A VR Component Architecture for Visual Multi-User Applications

Martin Leissler, Andreas Müller, Matthias Hemmje, Erich Neuhold

GMD – German National Research Center for Information Technology
IPSI – Integrated Publication and Information Systems Institute
Dolivostr. 15, 64293 Darmstadt, Germany
[leissler, anmuelle, hemmje, neuhold]@darmstadt.gmd.de

Abstract

The vision of cyberspace coined in books such as William Gibson's "Neuromancer" involves the basic principles of extensibility, openness, and shared 3D-environments. This paper describes a design for an architectural model allowing the construction of multi-user VR applications from flexible modules with the goal to fulfill these demands. First, so-called VR components are designed. Second, a framework is defined in which the VR components can be embedded and into which a multi-user infrastructure is incorporated. Furthermore, an implementation framework for the VR-components and the multi-user infrastructure based on VRML and Java is presented, which we call Vrmlets. An analysis of existing VRML-centric approaches shows that this API based approach is superior with regards to flexibility and scalability. Since the Vrmlets can be implemented on top of any existing software component model (such as COM/DCOM) they are ideal for the use in integrated visual development environments. Therefore, Vrmlets enable a consistent paradigm for the development of visualization applications with complex behavior.

Keywords: Information Visualization, Software Components, Multi-User, Shared Environments

1. Introduction and Motivation

The vision of **cyberspace** was born at a time, when hardly anybody could imagine that someday computers, which are fast enough for interactive 3D graphics would be sold in supermarkets. Cyberspace - that is more than an interactive 3D user interface such as provided by some computer games. The principle of **extensibility** is of decisive importance. This characteristic has two aspects: First, new elements should be easy to integrate into an existing world, e.g., via drag and drop, or they should be replaceable with other elements. On the other hand, the construction of new elements should be as easy as possible, because this is the only way to enable developers all over the world to create a rich and diverse virtual world.

A further principle is **openness**. In this case, "openness" means that such a system has clearly specified interfaces so that every developer can create own elements for the cyberspace. In this way, multi-user worlds of different manufacturers can be compatible with each other because the different implementations have the same interfaces.

With **VRML** (Virtual Reality Modeling Language, [11]), we have got a rich description means for interactive 3D graphics that was designed specifically for the use in the Internet. However, VRML provides no description means for multi-user applications. Existing multi-user 3D applications, such as computer games, are neither open nor extensible in the above sense.

2. Goal

The aim of this paper is it to design an architecture allowing the construction of VR multi-user applications from flexible modules. It continues and extends the work presented in [6] and [5]. The task can be divided into two parts: First, to design the modules to be referred to as **VR components** in the following. Second, a framework has to be defined in which the VR components can be embedded and into which the **multi-user infrastructure** is incorporated.

VRML, however, is not designed for a distributed application: The scene description in VRML is a local data structure, i.e. if the same scene is loaded to several clients, the modifications to this scene made by one client are not visible for the other users. Even the VRML messaging, i.e. the routing of events, is a purely local mechanism. Therefore, extensions are needed to realize a multi-user functionality. Possible approaches are presented in section 4.2.

The reader is assumed to have a basic understanding of the concepts of software component architectures, VRML, and Java.

2.1 VR components

The term "VR components" is to illustrate that these components do not only have a visual representation but that they are also interactive, i.e. the user can change their state. In addition, these components should be able to contain a complex behavior. The term "virtual reality" (VR) combines these characteristics in our opinion: First VR has a visible (3D) shape. Second, the user may act in the VR - therefore it is interactive. And, finally, there is also complex behavior "in the reality". These VR components are specified to the outside by a uniform interface and encapsulate both the state of the component and its visual representation within the VRML scene. The visual representation of the components (their appearance and behavior in the VRML browser) should be simply configurable and should be exchangeable at runtime if possible.

The aim is to enable a simple construction of visual multi-user applications from prefabricated components. It should be equally simple to create new components.

2.2 Multi-user infrastructure

The VR components are embedded in a framework, the so-called "multi-user technology" (MUTech), which provides the services necessary for the multi-user applications. These are in particular services for communication via messages (messaging), for object management and the replication of the state of the distributed application. The multi-user infrastructure should be well scalable in order to show good performance even with a great number of users and VRML components in the scene.

