Using CORBA Middleware to Support the Development of Distributed Virtual Environment Applications

S. Wilson, H. Sayers, M. D. J. McNeill

Virtual Environment Applications Group
School of Computing and Mathematical Sciences
Faculty of Informatics, Magee College, University of Ulster
Northland Road, Derry, BT48 7JL
Northern Ireland
<s.wilson,hm.sayers, mdj.mcneill>@ulst.ac.uk

ABSTRACT

In this paper we report on using Common Object Request Broker Architecture (CORBA) middleware as a means of supporting the rapid development of Distributed Virtual Environment (DVE) applications. We show how CORBA services can be exploited to provide many of the typical functional requirements that developers of DVE applications require, thus reducing the programming effort necessary to develop DVE applications rapidly. We also present the design, implementation and experimental results of NOMAD, our CORBA-based framework for developing DVE applications.

Keywords: Virtual Reality, Distributed Virtual Environments, CORBA, Real-time, Component-Based Design, Computer Graphics.

1. Introduction

Jaron Lanier originally coined the term Virtual Reality in the 1980s to describe a computer graphics system using immersive technology such as head-mounted displays and data gloves. The term Virtual Environment (VE) is used to describe the three-dimensional world in which the user is immersed when using a VR system. Although there is no standard definition for a virtual environment, VEs do share a number of characteristics including:

- The user is immersed in an alternative 3D graphical world that simulates a real or imaginary world.
- Users can navigate through the environment, usually assisted by a user embodiment or avatar.
- Users can examine, interact with and manipulate virtual environment objects.

Distributed Virtual Environments (DVEs) provide users with the illusion that they inhabit a shared virtual world where they may collaborate, communicate and interact with other participants as well as the environment. When the avatar of a remote participant moves though part of the DVE, the other participants will see that avatar move on their screens. To successfully perform this function several additional functional requirements not provided by stand-alone VEs must be met. These additional functional requirements include [1]:

- Support for multiple participants
  Multiple participants should not only be capable of inhabiting the environment simultaneously but also be able to communicate and interact with each other. Thus allowing them to share information and collaborate on common tasks.

- Data distribution and management
  To achieve acceptable frame rates the rendering processes of the remote participants must have local access to the geometry model, so this data must be replicated and/or distributed to clients [1]. As a result, changes in one copy of the shared virtual environment (e.g. a user moves their avatar) must be applied to every other copy of the shared virtual environment. The distribution, maintenance and synchronisation of the environment state represents one of the most demanding technical challenges associated with DVE application development [11].

- Event notification
  Mechanisms must be provided to notify clients of events such as object and environment state changes in a timely manner. Failure to do so may result in several unwanted phenomena such as network lag or high degrees of inconsistency between users. High levels of lag and inconsistency can ruin the illusion of an interactive shared virtual space.

- Network communication
  The efficient use of communication protocols is necessary to facilitate collaboration between multiple participants and distribute application components, environment data and subsequent updates to the environment model.

- Scalability
  The DVE should scale well and allow multiple participants to join the environment without seriously degrading the quality of service of the other connected participants.

Traditionally developers of distributed virtual environment (DVE) toolkits and applications have produced bespoke solutions to address the associated functional requirements and research challenges [3]. This has largely been due to the lack of supporting high level software libraries and APIs that provide the functionality required for developing DVE applications. Developing such a toolkit or framework is not straightforward, as one has to provide a diverse variety of functionality to satisfy functional requirements such as data distribution and management, network architecture and communication protocol configuration, remote object locking and synchronisation. Although these bespoke solutions offer appropriate solutions, they often still require considerable programming and design effort on the behalf of the developer. Such effort early in the development process often restricts changes in the design at later stages [1], and means that the development time is unnecessarily long.

