

A component oriented design for a VR based application

Michael Haller

Polytechnic University of Hagenberg
Media Technology and Design
AUSTRIA

haller@fhs-hagenberg.ac.at

Abstract

There are many toolkits that aid the process of constructing virtual environments. Each of these development systems has a special set of features, they support a set of hardware and they give developers and end users a level of abstraction.

This paper presents a new component based system for the development of VR based applications. With a pool of components everybody should be able to create a virtual environment. The paper gives a brief overview of the system and shows its applicability in a concrete VR based safety training program, called SAVE. Finally, this paper presents two prototyped modeling tools, which are used in the system for modeling new scenarios.

Keywords

Virtual Environment, Modeling Tools, Virtual Reality, Components.

1. Introduction

Virtual environments and virtual reality based applications can be very complex. Even more complex are the present VR systems and VR tools for modeling these environments [4, 10]. Often, they are difficult to expand and the authors of VR environments require programming knowledge for realizing the desired virtual scene [1]. And exactly the contradiction lies here: On the one hand developers should have programming knowledge, on the other hand they should have an exact technical knowledge about the composition of the scene, and finally they should be great designers – in other words, they have to be a genius.

In 1997 we had to implement a virtual reality based training simulation, called SAVE, for the Austrian OMV Refinery of Schwechat. **SAVE (Safety Virtual Environment)** was developed by the Institute for Applied Knowledge Processing (FAW) and by the Department of Computer Graphics and Parallel Computing (GUP) by order of OMV. The first prototypes were quite simple, and showed only a small perspective of the virtual refinery. In 1999 we had to develop other new scenarios for the training application [8]. We had to do a redesign. What we wanted to have was a framework where everybody should be able to model a virtual reality based

application with high complexity. The goal was to build a new model for the development of virtual environments, which would allow the simple configuration of a virtual world assembled from a set of components, which are connected together.

What we wanted to have was a design with a high reusability – were parts should be used in different VR applications, rather than starting from zero. We knew, that effective reuse requires much more than just code and library technology. Our vision was a transition from library-based reuse to kit-based reuse (cf. [5]) and we wanted to move away from the traditional modeling environments where programming knowledge was required.

Our new model shows how modeling rather than programming can be used to realize virtual environments. On the one hand the development of a virtual world is very simplified by the introduced component model. On the second hand a high reusability is supported. This work describes how the complex behavior of the synthetic three-dimensional objects can get combined of simple components. The virtual components which are connected together, can exchange events and messages between the input and output devices.

Furthermore, a concept is introduced, how the modeling tools for the development of the VR application have to look like. A closer description of the concept is given in [6, 7]. Finally, we show the benefits of our component based model by the SAVE application (see figure 1).



Figure 1: The feasibility of our component based model is introduced by SAVE, a safety training system refineries.

2. Component oriented design

In our VR system we wanted to have a component oriented design, which is not comparable with the commercial component models, such as JavaBeans or COM.

Our system is comparable with a set of different small LEGO components, which can be connected together. In our view, a visible component has a geometry and a property. Moreover, it is characterized by a behavior. Components can be very simple, but they can also be composed of other small components, so called *compound objects*. Our goal was to offer different simple and even complex objects provided by a repository.

In our system, all the components are described by prototype nodes based on VRML 97 with their input slots, output slots, and parameters, which allow a closer description of the object. By routing events from the output slots to the input slots of another node, customized functionalities and dependencies can be realized, e.g. if a switch has been switched on, a lamp shines, etc.

Once a component has generated an event, the event is propagated from the eventOut slot along any route to other nodes. Event notifications are propagated from sources to listeners by the corresponding method invocations on the target listener objects. The architecture has an event driven design and components only start information processing after receiving an event. After processing, the components can generate new events and the processing can be done in parallel.

Our system is based on the *producer/consumer concept*. No component (consumer) starts sending messages and processing information without having received a message from other nodes (producer). The only components, which generate messages, are called *active components* (i.e. timer, clock). They are triggered by system calls – but they do not start themselves by sending messages.

