Distributed Collaborative Environment Based on Web3D

Marek Kováč
Faculty of Informatics and Information Technologies, STU Bratislava

Abstract

This work introduces collaborative virtual environment developed on Java and Web3D platform. The aim of the project was to develop universal and extendable virtual environment, using simulation core and a set of plug-ins. Paper presents the developed architecture for wide usage in collaborative simulation and training. Also, several example applications are provided at the end o stress the usability of system.
Keywords: Distributed collaborative environment, VRML, Java, e-learning

1 Introduction

An idea of interaction with three-dimensional scene or more sophisticated virtual world is as old as first computers capable of modeling and rendering 3D objects. Many design, simulation, engineering and visualization activities would be much harder to achieve without on-screen representation of real or imaginary world in any context. Making these worlds interactive - with feedback to users and various ways how users can mutually communicate - creates foundation of the virtual reality (VR). Multi-user collaborative and simulation environments introduce new techniques for a collaborative task accomplishment and work.

Recent progress in hardware technology makes tools necessary for these environments much more affordable than few decades ago. This is a reason why the spectrum of applications and potential users of these software tools grows.

On-line collaboration can be employed in wide problem domain including training and learning applications, collaborative design, reconstruction of historical artifacts, events, crimes and many applications of telework and remote assistance.

This paper describes a system for various collaborative applications and its capabilities are demonstrated by examples of architectural and urbanistic design, further extended by e-learning applications (laboratory experiments and simulations). The description will mostly be aimed to techniques and architecture used for development. The key idea of this project was to design and develop an universal collaborative environment. It has no particular support for any kind of applications to be used, as it will be further clarified. On the other hand, system offers much extendability.

The product is built on Java platform, using VRML language for a visualization. While it may seem that Java is not suitable for applications with extensive computation, lower Java performance compared with compiled source code languages (C++ for example) is not a major bottleneck of the solution, as long as it is not used in a large-scale simulation. Visualization with VRML (Virtual Reality Modeling Language) browser was proved to be an easy way of creating 3D and interactive content which may be modified and used without sophisticated and expensive modeling tools. Moreover, this content can be controlled by EAI (External Authoring Interface) interface [1]. On the other hand, VRML can not be practically used for contemporary 3D graphics commonly presented in other applications which make direct use of API. Such approach can utilize hardware capabilities to offer the better performance, while VRML processing is constrained by the specification. For this project XJ3D browser[1] has been used. This browser is intended to be a tool capable of browsing X3D scenes, compatible with VRML specification. X3D is a relatively new standard and will become a replacement of VRML language. X3D is specified by ISO/IEC 19775:200x [2]. XJ3D browser is written in Java and can be easily integrated into Java application as user interface component.

2 Related work

One of the first distributed environments allowing multi-user cooperation was SIMNET [3]. Application was designed for military simulations. Later, this project was developed into NPSNET. This environment was compatible with DIS (Distributed Interactive Simulation) protocol specification. DIS protocol is also implemented in Java, in a project called DIS-Java-VRML [3]. Usage of DIS-Java-VRML has been explored. As this environment used security flaws for multicast communication from internet browser applet, it is not fully functional with latest versions of Java Runtime Environment. The causes of malfunctioning and general analysis of problems with usage of applet-integrated VRML browser are further analyzed in [4].

[1] available at www.web3d.org/xj3d

For the purpose of preserving data consistency on all nodes, all the distributed virtual environments systems utilize different communication and replication methods. There are various methods of virtual world distribution among the nodes. Aside from mentioned environments which served for simulation purposes, environments for collaborative work appeared recently. These support multi-user interactivity and cooperation. One of them, closely related to our project is M3D [5] which is used for a collaborative design. Also, many case-oriented systems appeared. These are generally harder to extend, but offer optimal handling in means of replication and response.
With the quantity of distributed virtual environments developed recently, methods of preserving data consistency and simulated objects distribution appeared. These methods are not always straightforward. According to [5], there are three methods for distributing objects between network nodes of distributed virtual environment.