3. Requirement Analysis

The following chapter is to show which typical elements may appear in a visual multi-user application:

- An **environment module**: This is a relation {object→visualization}. Users can select this assignment themselves to adapt the visualization to their taste. An environment module can contain any number of object types and therefore is a generalization of the visualization mechanism from existing virtual multi-user worlds, which allow the users to select only a model for their avatar.
- **Component architecture**: Objects in this scenario are software components with a standardized interface. This makes it easy to develop new components and to fit them into an existing world without problems.
- A **virtual guide**: Objects in a virtual world cannot only symbolize users but also messages or services, as a search service which is controlled by a software agent.
- **Persistent objects**: Users can create objects of any type dynamically, which are not deleted upon their logoff. The lifetime and existence of these objects is independent from the user, who created them.
- **Active objects**: Instead of a static hyperlinks, one can also imagine a software agents, which e.g. determines the participants whereabouts, which asks them whether they would like to receive "virtual visits", and which takes the user to them (if they agree).
- **Communication on semantic level**: A *query*, formulated by a user can be broadcasted to the other participants. Other users would see a completely different visualization (with another environment module). Here, not the concrete visualization is relevant, of course, but it is the query which must be transmitted via the network.

3.1 Identification of Deficits

Many of the above approaches cannot be realized or are hard to realize by means of the existing approaches, which is shown by the following list. Existing approaches are presented and discussed in section 4.2.

- **Rigid definition of visualization**: The facility of visualization adaptation is often very restricted. Mostly, the users may select a model for their avatar, other facilities are not provided.
- **Poor extensibility**: The objects in the world are often preset. Therefore, no new object types can be inserted at runtime. The extension of a world at design time is also difficult, since no tools can be used, such as broadly available GUI builders
- **Restricted behavior of objects**: Objects in existing multi-user worlds are often relatively static, their behavior only consists of defined gestures and movements in the 3D space, controlled through a human user. Much more flexibility would be desirable here: Market rates or stocks may change every minute, hence follows that their visualization must be generated and updated dynamically from a database. Therefore, the corresponding object in the virtual world contains no fixed data but e.g. a database query and it is not controlled by a human being but by a software agent.
- **Interaction only on visualization level:** Many approaches of multi-user worlds aim exclusively at the visualization level and provide efficient messaging services for the distribution and replication of visualization information. Information on semantic level (such as the above query) first would have to be converted to the defined data types before it could be distributed and replicated.

3.2 Derived Requirements

Next, we derive requirements on an architecture from the above deficits.

1. Easy extensibility: It should be as easy as possible to add new object types to an existing multi-user application. Ideally, this should also be possible at runtime. The multi-user infrastructure should be extensible as well, e.g. by new messaging services, authentication schemes, etc.

2. Modifiable visualization mapping: The visualization of an object should not be firmly defined, but should be modifiable at runtime. A search service, for example, can be visualized through a servant or a filing box.

3. Configurable visualization: The appearance of a visualization should be parametrical to enable a versatile application. For example, the user should be able to define the color of the clothing and the hair of an avatar via parameters.

4. Complex object behavior: Objects should be capable of complex behavior such as required for an agent establishing a contact with the user.

5. Uniform view of state information: The multi-user infrastructure should be able to distribute and replicate information both on visualization level and on the level of application semantics.

4. Concepts of Visual MU Applications

4.1 Concepts

4.1.1 The State of Distributed Applications

By multi-user applications we understand applications in which several users can participate simultaneously. These applications fall into the category of **distributed systems** which are discussed in [9] and [3]. A **multi-user application** involves the need to present a **consistent state** to every user. State management can be centralized in a global database or pursue a distributed approach (see also [9]).

1. Global state in a database: The simplest solution of the above problem is to keep the state of the multi-user application globally in a central database. The state data exists precisely once and the database management system (DBMS) provides mechanisms for concurrency control. However, for visual applications, this approach has the serious disadvantage that the latency for visual operations executed by a user on a client is too long. Shifting objects or navigating in the 3D space requires a smooth movement.