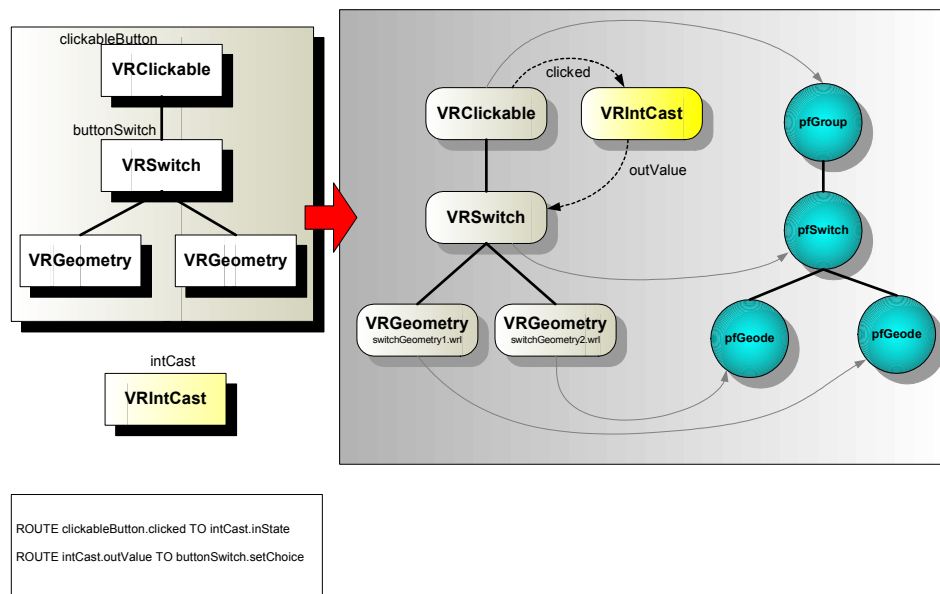


Figure 2: From the components scene graph through the meta scene graph to the Performer scene graph.

For the sake of being independent of a certain graphics library, we encapsulate direct calls to the actual graphics library in certain graphical classes. Instead of creating the scene graph using library specific calls we build a *meta scene graph*. Figure 2 shows the three graph types of a

simple example. In this case, a *clickableButton* consists of two separate geometries: one of the up-state (not pushed) and one for the down-state (pushed). If the user clicks the button, it remains in down-state until the user clicks it again.

The leftmost graph structure presents the VRML97 data structure generated by our parser. Note the thick black lines connecting the nodes. These lines represent parent children relationships. The graph structure in the middle of figure 2 constitutes the meta scene graph. Again, the thick black lines represent the parent children relationships. This structure is derived from the VRML97 scene graph. The right data structure is the scene graph of the graphics library which contains group-, transformation- and geometry-nodes. Note that the component communication network is not part of this scene graph.

Building the meta scene graph does not require any knowledge of the underlying graphics library. The graphical classes can easily be replaced by classes which encapsulate calls to an alternative graphics library and no further changes in the source are required for adding new graphical objects in the virtual environment. The mapping of the VRML source to the actual scene graph is performed in two steps. Firstly, our parser identifies all VRML nodes and creates a corresponding VRML node representation. Secondly, the VRML node is traversed recursively and each node is converted to its corresponding meta scene graph node representation. The result, in fact, is a meta scene graph. Finally, after a second traversal, the component communication network is established.

3. Modeling a scenario

All components used for describing a VR scenario are stored in a repository with different types. Graphical components represent visible objects of the virtual world. Logical components, i.e. AND, OR, etc., allow a connection of objects. For mathematical functions we offer an expression component. The so-called active components are not visible and not triggered by other components but rather they generate event sources.

At the moment we are developing different modeling tools for an intuitive modeling process. So, the developers can select objects and build up their virtual world – without any knowledge how to implement the behavior of the objects. The only thing they have to know is, how to build the world. At the moment there exist prototypes for modeling new virtual environments. The first tool is the **SceneBuilder**, where developers can specify the components of a scenario. Within the help of this tool the developer is able to place the components, rotate and scale them to match the desired layout. With the **RouteEdit**, a dependencies editor, dynamic objects can be connected by plugging their slots together.

