Components for virtual environments

Michael Haller, Roland Holm, Markus Priglinger, Jens Volkert, and Roland Wagner
Johannes Kepler University of Linz
Altenbergerstr 69
A-4040 Linz (AUSTRIA)
$[mhaller|rwagner]@faw.uni-linz.ac.at$
$[jv]@gup.uni-linz.ac.at$

Abstract

In this paper we describe how it is difficult to design virtual environments. We present an approach of a model for the design of a virtual world using a construction kit metaphor, and finally we show a virtual training environment (SAVE), which uses the presented methods.

1 Introduction

The design of virtual environment and virtual reality applications is a challenging task. While 2D WIMP interfaces often build on previous experiences with other applications through the use of common interaction elements (widgets) and the building of an appropriate mental model of the application through repeated use, these information sources are often unavailable for 3D interface designers due to the lack of standards. There is little knowledge about *how* virtual environments are designed, what issues need to be addressed, and little guidance about how design should be carried out [3]. Previous work has mainly focused on technical issues, such as improving runtime performance. But looking for better performance, which is of course a must for virtual environments, many developers do not look at the good design of the application in respect to the reusability. Essentially, the goal of this project is to develop a generic set of tools and a support strategy that will provide users with the means to create virtual environments - in a cost-effective as possible and deliver to low-end machines. The aim of this project is to develop methods and components that can be used to improve the design of interfaces for a virtual environment. In fact, our system is comparable with the principle of a *LEGO* kit box: the users can pick different simple components out of a repository and build their own complex environment. With our toolkit-components we offer a generic description of the definition for a virtual world. Our description of the virtual environment is very formal (it is based on VRML 97) and complete (even dependencies of objects can be defined). Obviously, the very nature of virtual environments contributes to the difficulty of describing and modeling interactions. Typically, they are a collection of static and dynamic components and they have a non-linear process and control flow. One of the most important questions and problems of this work is how the discrete event-based elements and the visual components of interaction can be expressed.

2 Design constraints and problems of virtual environments

One of the supreme aims of virtual reality and particularly in the virtual environment is the near reality cover. More realism can be reached with more interaction possibilities and more detailed environment. However, a number of constraints prevent the solution from more realism:

- machine constraints, e.g. render power limit the complexity of the 3D environment
- technological constraints

- human constraints for the creation of usable and more intuitive interfaces.

A general problem with the construction of virtual environments consists in the complexity of modeling *and* implementation. Therefore, different design methods and tools are necessary.

Of course, an expansion of the system should be possible. Monolithic systems are difficult to maintain and the opinion that hard coded and machine adapted C based applications are faster, is unsteady. As in the software development with components, it also must be possible with VEs to add new functions by installing new elements.

3 Design of virtual environments

As in the software development with components, it also must be possible with virtual environments to add new functions by installing new elements. In this paper a new system, the **C**omponent **O**riented **V**irtual **E**nvironment **S**ystem, will be presented. The central element in **COVES** is a mechanism, which is responsible for the component communication network. In fact, complex virtual environments are presented by simple, small units (components or nodes). Several advantages are connected to such a division:

- *Flexibility*: All elements can be combined in arbitrary order.
- *Reusability*: A component can be used in various configurations.
- *Extensibility*: If the system has a programming interface, arbitrary extensions can be made. If the components for a task are not implemented and installed, they have to be integrated as needed.
- *Communication*: If suitable interfaces are offered, the components can communicate with each other and use the common resources.

4 Approach

Our goal was to provide a very flexible system. Therefore, we used an object-oriented design to make the system more flexible and more readily understandable and extensible. The object-oriented concepts such as inheritance, polymorphism and dynamic binding provide a more compact and clear structure of the specification of complex systems, e.g. virtual environments. Moreover, the architecture has an event-driven design because the execution of logic occurs as result of an event being triggered. Thus, execution flow is as dependent upon the event model as data flow. The system resulting from this approach is more robust, more efficient, and easier to understand. The interesting facet of our system is the use of reusable modular components that can be linked at compile time to form a tailored system.

