External Interaction Management of VRML Scenes for E-Learning Applications

José M. Cubero⬀, Luis Salgado⬀, Jesús Bescós⬊, Francisco Morán⬀ and Julián Cabrera⬀

Grupo de Tratamiento de Imágenes

⬀ETS. Ing. Telecomunicación
Universidad Politécnica de Madrid
28040 Madrid - Spain
+34.913.367.353

{jcm,lsa,fmb,jcq}@gti.ssr.upm.es

⬊Escuela Politécnica Superior
Universidad Autónoma de Madrid
28049 Madrid - Spain
+34. 914.972.285

Jesus.Bescos@ii.uam.es

ABSTRACT

This paper describes an innovative approach to solve some of the problems that arise when integrating virtual reality capabilities into e-learning environments. The VRML representation of a scene includes, along with its geometric description, a full specification of the student-scene interaction logic. This representation is rendered by a browser, which also orchestrates the interaction according to the logic. Such a mechanism implies reprogramming and/or replicating partly the logic when modifying the interaction scheme of a single scene for different students. It also prevents any external access to student's actions or scene reactions, which is necessary for on-line evaluation or instruction. We propose to expand the standard interaction mechanism of VRML so that both the specification of the scene logic and the interaction flow are managed by an external and centralized entity following a client-server approach, hence solving the identified problems, while additionally increasing design efficiency and content protection.

Categories and Subject Descriptors

H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems - *Artificial, augmented, and virtual realities*; K.3.1 [**Computers and Education**]: Computer Uses in Education - *Distance learning*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems - *Distributed Applications*; K.6.4 [**Management of Computing and Information Systems**]: System Management - *Centralization/decentralization*; C.0 [**Computer Systems Organization**]: General - *System architectures*.

General Terms

Algorithms, Management, Performance, Design, Human Factors.

Keywords

Virtual environments, e-learning applications, external interaction, information management, client-server architecture, adaptability, VRML, Java.

1. INTRODUCTION

The current development of Internet applications, and especially of those based on computer graphics, allows for the use of e-learning systems including not only multimedia information but also Virtual Reality (VR) capabilities. Thanks to VR, a student can believe to be transferred to another place and enjoy interactive and immersive experiences, which can greatly improve her/his learning process. Low-cost VR experiences are possible with publicly available plug-ins for common web browsers, which can interpret and render interactive 3D scenes described in standard languages such as VRML [8][6] and X3D [9].

When immersed in a virtual world, a student experiences a scene which follows a behavior pattern established by the objectives imposed in the learning process. This pattern is implemented via an interaction logic controlling the interaction flow between the student and the virtual world, and therefore must be programmed and adapted for every new learning situation.

3D scene modeling languages, like VRML, allow to code that interaction logic in the same file where the geometric description of the scene is defined. The interaction management is usually event-based, an event being just some kind of basic information message being sent or received by any scene object or node in an asynchronous way: active nodes (like sensors) defined in the scene launch events that are just routed to other active nodes as a basic means to modify their characteristics (e.g., their position). In order to simulate more complex interactive situations, VRML considers the possibility of defining intermediate script nodes which accept input events, perform some processing according to the received information, and generate the corresponding output events, which are then routed to the desired nodes. These routing and scripting mechanisms, which conform the interaction logic implementing the scene behavior pattern, result in a traffic of events [12] that contains all the information related to user-scene interaction. This information flow is established between the student and the browser that interprets and renders the coded scene.

This scheme results in a lack of flexibility, from at least two points of view. First, every desired modification of the behavior pattern which controls the learning process implies the reprogramming of the interaction logic embedded in the same file (structure) as the geometric description of the scene. Second, the student's actions cannot be inspected on-line by an external agent (e.g., an evaluator) and the student's perception of the scene can-

