Reactive Virtual Environment System:
LivingWorlds server and VRML browser

Ryo Yoshida*
Tokyo Research Laboratory, IBM Japan Ltd.

Abstract

We describe a VRML browser-and-server system that we developed to support the LivingWorlds specification for multi-user environments. Since the browser supports VRML version 2.0 with quality-of-service (QoS) control, the LivingWorlds module was implemented by using the VRML 2.0 Java application interfaces. We discuss the concept of designing systems with QoS control, and the problems encountered and solutions reached in actually implementing a LivingWorlds client-server system.

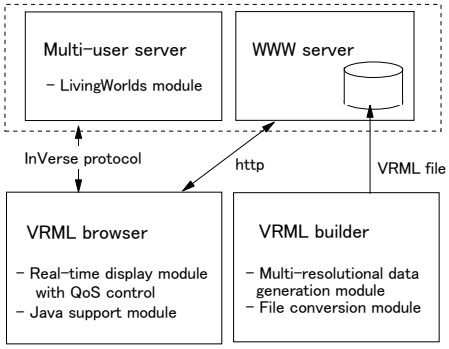

Figure 1: RVE system overview

1 Introduction

Several VRML browsers have been developed over the past year, during the formative period of the VRML 2.0 specification [5]. One is a plug-in type of HTML browser, another is a helper application of the first, and another is a non-VRML browser that can load and render VRML contents. Recently, VRML models have been created for various areas such as entertainment, manufacturing, and telecommunication, and thus the interests of people involved in VRML are moving in various directions, one of which is multi-user shared-state environments. Several research groups have created experimental distributed VR systems for multiple users. Their current goal is to support large numbers of participants simultaneously connected to a network. In a multi-user environment, a user can join a discussion in a virtual meeting room, for example, go shopping in a virtual mall via an "avatar," which is the user's agent in the virtual world.

Several working groups (WGs) have been formed to discuss about various themes in line with the interests of participants. Groups working actively on the multi-user environments, include the Universal Avatar WG, the Humanoid Animation WG, and the LivingWorlds WG. We are involved mainly with the LivingWorlds WG, in which the most important activity is lower (system-level) technical discussion of topics including file syntax specification [4] based on the concept of the shared-state space and object model.

Our project started before the process of standardizing the VRML 2.0 specification got under way. Therefore, the browser we developed, although currently a VRML 2.0 browser, can also load files with a different file syntax that we defined. The system, called Reactive Virtual Environments (RVE), consists of the VRML browser and two additional modules, a LivingWorlds multi-user server and a VRML file builder. Figure 1 shows an overview of the RVE system.

In this paper, we restrict our discussion to multi-user systems (browser and server). In section 2, we give an overview of system architectures and related work. In section 3, we introduce our VRML browser and server, and de-

*1623-14 Shimotsuruma, Yamato-shi, Kanagawa, 242 Japan, e-mail:yoshidar@trl.ibm.co.jp

scribe our implementation, which conforms to the LivingWorlds specification. Section 4 discusses the direction of our activity, and Section 5 is a summary of the paper.

2 Multi-user systems

2.1 System architectures

In general, there are three architectures for building systems to realize a shared-state virtual world. The first is a client-server model. This model makes it conceptually simple to build a shared-state world, and it is therefore used as a basis for multi-user systems. In this model, there is always only one reference world. The server maintains a master of all of the objects in the world, and the client basically copies the master from the server when it initially loads the world.

The second architecture is a peer-to-peer model in which all the clients maintain the same world among themselves, instead of copying the master from the server. The reference world is maintained by each of the clients.

The third architecture is a hybrid of the above two models, using both the server-client model and the peer-to-peer model. The systems in this category have a wide variety of implementations; for example, the client may have a replica of the world and maintain it for reference, although the server also has the master world.

Each model has strong and weak points. We use the client-server model for our RVE system, since it generally minimizes the required network bandwidth and computational power.

2.2 Related Work

There has been considerable research on multi-user shared-state environments. Sony's Community Place [2] is one system that supports to construction of a multi-user environment, using a client-server model. When the user changes the state of an avatar on a certain client, information on the avatar's position and orientation is sent to the server, and the server then dispatches that information to the chosen clients, using their algorithms for selecting the spatial areas of interest. The protocol linking server and clients is a proprietary Virtual Society Client Protocol. To build a VRML shared-state scene, the user has to write it with the Sony's proprietary defined nodes, and therefore the scene builder has to rewrite it when porting it to another system.