The core of the system is the repository of the components, where the structure of the different nodes is defined. In our virtual world all the components are described with prototype nodes based on VRML 97 specification, which helps the user to manage scene complexity by providing a method for defining higher level objects. We extended the VRML standard with our own prototypes, which describe the different dynamic objects of the virtual world, cf. lever, switch, lamp, spindle etc. All components or nodes have input slots (defined as `eventIn`), output slots (defined as `eventOut`), and parameters (`fields`), which allow a closer description of the object (see figure 1). The prototype nodes pretend an exact description of what has to be defined. Default values of the prototype are taken, if they are not be described and redefined. It corresponds to a repository or to a container of components, which define the different virtual objects.

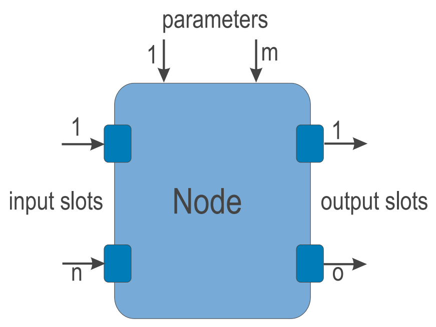

Figure 1: A node consists of different input slots and output slots

In the repository exist different types of components:

Graphical components represent a graphical object in the virtual world.

Functional components , cf. `AND, OR, NOT, PLUS,` $f(x)$, etc., which are used to connect the graphical components and to extend their functionality.

The next example shows the PROTO of the graphical object `VRLever` used in our virtual environment.

```
PROTO VRLever [
   eventIn SFBool set_value
   eventOut SFBool value_changed
   field SFNode case NULL
   field SFNode lever NULL
   field SFFloat leverDistance 0.1
]
{
}
```

All components represent an individual entity of the appropriate type. Obviously, no two base nodes represent the same entity and for all the components of the repository we provide a corresponding implemented class in our system. Proceeding these components, the user describes the virtual world in a *scene description file* using the prototypes of the repository. Finally, users can define dependencies and connect the components together. In fact, our components can be connected to each other by labelled, directed arcs. Again, based on VRML 97 we used the ROUTE functionality.

At the moment we are designing two different modeling tools, where the users should have a graphical programming interface for an environment which can be modeled more easily. Once defined the scene description file, our system generates the the virtual world - it generates the Performer scene-graph and maps the routing commandos in an internal network.

Most node types have at least one `eventIn` definition and thus can receive events. Incoming events are data messages sent by other nodes to change their state within the receiving node. Some nodes also have `eventOut` definitions. These are used to send data messages to destination nodes that some state has changed within the source node. Once a component has generated an initial event, the event is propagated from the eventOut producing the event along any ROUTE to other nodes. These other nodes may respond by generating additional events, continuing until all routes have been honoured. Event notifications are propagated from sources to listeners by the corresponding method invocations on the target listener objects. Each event source can have multiple listeners registered on it. Conversely, a single listener can register with multiple event sources. The node concept allows to add behaviors to the scene. All nodes contain programmatic logic that translates and propagates input events into output events. By routing events from the output slots of a node to the input slots of another node, customized functionality and dependencies can be realized, *e*.g. if a switch has been switched on, a lamp lights, etc.

With the listener concept our system becomes very flexible. With the different nodes and their corresponding slots we can define any imaginable virtual environment and define different scenarios, where dependencies of the dynamic objects can be defined. The node concept is so flexible, that it has also been adapted for the input devices, such as tracking devices and button input devices. A class with the physical description (driver) and a logical class for the internal node presentation is integrated in the node network.