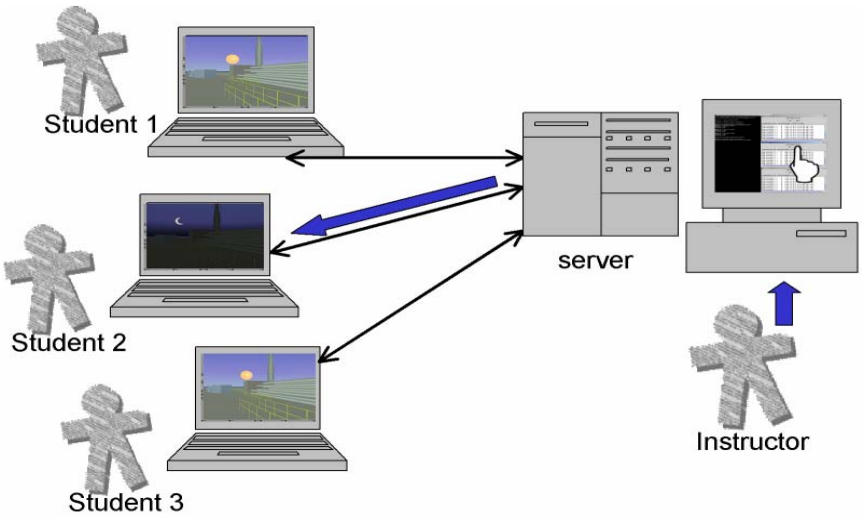

Figure 1. Example of modification of the student virtual environment conditions

not be modified on-line according to the criteria of an external agent (e.g., an instructor). These problems prevent any kind of dynamic adaptation of the learning process in response to the student's actions, which is one of the main features that e-learning technology can provide over traditional educational paradigms.

In order to solve the first problem, some works propose to separate the interaction logic from the local machine where the scene is rendered [1]. We propose to deepen into this approach and solve all the identified problems via the inclusion of a client-server scheme not just intended to centralize course management (an architecture frequently used in e-learning systems [12][2][3]) but specifically targeted at the management of the interaction information flow. This innovative approach completely decouples the behavior pattern of the scene from its geometric description, hence reinforcing the reusability of the models, which may be linked to different logics, as well as the flexibility in the design, as parallel developments of the geometry and the logic can be carried out almost independently.

Moreover, although the scene geometry has to be available in the client for rendering, it is mostly useless (at least, from an educational viewpoint) without the behavior pattern, which is kept at all times in the server, so the overall scene is inherently protected against undesired copies or misuse.

Additionally, our approach enables to achieve on-line inspection and adaptation of the interaction flow between several students (i.e., several clients), each having her/his own virtual learning environment independent from the others, and a single evaluation and instruction center (i.e., the server). An example is shown in Figure 1. Three students are working on the same exercise developed in a specific virtual environment. In a specific moment, the student number 2 shows a higher level of experience. Then, the instructor, located at the server machine, decides to increase the difficulty of his/her exercise changing on-line the illumination conditions of the scene, from daylight to nightlight. The student number 2 continues his/her exercise in those conditions while the rest of the students develop the exercise with the initial illumination. Our approach allows this kind of on-line interaction.

The architecture proposed in this paper is being used in the ongoing EU-funded research project "Self Learning Integrated Methodology –Virtual Reality Tool" (SLIM-VRT, IST-2001-33184) [11]. Section 2 summarizes the VRML standard interaction mechanism and Section 3 describes the overall architecture proposed to modify it. Section 4 delves into some interesting implementation details and Section 5 describes the performed tests. Finally, Section 6 concludes the paper.

2. VRML INTERACTION MANAGEMENT

The VRML standard mechanism for managing the interaction between the user (i.e., the student, in an e-learning application) and the virtual scene can be summarized in three steps:

- Identification and unique labeling of the scene objects that will provide interaction capabilities.
- Definition, for these objects, of the input events they are sensible to and/or the output events they generate.
- Establishment of routes between pairs of objects via linking the input events of one with the output events of another. According to these routes of interaction, the VRML browser delivers properly each generated output event to the object designed to receive it. The receiver object can be directly modified using the information provided by the output event but, to simulate complex situations, a script can process the original output event and yield one or more output events based on it. The system described here exploits this mechanism by the use of the Scripting Authoring Interface (SAI).

The management of a VRML scene generates events continuously. VRML does not distinguish between discrete events, such as those generated as the user activates a *TouchSensor* node, and continuous events, i.e., those generated by sampling over time conceptually continuous variables, such as those generated by a position sensor, a time sensor or the user's action on mobile objects, typically registered by the so-called "drag sensors" (*CylinderSensor*, *PlaneSensor* and *SphereSensor*). For these continuous events, an ideal implementation of a VRML browser would generate infinite samples. Actually, the sampling frequency depends on the particular implementation of the browser, and typi-

cally matches the number of rendered frames per second. In most applications, the number of such events is much more than needed to control the user's actions [12], so there is a need to filter them out to improve significantly the efficiency of the final system.

