction within a 3D virtual environment. Most tasks in a VE are described as interaction techniques, complex ensembles of direct manipulation and multimodal information that carry out a task, possibly consisting of several sub-tasks. Examples of interaction techniques in interactive VEs include selection techniques such as ray casting, but also navigation techniques such as a walking metaphor.

Keeping in mind the requirements of an easy to learn, easy to read high level notation for the description of an interaction technique, but also providing automatic execution of the generated models, we have developed NiMMiT. Basically, a

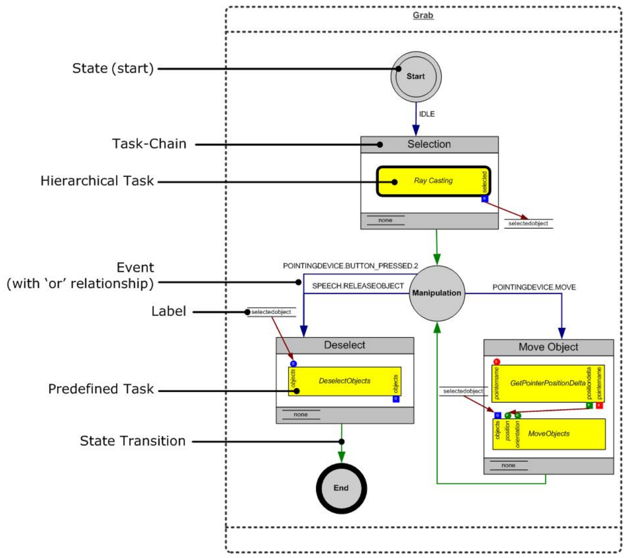


Figure 2: An Example of a NiMMiT diagram.

NiMMiT diagram can be seen as a state transition diagram, from which the following features can be recognised in figure 2: at a certain time t, the interaction technique resides in a certain state S, responding to a certain set of events E. These events can be multimodal by nature: gesture recognition, speech recognition, button clicks, etc. The recognition of an event triggers an activity, a linear set of tasks called a task chain. After a task chain has been completed successfully, a state transition to a new state is performed, such that the interaction responds to another set of events E'. NiMMiT also offers data flow between subsequent tasks and between tasks in different task chains by means of labels, which can be seen as high level variables.

NiMMiT allows defining the user's interaction by combining tasks that logically belong together. A number of predefined tasks such as collision detection, are already available, but designers can also define their own custom tasks for an application. For a more in depth discussion on NiMMiT, we refer the interested reader to [9] and [6].

3.2 Context Detection and Switching

A possible approach to define a context system is by adopting 'Event-Condition-Action' rules [2, 13]. A certain *event* or combination of events can signal a context switch. After the event has been recognised, certain *conditions* have to be met before switching the context. When these conditions are fulfilled, next it might be necessary to first perform some *actions* before finalising the context switch. For instance, a user may stand up from his chair (*event*). Before executing a context switch, we must ensure that he wears the tracked gloves (*condition*). If this condition is met, we disable the toolbars that are needed in the desktop setup and connect the cursor to make the glove visible (*action*). Note that the designer has the freedom to decide which actions are defined as events and which parameters are checked as a condition. In the example above, we assumed that 'standing up' is recognised as an event, but if this should appear technically difficult to implement, it is also possible to listen to a more general event, e.g. the movement of a tracker mounted to the user's head (*event*), and assert for the tracker's position in order to decide whether the user is seated or standing (*condition*).

Beer et al. [2] suggested that it is advisable to have a visual builder tool supporting the creation of context rules, or even the creation of whole interaction scenarios without programming. This approach of visually modelling context through 'Event-Condition-Action' rules allows non-technical people to control smart environments and devices without the knowledge of the entire complexity of a system. In the next section, we propose to use NiMMiT as a graphical notation and CoGenIVE as the accompanying tool support to model context using 'Event-Condition-Action' rules.

3.3 Context Detection and Switching Using NiMMiT

For the implementation of the 'Event-Condition-Action' rules using NiMMiT, we have chosen to split the context system in two separate parts. This must keep the design as modular as possible. One part, driven by one NiMMiT diagram is responsible for the *Context Detection* while another diagram handles the *Context Switch*. According to the 'Event-Condition-Action' paradigm, the context detection part identifies the events that may cause a context switch, checks whether or not the conditions are fulfilled and finally triggers a 'context switch event'. This event is at its turn recognised by the 'Context Switching' part.

