Distributed Interoperable Virtual Environments

Felicio Vanderlei Deringi Jr.¹
Mario Massakuni Kubo¹
Antônio Carlos Sementille²,³
Simone Santos Casolli⁴
Claudio Kirner⁴

¹Grupo de Realidade Virtual – GRV
Departamento da Computação – DC
Universidade Federal de São Carlos – UFSCar
Caixa Postal 676 CEP 13.565–905
São Carlos – SP – Brasil
fderiggi@yahoo.com
mkubo@dc.ufscar.br

²Laboratório de Sistemas de Tempo Real – LSTR
Departamento da Computação – DC
Universidade Estadual Paulista – UNESP
Caixa Postal 473 CEP 17.033–360
Bauru – SP – Brasil
semente@bauru.unesp.br

³Instituto de Física de São Carlos – IFSC
Universidade de São Paulo – USP – São Carlos
Caixa Postal 369 CEP 13.560–970
São Carlos – SP – Brasil

⁴Faculdade de Informática – FI
Fundação Eurípides Soares da Rocha – FESR
Caixa Postal 2041 CEP 17.525–901
Marília – SP – Brasil
{casolli, ckirner}@fundanet.br

Abstract
This paper discusses the implementation of three types of network support based on distributed virtual environment communication models using World2World toolkit, CORBA platform, and its CORBA Event Service as well. These supports encompass an integrated communication environment on a network that is suitable for distributed virtual reality applications. Finally, performance analyses from such supports are presented.

Keywords: Virtual Reality, Distributed Virtual Environments, Multicasting, CORBA.

1. Introduction
Virtual Reality Systems (VRS) have been popular in the development of new ways of human–computer interaction, through the integration of the most diverse areas (Multimedia, Hypermedia, Computer Graphics, CSCW – Computer Supported Cooperative Work and Scientific Visualization, among others).
A VRS must foster the integration human/computer resources (hardware and software), using tridimensional effects, were users are able to interact, visualize and manipulate the virtual environment in real time, by means of multisensorial devices. Thus, the resources and implemented techniques in a VRS simulate suitable conditions, enabling, for example, the touch of objects in a virtual world with reactive sensations.

Most VRS applications are inherently distributed. Tanenbaum [Tanenbaum, 1995] defines the Distributed System as a collection of independent computers, which cause the system users to have the impression of using a sole computer. From VRS researchers’ point of view, the Distributed Virtual Environment (DVE) concept involves the execution of programs referring to a virtual world in several computers connected by a communication network. The system supports multiuser interactive environment, where users geographically dispersed can cooperate and share computing resources in the real time [Benford, 1994] [Kubo, 1997]. DVE are required to have [Deriggi, 1998] [Van Dam, 1993]: virtual environment rapid updating rates (depending on the application, the environment may be updated with rates that vary from 10 to 60 frames per second); maximum latency (delay of up to 100 ms – strongly coupled interactions and 300 ms for the remaining interactions); treatment of multiple input and output devices; simulation of a great number of objects, including those with complex behaviors; virtual world realistic representation (including multimedia and hypermedia integration); long distance data distribution; support to computing resources and heterogeneous networks; and scalability.

In this article, we first present some DVEs related aspects. Afterwards, we describe the architecture, information and analysis on the performance of communication supports developed by integrating WorldToolkit with World2World and the CORBA platform as well.

2. Distributed Virtual Environment Taxonomy

For the construction of the DVE network support, issues related to communication and structure utilized, such as bandwidth, protocols and communication modes must be analyzed.

Bandwidth consists of the total capacity of data transference by the network [Macedonia, 1997]. In the DVEs, the bandwidth defines the size and richness of the application. The traffic demand in the network tends to be small, since the images are locally generated – the network transmits in the majority of the times the position and orientation information of objects. Nevertheless, as the number of DVE users tends to be high – thousands – a high bandwidth is therefore necessary.

Communication protocols are a very important factor in DVE, because they define the communication mode among processes, implying reliability and velocity in the communication. The main protocols utilized in the DVEs are the IP on the network layer and the TCP or UDP in the transport layer. The IP provides a way of datagrams transport from the origin to their destiny, regardless of being these machines in the same or in other intermediate networks [Tanenbaum, 1996]. The TCP makes sure that the message reaches its destiny, while the UDP has no such guarantee. The choice of utilizing either, TCP or UDP will depend on application objective.

