An Event-Condition-Action approach for Contextual Interaction in Virtual Environments

Lode Vanacken[1], Joan De Boeck[1], Chris Raymaekers[1], and Karin Coninx[1]

[1]Hasselt University - tUL - IBBT, Expertise Centre for Digital Media (EDM),
Wetenschapspark 2, B-3590 Diepenbeek, Belgium
lode.vanacken,joan.deboeck,chris.raymaekers,karin.coninx@uhasselt.be

Abstract. In order to support context-dependency in model-based development, three components need to be realised: Context Detection, Context Switching and Context Handling. Context detection is the process for detecting changes in context, while context switching brings the system in the new state that needs to be supported. Finally, context handling adapts the interaction possibilities to the current context. In this paper we discuss an approach for context detection and switching for virtual environments that is based on the Event-Condition-Action paradigm. Both context detection and switching are split-up and supported by our graphical notation for the design of multimodal interaction techniques. The main advantage of this approach is that we provide the designer with a flexible context system, supported by scalable diagrams.

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

1 Introduction and Related Work

The development of interactive computer applications takes much time, especially for the design and implementation of the user interface. This is in particular true for 3D multimodal interfaces for Virtual Environments (VEs). The process of creating or selecting interaction techniques for such interfaces is not straightforward. A large amount of possibilities exist with regard to input and output devices and the combination of these with respect to the interaction techniques being designed. One possible approach that can be applied in order to simplify the development process is using model-based user interface design as described in [4, 10].

In model-based user interface design (MBUID) different models are used through gradual progression. Typically, such a process starts at the level of a task model, moves over several other models, such as the dialog model up to the final user interface. Models can be transformed from one model into another or can be combined. Specifically for the design of a VE, it is necessary to model the multimodal interaction techniques, such as 'object manipulation'. Several high level notations exist, which can be used for this purpose [5, 7, 10]. In our research we use NiMMiT (Notation for Multimodal Interaction Techniques) [5] for describing the interaction.

The use of context information in a MBUID process when developing mobile or hand-held interfaces, already has been studied thoroughly [2, 8]. Indeed, a mobile application can have different locations, platforms or services which allow other features or actions to be available, resulting in interfaces that must adapt to each context. The same can be true in a VE application where the context is often defined by the available devices (and input/output modalities), external parameters such as the experience level of the user, or whether the user is seated or standing. All those parameters may have their influence on the interaction with the system.

Without special facilities for context in the diagrams, these features have to be supported in an ad-hoc manner. Clerckx [3] distinguishes several levels at which context may have its influence. A context system can be represented at the task or the dialog level, but in this work we focus on the dialog level. Such a context system consists of three components. Firstly, changes in context need to be sensed through a context detection system. Next, the system needs to react upon this change. The context switching makes sure that only parts of the system that react on the new context are active, while interaction techniques can act upon the current context through the context handling system.

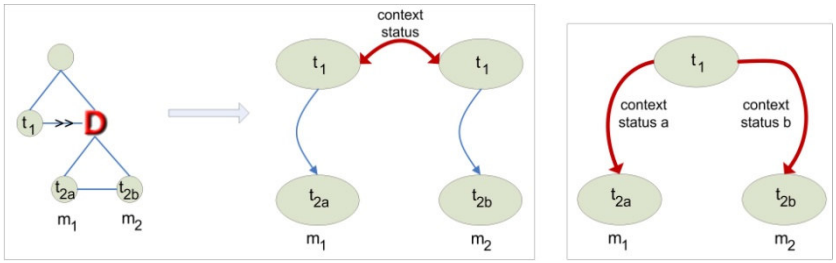
We will first briefly discuss our approach to handle contextual knowledge at the dialog level as earlier proposed in [9]. In section 3 we elaborate on how this approach can be improved to dynamically support context switches using the *Event-Condition-Action* (ECA) [1, 6] paradigm. Using this paradigm we achieve a scalable approach which can be supported by (semi-)automatic generation. As we will be using NiMMiT as a high level notation for interaction techniques, we can reuse the existing run time system, and as we can assume that designers may already know this notation from the design of their user interaction, they do not need to learn a new notation to model context information.

2 Runtime Context

In earlier work [9], we introduced handling context knowledge using NiMMiT. The graphical notation NiMMiT, inherits the formalism of a state-chart in order to describe the (multimodal) interaction within the virtual environment. Furthermore, it also supports data flow which occurs during the interaction, as well. A more detailed description of NiMMiT can be found in [5].