3.3.1 *Context Detection Diagram*

In the 'Context Detection' diagram (figure 6) a given context is typically represented by a state. Hence, the relevant events that can evoke a context switch are available for each context. In general, these events activate a task chain, which checks the *condition* using one or more tasks. When the condition is not met, the original state must be restored. Otherwise, before moving on to a new state reflecting the new context, the task chain has to fire a 'context switch event', which is handled in the 'Context Switching' diagram (see section 3.3.2).

The implementation of the condition check can be achieved in different ways, using different structures in NiMMiT, such as conditional state transitions. However, we prefer to exploit NiMMiT's exception handling primitives in order to break a task chain and return to the original state. We believe that this approach will result in the most easy to read diagrams.

The tasks that are responsible for checking the condition, must collect their result in a single boolean label. This label is passed on to a predefined task 'ContinueTaskChain', which checks whether or not the condition is fulfilled. When the boolean contains the value 'true', nothing happens and the task chain is continued. When the value is 'false' however, this tasks throws an exception and activates the exception handling mechanism of the NiMMiT task chain. In this particular situation, the simplest form of error handling, interrupting the current task chain and restoring the previous state, suits the needs. This means that when the condition is not met, the original state is restored and no context switching event is generated.

We want to stress that, although the designer may decide to adopt other patterns in order to interrupt the state transition, the proposed approach exploiting NiMMiT's exception handling, keeps the diagrams simple with no superfluous structures. The approach results in patterns that are uniform across different projects, which is a requirement for the tool support generating template diagrams.

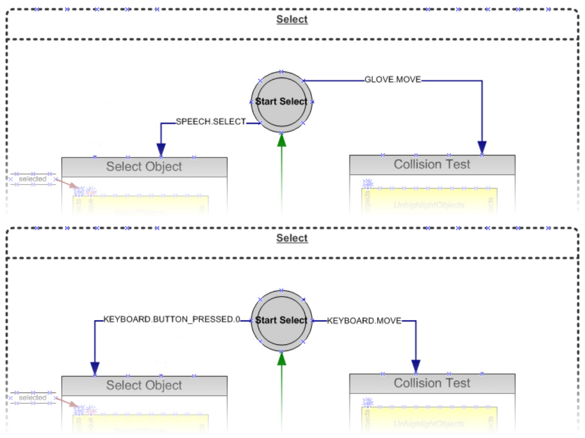
3.3.2 *Context Switching Diagram*

A second NiMMiT diagram, responsible for the *action* (figure 8), consists of only one state which contains the 'context events' representing the possible context switches. The occurrence of a 'context event' activates a task chain that contains the code that has to be executed before the context switch can effectively be performed. This code may include altering objects or controls in the world, enabling or disabling certain devices, etc. The last action in this task chain must explicitly change the context, in order to notify other running NiMMiT diagrams defining the actual user interaction to adapt to the new context (see section 3.3.3).

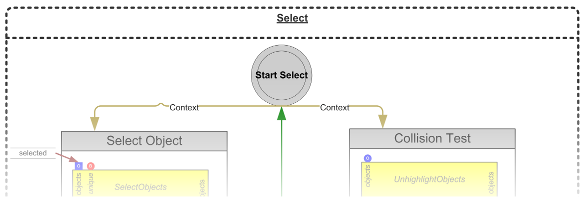
Independent of the nature or the number of contexts, the NiMMiT diagrams for context switching share a similar pattern, as well, which makes the approach suitable for the generation of template diagrams. Obviously, for specific purposes the designer is free to alter the generated diagrams, e.g. if he wants to restrict possible context switches.

3.3.3 *Context Handling*

Besides being able to define the context system using NiMMiT diagrams, we also need to be able to create context-aware NiMMiT diagrams that represent the interaction techniques. Therefore, we use the earlier introduced concept of a context arrow which represents a certain context while tying events to that context [30].



(a) Events active in two different contexts.



(b) Events were attached to context arrows.

