Facilitating Adaptation in Virtual Environments using a Context-Aware Model-Based Design Process

Johanna Renny Octavia, Lode Vanacken, Chris Raymaekers, Karin Coninx, and Eddy Flerackers

Hasselt University - tUL - IBBT
Expertise Centre for Digital Media
Wetenschapspark 2, 3590 Diepenbeek, Belgium
{johanna.octavia,lode.vanacken,chris.raymaekers, karin.coninx,eddy.flerackers}@uhasselt.be

Abstract. Designers and developers of virtual environments have to consider that providing adaptation in virtual environments is important to comply with users' different characteristics. Due to the many possibilities of adaptation that a designer can think of, it is necessary to support the integration of adaptation in the application in a rapid and practical way. We propose to achieve this by adopting the VR-DeMo model-based user interface design (MBUID) process which supports context. In this paper, we strive to integrate adaptation in virtual environments using a context-aware design process and present a validation of this approach with two case studies, namely supporting the adaptation of switching between interaction techniques and adapting the interaction technique itself. These case studies learned us that adaptation can be easily realized using our context-aware model-based design process.

Key words: adaptation, virtual environments, context-aware, model-based user interface design

1 Introduction

Working in virtual environments brings along complex interaction for its users. One difficulty may develop from the vast range of possible 3D interaction techniques employed in virtual environments. This may be cumbersome because the user can not recognize the interaction technique immediately [1]. Users may encounter extraneous cognitive load when having to determine which interaction technique is best for them to use in a certain environment condition. Providing adaptation in virtual environments can be seen as a prospective solution for overcoming this particular problem. Depending on the context of use, the user interface can be adapted for a specific user, based on the user's characteristic [2]. For example, by offering the user only a limited number of interaction techniques derived from the information of his/her preference and ability.

It is our intention to simplify user interaction and make it more natural and intuitive by integrating adaptation into 3D user interfaces in virtual environments. However, the scope of adaptation can be very broad in the sense that there can be many kinds of adaptation to be implemented. Ideally, we would like to support as many adaptations as reasonable from the usability perspective. On the other hand, the integration of such an adaptation in the application may require a lot of ad hoc programming and can bring designers and developers into a lengthy, time-consuming iterative design process. Therefore, we adopt a model-based user interface design (MBUID) process to enable quick prototyping (i.e. by easily modifying diagrams) to investigate the possibility of adaptation in virtual environments. Model-based design approaches have been regularly used to realize implementation of adaptive user interfaces for a broader range of applications over the years, including visual adaptation of user interfaces [3], adaptation for plasticity [4] and website adaptation [5].

In our MBUID approach, we consider adaptation as a form of context. We define context as an artefact influenced by different factors such as user, environment, platform and others. In virtual environments, context can be defined by the internal parameters (e.g. virtual world) and external parameters (e.g. user's characteristics) applied in a certain situation. As we wish to modify those parameters for the purpose of adaptation, we can look at the virtual environment as a context-aware system. Customarily, designing context-aware systems results in low-level programming code that is difficult to maintain. Using MBUID process that supports context, designers and developers are enabled to create context-aware systems in a higher and more abstract level.

In this paper, we describe an approach to integrate adaptation in virtual environments using a context-aware design process. We use the VR-DeMo model-based process [6] in the approach, together with the CoGenIVE tool [7] to support designers and developers in applying our MBUID approach. Further, we will discuss different kinds of adaptation that we want to provide and how our system can be used to support adaptation. The validation of the proposed approach by means of two practical case studies is also discussed in this paper.

2 Adaptation in Virtual Environments

We recognize the significance of supporting adaptation to enhance user interaction in virtual environments. However, we also notice that adaptation in virtual environments is explored less often than in WIMP applications probably partly due to the fact that interaction in virtual environments is more complex. Yet, adaptivity of interaction in virtual environments has been investigated through learning user behavior [8] and user's preferred method of interaction [9].