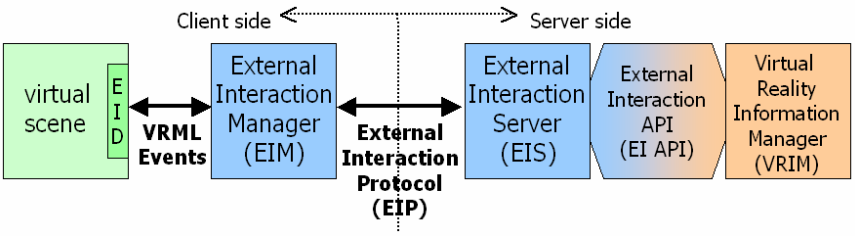

Figure 2. Interaction Management System overview

3. SYSTEM ARCHITECTURE OVERVIEW

Figure 2 presents a block diagram of the proposed client-server Interaction Management System (IMS) architecture. It allows user-scene interactions to be managed by an external entity called Virtual Reality Information Manager (VRIM), which implements the virtual scene interaction logic. It typically runs on a remote server and should operate in real time, network bandwidth permitting.

The IMS implements a bidirectional event-based communication between the virtual scene and the VRIM on top of a TCP/IP link. It involves several modules and interfaces: the External Interaction Manager (EIM) at the client side, and the External Interaction Server (EIS), together with its External Interaction API (EI API), at the server side.

The EIM is a module that, on one side, handles the exchange of VRML events with the scene through the External Interaction Definition (EID) SAI and, on the other, establishes a TCP connection with the EIS, which later follows the External Interaction Protocol (EIP). The VRML events are hence directed from/to the EIM to/from the EIS, which further communicates with the VRIM via the EI API.

The remainder of this Section gives a functional overview of these interfaces, modules and protocols. Section 4 gives a more detailed description of the modules, of which most have been implemented in Java to ensure portability and to take advantage of the Java support offered by VRML.

External Interaction Definition

The interaction, that is managed outside of the virtual scene, has to be defined anyway according to the VRML standard mechanisms introduced in Section 2: it is necessary to specify which objects will be subject to the external interaction, and which type of information they will either provide or accept. This specification, the set of rules and labels that should be added to the geometric description of the scene, is identified as External Interaction Definition (EID) in Figure 2. It consists of the list of the VRML events of the objects allowing external interaction management, and of several parameters required for the proper operation of the EIM.

External Interaction Manager

The External Interaction Manager (EIM in Figure 2) is the script in charge of establishing a TCP connection with the EIS in order to exchange events from the VRML browser, where the scene is rendered, to the VRIM and vice versa.

The EIM procedures are called from a VRML *Script* node of the scene. When the scene is rendered, the EIM establishes a TCP communication with the EIS, which should be available via an IP address (or a valid DNS name) defined by an input parameter, also referenced in the VRML *Script* node. Every event routed from the scene to the EIM is processed by this module and sent to the EIS through TCP messages that follow the EIP. When the browser is closed or another VRML scene is loaded, the EIM detects it and automatically shuts down the TCP connection with the EIS.

External Interaction Server

The main function of the External Interaction Server (EIS) is to act as a TCP server accepting connections from multiple EIM clients, and holding bidirectional communications of events with them according to the EIP.

External Interaction Protocol

The External Interaction Protocol (EIP) regulates the communication of asynchronous messages between the EIM and EIS to allow the transmission of events produced by the user interaction within the virtual scene to/from the VRIM.

External Interaction API

The External Interaction API (EI API) is the interface between the EIS and the VRIM that allows an independent and possibly parallel implementation of both modules. The EIS functionality (i.e., manage the TCP communication) is completely independent from the particular virtual scene, whereas the VRIM implementation fully depends on it, as it manages the user's actions and the corresponding scene reactions. Therefore, the EI API allows a flexible, efficient and updatable implementation of the interaction management.

Virtual Reality Information Manager

The Virtual Reality Information Manager (VRIM) implements the behavior pattern of the scene. It is able to manage simultaneously the information provided by different instances of a scene being rendered in different clients and interacted upon by different users.

The VRIM usually receives VRML events related to the user-scene interactions through the EIM, EIS and EI API, and sends back VRML events in response and through the reverse path. But the VRIM is also able to generate itself messages (events) to be sent to the user at any time, e.g., upon the orders of an instructor.

