INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

Université d'Ottawa · University of Ottawa

VESIR-6

Virtual Environment Supporting Multi-user Interaction over IPv6

by

Mejdi Eraslan, B.Sc.

A thesis submitted to the
School of Graduate Studies and Research
In partial fulfillment of the requirements for the degree of

Master of Applied Science
In Electrical Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Information Technology and Engineering (Electrical & Computer Engineering)

Faculty of Engineering
University of Ottawa

April, 1999

Abstract

Virtual environments (VEs) are interactive computer simulations that immerse users in an alternate reality. Multi-user VE applications simulate the experience of real-time interaction among multiple users in a shared three-dimensional (3-D) virtual world. The Internet has a growing importance as a communication medium for presenting multi-user VEs and interaction among distributed VEs. In this thesis, we identify the requirements for highly interactive multi-user VEs and limitations of collaboration over the Internet. We focus on the design, implementation, and evaluation of network communication infrastructures for multi-user virtual environments, also exploiting the multicast and quality of service (QoS) support introduced with IPv6, the next generation Internet protocol. We propose an architecture, VESIR-6, that combines the 3-D navigational and interactive capabilities of virtual worlds with the communication capability of the next generation IPv6 protocol to bring people together in one virtual world, regardless of their physical locations, in a platform-independent manner. The goal of the project is to get all the parties together in one virtual world, allowing them to see each other, to interact with each other, and to affect that common world in a secure and high-quality network infrastructure over the Internet.

Acknowledgements

I would like to thank Dr. Nicolas D. Georganas for supervising me throughout my masters and for the well-equipped lab environment he provided to facilitate our research. I would also like to thank my friends and colleagues for their valuable input. I would also like to acknowledge the financial support from NSERC, which partially funded this research project.

I wish to thank my wife for her love, patience and support, which helped me to complete this work.

List of Acronyms

3-D	Three-dimensional
6BONE	IPv6 Backbone
API	Application Program Interface
ATM	Asynchronous Transfer Mode
AWT	Abstract Window Toolkit
CANARIE	Canadian Network for Advanced Research in Industry and Education
CSMA/CD	Carrier Sense Multiple Access / Collision Detect
DiffServ	Differentiated Services
DLL	Dynamic Link Library
DVE	Distributed Virtual Environment
EAI	External Authoring Interface
HTML	HyperText Markup Language
ICMP	Internet Control Message Protocol
IGMP	Internet Group Management Protocol
IntServ	Integrated Services
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
JNI	Java Native Interface
LAN	Local Area Network
MBONE	Multicast Backbone
QoS	Quality of Service
RSVP	ReSource reserVation Protocol
SLA	Service Level Agreement
TCP	Transmission Control Protocol
TTL	Time To Live
UDP	User Datagram Protocol
VIA	VESIR-6 Interaction Agent
VIAP	VESIR-6 Interaction Agent Protocol
VE	Virtual Environment
VESIR-6	Virtual Environment Supporting Multi-user Interaction over IPv6
VNCP	VESIR-6 Network Communication Protocol
VRML	Virtual Reality Modeling Language
WAN	Wide Area Network
WWW	World Wide Web

Table of Contents

List of Figures

List of Tables

List of Listings

1 Introduction

Virtual environments (VEs) are visualization systems that allow users to discover new ways of experiencing real-world and simulated phenomena. The terms virtual reality, virtual world, and virtual environment are used interchangeably, and they generally refer to the technology for moving through and interacting with a 3-D computer generated environment such that the experience is perceived to be real. 3-D graphics-based VEs and visualizations will offer new insights into the structure of information that will be useful in a vast number of fields, including education, manufacturing, commerce and medicine. VEs will quite likely change the nature of computing, which will move from text-based dialogues using arcane computer languages to graphically based visualizations and spatial metaphors that symbolically represent text and numerical data. People will use natural hand gestures and head movements to execute commands for navigating through and manipulating objects in virtual worlds.