- *active replication* – all objects are copied to existing sites and method will guarantee resulting state independent of site and event order.
- *replication on demand* – only those sites, which affect the object will get a copy of it and all other sites will receive the update.
- *migration* – objects are distributed but not duplicated. Distribution algorithm may involve load balancing. The only information which is duplicated may be for example object visual representation, which can not be transferred to all sites effectively.

Depending on the distribution method used, there are several approaches for preserving data consistency and multiple access to objects by more than one process at time. Basically, the consistency approach may be strict, and does not allow unresolved conflicts to occur, or relaxed one. Relaxed approach allows local consistency conflicts occurrence, but reduces the cost of communication.
Multiple access to objects may be considered as the most significant source of consistency problems. There are many methods to ensure correct multiple access. These methods vary from centralized processing, and exclusive object locks to reversible update execution [5].
Communication topology and protocol is another key issue to be solved with a respect to scalability and responsiveness of environment. Basically, it can be considered as a tradeoff between these parameters of designed collaborative virtual environment. While a point-to-point communication can be suitable for small applications, performance may degrade significantly when a number of nodes involved in simulation grows. Multicast and one-to-many approaches are more suitable for the large scaled experiments, but the consistency model performance may be inhibited by unreliable multicast protocols. In [6], partial solutions involving a selective reliability of transferred messages is presented. Selective reliability is however not always suitable, as it has tightly related to a behavior models of simulated objects in virtual environment.
As most universal topology for virtual environments, client-server architecture may be considered, unless there are no requirements for multiple message round trips while processing a single message. This approach enforces centralization of some system components on the server.
Such approach enables using the active replication, because event order and updates construction can be done by the server. Consistency control is strict. It uses exclusive object locks to prevent inconsistencies.

3 System architecture

Developed system described in this paper is basically, a multi-user client-server application. Clients are nodes in which users can interact with a scene, and parts of world are simulated. Object updates are distributed to the nodes, which contain replicated simulation objects. These passively accept parameter updates and collect the input from user. The server application serves as a model data provider. All available objects for the simulation are stored and distributed by the server application. Therefore, server application provides an access to object gallery, which serves as data repository for clients to download model and graphical data.

3.1 Model and scene separation

Efforts have been made to separate the simulation and the model representation from its visualization. VRML offers ways to use scripting or programming structures directly inside the VRML scene. These mechanisms are called EAI (External Authoring Interface, providing an access from outside browser) and SAI (Script authoring interface, providing the access to objects inside the scene as a part of a scene definition). These mechanisms are supported only for a limited number of languages, although VRML specification does not limit them. (namely, JavaScript for SAI and Java for both EAI and SAI). This approach makes impossible to use an alternative visual platform, what is a major disadvantage. Developed system uses rendering and input sensor capabilities of VRML only. Input and output events are further transferred by EAI used in the visualization layer. No direct processing of events inside scene nodes is performed. Therefore, the behavior of objects is defined outside the scene and VRML browser can be viewed as a visualization and input platform only. It reduces several interdependencies between the core and visualization.

3.2 Network communication

The communication is built on connection-based TCP/IP sockets, which guarantee message delivery and order. The major advantage is avoiding complex message delivery verification and ordering when compared to the UDP multicast. The only problem is a redundancy of transferred messages, as these must be distributed by server for every existing connection. This creates otherwise unnecessary network bandwidth consumption.
Connection based communication is suitable for transferring static model data, including the VRML models and textures, where bandwidth is not as critical as in simulation messages exchange. These data can be cached on local client node, provided that the application verifies whether there are not new versions in server gallery.

3.3 World distribution

Server application contains critical synchronization information and runs implementations of simulation objects. The application logic of these objects is executed on the server. Client nodes contain only proxy objects with current parameter values. Client nodes can contain additional application logic not directly related to the objects in simulation, which is realized by plug-ins. These will be described later. Client nodes are responsible for sending all updates of objects' state performed locally to the server. Server then redistributes updates to other nodes.

Although this is a simplest distribution which could be used, it solves majority of problems with consistency.

3.4 Data persistence

It is possible to store current state of selected objects or whole virtual world for later use. This enables users to continue with once started experiments even after the simulation ends and model is discarded. Also, this approach allows users to store and later reuse parts of model within the same scene. For instance, if there is a scene with building architecture with recurrent components, these can be stored and copied to make the composition of the building. When node is connected to the simulation which is already running, it obtains current state of the simulation model and after setting up a local copy, it receives updates occurred meanwhile.