Figure 3: Context Handling

In figure 3, an example of this approach is depicted. In figure 3(a) we see that in the 'Start'-state several different events (modalities) could trigger the execution of the task chains. Using context information, there is the ability to attach a context to a certain event or modality in such a way

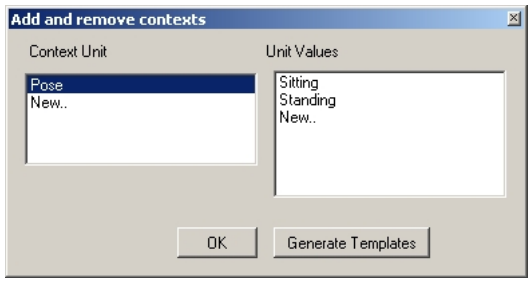


Figure 4: Dialog in CoGenIVE to indicate the possible contexts and generate template diagrams.

that, depending on the context only the according events are active. If for example 'GLOVE.MOVE' is intended to be used in the immersive setup, one can attach the 'immersive'-context to the event-arrow 'GLOVE.MOVE'. Similarly, the event 'KEYBOARD.MOVE' can be used in the 'desktop'-context.

Note that if there was no support to couple events to a context the same diagram should be created twice with different events (as in figure 3(a)) which obviously would make maintenance much harder.

Adding this contextual knowledge to NiMMiT, transforms the view of the diagram according to the context. A part of the resulting diagram containing context arrows is shown in figure 3(b).

3.4 Tool Support

The aforementioned approach has been integrated in CoGenIVE, the tool supporting the adopted VR-DeMo MBUID process. The integration starts with a dialog that defines the different contexts (figure 4). In definition 1, we defined 'Context' as a multidimensional vector of observables, called 'Context Units'. In figure 4, it can be seen how context units and values for each unit can be defined. The combination of a particular value *for each* context unit then defines the context. Hence the maximum number of available contexts is the cross product of all unit values, grouped per context unit.

As the 'Context Switching' diagram is triggered by context events, for each context, a matching event is created and made available in the 'events'-pane in the interface (Figure 5). Here the events can be picked by the user in order to be used in a NiMMiT diagram.

From within the 'add/remove context' dialog, a template 'Context Detection' and 'Context Switching' diagram can be generated. The template for the 'Context Detection' diagram defines a state for each context. Each destination context (state) also gets a task chain already containing a task throwing the according context event (this is the last task in each task chain). Finally, each context has an (empty) event arrow to each task chain (excluding a transition to itself). The designer now only has to add the desired events to the event arrows and eventually add extra conditions into the task chains, resulting in a diagram similar to figure 6 and figure 7.

The template for the 'Context Switching' diagram is fairly simple. There is only one state. Next, for each context switch, there is an event arrow pointing to a task chain. This task chain already contains a task to explicitly switch the context (typically the last task). Finally a state transition to the (only) state is performed. In this diagram, the designer only has to add the custom actions that have to be performed before the context switch takes place. This results in a diagram similar to figure 8.

In summary, for a situation with n different contexts, a typical context detection template contains n states, n task chains with n state transitions and finally $(n-1)\cdot n$ event arrows. The 'Context Switching' template contains only one state with n context events and n task chains. It may be clear that, with regard of the generation of these templates, the automatic layout management is a substantial problem which currently falls out of the scope of this paper.

Regarding the context handling, CoGenIVE provides the possibility for the designer to indicate in a NiMMiT diagram, defining the user interaction, when certain events explicitly belong to a given context. To simplify the visual representation, the events are collapsed in a 'context arrow', as can be seen in figure 3(b). The tool support allows the designer to view the resulting diagram for each particular context.

4. CASE STUDY

In order to evaluate and motivate our approach, we elaborated upon a practical case study, which is described in this section. The result is a virtual environment application, represented by a 3D stereo projection. A user can interact either seated using a 3D mouse and a Phantom haptic device, or standing using two pinch-gloves which are tracked with a Nest of Birds magnetic tracker. The first alternative setup offers the user the experience of having force feedback, but in a very limited workspace. The other alternative offers an intuitive two-handed interaction paradigm with a much larger workspace, but without force feedback. A couple of possible interaction techniques have been implemented using the NiMMiT notation: Two Handed Scale, Flying Vehicle Navigation and Virtual Hand Object Selection and Manipulation [3]. In this case study, we consider changing from either seated to standing or vice versa as a context switch, enabling or disabling the appropriate devices. Figure 1 gives an impression of the described setup.