Another example of a multi-user system is MERL's SPLINE [1], which is positioned as a platform for shared-state environments. And world named "Diamond Park" was built by using SPLINE. The Open Community specification has the API, which is available with the SPLINE module and has comprehensive interfaces between the network layer and the world model layer. The scene generated by the system seems to be a multi-user shared-state world, but not a VRML world. If an implementor wants to develop a new browser or a new server using the Open Community API, he/she might be need to use the SPLINE's modules, since in practice it is not an open specification.

3 RVE architecture

The RVE system follows a client-server model. We developed the browser partly in C/C++ to display the scene-graphs, and partly in Java to support the Java scripting API and to communicate with the server. The server was developed only in Java.

Our project has two principal objectives (1) to develop a VRML browser that can process interactions with users without serious delay and can also display large amounts of data in applications such as CAD geometry, and (2) to develop a client-server system that can share the state of a world visible to many users simultaneously across a network. Recently, multi-user environments, including systems conforming to the LivingWorlds specification, have been built with the low-cost home computers, whereas in the past, distributed VR systems usually included many very high-performance and expensive computers. It is thus now possible to form a multi-user environment by using computers and connections with a wider variety of performance levels, such as low-cost PCs without any graphics accelerator boards connected to a network by modems, and high-performance graphics workstation connected via LANs. To realize a situation in which every user can see the same world almost simultaneously in such a heterogeneous environment, the client needs to coordinate its own load to be able to reflect the results of user interactions by updating the display screen of a world at an appropriate frame rate. Other-

wise, a user on a low-performance client PC will always have a delayed view of events and will miss some occurrences in the world, since they will be reflected to the client's display screen with a delay. To solve this problem, we applied the quality-of-server (QoS) control technology to the VRML browser. The basic concepts of the technology are as follows:

- According to the performance of the machine, the appropriate level of detail is selected and the display frame rate for updating the scene is coordinated to reflect the changes in scene promptly without causing the frame rate to fall during display. The load of rendering the scene in the next frame is estimated from the load of the previous frame.

- For low-bandwidth networks, redundant event information transmitted from the server to clients is eliminated to reduce the network traffic.

For example, in the case of a low-performance PC connected to a network by a modem, a smaller LOD set of VRML data, that is, the coarser three levels from the five LOD data is selected and sent to the client when it initially loads a world. At the time, the browser itself notifies the server of the performance level for the data set selection. Once the browser starts rendering of scene, when generated by dragging a mouse by the user, a series of the events is able to partially be culled. Thanks to the technology, even a low-performance PC user can join a multi-user (i.e. heterogeneous performance) environment without viewing the delayed scene. Figure 2 shows the concept of the RVE QoS control.

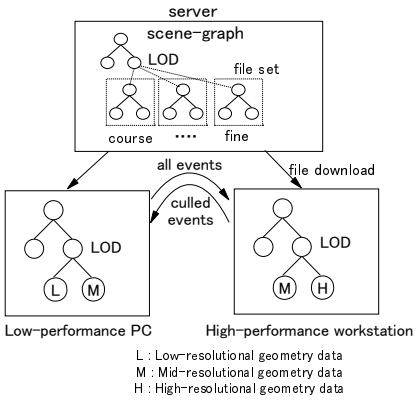

Figure 2: QoS-able system

3.1 RVE browser

The RVE browser supports the VRML 2.0 specification; that is, it can load a scene-graph containing the VRML standard nodes, prototyping, and ROUTE. As the API of the Script node, the browser provides the Java Standard Application Interface (JSAI) by Java version 1.1.

By loading the initial VRML world, our browser works as a helper application of a HTML browser. The current version of the RVE browser is not a plug-in module of any HTML browser, because the system we developed is an experimental one for applications throughout a network, and we do not want to be bound by any of the available security policies we can see for the Java virtual machine of a WWW browser. When the system reaches a securer level, we might have to apply something like those security policies.

When the user does not enter a world that includes a shared-state space, the browser functions simply as a stand-alone VRML 2.0 browser. In particular, it enables the user to see VRML objects in front of a background panoramic image generated by the PanoramIX technology, if the input file contains a Background node with the PanoramIX control file name. The technology for generating such panorama images is available on the Web [3].