Our approach to integrate contextual information was inspired from earlier results of research in the area of model-based design [3]: (1) incorporating context in task and dialog modelling and (2) adding modality constraints to tasks. We discuss a combination of these two approaches, enabling context-aware selection of modalities.

Consider for instance a VE application that has two setups (external contexts), which differ in devices/modalities to be used. An application, that can be used in either an immersive setup using stereo vision and gloves or in a desktop setup with a keyboard and a mouse, illustrates this idea.



(a) (b)

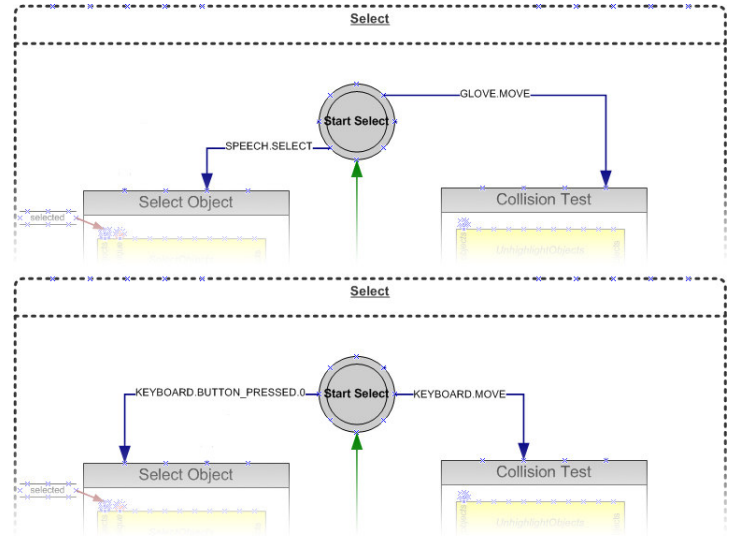
Fig 1. (a) Combining modality constraints with the decision task notation (task model and dialog model). (b) Merged dialog models.

One way to incorporate this, is to introduce the contexts at the task level [3], this means that according to the contextual constraint (m_1 and m_2), a different task is selected, as is illustrated in figure 1(a). Task t_2 is divided into two distinct tasks t_{2a} and t_{2b}. This approach expands into two different dialog models, one for each context. Both models contain two states, containing one task each, in this example. According to the context m_1 or m_2, the appropriate dialog model is selected.
This solution is suitable when a context switch enables other tasks or requires other interaction techniques and thus affects the task model. The drawback is the duplication of the dialog model for each context. Alternatively, in a typical VE situation as in our example, the tasks may remain the same for each context. In this situation defining 'context' at the dialog level is considered as a more efficient approach.

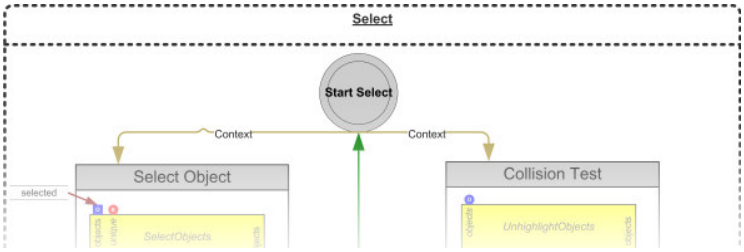
Moreover, to overcome the problem of duplication both approaches may be combined: instead of having two distinct dialog models, we simply merge them together and make a distinction only where a difference is made by a context status. This is illustrated in figure 1(b). The two states containing t_1 are merged into one state in one and the same dialog model, but a choice is made to the appropriate state transition depending on the context. In this way the decision at the task level is modelled at the dialog level.

In our former work [9], we implemented this approach at the dialog level using our interaction description model: NiMMiT. In figure 2 an example of our approach is depicted. In figure 2(a) we can see that in the 'Start'-state several different events (modalities) could trigger the execution of the task chains. Using 'context' information, we are able to attach a certain context to a certain event or modality, such that, depending on this context, only those events belonging to that context 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 2(a)) which would make maintenance much harder.

Adding this contextual knowledge to events transforms the view of the diagram depending onto which context of the diagram we are viewing. A part of the resulting diagram containing context arrows is shown in figure 2(b).



(a) Events active in two different contexts.



(b) Events were attached to context arrows.

Fig 2. Our approach to contextual knowledge at the dialog level.

3 Defining a Context System

3.1 Context Detection and Switching

Our previous work concentrated on context handling. The detection and switching was handled through explicit user interaction. In order to realise a context-dependent interaction, it is also necessary to automate this detection and handling of the context. The following section therefore discusses how this has been integrated into our existing system.