2. Replicated state: To tackle the above problem, state information has to be **replicated,** i.e. it is redundantly available at various places. This enables short latencies because only local data structures have to be accessed, though it requires mechanisms to keep the replicated state information consistent. This is referred to as **synchronization.**

4.1.2 Structure of a Visual Application

In a **visual application,** two layers can be identified as outline structure: a semantic layer and a visualization layer. The first layer describes what is to be visualized, e.g., the relation within a database. The visualization layer describes what is visible on the screen. One knows nothing about the semantics of the application on this level. Typical objects of this layer are buttons, lists and labels in the 2D GUI area and scene graphs in the 3D area. Both layers contain objects that can carry state information.

The **visualization mapping** assigns corresponding objects of the visualization layer to the objects of the semantic layer. Visualization mapping: [semantic object ➔ visualization object]. It defines how to represent abstract objects visually. Users, for example, can be represented through their name in a text field.

4.1.3 Structure of a Visual Multi-user Application

A **visual multi-user application** combines both characteristics: Every client has two layers, the semantic layer and the visualization layer. Replicated state information can be contained in each of these layers.

There is a component called **MUTech** (multi-user technology) to manage and distribute state information on both layers: If an operation on one of the two layers is executed on a client, the local replicate of the state has to be changed and this modification has to be propagated to the remaining clients.

4.2 Existing Models

The following is to present and evaluate two models for VRML-based multi-user worlds. Both models have in common that they are "VRML-centric" which makes them easily comparable.

4.2.1 The VSPLUS Model

The VSPLUS model is described in [1] as a "high-level multi-user extension library for interactive VRML worlds". The aim of this approach is to achieve a simple design of interactive VRML worlds (if possible without programming effort). Existing interactive single-user content should be easily converted to multi-user operation.

This model offers a simple and elegant extension of VRML event handling and allows a simple enhancement of existing VRML worlds with multi-user functionality. However, only VRML data types can be handled as a state to be routed. In addition, there are no filtering mechanisms that suppress messages completely; This approach is not suitable for complex applications; it is rather intended for simple applications whose functionality is included completely in the VRML scene and which are not designed for a very large number of users.

4.2.2 The Living Worlds Model

The Living Worlds originates from the VRML working group with the same name [7] and is an interface specification between any multi-user infrastructure (MUTech) and VRML.

The Living Worlds model is considerably more complex than the VSPLUS model. Therefore, it is much more difficult to adapt existing contents to multi-user operation. However, the Living Worlds model is also far more flexible and more extensive: It contains a spatial partitioning mechanism, provides locking of objects and even facilities for authenticating messages and access control on the basis of user authorizations. The Living Worlds model is deliberately constructed to leave many possibilities of performance optimization open.

An aim of Living Worlds, namely to define flexible, versatile constructs for building multi-user worlds in VRML, is achieved. However, Living Worlds is extremely complex which complicates both the building of MUTechs and the realization of applications. The complexity of Living Worlds clearly shows that API-oriented approaches provide more flexibility because they enable a uniform view of state data (see section 2). For this reason, the model developed in this paper pursues an API-oriented approach. It is to be presented in the following section.

5. The VR Component Model

This section describes the abstract model developed in the present paper. It is divided into two parts: The first describes the so-called VR components; the second part describes their embedding into a multi-user infrastructure.

5.1 VR Components

Let a VR application consist of VR components, which encapsulate functions both from the semantic layer and the visualization layer. They are accessed via an API and therefore, the VR components can be used exactly in the same way on the application side as conventional classes. To be usable for many purposes, these components must be very flexibly configurable which applies in particular to their visualization.

If these VR components are implemented as software components (e.g. JavaBeans or COM objects), they can be used in graphical development tools such as Microsoft’s "Visual Studio" or Symantec’s "VisualCafé".

5.2 Multi-user Infrastructure

The VR components should be embeddable into a multi-user infrastructure (MUTech). The resulting requirements are as follows:

- The VR components must be able to make their state **persistent** in order to export it to a MUTech component.
- If the application changes the state of the VR component by method calls, the component must generate and dispatch an **update message.** Equally it must be able itself to receive, decode and process such update messages.
- The components use the **messaging services** provided by MUTech for these update messages.