In the current implementation, once the user enters the shared-state space, the browser contacts the server via a proprietary protocol (the InVerse protocol) that works on the IP. The LivingWorlds module runs between the communication layer and the VRML Java API layer. Figure 3 shows the client architecture.

3.2 RVE server

The RVE server supports the LivingWorlds specification. The function for communication between clients and server is provided by InVerse. InVerse was developed by IBM Software Solution (Raleigh). Its communication infrastructure has two components: one controls the sender and receiver of the shared object's information, which changes every moment, while the other controls the registration and management of the clients. The LivingWorlds module functions as a shared-state object holder on

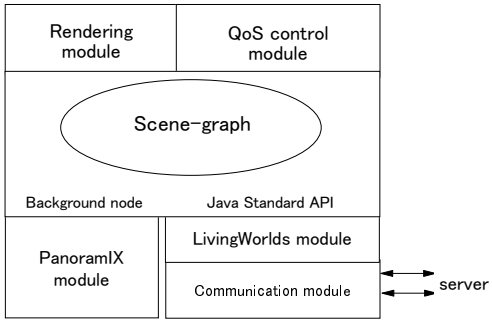

Figure 3: RVE client

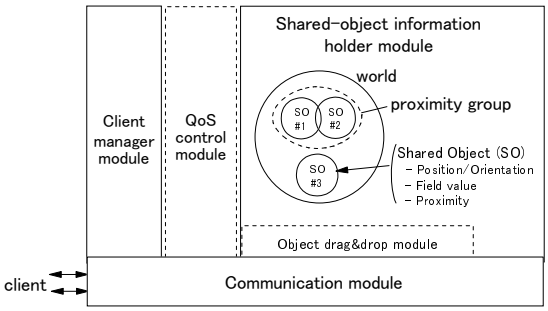

Figure 4: RVE server

the communication infrastructure. The server maintains the information about the position and orientation, as well as the specified field values of shared objects routed to the LivingWorlds AssociativeStringArray. When the user enters the shared-stated world or moves around in the world, the browser sends the position and orientation information to the server, and the server then distributes the information to all or some of the clients. Since the system uses the concept of a proximity group of avatars, the information is sent to the chosen clients on which user's avatars are located in neighboring in the world. The server can currently maintain a single scene; however, if the user wants to create two independent worlds, he/she needs to run two servers to maintain these respective worlds. Figure 4 shows the server architecture.

In the client-server system we designed, the server maintains one complete and independent

```
#VRML V2.0 utf8

EXTERNPROTO Zone [
  field    SFNode private
    .....
] "Zone.wrl"

EXTERNPROTO PrivateZone [
  field        SFNode zone
  field        SFNode muTech
  exposedField MFNode children
    .....
] "PrivateZone.wrl"

EXTERNPROTO IBMZone [
  field SFString nameServer
  field SFNode   zone
  field SFNode   privateZone
    .....
] "IBMZone.wrl"

DEF Z Zone {
  private DEF PZ PrivateZone {
    muTech DEF MZ IBMZone {
      nameServer "lwsvr.trl.ibm.com"
      zone USE Z
      privateZone USE PZ
    }
    zone USE Z
    children [ ..... ]
  }
}
  .....
```

Figure 5: VRML file incorporating LivingWorlds nodes.

scene including one Zone, and a client can connect to multiple servers. Each server manages the information of every client on which users see the world maintained by the server. The client loads several independent scenes from the respective servers, then displays them as a world; that is, the client can be managed by the multiple servers to which it is connected.

3.3 LivingWorlds module

Figure 5 shows an example of a VRML file for the shared-state world, which was written for the RVE system. All the nodes defined in the LivingWorlds specification are written in PROTO/EXTERNPROTO, and several VRML standard nodes, including the Script node working with Java program. Each Java file is required to be compiled into a class file that includes invocations of the Java Standard Application Interfaces (JSAI) instead of the External Application Interfaces (EAI). Here, we note that

nodes related to the "lock" have not been implemented, because to implement its function we may need to define what should be protected, whom we have to protect it from, and how we should protect it. According to the LivingWorlds concept, the function of Zone, PrivateZone, and IBMZone is to represent the shared-state space. The function of SharedObject, PrivateSharedObject, and IBMSharedObject is to represent the avatar that we can see as an agent of the user in a virtual world. The function of Selector, SelectorItem, all converters, and AssociativeStringArray is to represent the avatar's behavior. They enable the user to select a behavior, from a menu generated by Selector, that he/she wants to make his/her avatar act out. AssociativeStringArray then holds the information about which behavior the user selected, and distributes it throughout the network. The details of the procedure are stated in the LivingWorlds specification.