It is essential that the adaptation should occur according to the current context such as the environment condition of the virtual world and the preferences and abilities of the users. As mentioned earlier, there are many possible forms of adaptation that can be implemented in virtual environments. In this work, we classify adaptation into three types: (I) switching between interaction techniques, (II) adapting the interaction technique itself, and (III) enhancing the interaction technique with modalities. Through our case studies, we would like to investigate the first two types of adaptation as we already experimented with the third type of adaptation in our previous work [10, 11].

The first type of adaptation, switching between interaction techniques, involves the action of offering the most suited interaction technique for a user in a certain situation (e.g. environment condition of the virtual world, position of the user). When performing one task in virtual environments, users have the opportunity to choose from several possible interaction techniques to execute the particular task. For example when executing a selection task [1], users may choose between a technique using a virtual hand metaphor and a technique using a virtual pointer metaphor. The choice of interaction techniques can also vary depending on the environment condition or view of the virtual world. In particular, users may prefer one technique over the other when selecting an object in a dense environment, which might be different when in a sparse environment. So, we are interested to investigate the possibility of enabling users to switch between different techniques while performing the same task, as a result of adapting to different environment conditions. We will discuss this in the first case study.

The second type, adapting the interaction technique itself, is basically adjusting the parameters of the interaction technique, which could influence how it should be performed (e.g. motion sensitivity, personal preference). Users have different levels of physical abilities that may affect them when performing an interaction technique in virtual environments. For instance, a user may have a lot of tremor in his hand that decreases his efficiency in accomplishing a selection technique and lessen the accuracy of object selection. To resolve this, we could change certain parameters (e.g. viscosity, force strength) of a task as an adaptation to the user's specific attributes (e.g. degree of tremor). Therefore, we are also intrigued to examine the opportunity of adapting the interaction technique itself by adjusting its sensitivity according to the user's need. This will be the focus of our second case study.

Prior to determining how the adaptation should behave, the designer should assess what the exact need of adaptation is. This includes constructing a user model at design time and acquiring information about the user itself from that model at runtime. A user model plays an important role as a base in the adaptation process of user interfaces to comply with different needs of users. It contains significant information and knowledge about a user's characteristics, which is learned from his traces of interaction with a system/an interface [2]. User characteristics are generally modeled as part of the context description [12]. By closely collaborating with the user model, the adaptation engine will designate the suitable adaptation for a particular user in a particular context.

We are aware of the vital contribution of a user model in providing adaptation in virtual environments. Our previous work [13] constructed a general user model for a target acquisition task in virtual environments. This user model is intended for first-time users, who have no interaction history whatsoever, to benefit straightaway from adaptation in virtual environments. We believe that the idea of supporting adaptation in virtual environments may be realized with the help of context-aware systems.

3 Designing Context-Aware Virtual Environments

In previous work, we presented an approach to design context-aware multimodal virtual environments using the VR-DeMo model-based user interface design process based on an 'Event-Condition-Action' paradigm [11]. The context system made it possible to switch between different modalities for a certain interaction technique based on the context. In this section we shortly introduce this approach and discuss how it can be used for other types of adaptation as well.

The main difference, compared to traditional context-aware systems [14, 15], is that the context system resides at the dialog level. More in particular, it is realized using our interaction description model 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 [16].

To support context-dependency in model-based development, three components were realized: 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.

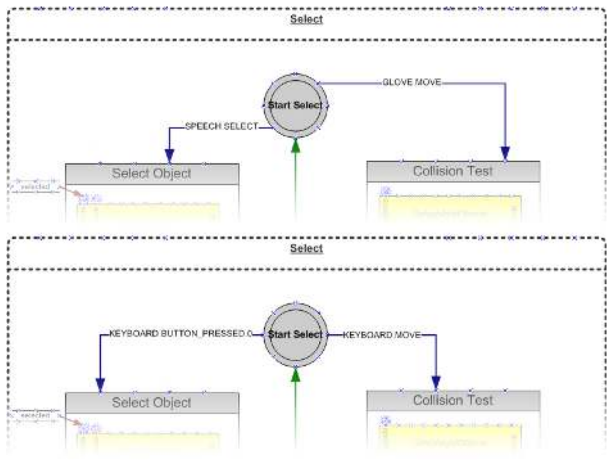
With regard to adaptation, the context detection part provides the link with the user model, it will query the user model for information about the conditions which it is checking. The context switching part mostly takes care of actions which should be performed such that the adaptation can occur correctly or that it changes parameters which will adapt the interaction techniques. Finally, the context handling part makes sure that the adaptation can be realized.