Our approach focuses on using the Adaptive Communication Environment (ACE) framework [24] and its CORBA ORB TAO [9] to address many of the functional requirements of DVE application development. In section two of this paper we review related work in this field. In section three we provide a brief overview of the CORBA architecture and how these may be exploited in DVE application development. In section four we present the design of NOMAD, our CORBA-based software framework for the development of DVE applications. Section five contains experimental results obtained from a multi-user DVE simulation implemented with the NOMAD framework. Section six presents an overview of alternative approaches and future extensions to the NOMAD framework. We finish the paper by presenting our conclusions in section seven.

2. Related Work

With the recent exception of [20,3], VE toolkits such as Maverik [13], DIVE [14], MASSIVE [15] and CAVERN [16] have avoided exploiting the CORBA architecture in the development of DVE applications. Early versions of the CORBA specification were unsuitable for supporting the distribution and management of data within distributed interactive graphical applications for a number of reasons. Firstly, to achieve acceptable frame rates the data used in the client-side rendering process requires the data to be local. Therefore pure client server approaches, where all the environment data is stored centrally at one site, are unsuitable, as some amount of the data must be replicated. Secondly, until recently CORBA only provided synchronous remote method invocations which are unsuitable for graphical applications where rapidly changing shared data must be asynchronously updated. Thirdly, the early versions of the CORBA specifications lacked the necessary levels of support for functionality such as load balancing, fault tolerance, real-time characteristics and quality of service specifications making CORBA unsuitable for deployment within time critical applications. However due to the adoption of asynchronous messaging [6] and research into real-time CORBA [7,8,9,10,21], CORBA now represents a technology mature enough to provide much of the functionality required in DVE development.

Typically, most DVE applications and DVE development toolkits use distributed callbacks extensively as a method of event notification [1,13,14,15,16]. Distributed callbacks are often not straightforward to program and have some inherent problems regarding object reference equality, persistence, callback failure and notification scalability [17]. CORBA provides two object services that support event notification, the event service [4] and the notification service [18].

3. The Common Object Request Broker Architecture

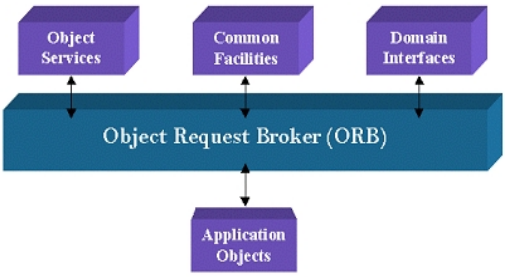
In 1989 the Object Management Group (OMG) was formed to "*establish industry guidelines and detailed object management specifications to provide a common framework for application development*" [2]. To date the OMG have produced a number of specifications most notable of which is the Object Management Architecture (OMA) and its core the Common Object Request Broker Architecture (CORBA). The CORBA specification describes how applications and their objects can be written in different programming languages yet still interoperate across heterogeneous operating systems and networks. The key elements of the architecture are:

- **The Object Request Broker (ORB)**
  The ORB allows application objects to communicate with other application objects and elements of the OMG reference model in a network and platform independent manner. The primary responsibility of the ORB is to resolve requests for object references, enabling

application objects to interoperate. Once a client has obtained a reference to an application object it may invoke methods of that object. The ORB is also responsible for translating method input parameters into a format that may be transmitted across the network to the remote object. The ORB also unmarshals parameters once they are transmitted across the network into a system-independent format the receiving object can use.

- **Object Services, Common Facilities and Domain Interfaces**
  Object services provide a core set of commonly used interfaces for object management (object creation and location). Examples of object services include the event, naming, time, trading, property and life cycle services [4]. Common Facilities are services that many applications may share, but which are not as fundamental as the Object Services. An example is the mobile agent facility [5], which specifies the interaction between various manufactures agent systems. Domain Interfaces provide standard interfaces for specific application domains. Such interfaces exist for application domains such as finance, manufacturing, telecommunications and health care.

- **Application Objects**
  These objects are created by application developers to perform specific tasks for users. Most distributed systems consist of a large number of application objects that typically utilise functionality provided by the other components of the OMG reference model such as the naming service of the object services component.