For us, there were two issues involved in implementing LivingWorlds modules by using the RVE client-server system. One is prototyping with the script node, and the other is processing events between nodes. Moreover, there is one important issue involved in authoring a world.

The prototyping problem is caused by the syntax of VRML, on which the exposedField declaration is not allowed for the Script node. In prototyping the various LivingWorlds nodes, we solved the problem by applying PROTO, which has no prototype definition. In the LivingWorlds nodes, those applied node are the one whose PROTOs have an exposedField VRML field-type, such as SharedObject, PrivateSharedObject, or Selector. For instance, the prototyping of Selector has a selectorItems field, and therefore this field was divided into two fields, a selectorItems field and a set_selectorItems eventIn field, for the Script node.

The second issue concerns the VRML event flow. The problem is caused by the event cascade processing when two events have the same time-stamp. For instance, suppose that an event is transmitted between PrivateSharedObject and SmoothMover. The event sent to PrivateSharedObject from the network is forwarded to SmoothMover, which then returns the value to PrivateSharedObject. The value is calculated on the basis of value input to SmoothMover. Here, to be exact, the ROUTE of the event is not the loop defined in the VRML specification; however, the PrivateSharedObject

```
#VRML V2.0 utf8

PROTO Selector [
  exposedField MFNode selectorItems []
  eventIn      SFBool enabled
  field        SFBool isTearOff FALSE
  eventIn      MFNode addItems
  eventIn      MFNode removeItems
] -
  DEF G Group - "
  PROTO ExposedFields [
    exposedField MFNode selectorItems []
  ] - "
  DEF EF ExposedFields -
    selectorItems IS selectorItems
  "
  Script -
    field SFBool isTearOff IS isTearOff
    eventIn MFNode set_selectorItems
                   IS set_selectorItems
    eventIn SFBool enabled IS enabled
    eventIn MFNode addItems  IS addItems
    eventIn MFNode removeItems IS removeItems
    field   SFNode exposedfields USE EF
    field   SFNode group USE G
    ......
  "
"
```

Figure 6: PROTO of Selector

node has to be evaluated twice to end the entire processing of an event cascade routed to the node, even though every event has the same time-stamp (Figure 7). To solve this problem, in our browser, while events with the same time-stamp are evaluated at the same time principally, they are evaluated one by one if the VRML file includes a loop that needs to be evaluated twice.

The last issue is related to AssociativeStringArray. In the current specification, it is generally difficult for us to set two val-

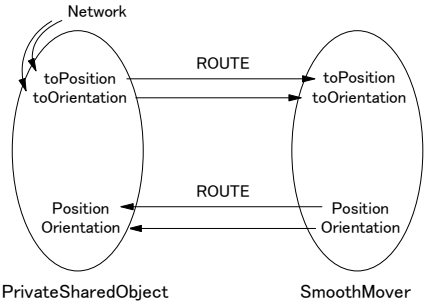

Figure 7: Event loop between two nodes

```
EXTERNPROTO SFNodeToASA [
  exposedField SFNode shared   NULL
  field        SFString tag    ""
  field        SFString access ""
     # "Pilot","Anyone", or GroupID
  eventOut     MFString output
] "urn:..."

EXTERNPROTO ASAToSFNode [
  eventIn      MFString input
  field        SFString tag    ""
  field        SFString access ""
     # "Pilot","Anyone", or GroupID
  exposedField SFNode shared   NULL
] "urn:..."
```

Figure 8: Proposed prototype

ues to the same AssociativeStringArray simultaneously, since this causes the fan-in problem. Without a solution, a user is not allowed to create a scene in which the shoulder joint and elbow joint of an avatar are simultaneously bent at arbitrary angles. We proposed prototypes of SFNodeToASA and ASAToSFNode with an owner field named "access" for access control. With this syntax, a node described as a value of the SFNode "shared" field is considered as a shared node; that is, all the field values of this node are conceptually the same for all the clients referring to the scene. These prototyped nodes are used in the same way as the converters (SFxxxxToASA and ASAToSFxxxx) described in the LivingWorlds documentation. This description does not cause the fan-in problem, since the field value of this prototyped node may not be frequently changed. However, there is a concern that an unnecessary field that a world author does not intend to share is always shared as a result of the description for each node, so it may be necessary to consider using a different syntax.