In the next sections we will shortly describe the three different parts and an extension on the approach which enables us to also support different context-aware interaction techniques at the dialog level. The case studies elaborate on the practical use of the context system.

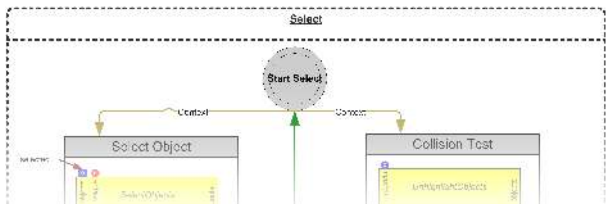
3.1 Context Detection and Switching

The realization of the context detection and switching part is performed by adopting 'Event-Condition-Action' rules [17, 18]. A certain *event* or combination of events can signal a context switch. After the event has been recognized, certain *conditions* have to be met before switching the context. When these conditions are fulfilled, it might be necessary to first perform some *actions* before finalizing the context switch. For instance, a user may stand up from the chair (*event*). Before executing a context switch, we must ensure that the user wears the tracked gloves (*condition*). If this condition is met, we disable the toolbars needed in a desktop setup and connect the cursor to make the glove visible (*action*).

The context detection and switching will be performed using NiMMiT. Usually, one diagram will be used for detection, listening to the events and checking the conditions, and one for switching, performing the actions. We use two diagrams so that the solution is as modular as possible. Of course, it is possible to integrate both diagrams into one diagram if either the detection or the switching diagram is very simple.



(a) Two NiMMiT diagrams with the same states and event flow, but different events depending on the context



(b) Events were attached to context arrows

Fig. 1: Context Handling [10].

3.2 Context Handling

The context handling part finalizes the context system. The context system needs to be able to create context-aware NiMMiT diagrams that represent the interaction techniques. In this section, we will shortly give an overview of the approach using an example containing context-aware modality selection [10].

In Figure 1, an example of this approach is depicted. Figure 1a shows that in the 'Start'-state several different events (modalities) could trigger the execution of the task chains. Using context information, it is possible to attach a context to a certain event or modality in such a way that depending on the context only the corresponding 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.

It is important to note that if there was no support to couple events to a context then the same diagram should be created twice with different events (as in Figure 1a) 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 1b.

The approach presented has only been used to switch between modalities at the dialog level. No support for switching between different interaction techniques has been discussed yet. In the next section, we will present an approach to perform context-aware interaction technique selection at the dialog level. It is important to perform this to enable switching between interaction techniques.

3.3 Context-Aware Interaction Technique Selection at the Dialog Level

In Section 2, we introduced three different types of adaptation. The third type (enhancing the interaction technique with modalities) has been realized and investigated in prior work [10, 11] and the second type (adapting the interaction technique itself) can be realized in the switching part of the context system without any influence on the model-based design process. To support the first type (switching between interaction techniques), traditionally the task level would be used because of the fact that an interaction technique is similar to a task.

To show the approach which enables designers to switch between interaction techniques using context information, we will use the first case study to explain how the above context system can be used and how it differs from the traditional approaches.

Certain selection techniques are sometimes better suited for a certain context. To exploit this, it should be possible to switch between different interaction techniques. In Figure 2, a task model is presented in which the selection technique can be changed depending on the context. Transforming this task model to the corresponding dialog model will give a similar model as in Figure 3a. When more alternatives would be present, state explosion could occur [14]. To avoid this, we use a combination of our dialog model and NiMMiT.

