A PLATFORM FOR SHARING EDUCATIONAL VIRTUAL ENVIRONMENTS

Ch. Bouras [1, 2], D. Psaltoulis[1], Ch. Psaroudis[1], Th. Tsiatsos [1, 2]
[1] Computer Engineering and Informatics Dept., Univ. of Patras
GR-26500 Rion, Patras, GREECE
[2] Computer Technology Institute
61 Riga Feraiou STR - GR-26121, Patras, GREECE
Tel: +30-61-960375, Fax: +30-61-960358, E-mail: bouras@cti.gr

Abstract: *In recent years, a lot of work has been done on multi-user virtual environments. This work led to platforms that aim to support 3D virtual communities. However, little research has been done on specific services and functionality, which affect significantly the design of a system. The growing need for communication, visualisation and organisation technologies in the field of e-learning environments, has led to the application of virtual reality and the use of multi-user real-time communication platforms to support these needs. In this paper we present the design and implementation of a platform, suitable for educational virtual environments. Apart from the platform itself, we present the technological choices, a new method for sharing virtual environments and a new protocol for educational virtual environments.*

KEYWORDS: Educational Virtual Environments, Platforms, Protocols, VRML, Java, Networked Virtual Environments, Multi-user Communication, E-learning, EAI.

INTRODUCTION

E-learning and especially collaborative e-learning is one of the emerging needs of the information age. Also the use of Virtual Reality (VR) technology in the educational process is considered useful and effective. Furthermore, the need for collaborative e-learning leads to the development of environments that support collaboration and interaction of learners and teachers for educational purposes. This led us to design and to develop a Virtual Environment (VE) for educational purpose [1]. We call these environments Educational Virtual Environments (EVEs). An EVE is a Collaborative VE-system (CVE) aiming not only to collaborative tasks but also to additional educational tasks such as synchronous and asynchronous learning. An EVE is a set of virtual worlds or a virtual world that offers educational functionality to its users. Users' graphical representations (avatars) populate the EVE. They are provided with additional characteristics such as gestures, interaction, movements and sound. In order to implement an integrated EVE, some basic requirements should be satisfied. These requirements are can be divided to technical and functional. Technical requirements are the scalability and interoperability of the system, the consistency of the environment, the quality of services, and the effective management of system failure. From the functional point of view the proposed environment should be easy to use, support of application sharing, audio and text communication, transmission of users' avatar and users' profile to the rest of the users as well as transmission of the educational material. Furthermore it should support multi-modal interaction between the users through visual communication, realistic user representation, and real-time display of users' movements, visualization of the learning environment as realistically as possible, and an easy way for sharing the virtual environments. Moreover it should give the lecturer the capability to administer her/his own courses and to monitor the learners' progress and participation. To achieve these goals, we have to use a platform for educational virtual environments in order to offer educational services to users in a sufficient way. In our days there are several platforms that support on-line virtual communities. However, current platforms rarely support both educational communities and the above goals. Other research platforms emphasize on specific research issues such as facial communication [6], support of heterogeneous network [5], [3] or reliability, being and limited in breadth [4]. This gives rise to a proliferation of independent and often partial systems. The solution of integrating or combining work from different groups may be very difficult because of different philosophies and assumptions, making this solution inappropriate for educational purposes. The above reasons led us to design and develop a platform for educational virtual environments.