4.1 Context Detection

The 'Context Detection' diagram is illustrated in figure 6. In a first step the initial context is queried and an initial switch to that context is executed ('GetContext' and 'Fire-

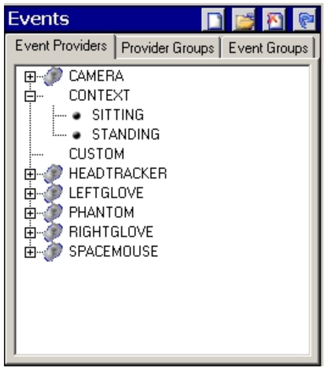


Figure 5: Events Pane in CoGenIVE, including context events

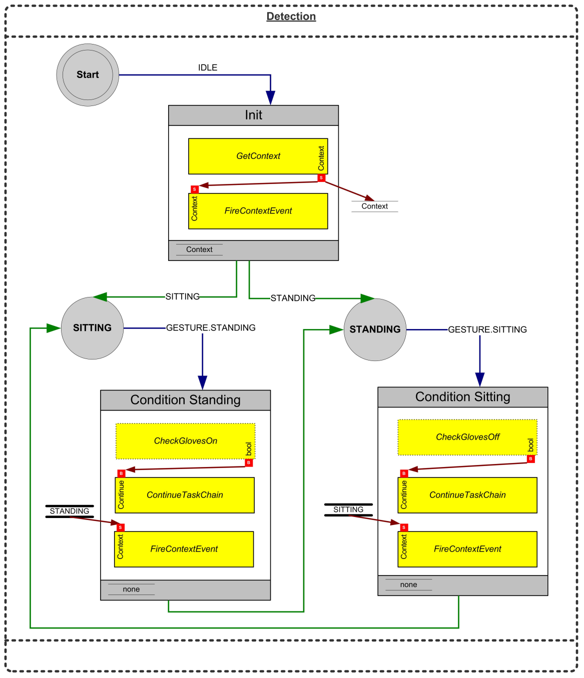


Figure 6: The 'Context Detection' diagram from the case study with the 'GESTURE' interface.

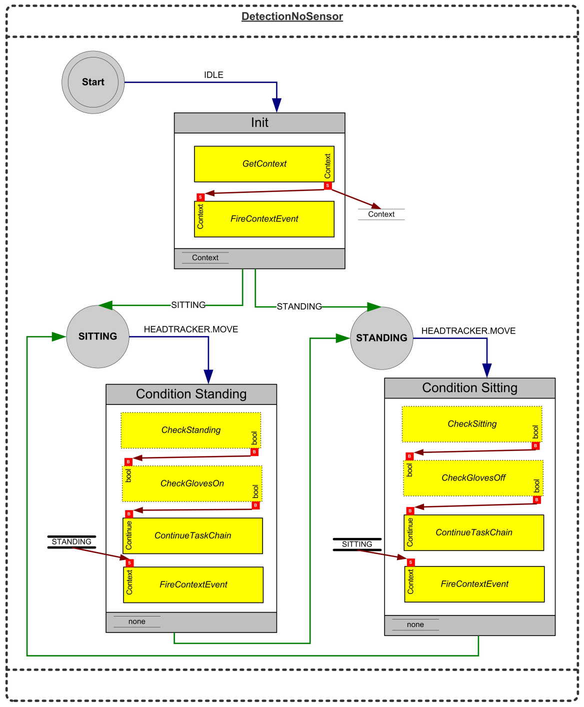


Figure 7: The 'Context Detection' diagram from the case study using a head tracker.

ContextEvent'). This additional state and task chain is used only to perform some kind of a 'virtual context switch' in order to execute the context switching tasks (*action*) in the initialisation phase. Next, the diagram performs a state transition to the state of the initial context.

The events that indicate the possible context switches are 'GESTURE.SITTING' and 'GESTURE.STANDING'. Whenever these events occur the necessary conditions are checked using the respective task chains ('Condition Standing' and 'Condition Sitting'). For instance, in figure 6 it is checked whether the user is wearing the gloves ('CheckGlovesOn'). Note that this task is a custom task, defined and implemented by the designer. For this particular implementation, we detect if there has been any motion in the gloves over the past 2 seconds. If there has been a movement, we can assume that the user has put on the gloves. Finally, when the condition is not met, the 'ContinueTaskChain' task interrupts the task chain, breaking the context switch. Otherwise, the appropriate context switching event is fired. Finally, a state transition to the other context-state is performed.

A possible problem with the proposed solution is the gesture interface which is supposed to be present in order to recognise the 'sitting' or 'standing' event. This approach assumes either an advanced gesture recognition engine, or it requires again a lot of ad-hoc code, implementing the gesture.