In the previous section, explaining the context handling, we used NiMMiT and context arrows to be able to differ between modalities depending on the context (see Figure 1). The same approach can also be used to select between tasks depending on the context. To avoid the extra dialog model in Figure 3a, we create a NiMMiT diagram representing the 'Selection' decision task, which then will be used in our dialog model. Merging the both context possibilities

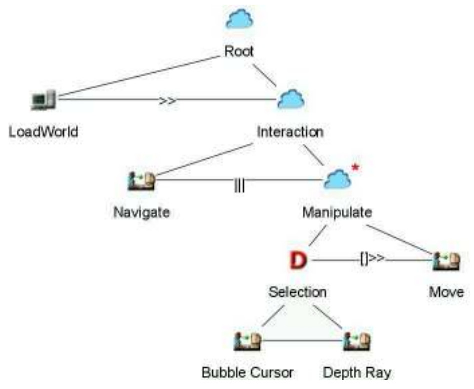


Fig. 2: Task model in which different selection interaction techniques (bubble cursor and depth ray) can be chosen depending on the context

into one NiMMiT diagram allows us to merge the two dialog models. This 'Selection'NiMMiT diagram will use context arrows from Section 3.2 to enable to correct interaction technique depending on the context.

The above approach, applied to the task model from Figure 2 can be seen in Figure 3. Instead of having two dialog models (Figure 3a), one is used containing the 'Selection' decision task (Figure 3b). This 'Selection'-task is represented through a NiMMiT diagram with 'IDLE'-event arrows to the possible interaction techniques (Figure 3c). In order to switch to the correct decision depending on the context, these 'IDLE'-event arrows are assigned to the bubble cursor or depth ray context (Figure 3d).

This strategy might be less intuitive to be performed by a designer. But if the designer is aware of this possibility, it is actually an easy way to support the first type of adaptation, switching between different interaction techniques depending on the context.

4 Case Studies

In this section we introduce two case studies which respectively show how the first and second type (see Section 2) of adaptation are realized using our context system. The same setup is used for both case studies as shown in Figure 4. The input device is a haptic device (Phantom premium) and a stereo screen.

4.1 Adaptation Type I: Switching between Interaction Techniques

Selection experiments [19, 20] showed that several aspects of the virtual world influence the performance of selection techniques differently. This means that there is no one best technique in all situations. This information can be used to indicate preferences and provide users with the 'best' selection technique at a right time. However, it should not be surprising that almost all users will