1. ARCHITECTURE

The basic idea of our architecture is to divide the processing load of necessary services of an EVE (such as shared objects, chat and audio communication, etc.) to a set of servers aside from communication of users or management of the virtual worlds as described in other models [3]. In addition, the whole system will serve as a virtual representation of the relevant theme and a presentation mean for the available material. The structure of an educational community implies a VE that can be separated into smaller parts (VEs). This provides a "segmentation" of the virtual community that enables us to design a communication model that consists of a number of message servers. Each message server hosts some specific VEs and, at the same time, it is a back-up server to a number of other VEs. The set of message servers constitutes a locus of control of the whole system. The proposed architecture is based on the **Message Server**, the **Application Servers**, and the **Client**. The message server has to accomplish three main tasks: (a) to transmit virtual world content, (b) to offer scalability to the system and (c) to keep the 3D world consistent. In order to provide specific functionality, we use dedicated application servers. According to previous described requirements the main applications that should be offered are chat and audio communication as well as shared objects in order to support the educational process and to present educational material. For these reasons we use three types of application servers: (a) **Audio server**, which is responsible for providing real-time streaming audio capabilities to the whole system. (b) **Chat server**, which is responsible for chat capability. (c) **Shared object server**, which contains specific objects that can be shared in the VE. The system clients interact mainly with message server. This model offers scalability (due to the fact that the processing load is divided, and servers for additional services can be added without affecting the end user), as well as concerted management and authentication procedure. Furthermore the users do not have to possess excessive system or network characteristics. In addition there is no central point of failure. Also the model is flexible. For example, if the number of users is small, some of the servers dedicated to one service can be consolidated in a message server. For these reasons we believe that this model is well suited for educational purposes.

2. PROTOCOL FOR EDUCATIONAL VIRTUAL ENVIRONMENTS (pLVE)

In order to support the above architecture a suitable protocol should be designed and implemented. Many protocols for shared virtual environments have been presented [3]. Most of the existing protocols are well suited for shared virtual environments. However, in order to support EVE, it needs the manipulation of special data types, as mentioned in the previous described requirements of EVEs. These data types are: **Triggers** (messages, that need little bandwidth to be transmitted), **Streams** (that are used for transmitting audio and video data),

State update messages (that are responsible for the consistency of the 3D scene), and **Files** (that are the 3D virtual worlds, user's avatars and additional educational material provided by users). The need to support the above data types guides us to design and implement a protocol for the interaction between the components of the previous described architecture. This protocol, which is named pLVE, is presented in [2].

Some advanced issues that the protocol should resolve are the reliability of the communication and the consistency of the virtual environment. As mentioned before, the main component of this architecture is Message Server. In order to deal with these issues Message Server is separated into three modules, **InitServer** (which maintains a list of shared nodes), **ConnectionServer** (which administrates connection of clients to Message Server) and **VrmlServer** (which reflects every event-message it receives back to all connected Clients). The packet that the pLVE uses has the format, which is depicted at Figure 1:

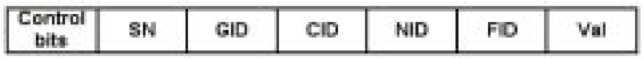

Figure 1: pLVE packet structure

Sequence Number (SN): This is the number that VrmlServer assigns to each packet that receives. Clients use this number in order to preserve the correct order of the received packets.

Group ID (GID): It is the ID of the VE that the packet is going to. The use of this field makes possible the co-existence of more than one VE in the same server.

Client ID (CID): It is the ID of the client that sends the specific packet. It is used mostly for security reasons.

Node ID (NID): This is the unique name of a shared node. It is the same name that is defined in the .wrl file with the DEF statement.

Field ID (FID): This is the name of a shared event that belongs to the node with the previous NID.

Value (Val): It is the value of the FID field, which we want to transmit to VrmlServer. In essence, this is the data that we want to share with all other users.

2.1 Achievement of reliability

One of the main problems in Networked Virtual Environments is dealing with is the loss of packets due to network problems. This, in combination with the non-existence of a central point of failure, illustrates a stable and reliable system. To accomplish the above, pLVE uses the following mechanisms:

- The use of UDP for sending and receiving event packets, implies a non-reliable communication, therefore the loss of event packets is possible. There are two possible cases of loosing an Event:

  Case 1: A client sends an event to VrmlServer, but it never reaches its destination. This protocol ignores these Events. This approach is suitable for real time collaboration where a late Event is worst than no Event at all. The user is free to resent the same Event, if he still wants to, or ignore it, if something has happened during the time gap, that made him change his decision.

