Multi-User Layer in the EVE Distributed Virtual Reality Platform

Ch. Bouras
Computer Engineering and Informatics Dept., Univ. of Patras, Greece and Research Academic Computer Technology Institute, Greece
bouras@cti.gr

D. Psaltoulis
Computer Engineering and Informatics Dept., Univ. of Patras, Greece
psaltoul@ceid.upatras.gr

Ch. Psaroudis
Computer Engineering and Informatics Dept., Univ. of Patras, Greece
psaroudi@ceid.upatras.gr

Th. Tsiatsos
Computer Engineering and Informatics Dept., Univ. of Patras, Greece and Research Academic Computer Technology Institute, Greece
tsiatsos@cti.gr

Abstract

In this paper, we present the design and implementation of a VRML97 multi-user layer, which is introduced in the EVE distributed virtual reality platform. Main consideration of our multi-user extension is the ease of transforming single-user virtual worlds to multi-user virtual worlds, as well as the conformity with any standard VRML97 browser. Furthermore, we present the EVE' s communication platform, which can be used in order to support Collaborative Virtual Environments (CVEs). Main issues regarding the effective network communication as well as the initialization of the 3D scene are discussed in this paper.

1. Introduction

Nowadays the use of CVEs is one of the most promising uses of virtual reality. Overthe past decades the traditional user interfaces over the World Wide Web evolved into Virtual Environments (VEs) [6], in order to provide the users with a sense of realism. Furthermore, the technical improvements, in both 3D graphics cards and high-speed networks make this evolution easier. However, the VEs are single-user virtual worlds and the user can only interact with the environment. The need for new services, such as collaborative work and distance learning, leads to the development of CVEs. The central concept of CVEs is a multi-user virtual world, i.e., a computer generated space where participants can meet and interact. Users are embodied in the world by an avatar, which provides them with a 3D representation that follows the movement of their viewpoint. The essence of CVEs is that the shared space defines a consistent and common spatial frame of reference. From the technical viewpoint, Virtual Reality Modeling Language (VRML97) [8] is the standard technology for the creation of virtual environments and their distribution over the web [4]. This technology has been used by a wide range of VEs and is supported by various editors and 3D authoring tools [3]. However, VRML does not provide support for multi-user virtual worlds. Thus, there is a definite need for a multi-user extension behind the VRML. This extension should satisfy at least the following requirements: (a) conformity with a standard VRML97 browser, and (b) easy transformation of a single user world to a multi-user 3D world. The first attempt for standardization of a multi-user extension has been made by the Web3D's Living Worlds (LW) working group [8]. It is designed to work with any standard VRML97 browser. However, the sharing description is blended with the geometry description. The creator of the worlds should insert new nodes into the existing VRML97 scene graph and modify the desirable shared routes in order to transform a single-user VRML world to a multi-user VRML world. This results in a very complex creation of multi-user worlds. It also requires programming skills to carefully update the states of a shared object because high network traffic is generated if events are often sent to change them [1]. Two other approaches have also been proposed: VSPLUS and SPIN-3D. The VSPLUS [1] proposal offers easier transformation of a single user world to a multi-user 3D world than the LW. However, VSPLUS is a simplification of the LW project. As a result VSPLUS does not support the whole functionality that is described in the LW specifications. Furthermore it works only with predefined multi-user worlds and it does not support dynamic shared objects. The SPIN-3D [6] introduces a new approach on VRML data sharing. More specifically, SPIN-3D proposes a field substitution mechanism rather than a node insertion mechanism (that both LW and VSPLUS use). The main idea behind SPIN-3D is the

substitution of object fields that need to be shared by new fields with network behavior embedded (called SharedNodes). It supports easy transformation of a single user world to a multi-user 3D world, as well as manipulation of dynamic created shared objects. However SPIN-3D is not compatible with a standard VRML browser. The above reasons led us to introduce a new approach for VRML data sharing, which is described in this paper. Main consideration of our multi-user extension is the easy transformation of single-user virtual worlds to multi-user virtual worlds, as well as the conformity with any standard VRML97 browser. Furthermore, we present the EVE' s communication platform, which can be used in order to support CVEs. This platform is based on the use of VRML for representing the virtual environment, VRML-External Authoring Interface (EAI) [8] for accessing the virtual worlds from external applications, and Java networking for the network communication. Main issues regarding the effective network communication as well as the initialization of the 3D scene are also discussed in this paper. This paper is structured as follows. Section 2 presents the Educational Virtual Environments (EVE) communication platform. Section 3 describes the VRML data sharing mechanism used in EVE platform. Sections 4 and 5 discuss main technical issues such as initialization and reliable message transmission across the network. Finally, section 6 describes our vision for the next steps, and some concluding remarks.