Multi-user VEs are an evolving field of applications, that are distributed in nature, where different parties interact within the virtual world concurrently. Multi-user VE applications have evolved from distributed interactive simulation experiments for military purposes. Improved computer graphics, and platform independence of the Virtual Reality Modeling Language (VRML) [1] and Java [2] standards opened the doors for a new kind of multi-user applications including online training, teleconferencing, telemedicine, and electronic commerce in a 3-D virtual environment. Internet could potentially be deployed as the communication and message distribution mechanism for these applications. Internet-based VEs can bring a large number of participants together in a simulated 3-D space and let users explore virtual worlds and interact with each other. Standards for texture and 3-D information exchange between the viewer and application have already been published, and the software tools to browse information on different platforms are publicly available. VRML and Java allow rendering of 3-D objects within the web browsers.

VRML has already established itself as a standard for the exchange of 3-D descriptions on the Internet. However, it is a completely static description, with no support for virtual worlds with several users and applications, or with a high number of

dynamic objects. In this thesis, we want to examine how VRML can be extended to fit the requirements of an interactive, distributed, multi-user virtual environment.

This thesis considers two major fields where VRML needs to be extended in order to support multi-user VEs: the network components to support multiple users and to achieve consistent worlds, and an object-oriented interaction model via some intelligent agents, which allows the modeling of interactions and behaviors, that can be extended to support complex applications.

1.1 Networking VEs: Challenges

The fundamental aspects of networking VEs include accessing the network, generating interactive graphics, sharing state, and managing the interactions between the various participants across the network. Multi-user VEs can support the full range of desired interactions among users. Networking VEs within a LAN is fairly easy, since there is plenty of bandwidth and very low latency. Networking VEs across a wide area network, such as the Internet, is a much more complex task, and requires an engineering design that must balance a variety of issues, including bandwidth, latency, data consistency, and scalability.

Networked Multi-user VEs raise challenging research questions concerning how users interact with each other, applications, and other users, and how the system scales when the number of users increases. Representation of users in the virtual world, reliable distribution of control information such as object discovery or removal, distribution of computation, networking, synchronization, and maintaining consistency are among other problems that must be solved when designing a scalable multi-user VE.

Distributed Multi-user VE systems must accommodate delays and delay variance compatible with human interaction times in order to preserve an accurate order of events and provide a realistic simulation. While the users may be geographically distant from each other, the VEs running at different sites must be operating and interacting as though they were in close proximity to one another.

The amount, type, and sensitivity level of information that must be exchanged in a distributed VE drives the communications requirements, and depends on the number and type of participating users and the nature and level of interaction among those users. The packets exchanged among users could include the location of each object, its direction, orientation, or text-based messages. A typical moving object would generate about 10

packets/second. With so much information being exchanged among users at different locations, multicasting is required to minimize network bandwidth used and to reduce input to individual users so that each user receives only those packets that are of interest to him/her.

The multi-user VE applications are characterized by the need to distribute a real-time application over a shared wide area network in a scalable manner, such that the users are able to interchange state data and event updates with sufficient reliability and timeliness to sustain the 3-D virtual world containing a large number of mobile (moving) objects. They require bulk transfer of virtual world data, low latency highly-reliable delivery of control commands and state updates, and low latency semi-reliable delivery of event updates. These requirements derive the need for real-time multicast with established quality of service in a shared network.

Furthermore, achieving multi-user interactivity in distributed VEs over the Internet with scarce bandwidth resources is a great challenge. There are several design issues to be considered in developing a large-scale distributed VE over the Internet. The most obvious limitation is the considerable network bandwidth utilization, particularly as the system incorporates more hosts. Most hosts are connected to the Internet via low bandwidth dialup lines, with a bandwidth of no more than 56kbs. Low bandwidth operation is essential for a DVE transport layer to be immediately available to the currently existing infrastructure. Although local area networks (LANs) are fast enough to support the distributed VEs on a large scale, we need to be careful in designing the communication architecture to enable multi-user VE applications run over the Internet with most users connected to the system by dialup modem connections of 56kbps. A second problem related to DVEs is the network latency. By the time an update arrives at a particular receiver, it may be several hundred milliseconds old, and the source may have already placed a new update in transit. Suppose that you are performing a combat simulation and trying to shoot at your enemy. Because of the network latency, the enemy never seems to be where you think he is. You see him in the scene at his relatively old position, and when your bullet arrives at him after several hundred milliseconds, he might have already moved. This latency becomes more critical when dealing with medical applications and immersive environments.