The OMG Reference Model Architecture
Figure 1

One of the cornerstones of CORBA is the Interface Definition Language (IDL). As the name suggests IDL is a definition language, not a programming language. IDL is used within CORBA development to define the public interface of an object in a language-independent manner. With IDL language mappings for a number of languages including C, C++, Java, COBOL and Smalltalk it is possible for application objects implemented in different languages to interoperate. IDL is one of the key elements within CORBA that supports the development and integration of heterogeneous systems.

The CORBA architecture has several advantageous features that can greatly simplify the development of DVE applications. The advantages include:

- **Location Transparency**
  To access the services of a remote object a DVE application developer has only to obtain a reference to that object. They need not concern themselves about the location of the object.

- **Component-based Development Approach**
  As CORBA supports language mappings for popular object oriented programming languages such as Java and C++, DVE developers can create distributable object and components that can be reusable and portable.

- **Support for Commonly used Services**
  CORBA provides an array of services that DVE application developers can use distribute and managing data in a DVE. Common services such as the naming and trading services [4,19] provide functionality to support object location, whereas the event and notification services allow update messages to be sent to clients in an efficient and timely manner.

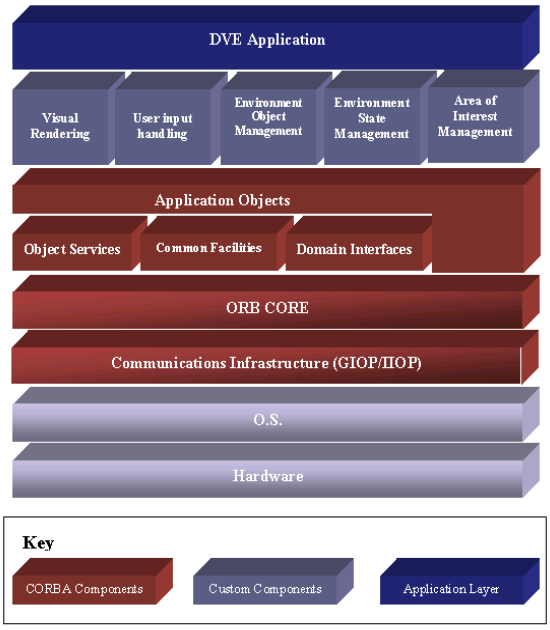
- **Interoperability**
  CORBA allows a DVE application developer to seamlessly develop DVE applications consisting of objects written in different languages that can interoperate across heterogeneous networks, platforms and operating systems.

- **High level of abstraction**
  CORBA provides it services at a high enough level to allow application developers to ignore the underlying complexities of the architecture, thus speeding up the development process.

4. The NOMAD Framework

We have developed a framework or toolkit based on CORBA to accelerate the development of distributed virtual environment applications. A framework is a reusable, “semi-complete" application that can be specialised to produce custom applications [22]. NOMAD consists of a group of components and application objects that together form an object oriented application framework to support DVE application development. NOMAD exploits CORBA to provide much of the core functionality required by DVE application developers. Several distinct

specialised components reside 'above' the layer of core NOMAD application objects. These components provide specialised high level functionality such as environment object management, area of interest management and event notification. They are designed in such a manner that they can either be dynamically configured to address specific development needs or replaced entirely with custom components developed by the user.



The Nomad component Architecture
Figure 2

4.1. Server Hierarchy

One of the key concepts employed by the NOMAD framework is the concept of *locales*. To provide scalability our approach divides the environment data into more manageable segments (locales), an approach originated by SPLINE [12]. Each locale represents a geographical region of the virtual environment. We implement this concept by utilising a range of CORBA services to develop a variety of distributed objects to coordinate the management of the CVE application. We call these components the *environment server, session managers, area managers* and *locale managers*.