3.5 Model composition

System was designed to be open for any extensions concerning kinds of objects that can be a part of the simulated model. These objects are called entities. Every type of entity is defined by its XML description, implementation class and if needed, geometrical and visual representation written in VRML. Implementation class is a class written in Java language. This class implements a behavioral logic of the entity and has direct methods for setting and retrieving object parameter values. The XML definition contains information about entity implementation class, complete parameter list, list of parameters which affect the visual representation. Such definition is called *type template.* Instantiation with parameter values creates an instance, which can be a part of model, or in serialized form, part of object gallery.
VRML definition of entity appearance is not compulsory, there may be non-visible entities in model. VRML object is written as a PROTO definition (which may be repeated in the scene) and an instantiated node with name, which is composed from the entity identification. These nodes are inserted and removed dynamically during the execution of the simulation, using EAI API. As the standard EAI specification does not allow direct node removal, removing node is performed as re-composition of world without omitted node, what may be sometimes ineffective.
When the entity instance is created on the node, object creation request is sent to the server. After an object has been successfully loaded, new object information is spread to active nodes, which download graphical content on demand (if local cache does not contain current copy). Implementation class instance is loaded and initialized.
As the entities of simulated model are completely standalone objects, some mechanism for communication between them is needed. Entities should not be aware of other objects existence. Therefore, a mechanism of parameter links has been developed. Parameter link is a logical connection between value of parameter of source entity and value of parameter of destination entity. Parameters must be of the same type. When a value of source changes, the destination parameter is updated to a new value. This concept was borrowed from VRML ROUTE statement [1]. Basically, it allows an interaction between objects without being aware of it. Parameter links are unidirectional.

3.6 Consistency control

As more than one user can manipulate an object, mechanism for securing consistency in object state is required. It is enforced by an exclusive lock granted by the server to client node which changes the state of object. Lock can be granted when an attempt for parameter modification of the object is made and no other client node is the owner of lock. After user stops manipulating the object, after certain amount of time, lock is released and server can grant it to another or the same node. Although this approach may appear restricting, it is generally possible to build the

simulation object in such manner that it does not directly disturb the usage inside simulation.

3.7 Plug-in architecture

For better extendibility, architecture of core and plug-in has been used. Core described in above paragraphs contains basic functionality for distributed collaborative environment without any support for specific kind of applications. Support for applications is realized through plug-ins. These are application modules that operate inside the server and client application. The server and client plug-in module communicates through the standard client connection by data which are not interpreted by the core, but dispatched to plug-ins. Plug-ins communicate with application through a standardized interface of client and server application which is called *context object*.
Plug-ins provide various functionality, for example, dynamically generated user interface used for modifying parameters otherwise inaccessible from visual scene, specialized geometry generators, refresh of parameters, or bounding box selection (for the architectural studies).
The mentioned plug-ins are loaded on demand as specified in object type template. Also, with the specification of a loaded plug-in, context data are contained in type template, they may be used for the initialization of plug-in at server or client side.

4 Applications

In this part, example applications and their description will be shown.

4.1 Urbanistic study example

Following example demonstrates usage of the virtual environment in a collaborative decision concerning placement of railroad segment through residential zone. Model contains prepared scene with buildings and rail segments can be inserted. All objects can be positioned, added or removed from scene. Multiple participants can gradually go to the decision, where to lay the track.
Application uses selection box plug-in to allow group modification of transformations of the objects. There are two modes in which the selection may be used. It can behave as a simple block which is scaled, moved or transformation changes can be applied to all selected objects separately. Figure 4.1 shows the window of client application.