Maintaining the illusion of a single system in a multi-user VE is challenging because of the messaging required to exchange information among participants. For

example, networks impose a noticeable delay from the point that a message is transmitted to the point that a message is actually received at its destination. Moreover, different messages may incur different delays depending on the state of the network. Each host must therefore attempt to present a consistent real-time view of the VE and contend with the fact that all of the incoming information about users could already be out-of-date. The network delays are particularly difficult to handle when multiple users interact with each other directly.

1.2 Thesis Contributions

Our research objective in the thesis is to develop and evaluate an end-to-end architecture for supporting emerging DVE applications. We propose Virtual Environment Supporting Multi-user Interaction over IPv6 (VESIR-6), a scalable architecture, as a general framework that supports distributed multi-user VEs over the next generation Internet protocol. VESIR-6 overcomes the challenges in networking VEs, minimizes the computational and bandwidth requirements of VE members, exploits the authentication and privacy capabilities of IPv6 to bring security and data confidentiality to the transactions, and integrates multicast communication with QoS support over IPv6 to provide a reliable means of collaboration among the objects within the VE.

VESIR-6 is composed of independent modules with different functionality. User interface modules presents a friendly interface to the user to control the objects in the VE, the interaction agents acts like a bridge between the application and the objects in the scene, and the networking module provides access to the underlying IPv6 network. This flexible system design makes it easy to extend the system by modifying the appropriate modules. In order for VESIR-6 to be platform-independent, we designed and implemented the user interface in Java. The interaction agents are interacting with both the user interface and the VE which is written in VRML, so they implement the Java-VRML interface. The networking module has to be written in C/C++ since most of the IPv6 implementations are using C socket calls for network communication. The most challenging part of the thesis was to combine these modules that are implemented in different programming languages successfully. The arising JNI (Java Native Interface) and EAI (External Authoring Interface) standards helped us to integrate the pieces. We developed our own protocols for interaction (VIAP, VESIR-6 Interaction Agent Protocol)

and message transfer over the IPv6 network (VNCP, VESIR-6 Network Communication Protocol) in order to improve the response time of the application.

The most obvious contribution of VESIR-6 is the "networking module" that encapsulates low-level network communication aspects and thus provides transparent access to the underlying IPv6 network for the DVE application developers. The VESIR-6 networking module provides simple abstractions for ownership and QoS management in addition to connection and group management primitives. Although developed simultaneously with VESIR-6, the networking module can be deployed in any DVE application.

The basic approach of VESIR-6 for achieving rapid and yet reliable communications is to combine the best features of central server and multicast approaches. The group manager is a central database that keeps track of the users and object in the system and thus provides a consistent view of the VE. The event updates are directly multicasted to the users.

The most outstanding feature of VESIR-6 is that it employs the latest technologies to optimize the use of scarce network resources. Among those technologies, the most important are the IPv6 and Differentiated Services standards. The proposed end-to-end QoS architecture in thesis shows the benefits they would bring to real-time interactive multimedia systems.

1.3 Organization of the Thesis

The remainder of the thesis is organized as follows:

In Chapter 2, we discuss the aspects critical, from a communication point of view as well as data distribution, to large-scale distributed VEs. We identify the network layer service requirements of multi-user VE applications, and then see how IPv6 satisfies those requirements with its inherent multicasting, security, and QoS capabilities, and our motivation to use it.

We address the network implications and requirements to enable large scale networked multi-user VEs in Chapter 3. We propose an information delivery architecture that meets the scalability requirements of multi-user VE applications, and address the low-level networking issues in VESIR-6. We design our communication model step by step to provide a scalable and flexible network topology. We do some performance

analysis and outline the benefits of the final model. We give the API supported by the network modules and see how it hides the networking issues from the application .

Chapter 4 introduces the proposed system architecture that is composed of independent modules to hide the complexity of each task within a well-defined API, and to ease enhancements to the system by modifying only the related module. The functionality of each module, together with the interfaces used to interconnect them are also given. The system architecture of the system is mainly composed of three parts: Networking module, user interface, and VRML agents. Each module constitutes a separate library with minimal dependencies.