  Case 2: An event reaches VrmlServer, where it is mirrored to all connected clients. A certain client though does not receive it. In this case a problem of consistency is created. Numbering every Event that reaches VrmlServer before transmitting it back to all clients solves this problem. If for a long period VrmlServer does not receive any Events, then a numbered “empty” event is sent to all clients every few seconds. When a client receives an Event that has a bigger than the expected number, it automatically understands that a previous Event was lost and it makes a request to InitServer in order to receive it. The client waits to receive and apply the lost event to the VE, before applying the next events. If a client does not receive any Events for a long period, even “empty” Events, then the message “Connection Lost” is printed on user’s screen. In this case the reconnection of client is demanded.

- Besides hosting a VE, each message server represents a backup server to a number of other Message Servers. The ConnectionServer of each Message Server is responsible for maintaining back-up information of other Message Servers. Due to the fact that it administrates the connection of clients, it contains all the necessary information for every connected client. Therefore, if a Message Server fails, then any of its backup servers can take over its processing load, preserving the integrity and functionality of the overall system.

2.2 Achievement of consistency

In order to preserve a consistent VE, pLVE should resolve the following issues:

- **Initialization**: A new participant of the VE should receive the last state of the 3D scene. We deal with this problem by maintaining an update instance of the shared objects that constitute the 3D scene in the InitServer of message server. When a new participant connects to the VE, InitServer sends the set of shared objects, making it consistent to other clients.
- **Synchronization**: All connected clients should receive the same Events in the same order. As mentioned before, numbering every incoming Event that reaches VrmlServer before transmitting it back to all clients accomplishes this cause.

3. IMPLEMENTATION ISSUES OF THE PLATFORM

3.1 General

The platform we developed is based on the client-multiple server model. The client communicates with server either directly or through a file (sve file) as depicted in Figure 2.

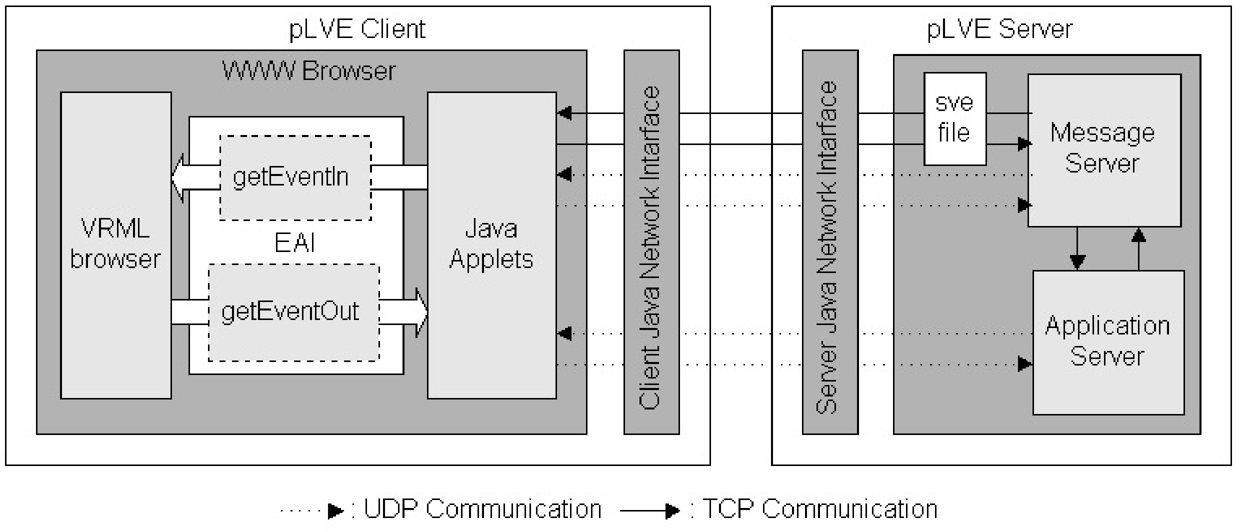

Figure 2: Client-server communication

In the following paragraphs we present a method for sharing events, and the implementation of the client and the servers. Before this it would be useful to refer to the adopted technologies (VRML, EAI, Java) for the development of our platform. Due to the fact that our platform is based on pLVE, it requires a 3D virtual environment. The development of this environment implies the need of a language for describing 3D objects. Such a language is Virtual Reality Modeling Language (VRML 97) [7], which is the international standard in this area. Also we use the VRML External Authoring Interface (EAI), which defines an interface between a VRML world and an external environment. It provides a set of methods that an external application can use to interact with, and dynamically update a 3D scene in real time. Another choice in the implementation phase was Java programming language, in order to create multithreading applications, to implement the network communication, and to endure interoperability to the system.