2. EVE communication platform

The EVE communication platform is presented in this paragraph in order to introduce the basic modules that are used in the description of the VRML97 multi-user layer, the network communication and the initialization of the multi-user virtual worlds. Having in mind that our platform aims to support CVEs, the main requirements that it should fulfill are the following: (a) to offer scalability; (b) to keep the multi-user worlds consistent; (c) to support specific services for interaction (such as text and audio chat). In order to meet the above requirements, we propose a new hybrid multi-server communication model [2], using two kinds of servers (the *Message servers*, and the *Application servers*) and a database. The basic idea of our architecture (Figure 1) is to divide the processing load of necessary services of a CVE (such as text and audio chat) to a set of servers (application servers) and to exploit dedicated servers (Message servers) for the users' manipulation, the management and the initialization of the virtual worlds. This is a different approach compared to other models [7], and aims to enhance the scalability of the system, because it not only alleviate the Message server from the processing load of the voice and text chat services but it also uses multiple Message servers in case of many concurrent users in many virtual worlds.

Server: The *Message server* is responsible for manipulating a set of multi-user virtual worlds, serving the latecomers in these multi-user virtual worlds as well as keeping the multi-user virtual world consistent by reflecting every shared event to all participants in the virtual world. It can also serve as back-up of another message server by keeping the current state of the multi-user virtual worlds of the other server [2] in order to offer scalability and stability. The message server consists of the following modules: the ConnectionServer, the InitServer, and the VrmlServer.

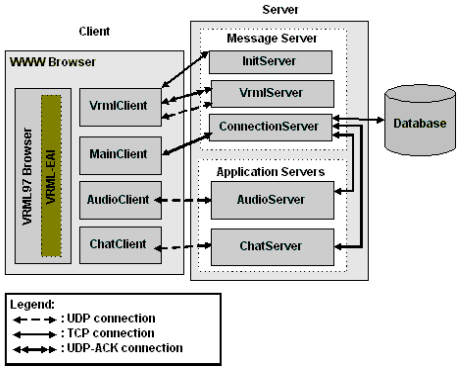

Figure 1: Architecture and Components

The *ConnectionServer* is the core module of the MessageServer. Every other server is connected to the ConnectionServer in order to be aware of the users that are connected to the system. Basic operations of the ConnectionServer are (a) to handle the connection requests of the users in a certain virtual world, (b) to communicate with the database in order to authenticate the users, (c) to accept the users into the system and notify the rest of servers for the new user. The *InitServer* is responsible for the initialization of the multi-user virtual world in each new client (new connected user into the system). This server holds the last state of the virtual world. The *VrmlServer*, is responsible for sharing the state of the multi-user virtual world as well as for sending update messages regarding the user avatar's position and orientation in the multi-user virtual world. The *Application servers* are responsible to offer specific functionality to the participants in the virtual world. Each application server is connected with the ConnectionServer in order to be aware of the connected/disconnected users to/from the system. So far the ChatServer (for text chat), and the AudioServer (for voice chat) have been implemented. The database of the system contains information about the users that have access to the system as well as the available groups that a user can join. Furthermore the database contains the role of each user per group.

Client: The client, in order to communicate with the above-described set of servers, consists of the following components: the web browser, the VRML browser, the MainClient, the VRMLClient, the ChatClient, the

AudioClient. The VRML browser is a plug-in, used for the users' navigation in the VRML world. Any VRML97, EAI compliant browser could be used. The other clients are java applets. The MainClient establishes the initial connection to MessageServer and presents the current connection status, and a list of the participants populating the same multi-user virtual world. The VrmlClient is responsible for the initialization of the 3D scene as well as for the interaction between the user and the 3D scene. The ChatClient is used for the text chat and the AudioClient for voice chat functionality.

3. VRML data sharing using EVE's approach

The basic goal of a platform for CVEs is the sharing of users' representations and actions in the worlds. So far there is no standard in this area. The LW working group of the Web3D consortium, which has been frozen, has made the first attempt for standardization. Other remarkable proposals are the VSPLUS proposal, which is a simplification of the LW proposal, and the SPIN-3D approach. As it can be realized from the introductory section, the afore-mentioned approaches for VRML data sharing have some limitations. LW is complex, VSPLUS does not support dynamic created objects and SPIN-3D requires a proprietary VRML browser.