Any NOMAD based DVE will typically consist of several server objects, each of which perform a specific set of tasks. At the root of any NOMAD application is the environment manager. The environment manager is a CORBA server object that manages the DVE at a high level, acting as the first point of contact for any client (user) that wishes to join the DVE. Client processes wishing to join the DVE will register themselves with the environment manager and issue a request to join the DVE. Based on the starting location of the client processes avatar, the environment server will pass the client request to join the DVE onto the next server object in the hierarchy, the session manager.

We envisage NOMAD supporting DVE applications with many users at locations geographically distributed over a wide area. Each region of the area has a session manager, a process which maintains a complete copy of the environment model. This approach allows a NOMAD DVE and its environment model to be extended and enriched simply by launching a new session manager at a remote site. The session manager coordinates and manages the VE model by delegating responsibilities to other CORBA sever objects such as the area manager and the locale manager. The main responsibilities of the session manager include:

- Sub-dividing the environment model into distinct regions.
- Creating area managers to coordinate the distinct regions of the environment model.
- Initialising clients who wish to join the session managers region of the DVE.

Depending on a number of factors including the size and complexity of the environment model and the maximum number of clients the session manager has been configured to support, one or more area managers are created by the session manager. An area manager is responsible for a number of activities including:

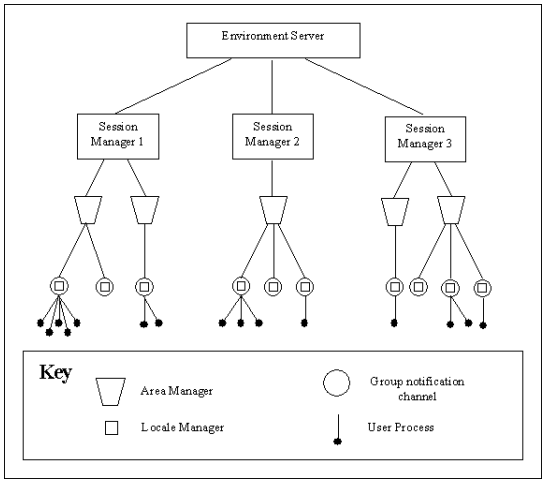
- Creating and initialising locale managers and assigning them elements of a particular region of the virtual environment to manage.
- Overseeing load balancing at a lower level by spawning new locale managers when connected clients begin to suffer degradation in performance due to the locale managers becoming overburdened by client requests and update messages.
- Dynamically resizing locales to maintain optimum levels of performance. If locales represent very small regions of the environment (such as a corridor or a single room), moving around this small area may cause degradation in interactivity from continually registering with new locale managers and downloading new portions of the environment model. In this case the locale manager would be assigned a larger portion of the environment. In this case a locale manager can be assigned a larger portion of the environment, thereby reducing these events.
- Distributing data such as interparticipant communication data to other area and locale mangers for eventual distribution to client processes.

CORBA server objects known as locale managers synchronise all client activity within the portion of the environment model represented by the locale.

Synchronisation of the portion of the environment model is achieved by maintaining a list of changes made to the persistent copy of the environment model since the initialisation of the environment. This persistent copy of the environment model represents a complete copy environment model before any changes in environment state occurred. Changes in the environment state are distributed to clients registered with the locale manager via the CORBA event service. Locale Managers are responsible for a number of duties including:

- Maintaining a dynamic copy of the environment state, which contains changes that have been made to the persistent environment information since the environment was initialised.
- Ensuring local client copies of the environment model are synchronised by sending environment state change data to a client who has requested the information.
- Registering new client processes when they join the environment and ensuring they obtain the entire necessary environment model object descriptions and updates.
- Managing the routing of interparticipant audio/video communication streams to clients within the locale and/or other locales.

Figure 3 shows how the server hierarchy may evolve when three session managers connect to a NOMAD environment manager. Each of the session managers subdivides its environment model into distinct regions and creates area manager objects to coordinate the management of these regions of the environment model. In turn each of these area managers further subdivides their region of the environment model into smaller locales, each of which is managed by a locale manager. Once the locale managers have been created and initialised, the environment server is ready to accept client requests.