4. DETAILED MODULES DESCRIPTION

4.1 External Interaction Events

To handle the different kinds of information, the generated External Interaction Events (EIE) are classified into three groups:

- Location Events, which indicate the position of the avatar, representing the user in the scene rendered in the client.
- Timing Events, aimed at transmitting temporal information of the client where the scene is being rendered.
- External Operation Events, which group the different VRML events generated by the user interaction with specific objects in the scene, together with the input events that will be sent by the VRIM as a response to the user's actions, modifying the appearance of the objects being rendered.

The first two types are intended to be used for tracking purposes, e.g., to evaluate the user's position (within the scene) or connection time (for how long the user is experiencing the scene), or how much time the user spends to perform specific required actions.

Communication with the EIS

The ClientTCP module establishes a TCP connection with the EIS. The IP address (or a valid DNS name) of the EIS is an input parameter obtained from the EID.

Filtering of Continuous Events

For most applications, the number of continuously generated events described in Section 2 is excessive to accurately control the user's interaction with the scene. The EIM design allows to configure the number of such events in order to the application requirements, thus reducing significantly the system communication overhead. The EIM sends events at intervals not less than a minimum value specified in the scene through the EID. For instance, the EOperationMgr module (see Figure 3) filters the *SFVec2f*, *SFVec3f* and *SFRotation* VRML events continuously generated by the *CylinderSensor*, *PlaneSensor* and *SphereSensor* VRML nodes, respectively.

The continuous Location Events generated by the user's motion are filtered by the LocationMgr module and, as the EIM knows if the scene is being rendered, it generates the Timing Events and filters them with the TimingMgr module.

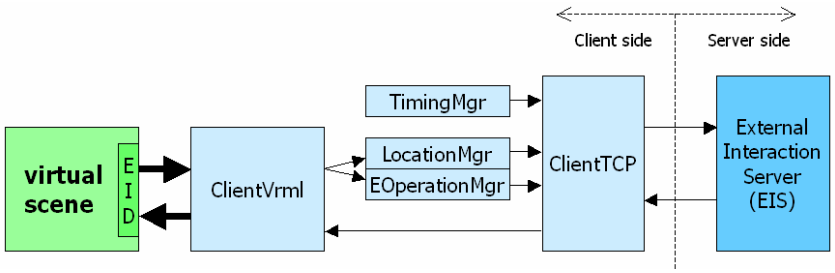

Figure 3. Block diagram of the External Interaction Manager

The last one helps tracking the user's interaction with specific objects in the scene, and to route the corresponding responses provided by the VRIM which modify the scene appearance according to the specific implementation of the behavior pattern.

4.2 External Interaction Manager

A detailed block diagram of the EIM is presented in Figure 3. The path followed by the different event types is identified through the arrows communicating the different modules, the thick ones representing VRML events and the thin ones the three kinds of EIE (Location, Timing and External Operation Events).

Interface with the VRML Scene

The ClientVrml module acts as the interface with the VRML browser. On one side, it extracts the input parameters and the VRML events generated by the user's interaction in the virtual scene. On the other, it updates the scene with the information provided by the External Operation Events received from the VRIM, typically as a response to some previous incoming event originated at the client, but possibly also as an event generated by an instructor behind the VRIM.

4.3 External Interaction Server

Figure 4 shows in detail the different modules that form the EIS. The path followed by the different kinds of events is identified through the arrows communicating the different modules.

The operation of the EIS can be described as follows.

The TCP server (EIServer in Figure 4) is set up initially to accept connections from the client (thick and continuous arrow).

When a new client connection is requested, the server launches a Client Manager (ClientMgr or simply CM) which handles the communication with a particular user. Every CM keeps waiting for the incoming events generated by the client (EIE from client) to which it is connected.

Whenever an event arrives, a specific Event Manager (EventMgr or simply EM) is created to process it by calling the appropriate set of methods of the EI API: Location or Timing or External Operation API.