5 SAVE (= SAfety Virtual Environment)

SAVE (SAfety Virtual Environment) is a Virtual Reality based safety training system for dangerous and hazardous facilities. The Institute for Applied Knowledge Processing (FAW) and the Department for Graphics and Parallel Processing (GUP) started in 1997 with the first prototype for a virtual training environment for an Austrian refinery [2, 1]. It is a multi purpose virtual reality software system that is mainly intended for employee training. SAVE was designed to use HMD technologies and demonstrated the possibilities of VR for safety training. SAVE is based on the technologies described above and supports

- real-time collision detection,

- simulation of dynamic behaviours of the objects,
- dynamic interactions between the user and the objects,
- interaction of the trainer, who can interact in the virtual scene and manipulate the objects, e.g. change the state of the valve.

The SAVE system offers a solution for these problems. It provides a framework and software system for a variety of training scenarios using VR technology. Each virtual training scenario comprises a scene in which the trainee can move freely and interact with objects like pumps, valves, and other control devices. *omVR* is the first application of the SAVE system, which provides an advanced technique for personnel safety training in refineries. It is based on the architecture described above and uses the components for the different virtual elements.

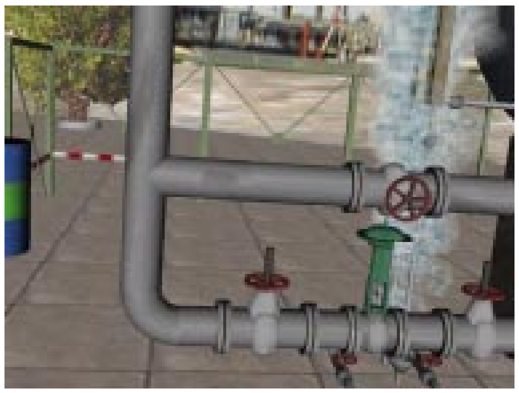

Figure 2: SAVE (SAfety Virtual Environment) is an VR application for safety training in a virtual refinery

The *omVR* training system consists of two major parts:

The scene simulator : This application runs on a graphics workstation and creates the user's view of the whole training scene. It also handles all interaction between the trainee and the objects in the scene, like switches, levers, tools, etc. Since every head movement is tracked using a tracking system, the application renders a new image corresponding to the trainee's viewpoint in the scene on every move, creating a strong immersive effect - after several minutes, the trainee believes to be inside the scene.

The trainer application : This program is used by the trainer to observe the whole training process and runs on a different machine, linked to the graphics workstation using a network connection. The trainer can control any part of the scene and react on the trainee's actions. Moreover, it allows to manage the trainee's training progress using a database, which allows individual training.

One of the most important aspects of our application is that the virtual environment of the trainee can be controlled by the education/training program. In fact, this relationship between the education/training program and the simulation/trainee program is the central idea of this educational system.

6 Conclusions

As Smith, Duke, and Massink [4] note, virtual environments are a mix of continuous and discrete components. What has been presented in this paper is an initial research for a framework with components for a virtual environment. The results have been realized in a refinery virtual environment, which used the different components. The next steps of the project is to define tools, which provide a very user friendly interface for assembling the components to a virtual environment. There are planned tools, where the user can choose of a huge pot of graphical and logical components which can be connected together. Doing so, the user creates his own virtual environment, which can be used for a new safety training.

References

[1] M. Haller, R. Holm, J. Volkert, and R. Wagner. A VR based safety training system in a petroleum refinery. In *Proc. of Eurographics'99*, pages 5–7, Milano, September 1999. 20th Annual Conf. of the European Association for Computer Graphics.

[2] M. Haller, G. Kurka, J. Volkert, and R. Wagner. omVR - A Safety Training System for a Virtual Refinery. In *Proc. of ISMCR'99*, number Vol. X, pages 291–298, Japan, June 1999. Topical Workshop on Virtual Reality and Advanced Human-Robot Systems.

[3] Kulwinder Kaur. *Designing Virtual Environments for Usability.* PhD thesis, Centre for Human-Computer Interface Design, 1998. 241 pages.

[4] Shamus Smith, David Duke, and Mieke Massink. The hybrid world of virtual environments. In *Proc. of Eurographics'99*, pages 297–307, Milano, September 1999. 20th Annual Conf. of the European Association for Computer Graphics.