The basic communication structures are unicast, broadcast and multicast. In the unicast structure the data communication mechanism is known as “one-to-one”, being performed by the communication between only two hosts. Broadcast is the data communication mechanism known as “one-to-all”, where a host sends data to all the network hosts. Multicast is the data communication mechanism known as “one-to-many” or “many-to-many” in a unique operation. The multicast importance derives from the restriction of users number in a same class, through group formation to enlarge bandwidth, reduce latency and increase the flow on the networks [Hagsand, 1996].
A DVE may use different communication models to support data and processing distribution: the Client-Server model consists in the direct communication between client and server or vice-versa, with no message exchange among clients; the “Point-to-Point” model consists in the direct communication among hosts, without the intervention of one or more servers; the Distributed Object Computing model, consisting in client communication with the solicitant’s object service and services receiver, is mediated by an agent-broker; and finally, the Hybrid model consists of the combination between Client/Server and “Point-to-Point” models, as Figure 1 shows. According to Kirner [Kirner, 1996], such models present advantages and disadvantages for a DVE construction.

![Diagram of communication models](image)

**Figure 1 – A DVE’s Basic Models of Communication.**

Construction, rendering and interaction of a DVE example (Figure 2) use the software WorldToolkit (WTK) [SENSE8, 1997a]. WTK contains an object-oriented library with high level functions to configure, interact and control real time simulations. The example application becomes distributed with a network support incorporated to WTK. In this way, WTK makes calls to functions in the support for the communication with distributed clients.

Three network supports, which are presented in sections 3, 4 and 5, have been developed for the integration with WTK.
3. Network Support utilizing World2World

WTK presents a self-mechanism for the construction of distributed applications. This mechanism is based on the programming paradigm Object/Property/Event along with the network support: World2World (W2W) [SENSE8, 1997b]. W2W is a network solution based on the Client/Server model, and is responsible for communication management, users’ connection and disconnection notification and sharing of the involved data in a distributed simulation.

Paradigm Object/Property/Event based simulations are comprised of objects and properties which can be attributed to these objects. Changes in the property values of these objects generate events, which inform the other users of this modification.

In order to allow multiple users to participate on a same simulation, each user (client) needs to receive certain updating done by other participants. For example, supposing that there is a graphic object in the simulation which may be manipulated by all participants, if a user moves this object, all the other users will see this movement as well.

In this example, the affected property (object position) must be shared among the user, who is modifying the value of this property, and the remaining users of the simulation, who would like to receive this new value. Each change made in the value of a property is known as an event, which is internally
generated for each change in the value of a property. Whenever a property is shared, the events allow the updated information to be automatically sent through the network to any other client who also shares the same property. The specification of events manipulating functions to manage the actions to be taken in response to a determined event, is possible.

The mechanism, in which events are transmitted to all clients who are sharing a determined property, is furnished by network support W2W.

W2W is comprised of a Server Manager and Simulation Servers. The Server Manager is the initial point of communication for the user who initiates a distributed simulation. The Server Manager verifies whether the client has login rights and directs the client to the proper Simulation Server, based on the simulation which the client is running. The Simulation Server stores and organizes the simulation data and distributes data updating to the users connected to this one server.

In Figure 3, we may observe the relationship among clients, the Server Manager and the Simulation Servers:

![World2World Architecture](image)

According to the network support W2W, an application example would have the following code extracts:

First set of functions:

```c
conn = WTCconnection_new(host, port, username, NULL);
WTproperty_share(avatar, WTNODE_TRANSLATION, grp, 0);
WTproperty_share(avatar, WTNODE_ROTATION, grp, 0);
```

These functions initiate a communication with the Server Manager and share the position and orientation properties of the application avatar.