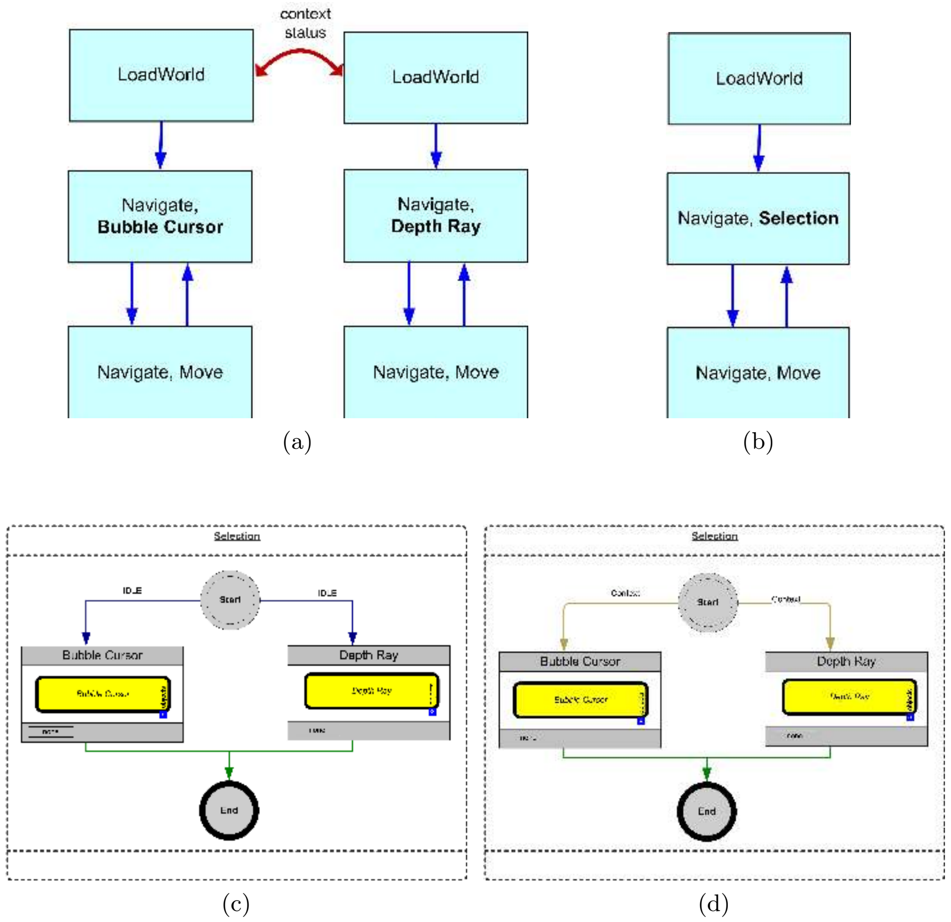


Fig. 3: (a) Two dialog models generated from the task model (b) The merged dialog model, containing the 'Selection' decision task (c) NiMMiT Diagram without context arrows, representing the 'Selection' decision task (d) NiMMiT Diagram with context arrows, assigned to the corresponding context (bubble cursor or depth ray), representing the 'Selection' decision task

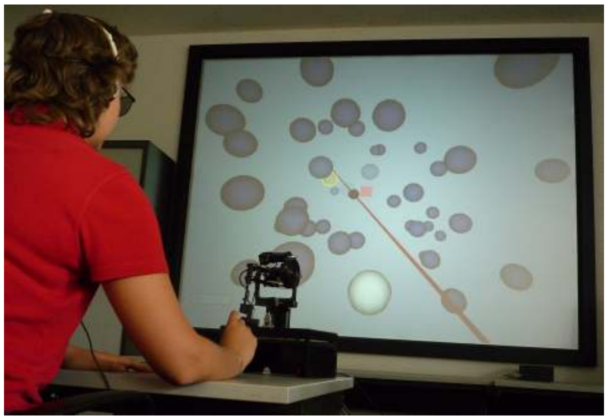


Fig. 4: The setup used in both case studies (stereo is disabled for visibility purposes)

slightly deviate from the expected performance for a certain selection technique in a certain environment condition. Because of this, adaptation seems to be a viable solution to explore. Although in general for all users we know the best technique for a certain situation, a particular user may perform better with another selection technique than the one proposed.

In this case study, we will offer the users two selection techniques based on different metaphors, the 3D bubble cursor (virtual hand) and the depth ray (virtual pointer). Not only because these techniques were found to perform better than the others in dense and occluded virtual environments [20], but also since we have constructed a user model [13] based on our selection experiment in different environment density conditions (sparse versus dense) and occlusion conditions (visible versus occluded). We will use the user model in combination with the current viewpoint of the user on the virtual world. Based on this viewpoint, we will derive the current environment condition (e.g. very sparse scene and almost no objects occluding each other). Then, we will provide the user with the selection technique which is best suited for him in this particular condition.

Context Detection and Switching Usually, the detection and switching diagram are split to have a more modular solution. However in this case study, no actions need to be performed after a new context has been detected. Therefore, we will combine the detection and switching diagram into one diagram since there is no reason to have a separate switching diagram.

First an initialization step (‘Start’-state to the ‘Init’-task chain) is performed (see Figure 5). This step queries for the initial context and executes an initial switch to that context (‘FireContextEvent’). This task makes the system fire a context event, which is usually used by the switching diagram to know which actions should be performed. We keep on firing the event because it could be used by other diagrams too. Since the switching diagram finalizes the context switch, we immediately perform the ‘SetContext’-task. Due to the fact that the different contexts (i.e. different selection techniques) are chosen by evaluating the same conditions in the ‘CheckViewpoint’-task, we only use one state instead of one state per context situation as we showed in previous work [11].