3.2 A method for sharing virtual environments

Having in mind the design of the protocol, we propose a new method of making a VE shared, with which sharing events is easy. To achieve this, we introduce a new file type with the extension .sve (Shared Virtual Environment). The sve file is a sub-layer between the network interfaces of client and server, which used for the initialization of the VE. It constitutes an abstract form of the .wrl (VRML) file, which contains the set of all shared nodes of the VE. Knowing that all other nodes are static, this set determines the current condition of the 3D scene. Thus, when a client connects to a shared VRML world, it only needs to receive this set of nodes in order to update its instance of the 3D scene. With the term **shared node,** we mean a VRML node that contains at least one shared event. Not all users can alter a certain shared node, though. For example, a user is able to send events concerning its avatar. Other users only receive those events in order to update its condition in their instance of the 3D scene. Moreover, it is desirable for some shared nodes, that only one user at a time to be able to access them. We accomplish this by adding the variable '*lock*' in each shared node. This variable can take the following values:

-1	The option of locking the shared node is disabled
0	The shared node is available for a user to lock it
>0	The user with ID = lock has locked the shared node

Furthermore, a shared node must have a unique name in the .wrl file assigned with the DEF statement of VRML. A **shared event** is defined as an event (EventIn, EventOut) that is visible to all users. The term '*visible*' indicates that when an event happens, it is transmitted to all of them.

3.2.1 Making a VRML world shared

One of the most attractive features of the prposed platform is the easy and practical way we provide for making a VRML world shared. System administrator must follow certain steps to accomplish this goal. These are: **(a) the creation of the .sve file,** and **(b) the modification of the .wrl file**: **(a)** After selecting the routes that must become shared, he must retrieve the EventIns and EventOuts that belong to each shared route and finally extract all nodes that contain at least one of the above fields. Then, he has all the necessary information to create the .sve file. This file must be created according to syntax rules presented in *Table 1*. Furthermore, if the use of some objects in the VRML world needs to be limited to one user at a time, then the variable 'Lock' of these nodes must be set to zero. In all other cases this variable should be omitted or set to minus one (-1). An example of an sve file is presented in *Table 1*. **(b)** Some minor changes are required in the wrl file in order to make possible the insertion of avatars into the world. In particular, two more nodes must be added in this file. The first node, avatarRoot, is a simple VRML node, which is the parent node of all avatars. The second node, addAvatar, is a script node that inserts a new avatar as a child to avatarRoot. To accomplish this, a route must be added from addAvatar to avatarRoot.

*Table 1. Syntax rules and an example of an *.sve file*

Syntax rules	**Example of an *.sve file**
<file>::= {<node>} | <node>{<node>}<route>{<route>}	
<node>::= **Node** <name> **{** <body> **}**	node BoxScript
<body>::= <lockvar> {<SharedEvent>}	{
<lockvar>::= **Lock** <intval>	eventIn SFBool isActive
<intval>::= **-1**, **0**, **1**, …, **32767**	eventOut SFColor ColorOut
<SharedEvent>::= <SharedEventIn> | <SharedEventOut>	}
<SharedEventIn>::= **SharedEventIn** <type> <name> [<value>]	node Box
<SharedEventOut>::= **SharedEventOut** <type> <name> [<value>]	{
<type>::= **SFBool** | **SFColor** | **SFFloat** | **SFImage** | **SFInt32** | **SFNode** | **SFRotation** | **SFString** | **SFTime** | **SFVec2f** | **SFVec3f**	Lock 0
	eventOut SFBool activate
<route>::= **ROUTE** <route_edge> **TO** < route_edge>	}
<route_edge>::=<name>.<name>	ROUTE Box.activate TO BoxScript.isActive
<name>::=Valid VRML String	