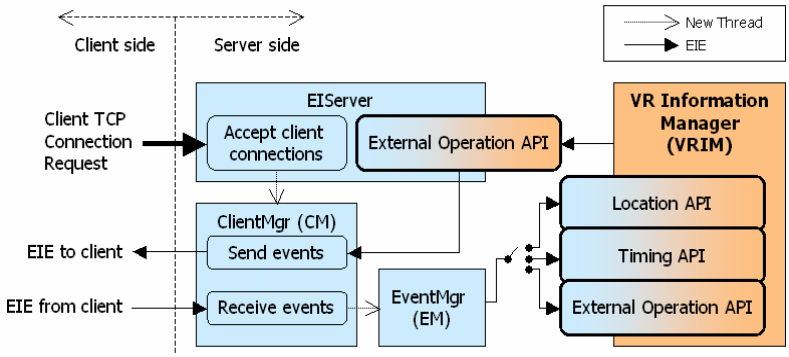

Figure 4. Diagram of the Java classes of the External Interaction Server

On the other side, the events received from the VRIM are passed to the appropriate CM, which forwards them to the client (EIE to client).

4.4 External Interaction API

The Location and Timing Events can only be used in a unidirectional way: from the scene to the VRIM, as they provide information solely depending on the user's operations at the client side. Therefore, the Location and Timing API must be unidirectional.

On the other hand, the External Operation Events do usually require bidirectional communication between the scene and the VRIM, as the VRIM uses this type of events to control the operations performed by the user in the scene (incoming events), and to send back modifications to the scene (outgoing events). Consequently, the External Operation API must be bidirectional, and appears twice in Figure 4, where the two paths of the incoming/outgoing events have been separated for the sake of clarity.

4.5 Virtual Reality Information Manager

The VRIM implements a method for every function of the EI API accessible from the EIS, i.e., the Location API, the Timing API and the External Operation API. These are the methods that finally implement the behavior pattern of the virtual scene, i.e., the number and type of events that will be generated as a response to the user's action or situation. This module, located at the server, compiles the interaction logic that, with VRML standard mechanisms, would have to be integrated either into the scene description file or in attached scripts, that would have to be located at the client before any rendering of the scene could occur.

Although the VRIM can be programmed using Java as the rest of the modules described above, it would perhaps be preferable to use other more efficient high level programming languages.

5. EFFICIENCY EVALUATION

As in any client-server architecture, the efficiency issues related to the number of clients that can be supported by a single server are particularly relevant. In the case of our system, efficiency tests have been carried out in which the computational load introduced by the IMS has been evaluated by measuring the ratio of the processor clock cycles used by the IMS at the server to handle the correct transmission of events.

Two main system variables were found to affect significantly the IMS server efficiency and have been considered during the tests: the number of clients to be supported, and the continuous event filtering parameter, i.e., the minimum interval between continuous events transmitted from each client to the server. In normal operation, user interactivity hardly generates a relevant number of discrete events compared with the number of continuous events (in particular, Location or Timing Events, which can be generated and transmitted as much as one every rendered frame).

Two different scenarios have been considered. In the first one, a predefined tour has been implemented in which the avatar is being moved continuously within a simple virtual scene. In this case, Location Events are continuously transmitted to the server. In the second, an automatic generator of External Operation Events has been implemented to simulate a user's interaction with an object generating continuous drag sensor events.

The overhead introduced by the proposed architecture was measured at both the client and server ends. As expected, the one imposed by the EIM at the client side was meaningless, while the one of the EIS at the server side could be really significant depending on the values of the system variables described above.

Table 1. Ratio (%) of processor clock cycles used by the server

		Number of clients							
		1	2	3	4	5	6	7	8
LEI (ms)	0	1	8	10	11	13	17	21	30
	40	1	2	3	4	5	7	8	10
	500	0	0	1	1	1	1	2	2
EOEI (ms)	0	8	15	30	50	BLOCKED			
	40	2	5	8	10	13	15	18	22
	100	1	3	5	6	8	9	10	12

The percentage of the processor clock cycles used by the server with the two scenarios was measured on a PIV@2.66 GHz PC with 512 MB of RAM, and is shown in Table 1. The first three rows correspond to the first scenario, where the LEI value is the Location Event (minimum) Interval in milliseconds. The last three rows correspond to the second scenario, and EOEI stands for External Operation Event (minimum) Interval.

As it can be observed, the main limiting factor is the EOEI, the overhead imposed by the Location Events being significantly lower for any number of clients. Therefore, as expected, it is mandatory to filter the External Operation Events. However, it has to be considered that these tests assume that the users at the different clients are generating non-stop continuous events, something that, apart from Location and Timing Events, never happens in real applications. According to the 12% in the rightmost, bottom cell of Table 1, it is clear that many more than ten clients can

be supported by a single server with the correct transmission of ten External Operation Events per second (i.e., one every 100 ms) under this artificially extreme situation. Therefore, it can be expected that the required frequency of event transmission be met in real applications.