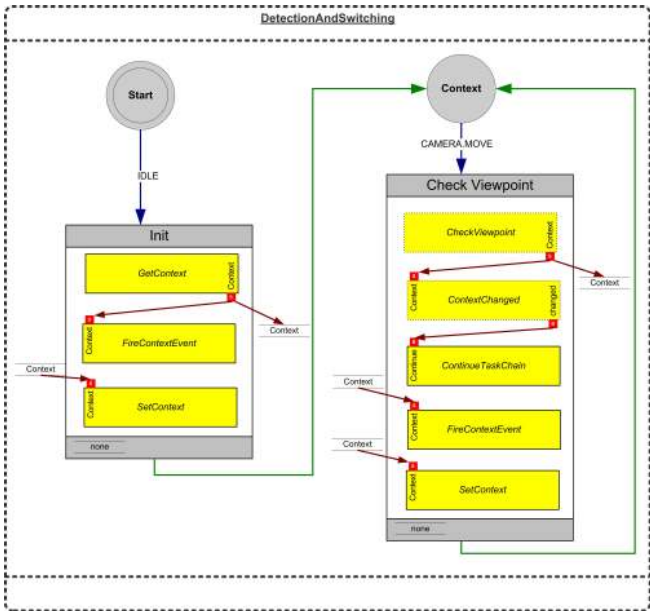


Fig. 5: Context detection and switching diagram - Case Study 1

We want to provide the user with different selection techniques depending on the viewpoint. Therefore the ‘CAMERA.MOVE’-event, the event fired when the viewpoint changes, indicates a possible context switch (*event*). When this event occurs, the viewpoint will be checked for the current environment condition. We will check for the amount of objects in the current viewpoint to assess the density of the objects. In combination with the density, we will also calculate how many objects overlap with each other so we can estimate the possibility that a user wants to select an occluded target (*condition*). By knowing the current environment condition, we can simply utilize the user model of the particular user to decide which selection technique should be chosen. The user model provides information about the most suitable selection technique for a user linked to a particular environment condition based on the user’s performance and preference (e.g. the bubble cursor technique in the dense and occluded condition).

When the ‘CAMERA.MOVE’-event occurs and the viewpoint is checked (‘CheckViewpoint’), we calculate the current context. To assure we only fire context change events when the context has changed, we add a construction that checks if the context has changed (‘ContextChanged’). If the context has changed, we output 'true' and then the context switches, else we output 'false'. The 'ContextChanged'-task can be seen as an extra condition to check before a new context is detected. The task that is responsible for checking the condition must collect its result in a 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 then the task chain is continued. When the value is 'false', this task throws an exception and activates the exception handling mechanism of the NiMMiT task chain. The current task chain is interrupted and the previous state is restored. This means that when the condition is not met, the original state is restored and no context switching event is generated.

Context Handling The handling part has to enable the correct selection technique depending on the context. We use the approach described in Section 3.3. The NiMMiT diagram presented in Figure 3d is the diagram which we will use in this case study, depending on the current context, the correct 'IDLE'-event arrow will be active so the correct selection technique will be activated.

There might be an issue when the context changes but a certain selection technique is already active. In the solution presented here, the selection technique currently active will not be aborted and the other selection technique can only become active after the selection is performed. While some applications might prefer this behaviour, it should also be possible to abort the current selection technique and enable the other selection technique. To realize this, an action would need to be added to change a label that tells the current running selection technique to stop immediately. The main drawback is that we need to change the selection technique itself by incorporating this aborting-check using the label.

4.2 Adaptation Type II: Adapting the Interaction Technique itself

In the second case study, we will show an example of the second type of adaptation where we change parameters of an interaction technique to influence the performance of the technique itself. In particular, we want to detect if a user's hand exhibits tremor during interaction and make such an adaptation to help reducing the hand motion tremor. For this purpose, we will use force feedback to enable applying viscosity to the input device which in the end will reduce the tremor. If no haptic device is available, it could also be possible to reduce the tremor using a low-pass filtering approach. A viscosity force is calculated using the velocity and a constant defined by the designer, where a high amount of viscosity feels like moving your hand in water.

We introduce three discrete levels of tremor (low, medium and high). In case of a low amount of tremor, we do not apply any viscosity because it is normal that the user's hand has some tremor during movement. For a medium and high amount of tremor, we apply two different constants that are defined manually.