3.2.2 EAI enhancement for distributing events over the network

As mentioned before, a user must receive all shared nodes in order to initialize its instance of the 3D scene. This implies the need of a practical way of handing over all nodes from server to client. Moreover, a solution to the problem of continuous update of the 3D scene according to received event-messages is demanded. Unfortunately, EAI provides methods only for local interaction with a VRML world; thus the enhancement of EAI is required. The most significant of those enhancements are the SharedNode, the SharedEventIn and SharedEventOut. In essence, a SharedNode, SharedEventIn, SharedEventOut is the representation of a shared node, EventIn, EventOut respectively in Java. Therefore, transmitting all SharedNodes to Client solves the problem of handing over the set of shared nodes. Furthermore, a SharedNode/ SharedEventIn/ SharedEventOut contains the same values as the corresponding VRML Node/ EventIn/ EventOut respectively. 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 that arrives from VrmlServer is applied on a SharedNode, the same event is applied to the corresponding VRML node, modifying the 3D scene.

3.3 Implementation of Client

Satisfying the requirements specified during the design of the platform, a client based on pLVE offers a useful and functional interface. It implements a transparent and simple to understand way of interaction with the VRML world, by concealing from users the distributed nature of the platform. Furthermore, it provides the users with the capability of exchanging messages when being in the same VRML world. *Figure 3* presents the architecture of the Client. The components of the Client are the Vrml Browser, the MainClient the VrmlClient, and the ChatClient. The **Vrml Browser** is a plug-in, used to navigate user in the VRML world. Any VRML97, EAI compliant browser could be used for this cause. Cortona is a tested and recommended solution. The **MainClient** is the applet that establishes and terminates the initial connection to Message Server. When user tries to connect, MainClient sends a connection request to ConnectionServer. The last one responds by sending either an integer, which represents the identification number (ID) of the Client when the connection is successful, or an error message when the connection cannot be established. Similarly, when user presses "Disconnect" button, MainClient sends a terminate request to ConnectionServer and the server replies with a "session ended" message. The **VrmlClient** is the applet responsible for the interaction of user with the 3D scene. It works in two phases. In phase one, it receives all shared nodes from InitServer, using the classes a SharedNode/ SharedEventIn/ SharedEventOut and initializes the VRML world. Then, it enters phase two, where VrmlClient sends and receives events to and from VrmlServer and updates the VRML scene according to the received events. The **ChatClient** is the applet that implements the exchange of messages among users in the same VRML world. When user types a chat message, ChatClient wraps the message into a packet and transmits it to ChatServer. Respectively, when a packet arrives from ChatServer, the contained message is retrieved and displayed on the user's screen.

3.4 Implementation of Server

In essence a server based on pLVE is a set of individual servers, each one designed to carry out a specific operation. It consists of instances of two server types, the **Message Server** and the **Application Servers**. Several Message Servers may co-exist each serving a different VRML world or backup other Message Servers in order to increase the scalability and reliability of the platform. This approach accomplishes the initial goal of distributing the workload in order to avoid network congestion. *Figure 4* presents the architecture of this server.