The next two chapters are continuation of the system architecture. They focus on management of the shared state and end-to-end QoS, respectively. In Chapter 5, we address the issues regarding the management of the shared state of the virtual world contents among participants. The group manager's functionality is explained in detail in this chapter. Chapter 6 outlines the resource management techniques that meet the performance requirements of the application. The proposed end-to-end QoS architecture is given in this chapter. This approach used by VESIR-6 guarantees reliable delivery while eliminating the need for acknowledgement traffic, hence optimizing the resource usage.

The implementation of VESIR-6 is explained in Chapter 7. We illustrate there how easily VESIR-6 could be integrated to the web.

In Chapter 8, we review the related work in designing DVEs over the existing IPv4 networks, and the limitations they are faced with. So far, unfortunately, there has been no work on integrating IPv6 into DVE applications.

Finally, Chapter 9 presents the conclusions.

2 The motivation for using IPv6

The current Internet protocol (IPv4) does not provide guaranteed performance levels or QoS support. Native multicasting, that is needed to distribute the update messages to the participants within a multi-user VE efficiently, is optional in IPv4 [3]. To move the Internet forward to a point at which it can truly support multi-user VE applications on a large scale will require the deployment of a network protocol that provides multicasting, security in terms of authentication and privacy, and QoS services.

IPv6 was introduced to achieve these goals. The IPv6 protocol makes it easy for applications to ask for various levels of assured service, enhanced security, and improved reliability. With IPv6 in play, organizations can expect certain assurances regarding quality of service (QoS), a term which means that an application can call for the specific reliability and speed it needs to do its job. IPv6 does this by providing a way for applications to request handling, by setting some fields of the packet header, as we will see in this chapter, throughout the WAN. In sharp contrast, an IPv4 system has no way to differentiate between data payloads that are time-sensitive, such as streaming video and audio, and those that aren't, such as e-mail and file transfers. IPv4 gives the same "best level of effort" handling to each packet.

The IPv6 protocol has already been implemented for a great number of platforms, and most router vendors have dual-stack code that can understand both IPv4 and IPv6 packets. 6BONE, the testbed to assist in the evolution and deployment of IPv6, has so far connected more than 400 research institutes and companies in 41 countries over native IPv6 [4].

We will first investigate the network service requirements for multi-user VEs and then see how the next generation Internet protocol, IPv6, addresses these requirements.

2.1 Network service requirements

Multi-user VE applications are distributed systems, meaning that they must contend with all of the challenges of managing network resources, including data loss, network failure, and concurrency. They rely on the network to exchange information regarding the current state of the VE. The basic service requirements of multi-user VEs from the network can be summarized as follows:

- *Low latency*: The maximum communication time between initiation and occurrence of an action, from an application perspective, should be predictable and bounded by a user perceived notion of interaction delay. Rapid network response times are necessary in DVEs in order to provide acceptable degree of illusion of reality to the user. Users of a widely scattered VE do not usually care about the network topology or the high level of security/encryption or firewalls that handle their traffic. What they care about is something more fundamental, such as: "Do I get acceptable response times when I access my mission critical applications from a remote office?". Acceptance levels for delays vary from an application to another. While a user would be willing to put up with a few additional seconds for a file transfer to complete, the same user would have less tolerance for similar delays when interacting in real-time within a VE.

  For DVEs, latency requirements between the output of a packet at the application level and input of that packet at the application level of another user should be sufficient to satisfy human perception. Hence, distributed multi-user VEs often require a latency of no more than 100 msec [5].
- *Real-time packet delivery*: Distributed VE systems are interactive applications, meaning that they must process real-time data input from users. If the DVE is to emulate the real world, it must operate in real-time in terms of human perception. To be effective, the system must present each other with the illusion that the entire environment is located on the local machine and that its actions are having a direct and immediate impact on the environment. The network should ensure that interactions work in real-time with low packet loss.
- *Reliability*: Reliability means that the system can logically assume that data sent are always received correctly, thus obviating the need to periodically re-send the information. Packet loss is a crucial issue in some cases such as a bank credit transfer at a virtual mall in the VE and should be eliminated to provide a reliable means of communication. The users in a multi-user VE should be able to assume that messages arrive at their destination successfully. The reliability of the best-effort datagram delivery service supporting the distributed VE should be such that 98% of all datagrams are delivered to all intended destination sites, with missing datagrams randomly distributed [6].