If, however, we do not have the convenience of a decent gesture interface, this same functionality can be implemented in the 'Context Detection' diagram, by using a more 'low level' event. Figure 7 shows an alternative version of the same 'Context Detection' diagram, but now listening to the raw movements of a head tracker, instead of the detection of an event. The position of the head can reveal very easy if a user is either sitting or standing by looking at the height (Y-axis). Therefore, an extra condition has to be added in the 'condition' part, checking for the head position.

4.2 Context Switching

The context event generated in the 'Context Detection' diagram is detected by the 'Context Switching' diagram. The diagram, shown in figure 8, contains one state, listening to all available context events. When the user stands up, the 'CONTEXT.STANDING' event is recognised (generated by the 'Context Detection' diagram), and the left task chain is executed. In this case study, the required actions are relatively straightforward: showing and hiding respective pointers for the new context. For instance, while the user is standing and wearing the gloves, the gloves have to be visible while the spacemouse and PHANToM have to be invisible and inactive.

Finally, the last task in the 'Context Switching' diagram explicitly changes the current context, so that the interaction description diagrams can handle the newly activated context.

4.3 Context Handling

The actual handling of the context, such as listening to other events, activating other task chains, etc., is defined in the 'Interaction Description diagram'. In this case study, the interaction diagrams listen to the appropriate events, according to the devices that are enabled in the current context. For instance, for the virtual hand selection technique, the move-events of the Phantom haptic device are disabled when the user is standing, avoiding unwanted response from this device. Alternatively, when the user is sitting, the move-events from the right glove are disabled. Disable, here, means that those events, with regard to the

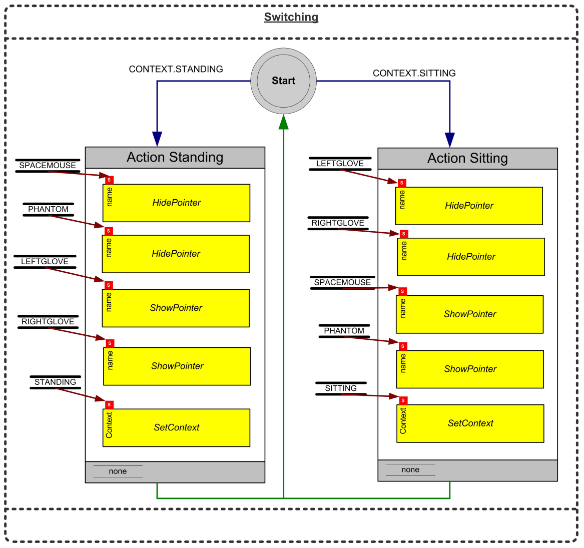


Figure 8: The 'Context Switching' diagram from the case study.

virtual hand selection technique, are not attached to the current active context.

5. DISCUSSION

The aforementioned approach describes a well-structured solution in order to support context detection, switching and handling using a single graphical notation, NiMMiT. Since we defined context as a n-dimensional vector of observables, called 'Context Units', very complex context systems, with several orthogonal units can be built. As the maximum number of context is the result of the cross product of all unit values, grouped per context unit, the number of contexts may grow rapidly with the number of units and values. This implies a possible explosion of the number of states in the diagrams, inevitably resulting in complex diagrams, even in spite of the proposed separation of the diagrams. We believe, however, that the amount of different context units and their values will remain limited in practical situations with regard to virtual environments, such that the complexity of the diagrams won't be catastrophical.

Alternatively, the other extremum can occur as well. One could argue that for very simple context systems, in which two contexts exist, where no specific conditions have to be checked, and no additional actions before the actual switch have to be taken, the discussed approach may be 'overkill'. In those very simple cases, however, both the 'Context Detection' and the 'Context Switching' diagram can be merged into a single diagram, containing both the basic events (coming from the devices), as well as the conditions and the actions.

Finally, a last point of discussion, is the fact that the proposed approach can be used to define quite powerful context handling. Indeed, in the example, we showed how the 'Context Switching' diagram has been used to perform simple visible actions inside the virtual environment such as enabling or disabling devices. Alternatively, it is also possible to assign values to a label (variable) in the 'Context Switching' diagram. Using input and output labels of the NiMMiT diagrams, these values can be exchanged between the different diagrams running in parallel. Imagine for instance an interaction diagram that has to use the position values for both the dominant and non-dominant hand. According to the handedness of the user, the function of both devices may be altered. An easy solution then is to use a label indicating which device is assigned to what hand. The mechanism of changing label values in the context handling diagram depending on the context, can be very useful for many types of contexts such as user preferences (e.g. strong or soft force feedback).