Second set of functions:

```c
WTproperty_new(avatar, "msg", WSTRING);
WTproperty_addhandler(avatar, "msg", msg_changed);
WTproperty_share(avatar, "msg", grp, 0);
WTproperty_sets(avatar, "msg", "Hello !!");
```
These functions create a new property to the avatar, designed as “msg”. This property will be used for the implementation of a “chat” window in the application. Afterwards, the “msg_changed” function for the introduction of updating events of “msg” property values is specified. Right after its creation, this property is shared and its value is altered to “Hello!!”, that is, by altering the “msg” property value, all the remaining clients, who are sharing this property, receive an event with the updating “Hello!!”. This way the sending of a message utilizing W2W is performed.

4. Network Support utilizing CORBA Platform

The utilization of CORBA platform as a network support for WTK applications promotes all the advantages of this platform to the application, such as: incorporation of object-orientation concepts, objects interoperability, clients have easy access, in a transparent way to local or remote object methods, besides the introduction of an Interface Definition Language (IDL) for the objects.

The use of IDL language for objects definition allows these characteristics in an element known by ORBs (Object Request Broker). The ORBs can be find from different manufacturers and written with different programming languages [Mowbray, 1995][Leavens, 1993][Schmidt, 1997]. ORB is responsible for the communication Client – objects. In this way, clients written in JAVA could access methods of objects written in C++ or Pascal, for instance. This capacity enables a myriad of applications developed in different languages to cooperate among themselves. IDL also makes possible the creation of Object Services, such as naming (or white pages), yellow pages, safety, among others.

By utilizing CORBA, the clients make calls to objects that manage the Virtual Environment (VE) in a transparent way, through ORB. The Virtual Environment Management Objects are responsible for the simulation database maintenance, and contain information about all clients who are participating in the simulation. The Simulation Architecture is illustrated in Figure 4.

![Figure 4 – CORBA Architecture for Distributed VR Applications [Deriggi, 1999].](image)

According to the architecture above, for the construction of CORBA network support, a Distributed Object which manages the users’ connection and disconnection in a simulation and is responsible for the management of object positioning data in the VE, was created. This object was called Position Server Object (PSO), and is located among the VE Management Objects for the architecture above.
Clients of a simulation are not able to communicate with themselves to send repositioning information of their avatars. This communication is performed by means of PSO, which constantly receives repositioning Data and stores them in its database, returning a table with the current position of the remaining VE avatars. This communication enables the client to correct the positioning of these avatars in its VE. These data are sent through the call to PSO’s NetviewPos() method.

Client connection and disconnection in the VE is performed through the calls to PSO’s NetAlive() and NetBye() methods, respectively. Method NetAlive() returns data about all the other clients in the application for the new client to construct its VE.

Thus, PSO plays the role of a passive server in a simulation, that is, only clients communicate with it. PSO does not directly send messages to clients, but may only return values as a result of calls to its methods.

Other objects could be added to the group of those called VE Management Objects, for example, the objects responsible for collision detection.

Both the application and PSO were developed utilizing Orbix software, Iona Technologies [IONA, 1997a], which implements CORBA’s architecture.

The IDL interface of the PSO is illustrated in Figure 5:

```idl
interface Net {
    exception IdRejected {};
    TableObj NetAlive (in long Id, in File Obj)
        raises (IdRejected);
    void NetBye (in long Id);
    TablePos NetViewPos (in long Id, in WT_pq Pos);
    TableObj NetGetObjects (in long Id);
    long NetTimeAck ();
} ;
```

Figure 5 – Application IDL Interface.

5. Network Support utilizing CORBA Platform and its Event Service

By utilizing CORBA platform in network support, one can notice that the VE management in the PSO in tasks such as connection and disconnection management (I/O), collision detection, etc., makes the client application lighter, improving its performance and making the interaction between user and VE faster, since the client sticks exclusively to this interaction and to avatar repositioning.

Thus, the main idea of this network support is to continue utilizing VE Management Objects so that information exchange among clients, such as repositioning messages, be performed directly among them, without interference of these Management Objects. In this manner, the previous support main problem is eliminated: the system’s bottleneck formed in PSO. The solution of this problem is possible within CORBA platform through its Event Service (ES), which performs the sending of messages by means of multicast.

CORBA platform Event Service is based on the providers and consumers’ concept that send and receive messages, respectively, through an event channel. This service is implemented by Iona Technologies, through the software called OrbixTalk [IONA, 1997b], being compatible with software Orbix.
By using OrbixTalk, the applications sending messages are called *Talkers* and those which receive messages are called *Listeners*. Any particular message flow is unidirectional: from one or more *Talkers* to one or more *Listeners*.