In this case, the **multi-user infrastructure** (MUTech) has the task to handle the state of the multi-user application. This includes the following:

- MUTech contains a directory of all objects and clients. All entities, objects and clients are distinguished by unique IDs.
- In order to create and delete objects dynamically, MUTech provides a defined interface. In addition, MUTech is responsible for informing clients which are added later (so-called latecomers) about the current state. For this purpose, MUTech must have stored replicates of all objects.
- MUTech makes the messaging services available to communicate state information to the connected clients. These messaging services should be filtered so that every client receives only the update messages relevant to it.
- In addition, MUTech is responsible for the authentication of users and the management of user authorizations.
- MUTech should be well scalable and show good performance even under high load.

5.3 The Internal Structure of VR Components

To avoid any inflexible specification of visualization the internal structure of VR components is divided into the two parts **controller** (C) and **visualization** (V) as shown in Figure 1:

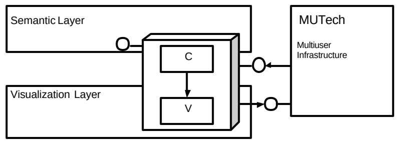

Figure 1: Structure of VR components

There is a defined interface between these parts so that a controller can be coupled to different visualizations upon runtime. The application accesses only the methods of the controller, the latter controls visualization.

5.4 Evaluating against the Requirements

This section it is concluded by describing the extent to which the model presented here meets the requirements on the VR components (see section 3.4).

- **Simple extendibility**: This is achieved by the component-oriented approach: The application sees and uses the VR component as a usual software component, which enables the dynamic addition of new components.
- **Modifiable visualization mapping**: This requirement is implemented by the flexible assignment mechanism of controller to visualization (see section 5.3). A modification is also possible upon runtime.
- **Configurable visualization**: The visualization parts of the VR components are executed as VRML protos, these can be configured in the proto interface with fields.
- **Complex object behavior**: Complex behavior can be contained in the controller part of the VR component.
- **Uniform view of state information**: This is given since state information is managed only in the controller part of the VR component. Therefore, the multi-user infrastructure uses exclusively this controller part.

6. The Vrmlet Architecture

This chapter is prefixed by a short introduction of the concept of spatial partitioning within the proposed Vrmlet architecture and then describes a possible implementation of the architectural model presented in section 5. It is an exemplary realization of the VR component model allowing us to perform experiments on the overall behavior and performance of such a system.

6.1 Spatial Partitioning Concept

In the Vrmlet architecture, a world can be divided into several partitions. This is due to the scalability requirement. Two aims are to be identified:

1. Filtering of update messages: To avoid a client being swamped with too many messages, a filtering of these messages should be possible by using the filter criterion of spatial partitioning.

2. Server scalability: To enable the management of a great number of clients and objects, state management can be distributed over several servers. As a result, the individual server has to manage a smaller number of objects and has only to distribute the messages for these objects to its clients. These spatial partitions are configured by the developer of the world and should be as flexible as possible.

In accordance with these two aims, two constructs are introduced: The concept of zones is used for filtering messages and the concept of districts for securing the scalability of servers.

6.1.1 Zones

1. Zones are invisible spatial partitions, which may overlap each other completely, or partially, i.e. other objects, e.g. avatars, can be located within a zone.
2. An object can be located in several zones simultaneously.
3. Zones can be active or inactive: Activation is triggered by the users when they approach these zones while navigating in the scene.
4. Zones are used for the **filtering of updates**: All replicated objects belong to some zone and receive updates of their state only if their zone is active.

Zones are spatial constructs and due to their static character they should preferably be declared in the VRML scene. "Static character" means that location, size and activation parameter of the zones do not change upon runtime. In addition, the world designer should be given leeway for optimizing the scene, for example, through specific zone activation scripts.

6.1.2 Districts

Like zones, districts are also invisible spatial partitions though **not overlapping** each other like zones. They are spatially disjoint, every zone is located within a district and every object belongs to exactly one district. Districts are used for the **causal and total order of messages**: All messages from objects in a district are sorted causally and totally by sending them via a shared sequencer.

Every zone belongs to exactly one district by definition. Therefore, the list of its zones can be stored for every district. Since every district can be located on a separate server, additional information such as the host name of this server as well as its port numbers have to be recorded either.

6.2 Implementation Architecture

Figure 2 shows the elements of the vrmlet architecture and their interrelations. Every named entity is presented separately.