An example of a server object hierarchy of a NOMAD DVE application
Figure 3

4.2. Object distribution and environment state management

Once the DVE has been initialised and all of the server objects have been created the environment server is ready to accept client requests to join the environment. Based on the starting location of the participant's avatar, the request to join the environment is propagated down though the server hierarchy to the appropriate locale manager. During this process, the session manager and the area manager may dynamically reconfigure the server hierarchy by creating a new server object to accommodate the addition of a new client to the environment. For example, if an area manager has reached the maximum number of clients it can support (which can be dynamically reconfigured), the session manager may create a new area manager server object and sub-divide the original region of the VE between the two area managers. Load balancing and dynamic reconfiguration of the NOMAD server hierarchy are two of the core elements supporting the development of scalable DVE applications.

Once the client request has reached a locale manager, the client is registered with the group communication channel. In addition the client process is provided with a copy of the current state of the environment model of the locale. The client process is also sent the location of the geometry models of adjoining locales. One locale is considered to be adjoining another locale if one can move into, see or hear the other locale. The caching of adjoining locale environment data ensures that clients minimise delays when they have to de-register and re-register with locale managers as they move from one locale to another.

To ensure consistency between remote clients, replicated copies of the environment model must be synchronised. When the state of an environment object changes, details of this update must be sent to all of the affected clients. As the virtual environment has been sub-divided into locales, the effected clients will typically only be those in the immediate locale and possibly the adjoining locales. Therefore the number of clients that have to be informed of the update will typically be a small percentage of the total number participating in the DVE as a whole, thereby keeping data communication to a minimum.

The NOMAD framework utilises the TAO real-time event service [23] to notify clients of environment state changes on a locale and global level. The OMG Event Service allows events to be sent to one or more recipients with a single call. In the event service objects that create events are called *suppliers* and objects that receive them are called *consumers*. Both suppliers and consumers are connected together via an event channel. The event channel is responsible for supplier and consumer

registration, error handling and the dispatching of events. In addition to the standard functionality, the TAO event service provides real-time extensions such as:

- Prioritisation of event dispatching
- Event filtering
- Specification of event notification criteria
- Periodic event notification

In NOMAD each locale manager maintains two event channels for that locale. When a client changes the state of an environment object it owns, the details of the change in state is sent to the locale manager via the locale managers consumer event channel. The locale manager then transmits this update to other users in that locale though its supplier event channel. Such an approach has several advantages. Large numbers of updates can be merged and compressed into one update message by the locale manager and transmitted to only the affected clients. Messages can also be filtered, ensuring updates are only sent to client processes that have a genuine interest in the update message.

4.5. Discussion of the NOMAD framework

The primary advantage of the NOMAD framework is it provides the developer of DVE applications with commonly required functionality at a high level of abstraction. This allows developers to focus on developing the DVE application without having to concern themselves with the underlying complexities of the communication and synchronisation of data or load balancing.

The multiple server approach offers several benefits over implementations that employ a pure client-server hierarchy. The VE can be easily extended during execution of the DVE by registering another session manager with the environment server. Load balancing can occur while the DVE is running by creating new server objects, dynamically balancing and reconfiguring the server hierarchy.

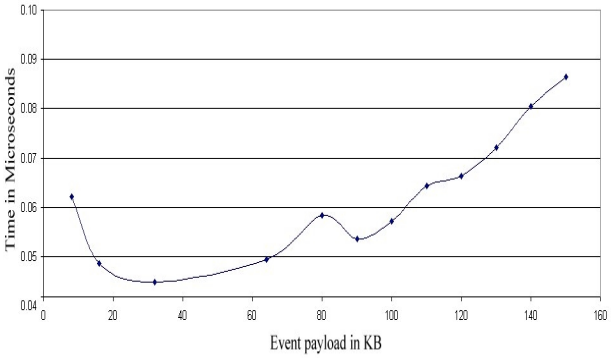
The effect of environment state changes are typically restricted to only the clients in the immediate locale and a subset of the adjoining locales. When updates are sent to the locale manager the opportunity arises to merge several update messages into larger packets, thus further reducing the number of update messages that have to be transmitted. This allows the efficient transmission of update messages to only those clients that are affected. As the locale manager is largely responsible for the transmission of environment state updates, it can control the flow at which updates are sent to clients.