In general, *m Talkers* can send messages in the same flow to *n Listeners*, without explicit knowledge of one another. One of the advantages of this approach is that new *Talkers* and *Listeners* may be easily added. The *Talker* does not need to manage a list of *Listeners*. According to OrbixTalk, events are notified to objects in the form of messages. Events of a particular kind are identified by OrbixTalk Topic Name (similar to the Event Channel of CORBA's Event Service) and the *Listener* specifies events of interest, informing OrbixTalk about relevant Topic Names.

OrbixTalk uses the UDP protocol on Multicast IP protocol to share information in between applications. This mechanism does not inform the applications when a piece of information is lost or arrives out of sequence. In order to provide reliability, OrbixTalk utilizes a protocol called "OrbixTalk Reliable Multicast Protocol" (otrmcp) on the UDP. This protocol makes sure that the messages sent from a particular application be reliably delivered to all applications in a respective multicast group.

Therefore, the network support architecture using CORBA and its Event Service is presented in Figure 6:

![Network Support Architecture utilizing CORBA and its Event Service](image)

In this architecture, those clients who wish to enter a shared VE send a *NetOpen* message to the I/O Server Object, which in charge of I/O management in the VE, provides a table with all VE participating clients' avatars. Thus, this one client is already able to receive repositioning messages from the remaining clients in a direct way.

After the new client’s connection in the VE, the I/O Server Object sends a *NetAlive* message through multicast to all simulation clients, notifying that a new avatar the has just connected the VE.

This way, the new client sends repositioning messages about its avatar and receives such messages from the clients as well, without the mediation of VE Management Objects, through *NetViewPos* message.
Whenever a client leaves the VE, the I/O Server Object receives a NetClose message, updates its avatar table and once again informs the clients with a NetBye message, through multicast.

The network support above mentioned has 2 IDL interfaces, one for communicating with Server Object and the other for event service (multicast communication, Figure 7).

```idl
// server.idl File
interface NetServer {
    exception IdRejected();
    TableObj NetOpen (in long Id, in File Obj) raises (IdRejected);
    void NetClose (in long Id);
};

// clients.idl File
interface NetHosts{
    oneway void NetAlive (in long Id, in File Obj);
    oneway void NetBye (in long Id);
    oneway void NetViewPos (in long Id, in WT_pq Pos);
    oneway void NetTimeReq (in long Time, in long Message, in long IdReq);
    oneway void NetTimeAck (in long Time, in long Message, in long IdReq, in long IdAck);
};
```

Figure 7 – Application IDL Interface.

According to this support’s architecture, besides providing I/O Services, the Server Object is a Talker in the application as it sends NetAlive and NetBye messages, thus, performing multicast communication as well.

Clients are at the same time Talkers and Listeners of a same Topic Name, to be able to send and receive positioning messages, forming a system from n to n, where the same clients send and receive positioning messages through multicast.

6. Performance Analysis of Developed Network Supports

The performance analysis of network supports is carried out according to the response time obtained of all clients in the simulation.

This response time is defined as the required time for the communication with all clients in a same simulation. In order to measure the response time, a time request message is sent to all the other simulation clients, waiting for acknowledge messages from these clients’ part. The response time is that measured since the request message is sent until the receiving of the last acknowledge message making sure that all clients have been reached.

Due to architectural differences in the three developed network supports, each support possesses distinct modes of obtaining a simulation’s response time:

- In the developed network support utilizing W2W (section 3), the system’s response time is measured through the time which a change in a property value may be acknowledged by all simulation clients. To accomplish this goal, two properties were created: the “req” property and the “ack” property. One change on the “req” property with the client identification, indicates that this client is sending a time request, in order to measure the system’s response time. The remaining clients receive an event indicating the change on this property, changing the property value “ack” with their identifier. The time counted since property ‘req’ change on this client’s part until the receiving of an event, indicating property “ack” last change on the other clients’ part, indicates the system’s response time.
• By utilizing CORBA platform for the construction of network support (section 4), the system’s response time becomes the time in which a change in a client is acknowledged by all the remaining clients, thus, as this support utilizes the PSO as a mediator, this time measure is performed as it follows: the client, that wishes to measure the response time, invokes PSO’s NetTimeReq() method. This method only returns a value to this client when all the other simulation clients have communicated with the PSO, making sure that these clients be informed about that client’s changes. In this way the response time of this system is obtained by the time count since the invocation to PSO’s NetTimeReq() method, until its return.