6. CONCLUSION AND FUTURE WORK

In this paper we presented an approach to design context-aware multimodal virtual environments using a model-based user interface design process. We applied NiMMiT, a graphical notation for the description of the user interaction in a VE to create a context system based on an 'Event-Condition-Action' paradigm. For the context system, two NiMMiT diagrams are used, one for detecting the context and checking the conditions, and a second diagram for performing the context switch. Using this approach, both diagrams remain simpler, more modular and better maintainable. In order to validate our approach we described a case study illustrating the proposed context modelling system. We believe that the proposed context system provides the designer with a powerful tool to create context-aware virtual environments while applying a model-based approach.

In the near future, our approach will be further explored using several other scenarios where user experience or preference can be modelled using context. Another interesting venue to investigate, is to make the NiMMiT diagrams self-evolving: instead of having a closed world assumption with regard to context this would allow for new context states to become available with the creation of new context detection and switching rules.

7. ACKNOWLEDGMENTS

Part of the research at EDM is funded by the ERDF (European Regional Development Fund) and the Flemish government. The VR-DeMo project (IWT 030248) is directly funded by the IWT, a Flemish subsidy organisation.

8. REFERENCES

[1] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.

[2] W. Beer, V. Christian, A. Ferscha, and L. Mehrmann. Modeling Context-aware Behavior by Interpreted ECA Rules. volume 2790, pages 1064–1073. Springer.

[3] D. A. Bowman, E. Kruijff, J. J. LaViola, and I. Poupyrev. *3D User Interfaces, Theory and Practice*. Addison-Wesley, 2005.

[4] D. Carr. Interaction object graphs: An executable graphical notation for specifying user interfaces. In *Formal Methods for Computer-Human Interaction*, pages 141–156. Springer-Verlag, 1997.

[5] T. Clerckx. *Model-Based Development of Context-Aware Interactive Applications in Ambient Intelligence Environments*. PhD thesis, transnationale Universiteit Limburg, June 2007.

[6] K. Coninx, E. Cuppens, J. De Boeck, and C. Raymaekers. Integrating support for usability evaluation into high level interaction descriptions with NiMMiT. In *Proceedings of 13th International Workshop on Design, Specification and*

Verification of Interactive Systems (DSVIS'06), volume 4385, Dublin, Ireland, July 26–28 2006.

[7] J. Coutaz and G. Rey. Foundations for a Theory of Contextors. In C. Kolski and J. Vanderdonckt, editors, *Computer-Aided Design of User Interfaces III*, volume 3, pages 13–33. Kluwer Academic, 2002. Invited talk.

[8] J. De Boeck, C. Raymaekers, and K. Coninx. A tool supporting model based user interface design in 3d virtual environments. In *Proceedings of the International Conference on Computer Graphics Theory and Applications (GRAPP08)*, Funchal, Portugal, January 22–25 2008.

[9] J. De Boeck, D. Vanacken, C. Raymaekers, and K. Coninx. High-level modeling of multimodal interaction techniques using nimmit. *Journal of Virtual Reality and Broadcasting*, 4(2), September 2007. `urn:nbn:de:0009-6-11615`.

[10] A. K. Dey. *Providing Architectural Support for Building Context-Aware Applications.* PhD thesis, College of Computing, Georgia Institute of Technology, Dec. 2000.

[11] P. Dragicevic and J.-D. Fekete. Support for input adaptability in the ICON toolkit. In *Proceedings of the 6th international conference on multimodal interfaces (ICMI04)*, pages 212–219, State College, PA, USA, 2004.

[12] B. Dumas, D. Lalanne, D. Guinard, R. Koenig, and R. Ingold. Strengths and weaknesses of software architectures for the rapid creation of tangible and multimodal interfaces. In *TEI '08: Proceedings of the 2nd international conference on Tangible and embedded interaction*, pages 47–54, 2008.

[13] R. Etter, P. Costa, and T. Broens. A Rule-Based Approach Towards Context-Aware User Notification Services. pages 281–284, 2006.

[14] V. Fernandes, T. Guerreiro, B. Araújo, J. Jorge, and a. P. Jo˙Extensible middleware framework for multimodal interfaces in distributed environments. In *ICMI '07: Proceedings of the 9th international conference on Multimodal interfaces*, pages 216–219, 2007.