The authors use these tools for specifying their *scene description files*. At the moment there exist two different tools, the SceneBuilder and the RouteEdit, which should in the future be affiliated into one modeling tool. Both modeling tools are implemented in Java and Java3D.

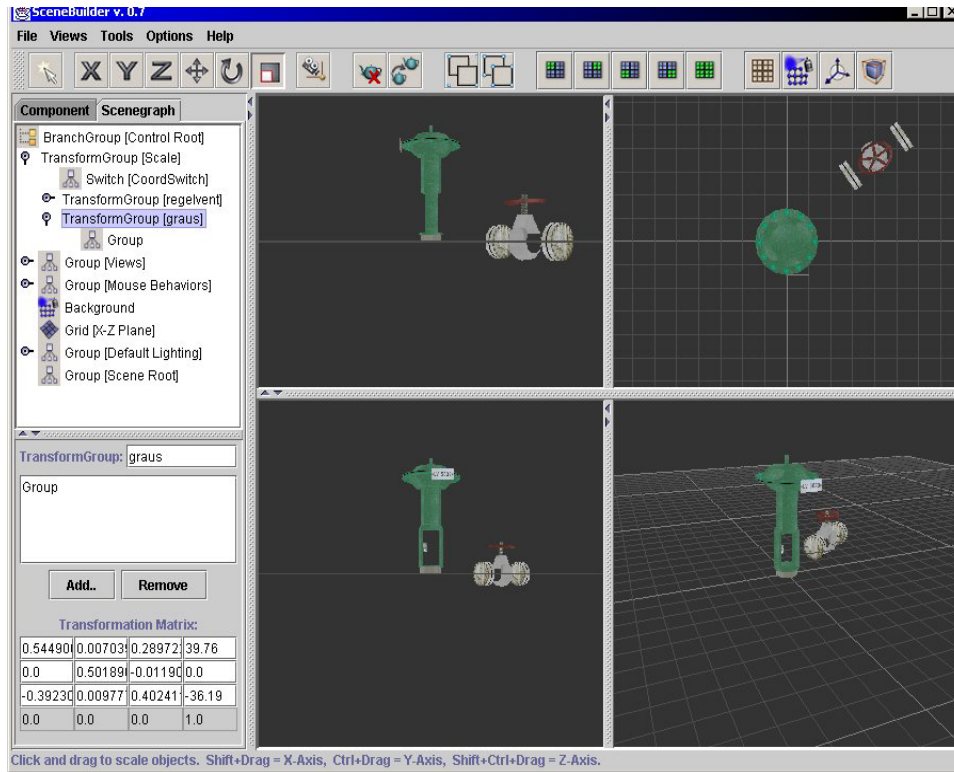


Figure 3: With the SceneBuilder authors can model their virtual world.