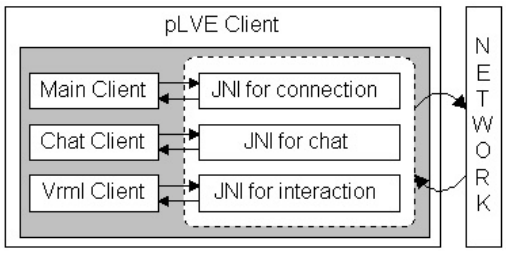

Figure 3. pLVE client

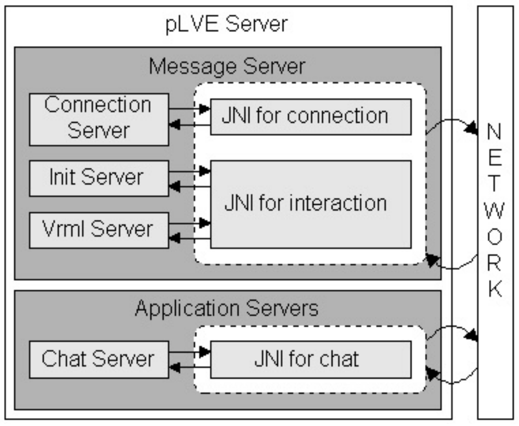

Figure 4. pLVE server

The main modules of the Message Server are: (a) **ConnectionServer:** When a new connection is requested, it either accepts it by sending a unique ID to Client or rejects it by sending an error message. Each time a connection is established or terminated, ConnectionServer informs InitServer, VrmlServer and its back-up servers for this event. (b) **InitServer:** When an "Add Client" event arrives from ConnectionServer, it transmits the entire list of shared nodes that maintains, to the newly added client. (c) **VrmlServer:** When an event reaches VrmlServer it is reflected back to all connected clients. Furthermore, it is transmitted to InitServer in order to update its instance of the VE. Regarding the Application servers, currently we have implemented one application server for chat communication, which called **ChatServer.** This server reflects every incoming message to all Clients that are in the same VRML world.

3.5 Avatars

Our platform supports the use of avatars that are compatible with the H-Anim (Humanoid Animation) specification [7]. Each avatar is contained in an individual wrl file. These files are stored at the InitServer, which is responsible for their handling. A user's avatar can be inserted into the VE using either the *createVrmlFromURL* or *createVrmlFromString* functions of EAI.

Each avatar is a VRML node that has a unique name. Therefore, in order to update the avatars' movement in the VE we use a proximity sensor in the VRML file to catch its position and orientation. We notice that the position and orientation fields of an avatar are shared Events, so we consider it as a Shared node. We send to VrmlServer the values of the position, orientation or triggers for gestures plus the ID of the client. These values are sent to other clients in order to update the condition of the avatar, which corresponds to the specific ID in their local VE.

4. CONCLUSIONS -FUTURE WORK

In this paper we introduce the term educational virtual environment and we propose a suitable architecture and a protocol to support this environments. Furthermore we describe some advanced issues concerning the nature of this environments and the way that we deal with them. Currently we are in the implementation phase of our platform In order to realize this platform we use open standards and technologies such as VRML 97 for the VEs, Java for the implementation of the servers and the multi-user clients and H-Anim for the construction of the avatars. Our next steps are to enhance the scalability of the platform supporting multicast communication between the message servers and the clients; to integrate audio communication (using the RTP protocol and the JMF tools); to integrate a T.120 compliant server for application sharing; and to implement an educational virtual community for testing and measurements. One experiment with particular interest are the number of simultaneous clients that can be served and the latency for a change by one client to be propagated to all other clients. We also believe that the educational issues that may come up while using this educational virtual community may be even more interesting than the technical issues and they will help us use in a more efficient way the new methods of communication and interaction that educational virtual environments offer.

REFERENCES

[1] C. Bouras, A. Philopoulos, Th. Tsiatsos: "e-Learning through Distributed Virtual Environments", Journal of Network and Computer Applications, Academic Press, July 2001.

[2] C. Bouras and Th. Tsiatsos: "pLVE: Suitable Network Protocol Supporting Multi-User Virtual Environments in Education", International Conference on Information and Communication Technologies for Education, Vienna, Austria, December 6-9 2000, pp.73-81.

[3] W. Broll: "DWTP - An Internet Protocol for Shared Virtual Environments", International Symposium on the Virtual Reality Modeling Language 1998 (VRML'98), ACM, ACM SIGGRAPH, pp. 49/56.

[4] Ch. Greenhalgh: "Implementing multi-user virtual worlds (panel session): ideologies and issues", Web3D-VRML 2000 fifth symposium on Virtual reality modeling language, Monterey, CA USA, February 20 - 24, 2000, pp. 149 - 154.

[5] I. S. Pandzic, Ch. Joslin, N. Magnenat-Thalmann: "Trends in a Collaborative Virtual Environment", International Conference on Software, Telecommunications and Computer Networks-SoftCOM 2000, Split, Rijeka, Dubrovnik (Croatia), Trieste, Venice (Italy), October 11-14 2000, pages 893-901.

[6] Virtual Life NETwork: http://miralabwww.unige.ch/~vlnet/.

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