Context Detection and Switching This case study will use two separate diagrams for detection and switching. The context detection diagram is very similar to the one of the previous case study (see Figure 5). However, there are two differences between the diagrams in both case studies. The first difference is the replacement of the ‘CheckViewpoint’-task by a ‘CheckTremor’-task, which tries to calculate the tremor of the hand. Calculating hand tremor is rather difficult and different mechanisms can be used depending on the target population [21]. We have implemented a straightforward approach, we use the velocity of the user’s hand as an indication for the degree of tremor. We buffer the velocity for a few seconds and check how many times its magnitude changes from speeding up or slowing down and vice versa.

The second difference is the removal of the ‘SetContext’-task from the context detection diagram, which is now inside the switching diagram. The switching diagram, represented in Figure 6, is fairly simple. It listens to the context events fired by the detection diagram and sets the corresponding viscosity constant depending on the degree of tremor.

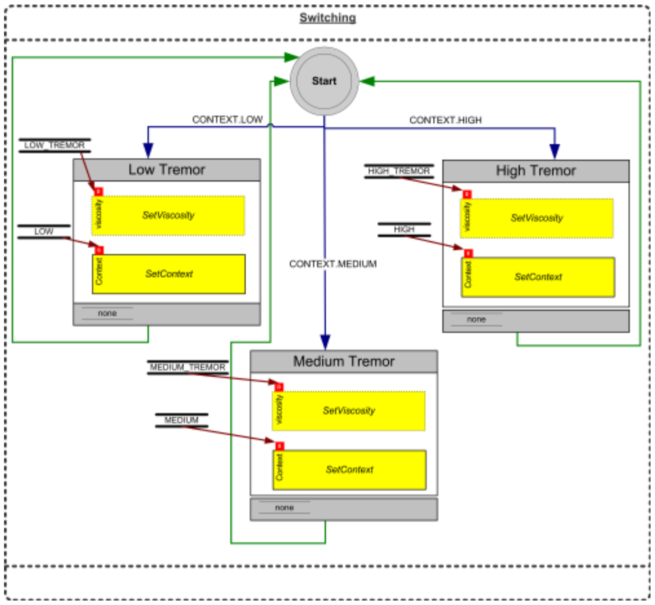


Fig. 6: Context switching diagram - Case Study 2

This case study provides three different discrete levels of adaptation with regard to the tremor (’LOW’, ’MEDIUM’ and ’HIGH’ degree of tremor). This solution might not be flexible enough, some designers may want to use a continuous scale instead. Note that other parameters might also have either a discrete or a continuous scale. It is possible to use continuous levels in the second case study. Instead of using three context levels, we would use two levels which indicate either ‘LOWER’ or ‘HIGHER’ degree of tremor. If we keep on detecting tremor, we could keep on firing the ‘HIGHER’ context as being detected[1] and keep on raising the viscosity constant. Otherwise if the tremor decreases, we can fire the ‘LOWER’ context as being detected (using the ‘FireContextEvent’-task).

Context Handling This adaptation only changes certain parameters of interaction techniques. It does not require another task chain to be executed depending on the context (adaptation type I) or other modalities to be enabled/disabled (adaptation type III). Therefore, the context handling part is not necessary to be applied in the system for realizing the second type of adaptation.

5 Conclusion

In this paper we showed how VR-DeMo, a context-aware model-based design process for virtual environments, can be used to support designers and developers to facilitate rapid development and prototyping of adaptation in virtual environments. We identified three possible types of adaptation and validated the proposed approach using case studies. These case studies showed that our context-aware model-based process is feasible to explore adaptation in virtual environments.

For future work, from the developers’ point of view, it might be interesting to further investigate how the user model could be more easily integrated in the context detection part of the system. From a usability perspective, we would like to further investigate how users react on adaptation that is realized by the system presented in this paper.

Acknowledgments

Part of the research at EDM is funded by the ERDF (European Regional Development Fund) and the Flemish government. The work described in this paper is funded by the transnational University Limburg (tUL).