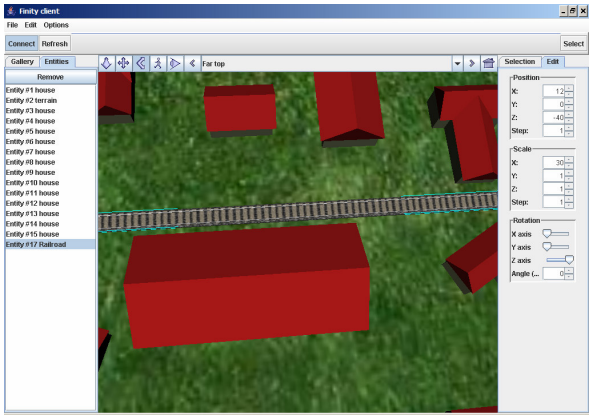

Figure 4.1: Laying track railway segment

4.2 Surface wave interference

In this application, client-side generated geometry of water surface is disturbed by parametrized wave sources controllable by the experiment participants. Surface is visualized and allows participants to observe the wave interference effects. It is possible to enable and disable wave sources. Due to maintainable performance, water surface is a matrix of 32x32 vertices, where only elevation of vertex is being recalculated.
Wave source has an adjustable frequency and amplitude, moreover its position on the water surface can be further modified. System allows modifications of several objects at one time.
Phase of waves is not synchronized on client nodes, as no transition effects are simulated. Any wave source can be accessed by a participant. Wave sources are distinguished by colors. Situation is displayed on figure 4.2.
Water surface object has parameters of four wave sources linked for immediate reflection of source state change.

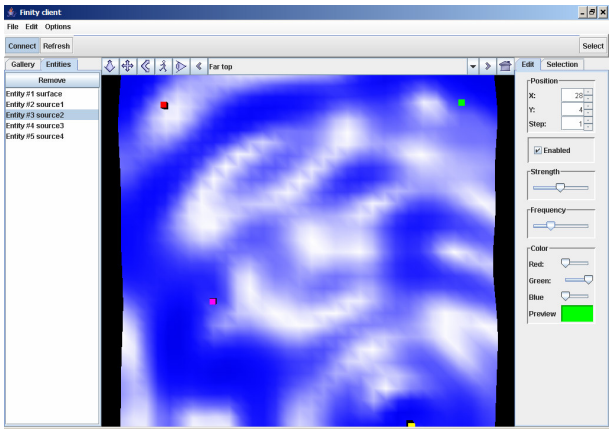

Figure 4.2: Wave interference experiment

4.3 Resistance characteristics

The experiment allows students to measure a characteristics of electrical resistor. Scene contains adjustable voltage source, ampermeter and resistor with parametrized resistance curve. An instructor can set up this characteristics to allow the measurement by students. Virtual electrical devices have selectable range. Situation is demonstrated on figure 4.3.

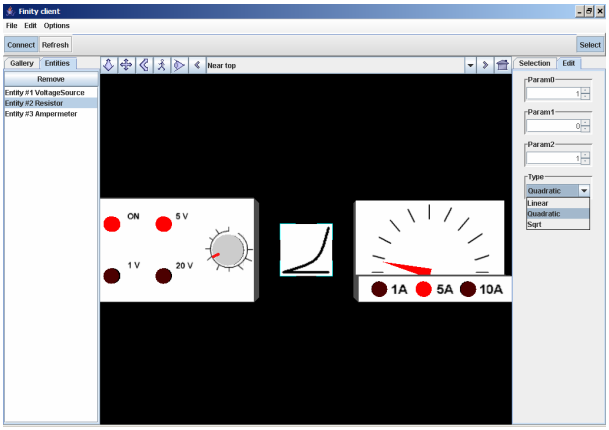

Figure 4.3: Measurement of resistance characteristics

4.4 Sail boat simulator

This application allows basic training in sail boat navigation. The task of instructor is to control a destination of navigation, wind strength and direction. The student uses rudder and sail displacement for navigation towards the destination. Navigation data can be viewed at compass which is situated in the scene together with sail ship model. Wind, ship and destination directions are displayed. Ship rotates around vertical axis but stays fixed at the position in scene. Latitude and longitude is calculated internally. Situation is demonstrated on figure 4.4.