The major disadvantage of the above design is that latency is introduced, as update messages are not transmitted directly between participating clients, but rather via the locale manager. This effect is further exaggerated if the locale manager is inundated with a high number of updates or if the locale manager processes the updates in some manner. Further work is required to quantify the latency and determine what can be done to minimise this effect.

5. Results

One of the key concepts of the NOMAD architecture is the sub-division of the environment model into locales as a means of increasing the scalability of the DVE. It is therefore important that the component responsible for the transmission of environment updates operates efficiently. As the update messages are transmitted though an event channel and not directly to their destinations, we conducted a number of experiments to determine the latency introduced by the event channel when sending varying amounts of data.

The tests described below were conducted on two CPU Pentium II 400 MHz PCs with 256MB RAM running MS Windows 2000. The network used had a bandwidth of 10Mbits per second and approximately 250 other PCs connected to it. During the execution of the tests many of the other PCs were also using the network. Although a dedicated network could have been set-up for the tests, it was decided that running the tests on a “live” network would provide more realistic figures. Each test involved sending 100 events in 10 bursts with a delay of 100 milliseconds between bursts from one supplier to one, five and fifteen respectively. The data transmitted represented environment state changes such as changes in avatar location and not environment scene data such as texture maps. In the first set of tests the consumers, supplier and ORB services all executed as separate processes on the same workstation. In the second set of tests the supplier and consumers resided on separate workstations. Neither the locale manager nor the consumer of the event performed any processing other than to keep track of timings. The payload (data packet excluding event header information) associated with each event ranged from 8KB to 150KB and the average latency incurred by the first consumer was recorded.

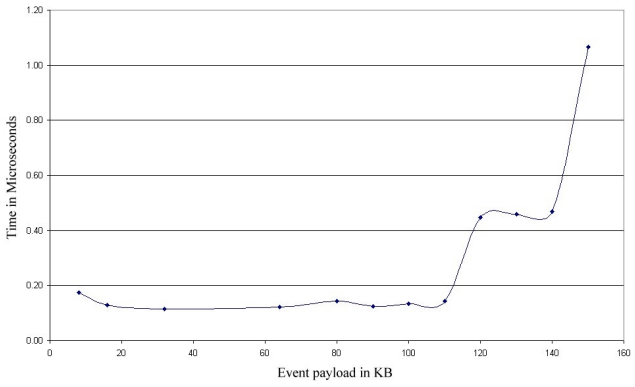


Latency of event delivery with 1 consumer
Figure 4

The chart in Fig.4. plots the average time taken to deliver an event to 1 consumer as the data payload increases. Unexpectedly the time taken to deliver the event did not initially increase, as expected. Instead the latency of event delivery continued to fall from 0.062 ms for 8KB to 0.044 ms for 32KB, before making a steady increase. Table 5 shows the payload size, latency and number of events per second. The time taken to deliver an event remains fairly constant until the payload reaches 100KB while the number of events per second falls from as the payload increases. At 100KB per payload, some 50 events per second are transmitted (TAO reduces the number of events per second in order to maintain performance).