6.2.1 Vrmlet

The vrmlets constitute the realization of the VR components from section 5. They are persistent and have a globally unique name (OID). This OID is used for addressing messages to objects. They also have a class name (their ClassID), which is used to identify the program code of these components and to download it if necessary. The ClassID can be a fully qualified Java class name or a COM CLASSID.

Vrmlets have a "locus of control" (LOC). This is the unique name of the client (i.e. a ClientID) on which the original of the object is located. If this client exits from the multi-user session, the vrmlet is removed everywhere. Moreover they can send and receive messages and have a visual representation, which is controlled by the mechanism described in Section 5.3.

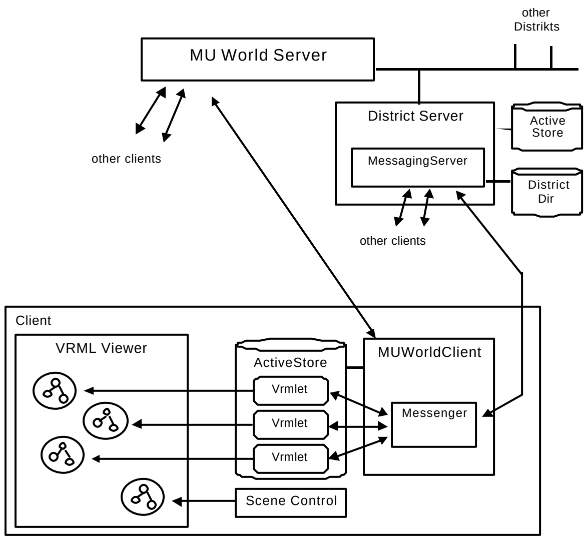

Figure 7: The Vrmlet architecture

6.2.2 Active store

The ActiveStore is a storage for distributed objects (**SharedObject - Vrmlet**) and has methods for integrating, deleting and accessing objects. The ActiveStore is initialized with a messenger which is an interface for sending messages. If an object is integrated into the ActiveStore, it is provided with a reference to this messenger which it uses for sending messages.

The task of the distributed MUWorld component is to synchronize the existing ActiveStores and to keep them consistent, i.e. that the same ActiveStores contain the same objects everywhere and that all objects have the same state. Further, if a new district becomes active, its ActiveStore must be downloaded completely.

6.2.3 SceneControl

The scene graph of the VRML viewer is managed here via External Authoring Interface (EAI) (see [4] [10]). The visualization of the vrmlets is integrated into the scene graph or is removed from it. The SceneControl observes when which regions become active or inactive using callback functions, triggered by position value modification. In addition to the zones, the active district is also managed here.

6.2.4 Messenger and MessagingServer

The messaging services, which are necessary for a multi-user application, are implemented here. The MessagingServer forwards all messages to its clients, located in the same district. It uses the DistrictDir, to filter the messages according to zones, which reduces the amount of messages to be processed by a client.

For every district, there is a separate MessagingServer in order to guarantee good scalability and performance, because some messages need only to be forwarded to a subset of the clients.

6.2.5 MUWorldClient

The MUWorldClient is the primary interface to the multi-user infrastructure, which contains a SceneControl and an ActiveStore. The MUWorldClient has methods for managing vrmlets, i.e. adding and removing them from the ActiveStore, which includes integrating the scene graph of the vrmlet into the overall scene or removing it.

The MUWorldClient has a callback interface with the methods "zoneChanged" and "districtChanged" which are used by the SceneControl. In the case of a "districtChanged" event, the ActiveStore of the new district is downloaded completely. If the client changes the zone, the DistrictDir, which manages the active zones of all clients, is updated. The MUWorldClient has methods to login and logoff at a MUWorldServer. Upon login, the user is authenticated through password prompting.

6.2.6 MUWorldServer

The MUWorldServer is the counterpart to the MUWorldClient. It is responsible for authenticating a new client upon login by using the UserDB. Upon login, the server assigns a unique ClientID to the client. The server keeps a list of the connected clients. A DistrictServer is startet for every district.

The MUWorldServer manages the ObjectMasterDirectory, which describes the assignment of objects to districts. Also, adding or removing vrmlets is a service provided by the MUWorldServer. It updates the ObjectMasterDirectoy as well as the DistrictDir and the ActiveStore of the DistrictServer which is responsible for the vrmlet.