6. CONCLUSIONS

The Interaction Management System (IMS) we propose allows for the external management of the interaction-related information generated by the actions of a student in a virtual scene, separating the logic ruling the scene behavior from its geometric description. An entity completely external to the scene, the VR Information Manager (VRIM), is able to monitor at a remote server the actions the student performs at the client, and manages remotely the behavior of the scene, modifying it when necessary.

The IMS helps to flexibly and efficiently model and program interactive 3D scenes. Modifications of the logic of the scene, which resides in the VRIM, do not imply modifications in its geometric description. This enables transparency for the student and parallel developments of the geometry and the logic of the virtual environment. The IMS is modular, with well defined interfaces that allow easy modifications and evolutions of the system.

Tests developed on the IMS show its correct functioning even in stressful situations, i.e., with too short a minimum interval between External Operation Events, which is the determinant factor for the computational load at the server. In real applications, with a reasonable number of clients and a normal value for that minimum interval, the correct transmission of the number of events required to effectively control the behavior of the scene can be guaranteed.

The architecture we propose is being used in the EU-funded research project "Self Learning Integrated Methodology –Virtual Reality Tool" (SLIM-VRT, IST-2001-33184) [11], where an e-learning application in a shipping environment is under development. The particularities of this application have been addressed on a case study based approach where the flexibility, reusability and efficiency provided by the separation of the geometric description of the 3D scenes and the management of their logic are proving to be extremely useful.

Future developments are foreseen in the following directions:

- Improved event filtering: currently, only the VRML events generated by drag sensors are filtered, but it could probably be advantageous to let the scene logic designer to add other events to be filtered, and perhaps even to establish priorities among events.
- Porting to X3D: the information types have been defined to be as general as possible in the IMS, so an evolution to support scenes described in X3D should only imply modifying the interface with the scene (i.e., the ClientVrml module in the EIM), the rest of the system being, in principle, independent of the language used to describe the scene.

7. ACKNOLEDGEMENTS

This work has been partly supported by the Plan Nacional de Ciencia y Tecnología of the Spanish Government under project TIC2001-3069.

8. REFERENCES

[1] Bouras, Ch., Philopoulos, A., and Tsiatsos, Th. *A Networked Intelligent Distributed Virtual Training Environment: A first Approach.* International Workshop on Intelligent Multimedia Computing and Networking (IMMCN 2000, a constituent of JCIS 2000), 2000.

[2] Bouras, C., and Tsiatsos, Th. *Distributed virtual reality: building a multi-user layer for the EVE Platform*. Elservier. Journal of Networks and Computer Applications, No. 27, 2004.

[3] Chover, M., Belmonte, O., Remolar, I., Quirós. R., Ribelles, J. *Web-based Virtual Environments for Teaching.* Eusrographics/SIGGRAPH Workshop on Computer Graphics Education (CGE 02), 2002

[4] Eckel, B. *Thinking in JAVA*. Prentice-Hall, 2000.

[5] Harold, E. R. *JAVA Network Programming*. O'Reilly, 2000.

[6] Hartman, J. and Wernecke, J. *The VRML 2.0 Handbook*. Addison-Wesley, 1996.

[7] Hughes, M., Shoffner, M., and Hamner, D. *JAVA Network Programming*. Manning, 1999.

[8] ISO/IEC 14772-1:1997. *Information technology - Computer graphics and image processing - The Virtual Reality Modeling Language. Part 1: Functional specification and UTF-8 encoding*. ISO/IEC, 1997.

[9] ISO/IEC 19775:2004. *Information technology - Computer graphics and image processing - Extensible 3D (X3D).* ISO/IEC, 2004.

[10] Qingping, L. et al. *A Novel Method for Supporting Collaborative Interaction Management in Web-based CVE*. Proceedings of the ACM symposium on virtual reality software and technology, 2003.

[11] SLIM-VRT (IST-2001-33184) EU-funded research project. http://www.gti.ssr.upm.es./~slim/.

[12] Wolff, R., Roberts, D.J., and Otto, O. *A study of Event Traffic During the Shared Manipulation of Objects within a Collaborative Virtual Environment*. The MIT Press. Presence, Vol. 13, No. 3, June 2004.