References

1. Bowman, D.A., Kruijff, E., LaViola, J.J., Poupyrev, I.: 3D User Interfaces, Theory and Practice. Addison-Wesley (2005)
2. Rich, E.: Users are individuals: individualizing user models. International Journal Human-Computer Studies **51** (1999) 323–338
3. Nilsson, E.G., Floch, J., Hallsteinsen, S., Stav, E.: Model-based user interface adaptation. In Davies, N., Kirste, T., Schumann, H., eds.: Mobile Computing and Ambient Intelligence: The Challenge of Multimedia, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2005)
4. Sottet, J.S., Ganneau, V., Calvary, G., Coutaz, J., Favre, J.M., Demumieux, R.: Model-driven adaptation for plastic user interfaces. In: Interact. (2007) 397–410

[1] The ‘ContextChanged’-task has to be removed from the detection diagram.

5. Ceri, S., Dolog, P., Matera, M., Nejdl, W.: Model-driven design of web applications with client-side adaptation. In: ICWE 2004. Volume 3140 of LNCS., Springer Verlag (2004) 201–214
6. Coninx, K., De Troyer, O., Raymaekers, C., Kleinermann, F.: VR-DeMo: a tool-supported approach facilitating flexible development of virtual environments using conceptual modelling. In: Virtual Concept 2006, Cancun, Mexico (2006)
7. De Boeck, J., Raymaekers, C., Coninx, K.: A tool supporting model based user interface design in 3d virtual environments. In: GRAPP 2008, Funchal, Portugal (2008)
8. Celentano, A., Nodari, M., Pittarello, F.: Adaptive interaction in web3d virtual worlds. In: Proceedings of the 9th 3D Web. (2004) 41–50
9. Wingrave, C.A., Bowman, D.A., Ramakrishnan, N.: Towards preferences in virtual environment interfaces. In: Proceedings of the 8th EGVE. (2002) 63–72
10. Vanacken, L., Cuppens, E., Clerckx, T., Coninx, K.: Extending a dialog model with contextual knowledge. In: TAMODIA 2007. Volume 4849 of LNCS., Springer (2007) 28–41
11. Vanacken, L., De Boeck, J., Raymaekers, C., Coninx, K.: Designing context-aware multimodal virtual environments. In: IMCI 2008, Chania, Crete, Greece (2008) 129–136
12. Preuveneers, D., Van den Bergh, J., Wagelaar, D., Georges, A., Rigole, P., Clerckx, T., Berbers, Y., Coninx, K., Jonckers, V., Bosschere, K.D.: Towards an Extensible Context Ontology for Ambient Intelligence. In Markopoulos, P., Eggen, B., Aarts, E., Crowley, J.L., eds.: Second European Symposium on Ambient Intelligence. Volume 3295 of LNCS., Eindhoven, The Netherlands, Springer (2004) 148 – 159
13. Octavia, J.R., Raymaekers, C., Coninx, K.: Investigating the possibility of adaptation and personalization in virtual environments. In Houben, G., McCalla, G., Pianesi, F., Zancanaro, M., eds.: UMAP 2009. Volume 5535 of LNCS., Springer (2009) 361–366
14. Clerckx, T.: Model-Based Development of Context-Aware Interactive Applications in Ambient Intelligence Environments. PhD thesis, transnationale Universiteit Limburg (2007)
15. Pribeanu, C., Limbourg, Q., Vanderdonckt, J.: Task modelling for context-sensitive user interfaces. In: DSV-IS 2001. (2001) 49–68
16. 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** (2007)
17. Beer, W., Christian, V., Ferscha, A., Mehrmann, L.: Modeling Context-aware Behavior by Interpreted ECA Rules. In: EUROPAR 2003. Volume 2790 of LNCS., Springer (2003) 1064–1073
18. Etter, R., Costa, P., Broens, T.: A Rule-Based Approach Towards Context-Aware User Notification Services. In: ICPS 2006. (2006) 281–284
19. Poupyrev, I., Weghorst, S., Billunghurst, M., Ichikawa, T.: Egocentric object manipulation in virtual environments; empirical evalutaion of interaction techniques. Computer Graphics Forum **17** (1998) 30–41
20. Vanacken, L., Grossman, T., Coninx, K.: Multimodal selection techniques for dense and occluded 3d virtual environments. International Journal on Human Computer Studies **67** (2009) 237–255
21. Duval, C., Sadikot, A., Panisset, M.: The detection of tremor during slow alternating movements performed by patients with early Parkinsons disease. Experimental Brain Research **154** (2004) 395–398