6.2.7 DistrictServer

The DistrictServer is implemented either as a separate server or as a data structure within the MUWorldServer. This makes it possible to distribute the DistrictServers over different computers to secure the required scalability.

If, however, all servers run on the same computer as the MUWorldServer does, it is more useful to use no separate processes for the DistrictServer since thread changes require less overhead than process changes.

7. Conclusions and Outlook

In this paper we have presented an architecture for VR components in conjunction with a multi-user infrastructure. This model was derived from a number of deficits within existing models for VRML based shared multi-user environments. Furthermore, an exemplary proof-of-concept implementation has been presented which uses Java components and VRML as a framework for the realization of the VR component model. The component model as well as the implementation of the Vrmlet architecture, however, is not limited to the usage of these components within shared multi-user environments. In fact the concept of VR components (Vrmlets) is rather generic in the sense that these components enable a consistent paradigm for the development of visual software components, which encapsulate complex behavior. Since these components can be exchanged at runtime they are ideal for the use in integrated visual software development environments. Since we used generic interface definitions in our specifications it would be easily possible to use existing software component architectures such as Microsoft COM/DCOM, EJB or CORBA.

The present implementation of Vrmlets lets plenty of space for improvements. In the following we propose some of these points as future work:

- **Embedding into ActiveX:** This means an extension of the Vrmlet-classes to become COM-objects as well as an appropriate extension of the MUWorldClient to become an ActiveX-container [2]. This would immediately enable developers to use existing visual development tools for the design of multi-user environments.
- **Integration of additional messaging services:** The multi-user infrastructure could be extended by additional, UDP based, messaging services. This would allow performing the time critical position updates of movable 3D-objects far more efficient than in the existing implementation.
- **Single user and multi user mode:** It should be easily possible to switch the multi user client between these two modes. Upon errors in the network communication the client should automatically change to single user mode.
- **Anchor nodes:** The VRML97 specification [11] defines so-called anchor nodes to load a new scene when clicking on an object in 3D. Regarding the Vrmlet architecture these nodes would have to be extended to allow the MUWorldClient to connect to nodes in the new scene and eventually to another MUWorldServer.

8. References

[1] Yoshiaki Araki: *VSPLUS: A high-level multi-user extension library for interactive VRML worlds,* Graduate School of Media and Governance, Keio University, Sony Music Entertainmant Inc.

[2] David Chappell: *Understanding ActiveX and OLE,* Microsoft Press, 1996

[3] George Coulouris, Jean Dollimore and Tim Kindberg: *Distributed Systems: Concepts and Design Second Edition,* Addison-Wesley, 1994

[4] External Authoring Interface Working Group http://www.vrml.org/WorkingGroups/vrml-eai/

[5] Leissler, M., Hemmje, M., Neuhold, E.: *Automatic Updates of Interactive Information Visualization UserInterfaces through Database Trigger,* Advances in Visual Information Management: Visual Database Systems published by Kluwer Academic Publishers, 2000

[6] Leissler, M., Hemmje, M., Neuhold, E.: *Supporting Image-Retrieval by Database Driven Interactive 3D Information-Visualization,* Proceedings of the VISUAL'99, Third International Conference on Visual Information Systems - Berlin (u.a.): Springer , 1999.

[7] Living Worlds Working Group: *Living Worlds - Making VRML 97 Applications Interpersonal and Interoperable,* http://www.vrml.org/WorkingGroups/living-worlds/

[8] Cris Marrin, Jim Kent, Dave Immel, Murat Aktihanoglu: *Using VRML Prototypes to Encapsulate Functionality for an External Java Applet* http://www.marrin.com/vrml/papers/InternalExternal/Chris_Marrin_1.html

[9] Erwin Mayer: *Synchronisation in kooperativen Systemen,* Vieweg Verlag, 1994

[10] Bernie Roehl, Justin Couch, Cindy Reed-Ballreich, Tim Rohaly and Geoff Brown: *Late Night VRML 2.0 with Java,* Ziff-Davis Press, 1997

[11] Web3D Consortium: *VRML97 Specification, ISO/IEC 14772 – 1:1997* http://www.vrml.org/technicalinfo/specifications/vrml97/index.htm