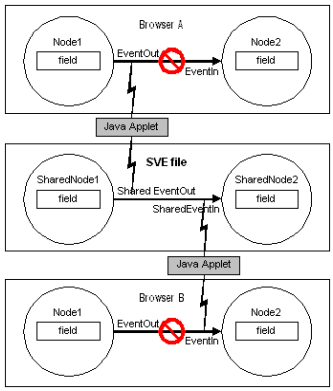

Figure 2: ROUTE removal

In order to overcome these limitations we introduce EVE's approach. Main characteristic of our method is the removal of ROUTEs (Figure 2) that we want to share (from the single-user world), rather than the "insertion" proposed by VSPLUS and "substitution" proposed by SPIN-3D. These ROUTEs are inserted in a new kind of file: the SVE file. Benefits of the EVE approach are the simplicity of use, the support of dynamic created objects, and the conformance with a standard VRML97 browser. Drawback is the need for one SVE file per multi-user 3D world. In the following paragraphs, the following concepts regarding the VRML data sharing are introduced: SVE file, SharedNodes, SFCreate and SFDelete Events. Afterwards, we describe the way that a single-user VRML world can be transformed to a multi-user VRML world, using our approach as well as how we manipulate the sharing of human created objects.

3.1 SVE file

In a multi-user VRML environment, a small part of objects and especially nodes, eventIn(s), eventOut(s) and ROUTE(s) is (are) necessary to be shared. Therefore, each new client in the virtual scene should be aware of both the shared objects and the current state of each shared object (especially if the client is not the first client that entered in the CVE). The concept of SVE file is introduced in order to: (a) Notify the clients what are the sNodes (we call sNodes the nodes of the single-user virtual worlds, that we want to share); (b) Help the initialization process, in order to keep the latecomers in the virtual world consistent with the previous actions on the 3D scene; (c) Support dynamic created objects; (d) Support necessary actions in a CVE such as text insertion, which that are not supported by VRML. Actually the SVE file has a VRML-like syntax and it contains all the sNodes and ROUTEs of a multi-user virtual world. More specifically, the SVE file must contain all the ROUTEs that have been removed from the single-user world along with their source and destination nodes. A sNode's definition in the SVE file should not include all the eventIns, eventOuts and fields supported by this node in the single-user VMRL file, but just the eventIns or eventOuts that are involved in a shared route. Some special nodes are also supported such as: **(a) text:** a node that 'implements text' is used for the text insertion in the virtual world from an external java applet. We decided to introduce this node due to the fact that text insertion in a virtual world is a common problem in VRML [8]. Although some effort has been done in order to overcome this difficulty, the available solutions are still far from functional. **(b) routeCreator:** a node that 'implements routeCreator' is used for the communication of the world with the client in order to dynamically create and delete shared nodes and routes. Apart from these special nodes, two special event types are supported: SFCreate and SFDelete. These event types are presented later in this paper.

3.2 SharedNodes

Using the SVE file, both clients and servers are aware of the sNodes (the nodes that the creator of the virtual world wants to share). Therefore, in order to share the events and the state of each sNode we need a method in order to update their state. For this reason we introduce the concepts of SharedNode, SharedEventIn and SharedEventOut. Actually, a SharedNode (SharedEventIn or SharedEventOut) is the representation (of an instance) of a sNode in Java. Therefore, the delivery of all SharedNodes to client solves the problem of delivering the set of sNodes. Furthermore, a SharedNode (SharedEventIn

or SharedEventOut) contains the same values as the corresponding VRML sNode (EventIn or EventOut) and a pointer to this node. So, the network behavior is embedded in the SharedNodes. Thus, when a user alters a value of a VRML node in the 3D scene, the same value of the corresponding SharedNode is updated concurrently. Respectively, when an event-message arrives from the server (VrmlServer) and it is applied on a SharedNode, the same event is applied to the corresponding VRML node (sNode), modifying the 3D scene. Furthermore, a SharedNode contains the parameter LastModified, which maintains the sequence number of the last event that has been applied to it. This parameter is very critical for the initialization process as described later in this paper. The SharedNodes can represent the following VRML node types: SFBool, SFColor, SFFloat, SFInt32, SFRotation, SFString, SFVec3f, SFVec2f, and SFTime. The SharedNodes can be lockable or unlockable. An unlockable node can be manipulated by every user. When a user locks, only s/he can manipulate this object until s/he unlocks it. The other participants in the virtual world can see the actions of the user that manipulates the shared object. In order to denote whether a shared object is lockable or not we use the boolean parameter "lock" in the SVE file when declaring a sNode.