- *Multicast*: Distributed VEs involve many-to-many interactions and go beyond the client-server paradigm. They require a capability of multicasting and managing multicast group membership at network level. This multicasting capability must be scalable to hundreds of sites, and potentially, to tens of thousands of users. Multicast group members must be added or removed dynamically, in less than a second, at rates of hundreds of membership changes per second. The network should support a mixture of best-effort and reliable multicast transports. Object attributes that often change continuously, for example position and rotation of the object, could be multicasted with best-effort whereas state updates like object discovery and removal need to be multicasted reliably.
- *Secure Networking*: Users in the distributed VE must be able to assume that their messages are safe from tampering, diversion, and exposure. The most obvious area where security is required in the network is in the exchange of end-to-end information. Imagine a user shopping at a virtual mall, using the Internet as the message distribution mechanism in the VE, ordering an item and providing a credit card number. The user should be confident in the sense of the credit card number being safe from exposure. Per-packet encryption and authentication mechanisms are needed for secure electronic commerce transactions or classified military simulations.
- *Service Differentiation*: The network protocol must be able to associate packets with particular service classes and provide them the services specified by those classes. It is desirable in distributed multi-user VE applications to treat different packets differently in the network. For example, state updates related to new users joining the virtual world are more important and urgent than event updates and can be assigned to a higher priority traffic class. Different levels of service should be provided to individual traffic flows and to some objects if necessary.

2.2 Exploring IPv6 for multi-user VEs

The current protocol used in the Internet, IPv4, is far away from satisfying the network service requirements of distributed multi-user VEs. Although there are some applications that provide collaboration among VEs over the Internet, they suffer from low quality of service, unsecured transactions, and design complexity.

IPv6 has been designed to enable high-performance, scalable internetworks that should operate as needed for decades. IPv6 corrects some of the inadequacies such as the inefficient routing of IPv4, and offers a number of enhanced features including a larger address space and improved packet formats.

We are motivated to use the next generation Internet protocol, IPv6, as the major drawbacks of IPv4 are addressed with the advent of IPv6. Multicast communication, QoS support, authentication extensions for network security, and privacy capabilities are added to the Internet thanks to IPv6 [7]. Among IPv6 features, the most important are undoubtedly a simple and rational design that allows it overcome all IPv4 limits, a practically infinite addressing space, the possibility of auto-configured hosts, an efficacious support for security and mobility of nodes, a design more suitable to transport real-time traffic, and the possibility to implement a gradual and painless transition from IPv4 to IPv6. We will now give a brief review of the new features in IPv6 that could be useful in designing multi-user VEs, and how we use them in VESIR-6. Note that this is not a complete review of IPv6, the reader is referred to look at [7] for the complete IPv6 specification.

2.2.1 Multicasting

Multicasting provides a very flexible and powerful mechanism for distributing messages to a large number of users with minimal network load and distribution effort. It takes advantage of the efficiency obtained when the network can recognize and replicate information packets that are destined to a group of locations. The network itself is responsible for providing duplicate copies to all destinations, thereby greatly reducing the amount of information flowing into and through the network. Any practical network needs multicasting to implement the required distribution of data to all participating users in a distributed VE.

The possibility of implementing multicast transmissions on the Internet was developed in 1988 with the advent of class D IPv4 addresses [8]. One of the limiting factors with the present IPv4 multicast is the optional nature of this multicast, particularly with respect to the routers. Islands of multicast-capable networks are connected over the Internet using tunneling and build the Multicast Backbone (MBONE) [9]. The historical channel rate limitation has restricted multimedia sessions to a largely unacceptable production quality. The historical overall rate constraint and the lack of a coherent

scoping mechanism have limited the number and diversity of simultaneous sessions. Much of the current MBONE is built upon uncoordinated multicast tunnels that are inconsistent with the underlying unicast topology. The use of tunnels, while enabling the initial deployment of multicast in the Internet, appears to limit its potential. The existing routing technology uses a flat routing architecture as that does not scale to a generally useful level.

There are currently only 2^{28} multicast addresses available in IPv4, and the multicast packets travel over the network as long as their TTL value has not reached down zero. IPv6 extends IP multicasting capabilities by defining a much larger multicast address space and a scope identifier that is used to limit the degree to which multicast routing information is propagated throughout an enterprise. Multicasting is an important feature of IPv6, and it actually replaces the IPv4 broadcast feature by supporting both functions.