The process of context detection and switching can be seen as an Event-Condition-Action process. A certain *event* or combination of events can signal a change in

context, possibly depending on certain *conditions*. Finally if the conditions are met, it might be necessary to perform an *action* such that the context switch is finalised. For instance, a user may stand up from his chair (*event*). Before executing a context switch, we must ensure that he wears tracked gloves (*condition*). If the 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 we assume that 'standing up' can be recognised as an event. If this would not be possible, 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*).

In order to keep the design as modular as possible, this Event-Condition-Action process is split up in two parts. One part is the context detection, the other handles the context switch. According to Event-Condition-Action, the context detection should identify what events to listen for, checks whether or not the conditions are fulfilled and it eventually triggers a 'context switch' event. This event is recognised by the Context Switching part.

The Context Switching part captures the 'context switch' event and executes the actions that are necessary before switching to the target context.

Dividing the process in two distinct parts, connected through the 'context switch' event, has the advantage that the code necessary for checking the conditions is separated from the action code. In the next section we will explain why this will lead to smaller and well-organised diagrams.

3.2 Implementation through NiMMiT Diagrams

In our research, interaction techniques are defined using NiMMiT diagrams. As NiMMiT offers a convenient way to describe systems in which events fire a set of tasks, we propose that both the context detection and switch can be implemented using NiMMiT diagrams, as well. Besides this reason, designers already know NiMMiT from the design of the interaction itself, which allows them to model context without having to learn a new modelling notation. Finally, the run time system to execute NiMMiT diagrams is already realised.

The Context Detection NiMMiT diagram defines a state for each 'context' where the relevant events that can evoke a context switch are available. The events activate a task chain, which checks the condition by a more complex set of tasks. When the condition is not met, nothing happens. Otherwise, before moving on to a new state, reflecting the new context, the task chain has to fire a 'context switch' event. A template of such a diagram can be seen in figure 3. Two contexts, SITTING and STANDING are represented using the two states 'Context-SITTING' and 'Context-STANDING'.

A second NiMMiT diagram, responsible for the action, contains a state for each context. In each state, the respective 'context events' are awaited. Upon occurrence of such an event a task chain is fired, containing the code that has to be executed before the context switch. This code might be enabling or disabling certain devices, showing or hiding objects or controls in the world, etc. The last action in this task chain, before moving to the new context state, is explicitly setting the context, so that the running

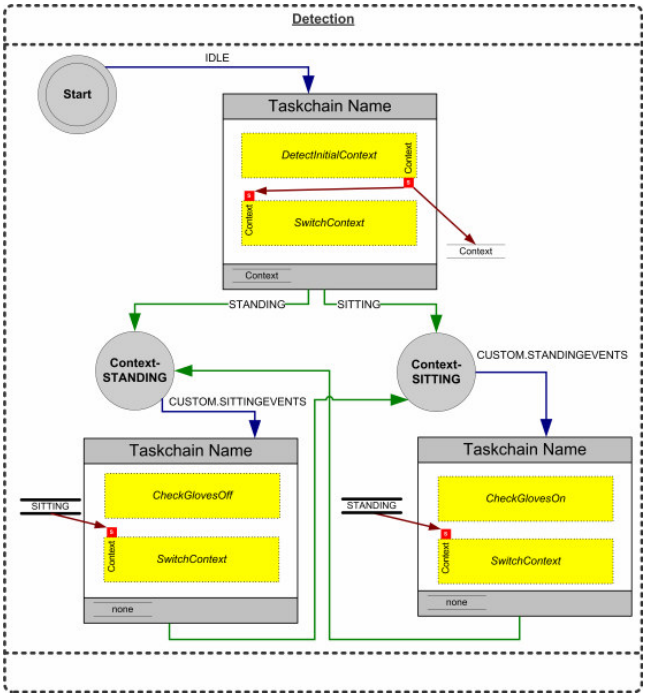


Fig 3. Implication of the proposed context system at the task level.

NiMMiT diagrams that define the user interaction can adapt to the context switch, and handle the new context.

The NiMMiT diagrams defining the context system, will share a similar pattern among different projects. Independent of the nature or the number of contexts, each context will be represented as a state in both diagrams. Each context transition will then be represented by an *event* arrow in the context detection diagram and a 'context event' arrow in the context switching diagram, invoking a task chain.

The similar patterns of these 'context' diagrams open the opportunity for an editor to generate a template diagram that can be completed by the designer. 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 Implications at the dialog level and task level

The proposed context detection and switching system is a process which is active during the entire execution of the application. This obviously has its implications on the task level and dialog level of the model-based process. For the dialog model, every state contains two new tasks which are performed concurrently with the normal tasks, these tasks include the context detection and the context switching diagrams.