3.3 SFCreate and SFDelete Events

In order to solve the problem of the initialization of dynamic created objects, especially for the latecomers, we introduce the concept of SFCreate and SFDelete events. The syntax of these events is the same with the syntax of VRML eventIn and eventOut. They are inserted in the SVE file (and not in the original VRML file). We use SFCreate to denote the event that creates dynamically a VRML object in the original VRML file. Actually, an SFCreate is an SFString, whose each new value creates a new SharedNode at the server (InitServer) as a child to the destination SharedNode. We have chosen SFString type due to the fact that this type is the more general event type of VRML. Therefore, using SFString type we can simulate every other type. In order to delete every dynamically created object the SFDelete event is introduced. This event is actually an SFInt32 event. The value of this event defines the actual position (in a stack of nodes) of the dynamic created SharedNode that must be deleted. Furthermore, if the value of the SFDelete event is -1, then all “children” SharedNodes of the destination SharedNode will be deleted.

3.4 Dynamic sharing

Sometimes an object should not be shared when the virtual world is loaded but at a later time. Some other times, an object is not predefined and its definition could not be inserted in the SVE file. These occasions show that it is necessary to have a way to make a VRML node and a ROUTE shared without accessing the SVE file. This can be achieved by using the connection between the multi-user VRML world and the EVE's server through the java applet (EVE's client) and the network. A special node that ‘implements routeCreator’ is used for this communication. An eventOut of type SFString informs the java applet about the dynamically shared routes. Moreover, this node can use another eventOut of type SFString to delete shared nodes and routes. All the routes that contain either a deleted node or a deleted event are also removed.

3.5 Easy transformation of a single-user 3D world to a multi-user 3D world

Main goal in the designing phase of the EVE platform is to offer an, as simple as possible, way of making multi-user a single-user VRML world. Indeed, the transformation of a single-user 3D world to a multi-user 3D world is quite simple using the EVE's VRML data sharing. The creator of the 3D world should follow the following steps:

```
Collision {children DEF AvatarRoot Group{children[ ]}}
DEF ProxSensor ProximitySensor {size 1000 1000 1000}
```

Figure 3: Necessary nodes for avatar representation

Step 1: Insertion of both a Collision node (which is the root of the users' avatars) and a Proximity sensor node (which is responsible for giving the position and orientation of the avatars in the virtual world). Using these nodes we can share the users' representation (Figure 3).

Step 2: Removal of the shared ROUTEs from the original VRML file (to add them into the SVE file).

Step 3: Creation of the corresponding SVE file. The SVE file should contain the definition of the nodes that we want to share (sNodes). More specifically, the starting node of a ROUTE (along with the corresponding eventOut) and the destination node (along with the corresponding eventIn) are inserted. Also, the ROUTEs that are removed from the original VRML file are inserted into the SVE file.

```
#VRML V2.0 utf8
Group { children [
DEF box Transform {children Shape{geometry Box {} }}
DEF sensor SphereSensor {}]}
ROUTE sensor.rotation_changed TO box.set_rotation
```

Figure 4: An example of single-user VRML world

In order to show the transformation of a single user world to a multi-user virtual world using the EVE's VRML data sharing, we use as example a single-user VRML world, which contains a simple box that can be rotated in any direction. The VRML code of this world is shown in Figure 4. After the steps 1 and 2 the VRML world has been transformed as it is shown in Figure 5 (the bold fonts are used for the code of the single-user VRML world). After the step the 3, the corresponding SVE file is created as it is depicted in Figure 6.

```
#VRML V2.0 utf8
# Step 1: Addition of Avatar Root and Proximity Sensor
Collision{children DEF AvatarRoot Group{children [ ] }}
DEF ProxSensor ProximitySensor {size 1000 1000 1000 }
Group { children [
DEF box Transform {children Shape{geometry Box{}}}
DEF sensor SphereSensor {}]}
# Step 2: Removal of the shared ROUTEs
#ROUTE sensor.rotation_changed TO box.set_rotation
```