It provides multicast capabilities that do not rely on a certain type of media being used and, in addition, requires that all routers implement those multicast capabilities. It provides a multicast service that is able to provide low delay many-to-many communication for groups consisting of thousands of hosts. Multicast is so powerful and preferable in IPv6 that broadcast addresses in IPv4 have been replaced by multicast addresses in IPv6. The management of multicast groups is ported to ICMPv6, and the IGMP protocol will become obsolete.

The availability of a large addressing space reserved for multicast addresses in IPv6 will improve the support of real-time multimedia applications. DVEs would benefit from the availability of the ubiquitous IPv6 multicast service.

2.2.2 QoS Support

Network-layer Quality of Service (QoS) products are still in the planning stage, but IPv6 lays the foundation so that a wide range of QoS functions may be made available in a highly open and interoperable manner. Under the IPv6 scheme, software compatible with quality-of-service plans will have to be included on every networked device, such as routers and switches, and in every computer that runs applications. The evolving IEEE 802.1p [10] and 802.1q [11] standards will define how applications ask for specific quality-of-service handling across a network. The standards had their genesis in the

RSVP (ReSource reserVation Protocol) [12], the User-to-Network Interface protocol [13] designed for ATM traffic, and the real-time transport protocol described in [14].

Multi-user VE applications frequently generate real-time traffic that is sensitive to queuing delays and to losses due, for example, to network overloading. The best-effort traffic is the only type of traffic that has been used on the Internet till now. It is based on the idea that network's task is to do everything possible to deliver each IP packet, without guaranteeing that packet is delivered within the required delivery time.

IPv4 performs its best-effort to deliver packets, but it does not guarantee anything at the upper layer, neither in terms of percentage delivered packets nor in terms of time used to execute the delivery. In short, IPv4 does not have a built-in concept of QoS. This is not suitable for applications such as multi-user VEs that have to provide real-time user interaction. IPv6, while remaining faithful to the IPv4 connectionless origin, introduces the concept of flow as a better integration mechanism toward QoS concepts and with RSVP.

The next generation Internet protocol is designed to have per-packet basis differentiated QoS without adding a new signaling protocol. The IPv6 specification states that every IPv6 header contains an 8-bit Traffic Class field and a 20-bit Flow Label.

2.2.2.1 Traffic Class

The traffic class field identifies and distinguishes between different classes or priorities of IPv6 packets. It enables the source node to differentiate packets it generates by associating different delivery priorities to them. This field is currently used by Differentiated Services to enable scalable service discrimination in the Internet without the need for per-flow state and signaling at every hop. Differentiated Services maps the traffic class field to a particular forwarding treatment, or per-hop behavior (PHB), at each node along its path [15].

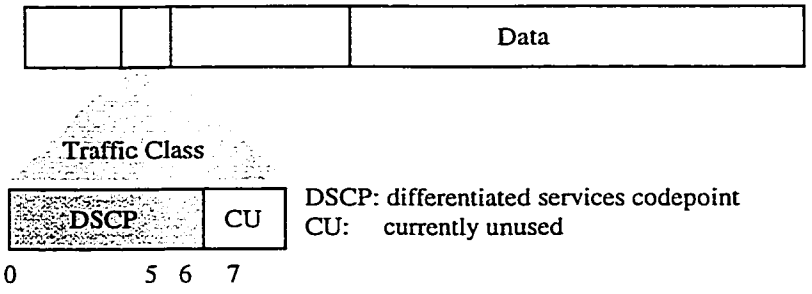

Figure 1: Traffic class field in the IPv6 packet header.

Even if an application does not request a QoS, differentiating the traffic generated by principal applications as a function of their real-time requirements is possible.

The 8-bit traffic class field can differentiate 256 potential traffic priorities, although only eight of them has been defined so far.

2.2.2.2 Flow Label

The IPv6 specification introduces the concept of traffic flows for better QoS management and to simplify the implementation of IPv6 on ATM. While a simple traffic class field, as part of the IPv6 protocol, can provide a QoS indication, the aggregation of packets to form a flow reduces the processing burden on the routers and eases the implementation of flow-based reservation mechanisms, such as RSVP, on top of IPv6 [16]. A flow is a sequence of packets in some way correlated and that therefore must be treated coherently by the IP layer. Packets belong to the same flow on the basis of parameters like the source address, the destination address, the QoS, accounting, authentication, and security.