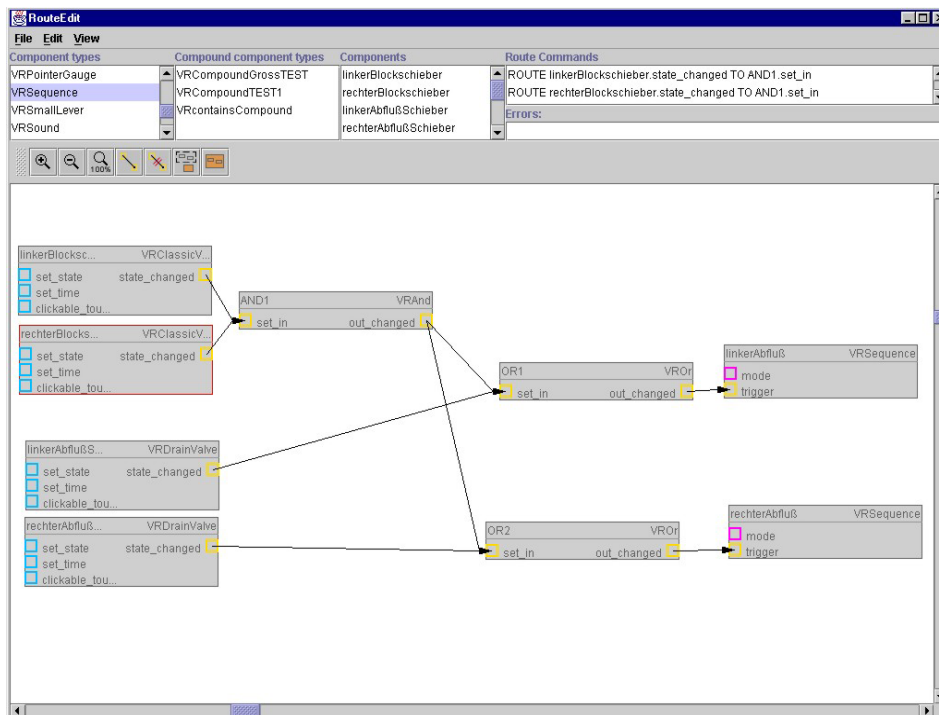


Figure 4: The author uses RouteEdit for the definition of dependencies between the components.

4. The VR simulation

The results of the modeling tools are the so called *scenario description files*, which describe the objects and their properties for the actual VR scenario. Starting from these files, our system generates the VR simulation. In our virtual environment all the components are described by prototype nodes based on VRML 97, which helps the author to manage scene complexity by providing a method for defining higher level objects. In fact, the scenario description files are the interface between the modeling tools (based on Java3D) and the VR simulation (based on Performer). Figure 5 shows a small part of our modeled virtual refinery, where the user can interact with.

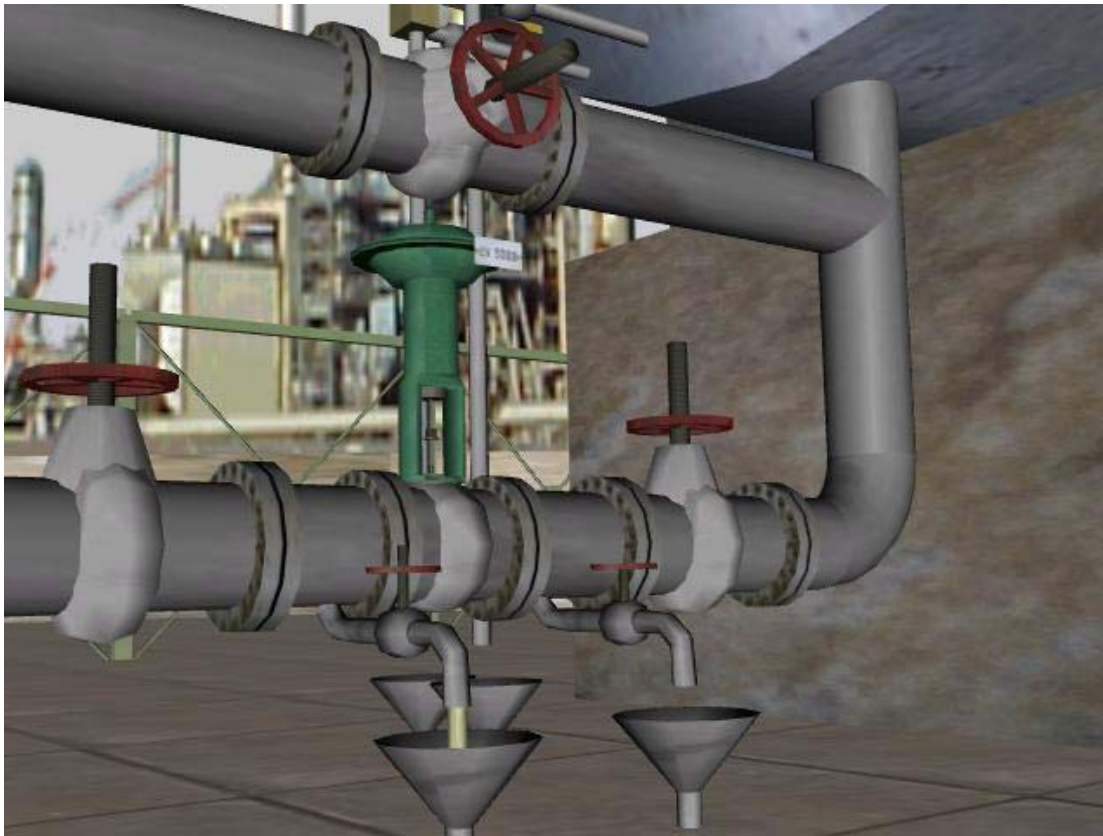


Figure 5: SAVE has many complex objects with even more complex . The dependencies of the components of this part of the refinery are described in figure 4.