Figure 5: An example of multi-user VRML world using EVE approach

```
node box {eventIn SFRotation set_rotation;}
node sensor{eventOut SFRotation rotation_changed;}
ROUTE sensor.rotation_changed TO box.set_rotation;
```

Figure 6: An example of SVE file

4. Packet Transmission across the Network

In our both server-client and server-server communication different types of communication are implemented, each having special characteristics that provide the requested functionality. In this way, we try to achieve the improvement of the overall performance, as well as to minimize the information that it is transmitted across the network. Generally speaking, the network communication in the EVE platform is based on TCP or UDP communication (Figure 1). Furthermore, we have implemented a UDP with one-way acknowledgements in order to transmit quickly and to ensure reliability for a specific class of messages. More specifically, we have divided the messages, according to the actions in the multi-user virtual world, in two classes: (a) Position messages (*posMes*), and (b) Important messages (*impMes*). PosMes are messages that define the position of avatars in the multi-user world. ImpMes are all the remaining messages. We separate the messages into two categories, because posMes have different characteristics and demands than impMes, and therefore different types of communication should be implemented for each type of messages. The following paragraphs describe in detail the three types of communication that are used in the EVE platform.

TCP Communication: We use TCP communication only when primary objective is the reliable transmission of information. In essence, we use TCP for the server-server (e.g. ConnectionServer-ChatServer) and for the initialization process of the client-server communication. For the first case, TCP is the only choice, as data loss might lead to inconsistency and therefore to problematic execution of the servers. Similarly, when a client is connecting to the server, it needs to receive vital information in order to initialize its 3D scene and any loss of information could lead to unwanted results.

UDP with One-way Acknowledgements: UDP communication is fast but not reliable. On the other hand, TCP is reliable but more time consuming; therefore, a middle solution should be implemented. The idea lies beneath the fact that in some cases packet loss is not very important, yet in others is catastrophic. To become more specific, in some cases, the lose of a packet that is send from a client to the server is not creating any problems, as none of the clients (not even the sender) will receive anything and therefore the consistency will remain. On the contrary, the loss of a packet that is sent from the server to a client could create a severe dependency problem. Thus, for these cases, we use UDP with acknowledgements for the server to client communication (one way). In our platform, we use this type of communication for the transmission of impMes messages, as we need to assure that all clients receive the same impMes messages.

Simple UDP Communication: Simple UDP is the fastest type of communication. It does not guarantees that data will be sent correctly, but it will send them very fast. This type of communications is used when primary objective is the vivid transmission of packets across the network, and when packet loss does not create any important problems. In our case, we use UDP communication for the transmission of posMes messages. As these messages define the position of avatars into the scene, the only effect that the loss of a message could create, is minor (and temporary) misplace of an avatar in the VRML scene.

5. Initialization of the 3D Scene

An important issue in a multi-user 3D environment is the initialization process of a new participant. When a new user enters a multi-user world, the shared objects should be updated in his/her client in order to become consistent to all other participants. The client accomplishes this, as follows: (1) It is notified about the sNodes list of the VRML scene; (2) It downloads the last state of sNodes, using SharedNodes; (3) It applies any updates of the sNodes, which have been made to the sNodes during the initialization period.

The initialization process in the EVE platform is based on the use of both SharedNode and SVE file. When InitServer starts its execution, it parses the SVE file and creates a list of SharedNodes, the SharedList. Then, the initialization process is implemented in the following steps:

Step 1: Transmission of SharedList from InitServer: Client opens a TCP connection to InitServer and it receives the list of SharedNodes (Figure 7-1).

Step 2: Connection to VRML server: When the client starts receiving the SharedList from InitServer, it is concurrently receiving any update message from the VRML server that has been applied to sNodes by other participants in the CVE. These messages are entered in the client's buffer and are not applied until the end of the transmission of the SharedList (Figure 7-2).