The network support built with the aid of CORBA platform and Event Service (section 5), due to the possibility of direct communication client-to-client, has a simplified manner of measuring the system’s response time, i.e.: the client who wishes to measure the response time sends a NetTime Req() message through multicast to all the remaining simulation clients. Upon receiving this NetTimeReq() message, these clients send a NetTimeAck() message in the same manner. Thus, the time measured since the sending of the NetTimeReq() message until the receiving of the last NetTimeAck() message indicates the system’s response time.

The performance graphs for the analysis of the three network support are comprised of time measurements (System’s Response Time) obtained in the following manner:

Initially, time measurements were taken by calculating the sum of time of ten response time measurements of the system.

Each time taking is performed 50 times in order to obtain an arithmetic mean which will be utilized in support comparison. The arithmetic mean is a standard deviation of the network support.

The time measurements are performed, primarily, with two clients in a same simulation. Afterwards they are repeated for 3, 4, and 5 clients. This way, according to this experience, it is possible to analyse, besides the performance, the scalability of the three network support.

The experience was performed by utilizing PC pentium computers, under operational system Windows NT 4.0, in a standard Ethernet local network with bandwidth of 10 Mbit/sec.

The results of these time measurements are presented in the Table 1 and following graphs (Figure 8):

Table 1 – The Times of the Network Support.

<table>
<thead>
<tr>
<th>Clients/ Network Support</th>
<th>W2W</th>
<th>CORBA</th>
<th>CORBA and ES</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Pcs</td>
<td>11.31 sec</td>
<td>2.89 sec</td>
<td>3.55 sec</td>
</tr>
<tr>
<td>3 Pcs</td>
<td>2.92 sec</td>
<td>3.82 sec</td>
<td></td>
</tr>
<tr>
<td>4 Pcs</td>
<td>3.11 sec</td>
<td>3.86 sec</td>
<td></td>
</tr>
<tr>
<td>5 Pcs</td>
<td>3.60 sec</td>
<td>4.03 sec</td>
<td></td>
</tr>
</tbody>
</table>
Based on these graphs, we observe that W2W has the highest response time in the performance measurements for 2 clients, where a reduced performance is observed. Furthermore, the support was not able to perform measurements for 3, 4 and 5 clients. W2W Simulation Server did not handle the massive number of messages successively received in the time measurements.

One can also notice that CORBA support has the lowest times in time measurements, presenting the best performance in the analysis, where it loses only in scalability to the third support (CORBA and Event Service). CORBA support presents a mean degradation of 7.73% at each new client added to the simulation, whereas CORBA and ES supports have a mean degradation of 4.27%, which is explainable by the bottleneck formation in the system (present in PSO).

CORBA and ES presented the best scalability among analysed supports, losing in performance to CORBA support, due mainly to the greater number of messages for the clients necessary to manage multicast communication.

7. Conclusion

This paper presented some factors to be taken into consideration in the construction of a Distributed Virtual Environment (DVE) as to the form of communication among its clients. The development of three types of network support for a DVE was presented, with variations among the communication forms used by the supports.

The performance analysis of these supports show the importance of the choice of the best communication form for a distributed application, and important issues to be considered, such as bottleneck and the support scalability.

Last but not least, it is put that such supports might obtain a better performance by using other techniques of Virtual Reality, such as Dead Reckoning [Macedonia, 1997], which avoids the sending of unnecessary positioning messages. This action diminishing drastically the system’s message quantity.

8. Acknowledgements

This paper has been supported by CNPq - PROTEM-CC-AVVIC (№ 680063/95-1) and by Fapesp (Process 95/6560-8). Felício Deriggi Jr. has been supported by CAPES and Mario Massakuni Kubo by Fapesp (Process 98/01940-5).
9. References