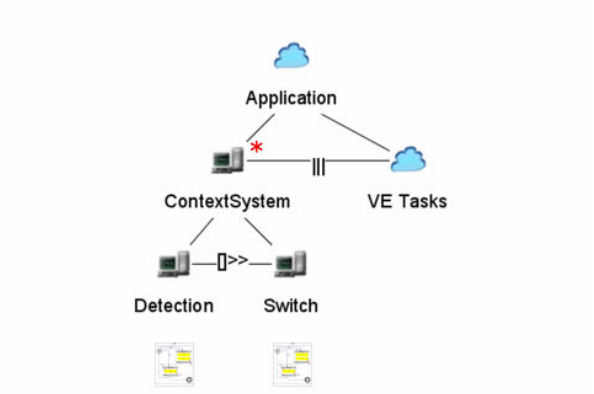


Fig 4. Implication of the proposed context system at the task level.

Considering the task level, this means that a new subtree concurrent with all other subtrees is part of the task hierarchy, as can be seen in figure 4. The ContextSystem tasks perform the detection and switching of the context, while the VE tasks may change their execution based upon this context switch, as discussed in figure 2.

4 Discussion

A possible problem related to describing context in a state chart, is the fact that these diagrams can suffer from a state explosion if the number of different contexts becomes too high. This especially is true if we consider a context as an n-dimensional vector of observed values. If these observed values are orthogonal to each other, this will result in an exponential explosion of contexts, and hence will make the diagrams hard to manage, even despite the division in two separate diagrams.

However, in some cases the contexts which are applied in a VE are rather simple, which means that either the condition or the action is not present. In that case it may be overkill to design two diagrams: one translating '(device) events' into 'context events', and another responding to those 'context events'. In this situation, the designer may decide to combine both diagrams and hence either add context switching code to the context detection diagram, or adding '(device) events' to the context switching diagram.

5 Conclusion

Context detection and context switching are necessary components of an overall approach for context-dependency in model-based development. In this paper, we presented our approach for context detection and switching.

Event-Condition-Action rules are used as a basis for the context system. Both context detection and switching are split up and supported by NiMMiT diagrams, this results in a scalable approach. The run time system for NiMMiT is already present and the designer also uses NiMMiT to design interaction techniques, therefore the usage of NiMMiT eliminates the overload of having to learn a new notation for context modelling or adding a new module to the run time system. As the general pattern of the context detection and switching diagrams is similar among different projects, the approach also opens the opportunity for a
(semi-)automatic generation of the diagrams by an editor. In the future, we would like to further investigate how our approach compares to other approaches and validate it using case studies.

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.

References

1. Beer, W., Christian, V., Ferscha, A., Mehrmann, L.: Modeling Context-aware Behavior by Interpreted ECA Rules. Proceedings of EUROPAR03. LNCS 2790, 1064–1073
2. Capra, L., Emmerich,W., Mascolo, C.: Carisma: Context-aware reflective middleware system for mobile applications. IEEE Trans. Software Eng. 29(10), 929–945 (2003)
3. Clerckx, T.: Model-based development of context-aware interactive applications in ambient intelligence environments. Ph.D. thesis, transnationale Universiteit Limburg (2007)
4. De Boeck, J., Gonzalez Calleros, J.M., Coninx, K., Vanderdonckt, J.: Open issues for the development of 3d multimodal applications from an MDE perspective. In: MDDAUI 2006.
5. De Boeck, J., Vanacken, D., Raymaekers, C., Coninx, K.: High-level modeling of multimodal interaction techniques using nimmit. Journal of Virtual Reality and Broadcasting 4(2) (2007)
6. Etter, R., Costa, P., Broens, T.: A Rule-Based Approach Towards Context-Aware User Notification Services. Proceedings of the IEEE ICPS'06 pp. 281–284 (2006)
7. Figueroa, P., Green, M., Hoover, H.J.: InTml: A description language for VR applications. In:Proceedings of Web3D'02, pp. 53–58. Arizona, USA (2002)
8. Sohn, T., Dey, A.K.: icap: an informal tool for interactive prototyping of context-aware applications.In: CHI Extended Abstracts, pp. 974–975 (2003)
9. Vanacken, L., Cuppens, E., Clerckx, T., Coninx, K.: Extending a dialog model with contextual knowledge. In: TAMODIA, LNCS, vol. 4849, pp. 28–41. Springer (2007)
10. Willans, J., Harrison, M.: A toolset supported approach for designing and testing virtual environment interaction techniques. International Journal of HCS 55(2), 145–165 (2001)