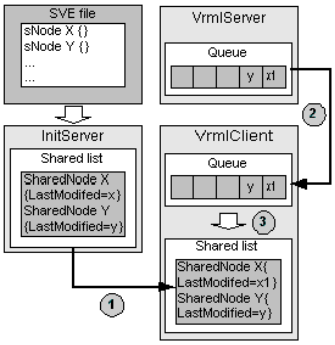

Figure 7: Initialization process

Step 3: Disconnection from InitServer and application of shared events: The client has completed receiving the SharedList and closes its connection to InitServer. Then, it starts applying the events (messages) that are buffered in its queue in the following way (Figure 7-3): The client retrieves from the SharedList the SharedNode that the event is supposed to be applied to. Then it compares the SharedNode's LastModified variable with the event's sequence number. If the second one is bigger then the event is applied on the SharedNode, otherwise it is ignored. For example, as shown in Figure 7, when the client receives SharedNodes X and Y their LastModified variables have values x and y respectively. While downloading the rest of the SharedList, two events are sent from VrmlServer with sequence numbers x1 (x1 > x) and y that concern SharedNodes X and Y respectively. After the end of the transmission of the SharedList, the client starts applying the events. Firstly, it compares x1-event to LastModified variable of SharedNode X. As x1 > x, the event is applied on the SharedNode and its LastModefied variable is updated to x1. On the contrary, when the client compares the value of y-event to the LastMidiafied variable of SharedNode Y, it sees that the event has already been applied to the SharedNode Y, thus it ignores it.

6. Conclusions

In this paper we present the EVE platform, which can be used in order to support every multi-user VRML virtual world. We also introduce EVE's approach for VRML data sharing. This approach is based on both the concept of the SVE file and the removal of the ROUTEs that we want to share from the single-user world, rather than the "insertion" proposed by VSPLUS and "substitution" proposed by SPIN-3D. The concept of the SVE file guarantees the VRML97 conformity, and ensures the proper initialization of the 3D scene in order to serve the late comers in the multi-user virtual world and. SVE file also extends in a transparent way the VRML behavior in order to support dynamic creation of shared objects as well as text insertion in a VRML world. Furthermore, using the EVE's approach, the transformation of a single-user world to a multi-user world is very simple. The communication architecture of the EVE platform is also presented. Main characteristics of this platform are the efficient communication across the network, which is based on three types of communication, as well as the scalability due to the hybrid multi-server communication model. The usage of Java for creating both servers and clients of the platform as well as the conformity with the VRML standard guaranteed the interoperability and openness of our platform. From the functional point of view our platform provides the basic functionalities in order to support CVEs. These functionalities are text and voice chat, user representation by human-like avatars, support of users' groups, as well as support of users with different roles and rights. Regarding our next steps, the first goal is to improve the initialization process for achieving better manipulation of dynamic created shared objects. We also plan to simulate the described architecture in order to measure its effectiveness and scalability. Furthermore, we would like to implement a VRML parser in order to facilitate the creation of the SVE file. Moreover, due to the fact that CVEs are increasingly gaining visibility in the field of collaborative e-learning, we are working on creating CVEs that support collaboration and interaction among users for educational purposes. A first prototype has been implemented so far, which is available at [5].

7. References

[1] Araki, Y. "VSPLUS: A high-level multi-user extension library for interactive VRML worlds". In proceedings of the third symposium on Virtual reality modeling language, February 16-20, 1998, Monterey, California, United States, pp.39-47.

[2] Bouras, C., Psaltoulis, D., Psaroudis, C., and Tsiatsos, T. "Protocols for Sharing Educational Virtual Environments". In proceedings of SoftCOM 2001, Split, Dubrovnik (Croatia) Ancona, Bari (Italy), October 09-12, 2001, Vol. II, pp. 659-666.

[3] Bouras, C., Triantafillou, V., and Tsiatsos, T. "Aspects of collaborative learning environment using distributed virtual environments". ED-MEDIA 2001, Tampere, Finland, June 25-30 2001, pp. 173-178

[4] Diehl, S., *Distributed Virtual Worlds, Foundations and Implementation Techniques Using VRML, Java, and CORBA*. Springer-Verlag, Berlin Heidelberg, Germany, ISBN 3-540-67624-4, 2001.

[5] EVE (Educational Virtual Environments) prototype, http://ouranos.ceid.upatras.gr/vr/.

[6] Picard, S., Degrande, S., Gransart, C., and Chaillou, C. "VRML data sharing in the spin-3D CVE". In proceeding of the 7th International Conference on 3D Web Technology-Web3D 2002, Tempe, Arizona, USA pp. 165 - 172, 2002

[7] Singhal, S., and Zyda, M,. *Networked Virtual Environments: Design and Implementation*. ISBN 0-201-32557-8, ACM Press, 1999.

[8] Web 3D Consortium, http://www.web3d.org.