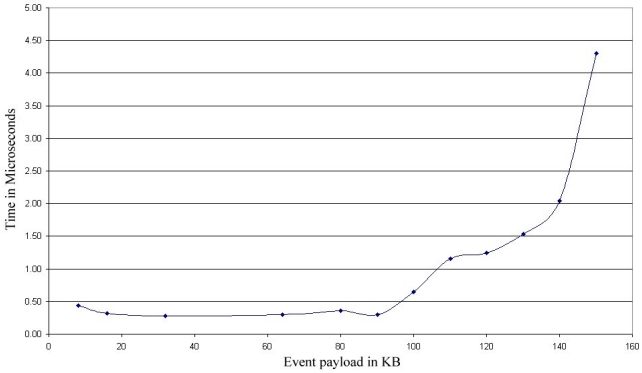
Payload on KB	Events per sec.	Event latency ms.
8	317	0.062
16	217	0.048
32	127	0.044
64	74	0.049
80	61	0.058
90	55	0.053
100	50	0.057
111	45	0.064
120	43	0.066
130	40	0.072
140	36	0.080
150	33	0.086

Throughput and event latency times for 1 consumer
Table 5



Latency of event delivery with 5 consumers
Figure 6

Both of the tests for 5 consumers and 15 consumers showed a similar trend as the test for 1 consumer, although the average delivery time of events increased with the number of consumers. In all three tests the event delivery time began to increase dramatically once the payload of the event was greater than 100 KB. This suggests that the optimum payload size should not be greater than 100KB to ensure reasonably high throughput. Although 100KB is not sufficient in size to send large data sets such as complete scene descriptions, it is more than adequate in size to transmit events notifying clients of changes in environment state.



Latency of event delivery for 15 consumers
Figure 7

When the tests were executed across a network there was a dramatic increase in event latency. Table 9 shows the latency (in microseconds) incurred when events of 100 KB in size are transmitted across a live network. Although these figures are much higher than those presented previously, the values are still quite small. Our experiments (run across a 10Mbit/sec network) show that the real-time efficient delivery of events up to 100 KB in size can be achieved by the NOMAD framework.

# Consumers	Latency ms	Payload KB	Events p/sec.
1	1921.71	100	50
5	2030.71	100	8.89
15	2124.50	100	7.64

Latency of event delivery with 5 consumers
Table 8

7. Conclusions

Due to the technical and research challenges associated with DVE application development there is a need for high level programming support. The CORBA architecture and its real-time extensions now provide a flexible and efficient solution to many of the technical issues in DVE application development.

As our results demonstrate, it is feasible to use the CORBA and TAO's real-time event service as a method to deliver environment state changes to a number of clients within a predictable, timely and efficient manner. By sub-dividing the environment model into many locales one can reduce the burden on each DVE server object whilst providing a scalable and efficient solution. However further research is required to determine the exact optimum throughput and event payload size. More tests need to be performed involving several locale managers

each with many connected clients. Furthermore, in a working system each of the client processes will be processing event data (rendering and updating their persistent store) in addition to sending and receiving events. Our next goal is to show how the NOMAD framework performs with a larger number of users supporting a real distributed virtual environment application.