4 Future Directions

We are investigating applications in the manufacturing and distribution industries, while of course the entertainment area is still one of the important areas for multi-user environments. For this purpose, the items we are currently working on are: (a) a model for processing events across the network for multi-user environments, (b) function extensions including file syntax and implementation for the multiple zones and the multiple servers, and (c) performance improvements for the browser and server.

The first item is related to both the system architecture and the LivingWorlds architecture. To reduce the network traffic, it is possible to notify other clients only of the target events that originally caused event cascades, while ignoring the other events transmitted in ROUTE. The former are generated at VRML sensor nodes and script nodes. As it is, the MUTechSharedObject requires the events to be transmitted by ROUTE according to the LivingWorlds architecture. We are working to put each architecture to its proper use in our system.

As regards the second item, function extensions we are adding several functions to make it possible to:

- Restrict accesses for changing the values of fields. This makes it possible to divide users into some groups that have different access authorities, so that group "A", for example, can change any fields of any nodes, but group "B" can change the fields of only some nodes.

- Transfer multi-resolutional data for LOD from the original server to the destination one. This makes it possible to transfer objects from one zone to another.

As explained in the previous section, the client displays several scenes downloaded from each server as a world. Each scene received from the servers generates a complete scene itself, so if the author designs the scenes well, they can be connected to give the appearance of a single world, such as a shopping mall with many shops. If the author has access control to change a field's value, he/she can build a world in which the shop owner can move an article for sale to another place, but a buyer in the shop can not take it away. If the article is moved from one zone to another, the LOD data set of the article must to be copied to the server maintaining another scene. This is because, when the article is introduced into the scene, it needs to be displayed on the screen of every user looking at the same scene in a heterogeneous environment. As a prototyped node for connecting scenes, we are considering the following:

```
EXTERNPROTO ConnectedPoint [
    field SFString url         ""
    field SFString rootName    ""
```

```
    field SFString nodeName    ""
    field SFString worldId     ""
    field SFVec3f connectPoint 0 0 0
}
```

If this type of node exists as a leaf node in two independent scene graphs, the browser interprets the node as one instance jointly owned by two graphs, that is, as one merged scene graph composed of two scene graphs. In this representation, the strict relation between two spaces is not defined, but the representation would be useful as a hyper-link in HTML, for example, the remaining definitions could be left to the world author.

As regards the last point, we need to improve the performance to display very large amounts of data, such as complex CAD geometries or bone structures captured from CT scans, with user interactions.

5 Conclusion

We have described the browser and server of the RVE system with the LivingWorlds implementation. The design of our system architecture is based on the concept of quality-of-service control in VR multi-user network environments. Problems in implementing the browser following the LivingWorlds specification were also described. We have completed the first stage of RVE system development, which is the implementation of the LivingWorlds client-server, and are now working on the next stage, which is the addition of functions for QoS control, access control, and object drag-and-drop function using multi-servers.

Multi-user environments will give all VRML users and all Internet users a chance to communicate expressively through an intuitive user-interface. We hope that such environments will accelerate the evolution of VRML.

References

[1] D. Anderson et al., "Open Community Overview", *http://www.merl.com/opencom/opencom.htm,* 1996.

[2] R. Lea et al., "Community Place: Architecture and Performance," *Proc. Second Symposium on the Virtual Readlity Modeling Language,* pp. 41-50, 1996.

[3] J. Lipscomb, "PanoramIX", *http://www.alphaworks.ibm.com/,* 1997.

[4] Mitra et al., "Living Worlds," *http://www.livingworlds.com/draft_2/index.htm,* 1997.

[5] VRML Architecture Group, "VRML 2.0 Specification ISO/IEC DIS 14772," *http://www.vrml.org/Specifications/VRML97/DIS/index.html,* 1997.

January 14, 1998
RT0234
Telecommunications 7 pages

Research Report

Reactive Virtual Environment System: LivingWorlds server and VRML browser

Ryo Yoshida

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242, Japan

Limited Distribution Notice

This report has been submitted for publication outside of IBM and will be probably copyrighted if accepted. It has been issued as a Research Report for early dissemination of its contents. In view of the expected transfer of copyright to an outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or copies of the article legally obtained (for example, by payment of royalties).