The complexity of a virtual scene is not driven by the dependencies of the objects. In fact, the complexity is given by the interaction between the user (trainee) and the objects. We use a 3D-joystick (with a Polhemus tracking system and two buttons) for the navigation and interaction possibilities. The first button is used for the navigation, the second button for the manipulation of objects, e.g. for opening valves etc. The SAVE system can be divided into two main modules:

- **The Trainee-Module:** This application renders the virtual environment of the trainee and it handles all interaction between trainee and the objects of the scene. Since every head movement is tracked, this application renders a new image on every move according to the trainee's viewpoint and orientation.

- **The Trainer-Module:** The trainer uses this program, which runs on a different machine, to observe the whole training process. The trainer can control any part of a scene and reacts to the trainer's actions.

SAVE uses different graphical objects, which can be utilized according to their functionality:

1. **Trainee manipulated objects:** The manipulation and the change of the behavior of different objects is very important for a virtual environment. Instead of abstract commandos the user should navigate and interact with the objects in a common intuitive way. All objects, which can be manipulated in SAVE are called *clickable objects* and are defined as components as follows:

```
PROTO VRClickable [
    eventOut SFBool clicked
    field MFNode children[]
] {}
```

If the clickable object has been selected by the user, it changes its internal state (cf. the valve will be opened) and it puts the new values on the output slots. Figure 6 shows the opening of a small valve.