References

1. B. MacIntyre and S. Feiner, "A Distributed 3D Graphics Library", *Computer Graphics* (SIGGRAPH 98 proceedings), 1998
2. The Object Management Group, http://www.omg.org
3. D. Dias, G. Fox, W. Furmanski, V. Mehra, B. Natarajan, H.T. Ozdemir, S. Pallickara, Z. Ozdemir, "Exploring JSDA, CORBA and HLA based MuTech's for Scalable Televirtual (TVR) Environments",Workshop on Object Orientation and VRML, VRML conference 1998, Monterey, California, 1998
4. The Object Management Group, "CORBAservices: Common Object Services Sepcification", ftp://ftp.omg.org/pub/docs/formal/98-07-05.pdf
5. The Object Management Group, "Common Facilities Specification - Mobile Agent Specification", ftp://ftp.omg.org/pub/docs/formal/00-01-02.pdf
6. Alexander B. Arulanthu, Carlos O'Ryan, Douglas C. Schmidt, Michael Kircher, and Jeff Parsons, "The Design and Performance of a Scalable ORB Architecture for CORBA Asynchronous Messaging", Proceedings of the IFIP/ACM Middleware 2000 Conference, Pallisades, New York, April 3-7, 2000
7. Douglas C. Schmidt, David L. Levine, and Chris Cleeland, "Architectures and Patterns for High-performance, Real-time ORB Endsystems", Advances in Computers, Academic Press, Ed., Marvin Zelkowitz, 1999
8. Douglas C. Schmidt and Fred Kuhns, "An Overview of the Real-time CORBA Specification", IEEE Computer, Special Issue on Object-Oriented Real-time Distributed Computing, edited by Eltefaat Shokri and Philip Sheu, June 2000.
9. D. C. Schmidt, D. Levine, and S. Mungee, "The Design of the TAO Real-Time Object Request Broker", *Computer Communications* Special Issue on Building Quality of Service into Distributed Systems, Elsevier Science, Volume 21, No. 4, April, 1998
10. The Object Management Group, "Real-time CORBA – A joint Revised Submission", 1999
11. S. Singhal and M. Zyda, "Networked Virtual Environments – Design and Implementation", Addison-Wesley, ISBN: 0-201-32557-8, 1999
12. D. Anderson, J. Barrus, J. Howard, C. Rich, C. Shen, R. Waters, "Building Multi-User Interactive Multimedia Environments at MERL", IEEE Multimedia, vol.2, no.4, Winter 1995, pages. 77-82
13. R. Hubbold, X. Dongbo, S. Gibson, "Maverik – the Manchester Virtual Environment Interface Kernael", *Proceedings of the third Eurographics Workshop on Virtual Environments*, 1996
14. O. Carlsson and O. Hagsand, "DIVE – A Platform for Multi-User Virtual Environments", Computer & Graphics, Vol. 17, No.6, pp 663-669, 1993
15. C. Greenhalgh, and S. Benford, "MASSIVE, A Collaborative Virtual Environment for Teleconferencing", *ACM Transactions on Computer, Human Interaction,. 2*(3), 1995
16. Leigh, J., Johnson, A., DeFanti, T., CAVERN: A Distributed Architecture for Supporting Scalable Persistence and Interoperability in Collaborative Virtual Environments, in *Virtual Reality: Research, Development and Applications,* Vol 2.2, 1997
17. M. Henning and S. Vinoski, "Advanced CORBA Programming with C++", Addison-Wesley, ISBN 0-201-37927-9, 1999
18. The Object Management Group, "The Notification Service Specification", June 2000, available at ftp://ftp.omg.org/pub/docs/formal/00-06-20.pdf
19. The Object Management Group, "The Trading Object Service Specification", May 2000, available at ftp://ftp.omg.org/pub/docs/formal/00-06-27.pdf
20. F. V. Deriggi, M.M. Kubo, A. C. Sementille, J. R. Brega, S. G. dos Santos and C. Kirner, "CORBA Platform as Supportt for Distributed Virtual Environments", *Proceedings of IEEE Virtual Reality*, 1999, pp 8-13
21. A. B. Arulanthu, C. O'Ryan, D. C. Schmidt, M. Kircher and J. Parsons, "The design and Performance of a Pluggable Protocols Framework for Real-time Distributed Object Computing Middleware", *Proceedings of the IFIP/ACM, Middleware 2000 Conference*, 2000
22. R. Johnson and B. Foote, "Designing Reusable Classes", Journal of Object-Oriented Programming, SIGS, 1, 5, 1988
23. T. Harrison, D. Levine and D. C. Schmidt, "The Design and Performance of a Real-time CORBA Event Service", *Proceedings of OOPSLA'97*, ACM, Atlanta, GA, October 6-7, 1997
24. D. C. Schmidt, "An Architectural Overview of the ACE Framework: A Case-study of Successful Cross-platform Systems Software Reuse", USENIX login magazine, Tools special issue, November, 1998