[15] P. Figueroa, M. Green, and H. J. Hoover. InTml: A description language for VR applications. In *Proceedings of Web3D'02*, pages 53–58, Arizona, USA, Februari 2002.

[16] F. Flippo, A. Krebs, and I. Marsic. A framework for rapid development of multimodal interfaces. In *ICMI '03: Proceedings of the 5th international conference on Multimodal interfaces*, pages 109–116, 2003.

[17] S. Irawati, S. Ahn, J. Kim, and H. Ko. VARU Framework: Enabling Rapid Prototyping of VR, AR and Ubiquitous Applications. In *Virtual Reality Conference, 2008. VR'08. IEEE*, pages 201–208, 2008.

[18] R. Kernchen, P. Boda, K. Moessner, B. Mrohs, M. Boussard, and G. Giuliani. Multimodal user interfaces for context-aware mobile applications. In *16th Annual IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, pages 2268–2273, 2005.

[19] J. Lee, G. Rhee, H. Kim, K. Lee, Y. Suh, and K. Kim. Convergence of Context-Awareness and Augmented Reality for Ubiquitous Services and Immersive Interactions. In *Computational Science and Its Applications - ICCSA 2006*, pages 466–474. Springer, 2006.

[20] C. Li and K. Willis. Modeling context aware interaction for wayfinding using mobile devices. In *MobileHCI '06: Proceedings of the 8th conference on Human-computer interaction with mobile devices and services*, pages 97–100, 2006.

[21] B. MacIntyre and S. Feiner. Language-level support for exploratory programming of distributed virtual environments. In *UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 83–94, 1996.

[22] F. V. Mario Gutiérrez, Daniel Thalmann. Semantic virtual environments with adaptive multimodal interfaces. In *11th International Conference on Multimedia Modelling, MMM2005*, pages 277–283, 2005.

[23] D. Navarre, P. Palanque, R. Bastide, A. Schyn, M. Winckler, L. Nedel, and C. Freitas. A formal description of multimodal interaction techniques for immersive virtual reality applications. In *Proceedings of Tenth IFIP TC13 International Conference on Human-Computer Interaction*, Rome, IT, September 12–16 2005.

[24] J. M. S. O., J. Serrat, K. Yang, and E. S. C. Modelling context information for managing pervasive network services. In *Proc. of the International Conference on Modelling and Simulation (ICMSŠ05)*, pages 35–39, 2005.

[25] D. Preuveneers, J. Van den Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, Y. Berbers, K. Coninx, V. Jonckers, and K. D. Bosschere. Towards an Extensible Context Ontology for Ambient Intelligence. In P. Markopoulos, B. Eggen, E. Aarts, and J. L. Crowley, editors, *Second European Symposium on Ambient Intelligence*, volume 3295 of *LNCS*, pages 148 – 159, Eindhoven, The Netherlands, Nov 8 – 11 2004. Springer.

[26] C. Raymaekers, K. Coninx, J. D. Boeck, E. Cuppens, and E. Flerackers. High-level interaction modelling to facilitate the development of virtual environments. 2004 May 12–14. Proceedings of Virtual Reality International Conference, Laval, FR.

[27] C. Rousseau, Y. Bellik, and F. Vernier. Multimodal output specification / simulation platform. In *ICMI '05: Proceedings of the 7th international conference on Multimodal interfaces*, pages 84–91, 2005.

[28] M. Serrano, L. Nigay, J.-Y. L. Lawson, A. Ramsay, R. Murray-Smith, and S. Denef. The openinterface framework: a tool for multimodal interaction. In *CHI '08: CHI '08 extended abstracts on Human factors in computing systems*, pages 3501–3506, 2008.

[29] T. Sohn and A. K. Dey. icap: an informal tool for interactive prototyping of context-aware applications. In *CHI Extended Abstracts*, pages 974–975, 2003.

[30] L. Vanacken, E. Cuppens, T. Clerckx, and K. Coninx. Extending a dialog model with contextual knowledge. In M. Winckler, H. Johnson, and P. A. Palanque, editors, *TAMODIA*, volume 4849 of *Lecture Notes in Computer Science*, pages 28–41. Springer, 2007.

[31] J. Willans and M. Harrison. A toolset supported approach for designing and testing virtual environment interaction techniques. *International Journal of Human-Computer Studies*, 55(2):145–165, August 2001.