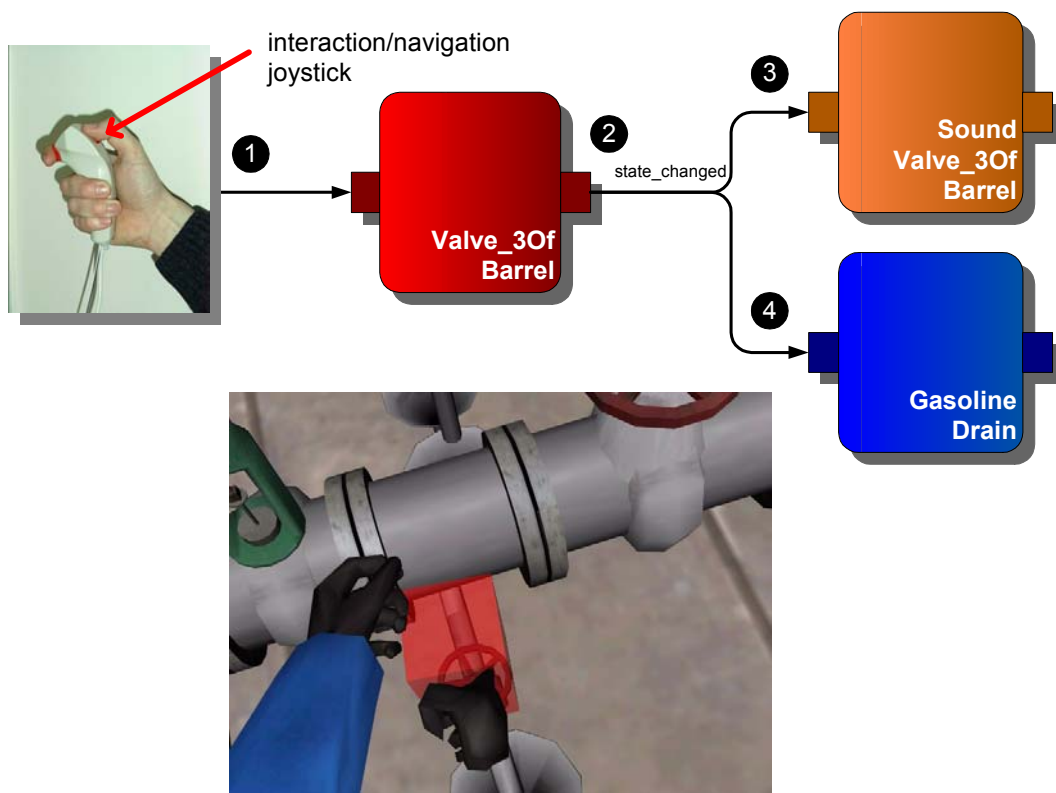


Figure 6: The trainee uses a simple two button joystick for the navigation and interaction. After clicking to the interaction button, the message will be propagated through the communication network.

Unfortunately, we do not have any force feedback datagloves. So, we had to find something similar to give the user a *visual feedback* of what he/she can do or of what he/she has done. This happens by showing a red transparent bounding box, which wraps the graphical object.

2. **Transportable objects:** The starting point of each training session is a lounge, where the trainee has to take on the equipment and the virtual clothes. First, the trainee has to take out of a cupboard different objects, he/she needs for the execution of the training. Independently of the task the user needs the standard equipment, such as a helmet, a pair of goggles, some gloves and a message peeper. Similar to the clickable objects, the transportable objects are wrapped by a red transparent bounding box, in case of the user points to them. Transportable objects can be taken by the user and can be moved either his/her virtual belt or his/her virtual left hand. Of course each transportable object can be put back to the cupboard.



Figure 7: The trainee has to pick up different objects, e.g. a H₂S-peeper.

3. **Trainer manipulated objects:** One of the most important aspects in SAVE is the integration of a trainer application (see figure 8). In fact, a trainer can interact in the virtual environment during a training session. A trainer can manipulate different objects such as build up some pitfalls or he/she can help the trainee in executing some difficult task. For the communication between the trainer and the trainee simulation we used a special component, the so called *VRConnectionNode*, which has no other work to do as to forward the incoming trainer messages (over TCP/IP) into the internal communication network of our system. So, if the trainer manipulates the level of a manometer, he/she has to change the slider on his/her trainer application. After that, this change will be directed by the network to the *VRConnectionNode* which forwards the incoming network message to the accordant objects (in this case to the manometer).

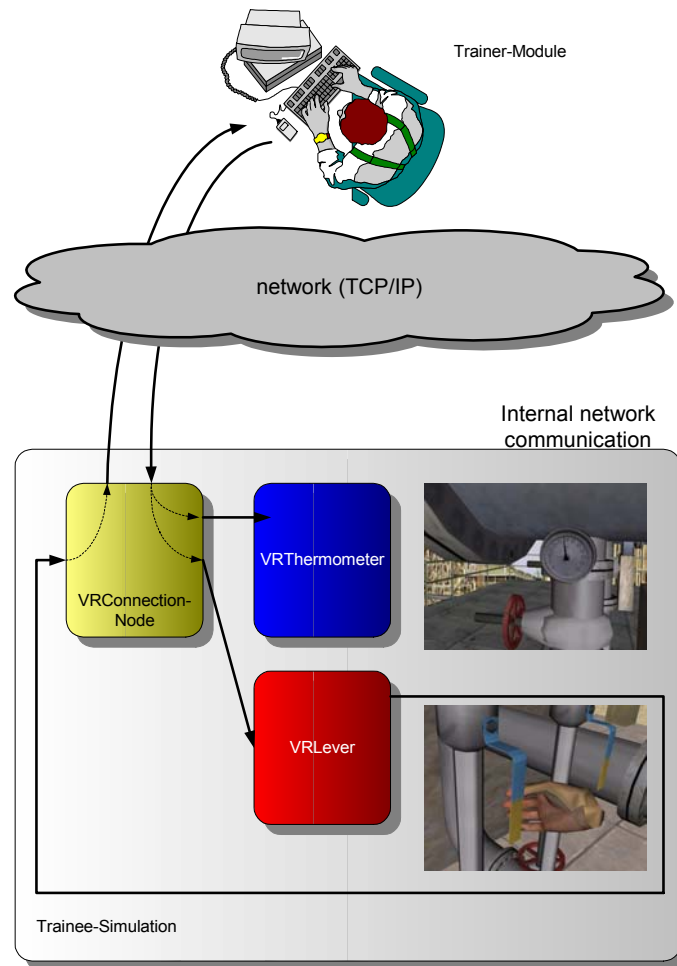


Figure 8: The trainer has the possibility to interact with objects of the virtual refinery.