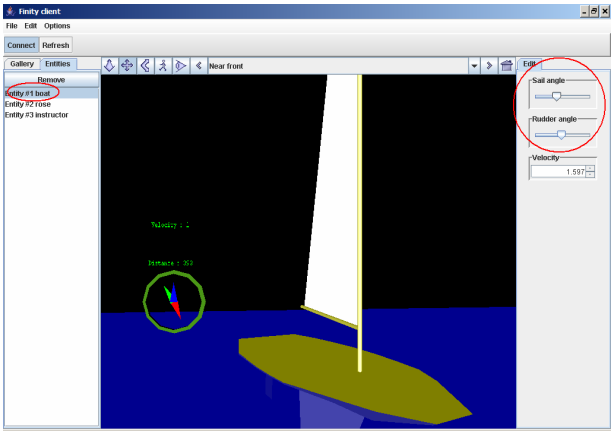

Figure 4.4: Sail ship simulator, student's view

5 Conclusion

Although majority of collaborative virtual environments are implemented on native platform, as performance can be a major issue, this project has its greatest advantage in platform independence. Integrated browser utilizes Java3D or OpenGL rendering engine. These are implemented on various platforms as well as runtime environment for Java. However, it has to be mentioned, that the performance against native rendering is lower by an order. Also, the attractiveness of content is limited by use of the low-detail modelling.

Efforts have been made to make the system as generic as possible. This means an abandonment of various optimizing techniques used in single-purpose environments. Specific features of system are implemented as extensions (implementation classes of entity types, plug-ins), using the common simulation core and communication technique.

Although there are no performance bottlenecks in the extent the system was used, using Java3D rendering engine causes considerable initialization delays and memory expenses.

Scalability of the system may be impaired by extendability as large-scale optimizing techniques are not used. In installation of about twenty network nodes with relatively simple scene degrading performance can be observed due to lack of any optimizing techniques.

6 Further work

As the development of this system has been finished as a product of master thesis, following suggestions outline problems, which basically are missing features in the system.

Currently, any participant has unlimited access to any object. This could be a problem when roles of participants are distinguished in the application. The problem in it's basic form shows up when using environment for laboratory experiment simulation with involvement of instructor, where basically any student can perform his task. Also, for tele-presence applications it may be not desirable to allow any operation of any participant over any object. Therefore, it would be appropriate to extend the system with participant role. This mechanism could be based on defining the access profiles for entity contained in the model and then associating access profiles with identified users.

Recently, new standard specification which is a successor of VRML97 has been published by Web3D consortium. This standard is called X3D and currently used browser supports both of these standards. Transition to X3D involves re-implementing of layer connecting simulation model and VRML EAI interface. Visual representations of entities need to be

rewritten into X3D. In time, when visualization layer was implemented, there were no documents describing X3D SAI (scene authoring interface) available. Also, there are not many model authoring tools for X3D yet. Such step could improve accessibility of scene content and possibly quality of graphic design. The only problem is that there can be less X3D modeling tools than for VRML.
Although system is highly extendable, when adding new functionality with implementation classes or plug-ins, knowledge of Java language is required. This may be unacceptable for model designer. Therefore, usage of scripting language could serve this purpose. As interfaces for communication with system core are defined, upon having access to Java calls from script, scripting language could be used. As a good example Python for Java would suit this purpose well.

7 References

[1] VRML97 Functional specification and VRML97 External Authoring Intergace (EAI) International standard Specification, ISO / IEC 14772-1:1997
at http://tecfa.unige.ch/guides/vrml/vrml97/spec/

[2] Extensible 3D (X3D) ISO / IEC FDIS 19775:200x
http://www.web3d.org/x3d/specifications/ISO-IEC-19775-FDIS-X3dAbstractSpecification/index.html

[3] Brutzman Don: DIS-Java-VRML.
http://web.nps.navy.mil/~brutzman/vttp/dis-java-vrml

[4] Horváth, Ľuboš: Viacpoužívateľské kooperatívna prostredie pre prácu s VRML modelom. Diploma work. FIIT STU Bratislava 2003.

[5] Galli R., Data Consistency Methods for Collaborative 3D Editing, Palma de Mallorca: Universitat de les Illes Balears, 2000. 231 pages. Dissertation work.

[6] Shirmohammadi, S: Collaborating in 3D Virtual Environments: A Synchronous Architecture. Ottawa: University of Ottawa.

[7] Watt A., Policarpo F. 3D Games Real-time Rendering and Software Technology. Addison-Wesley, 2000. 800 pages. ISBN 0-201-61921-0