5. Benefits of the component based system

First tests with the prototyped modeling tools showed that it is very difficult to design really intuitive tools for the modeling process. Nevertheless we have seen, that the modeling process becomes more and more intuitive and easier using the component oriented architecture. Even if the modeling tools are in a prototype state, they offer an intuitive user interface for a very easy modeling process. One of the most striking benefits of our system is the LEGO based architecture, where *everybody* without computer graphics skills can build his/her virtual world. As described in this paper, we tried to offer a modeling tool without any scripting languages. At the beginning we were convinced of a tool which does not permit a programming language as described in [1, 2, 3, 9]. But in the meantime we believe that in some cases it would be very useful to have an easy scripting language for defining some complex dependencies or object behaviors.

6. Conclusions and future work

We have proposed an approach to the virtual environment development process based on components. What has been presented in this paper is a framework using components which have

successfully used in the virtual refinery application SAVE. The system itself supports the simulation of dynamic behaviors of the objects and the full interaction possibilities of the user. SAVE uses HMD technologies and demonstrates the successful benefits of our component based architecture – also in real-time VR applications. At the moment we are developing the modeling tools, which should have high usability, a simple and intuitive interaction technique for object manipulation.

Acknowledgments

The author wish to thank all the members of the SAVE team, including Anton Dunzendorfer, Wolfgang Essmayr, Roland Holm, Johann Messner, Uwe Pachler, Markus Priglinger, Gernot Schaufler, and Erwin Stauder. This project was sponsored by OMV. Therefore, I wish to thank Alois Mochar and Franz Grion.

References

- [1] Bierbaum A. and Just Ch., *Software tools for Virtual Reality Development*, SIGGRAPH'98, Course 14, 1998
- [2] Blach R., Landauer J., Rösch A., and Simon A., *A Flexible Prototyping Tool for 3D Real-Time User-Interaction*. User-Interaction, Proc. of Virtual Environments VE98, Wien, 1998
- [3] Geiger C. Reimann C., Rosenbach W., *Design of Reusable Components for Interactive 3D Environments*. In Usability Centred Design and Evaluation of Virtual 3D Environments, Germany, April, 2000.
- [4] M. Gösele, W. Stürzlinger, Semantic Constraints for Scene Manipulation, in Proceedings Spring Conference in Computer Graphics'99 (Budmerice, Slovak Republic), pp. 140-146, ISBN 802231357-2, Apr. 1999.
- [5] Griss M. and Wentzel K., *Hybrid Domain-Specific Kits*, J. Systems Software, 30:213-230, 1995
- [6] Haller M., *Component based design for virtual environments*, PhD thesis (in german), University of Linz (Austria), December, 2000.
- [7] Haller M., Holm R., Priglinger M., Volkert J., Wagner R., *Components for a virtual environment*, In Usability Centred Design and Evaluation of Virtual 3D Environments, Germany, April, 2000.
- [8] Haller M., Kurka G., Volkert J., and Wagner R., *omVR – A Safety Training System for a Virtual Refinery*. In ISMCR'99, Topical Workshop on Virtual Reality and Advanced Human-Robot Systems, June 1999, Tokyo - Japan.

- [9] Monzy, The Alice Interactive 3D Graphics Programming System, Documentation, <http://www.alice.org>, University of Virginia, 2000.
- [10] Smith S. and Duke D., *Binding Virtual Environments to Toolkit Capabilities*. In Proc. of Eurographics'2000, Interaken, Switzerland, August 2000. 21st Annual Conf. of the European Association for Computer Graphics.