Flow label is used by the source to label sequences of packets for which it requests special handling by IPv6 routers, such as quality of service above best-effort or real-time service. Flow label is a random integer that is used as a hash key [17]. In conjunction with the source address, a globally unique flow identifier is created to identify traffic flows in case of collisions. This results in an efficient look-up process far superior to the traditional association of source address, destination address, as well as possible protocol identifier and port number in IPv4.

IPv6 flow labels can be used to identify to the network a stream of packets that needs special handling above and beyond the default, best-effort forwarding. Flow-based routing could give internetworks some of the deterministic characteristics associated with connection-oriented switching technology and telephony virtual circuits. For example, desktop video or audio streams could be given a flow label that tells routers they need a controlled amount of end-to-end latency. Flow labels can also be used to give traffic flows a specific level of security, propagation delay (e.g., satellite transmission), or cost.

An application that wishes to have a special handling for some packets could mark those packets or datagrams with the same flow label and set the traffic class field to the required service level. When an IPv6 node receives a packet, it examines the flow label field in the IPv6 header, and knows what the packet needs in terms of QoS.

The flow label could possibly be used to associate a particular flow with a specific reservation. The presence of a flow label in data packets may also help in expediting traffic through intermediate nodes that have previously established path and reservation states for a particular set of flows.

There is still ongoing research on the usage of the flow label. RFC 1883 specifies that problems related to flows are still experimental and subject to change when requirements for the Internet flow handling will become clearer.

Flow-based information is used in VESIR-6 for managing information flows at precommitted performance levels. QoS support over IPv6 provides a reliable means of collaboration among the objects within the VE.

2.2.3 Network-level Security

Encryption, authentication, and data integrity safeguards are increasingly a standard aspect of enterprise internetworking. The current Internet has a number of security problems and lacks effective privacy and authentication mechanisms below the application layer. The security in IPv4 is today managed through particular routers or computers performing the role of firewalls. These devices, however, introduce a number of new problems, including restrictive network policies, performance bottlenecks, and limited connectivity to the Internet or even between divisions of the same company. Traditionally, vendors in the IPv4 arena have been less than successful in adding robust security features to network layer components. This is largely due to the lack of interoperability caused by proprietary security extensions.

IPv6 remedies these shortcomings by providing native data security capabilities that are based on its flexible header extensions. Security features in IPv6 have been introduced mainly by way of two dedicated extension headers: the Authentication Header (AH) and the IPv6 Encapsulating Security Payload (ESP), with complementary capabilities [18]. AH enables the authentication of IP traffic for security purposes, and ESP fully or partially encrypts IP packets. The major advantage of network-level security in IPv6 is that the security services provided by the network layer are transparent to both applications and users.

2.2.3.1 Authentication Header

The IPv6 Authentication Header ensures that the data really have been sent by the specified source and they arrived at the intended destination [19]. This prevents malicious

users from configuring an IPv6 host to impersonate another in order to gain access to secure resources. It is used when the integrity and authenticity of the IPv6 packet or its payload must be protected but not necessarily the confidentiality of the packet itself. For example, in a money transfer at a virtual mall, the user wants to be sure that the amount being transferred is not compromised but she does not care that someone eventually learns what the amount is.

The native authentication of IPv6 gives the industry a standards-based method to determine the authenticity of packets received at the Network Layer. Because the authentication headers in IPv6 are defined in IETF standards, it is highly likely that network products from different vendors will achieve interoperable authentication services. IPv6 implementations are required to support the MD5 algorithm [20] for authentication and integrity checking, but since the specification is algorithm-independent, other techniques may be used as well. IPv6 uses a standard method to determine the authenticity of packets received at the network layer, ensuring that network products from different vendors can use interoperable authentication services. While the extension is algorithm-independent and supports many different authentication techniques, the use of keyed MD5 authentication algorithm is required to help ensure interoperability within the global Internet.

2.2.3.2 Encapsulating Security Header

AH does not provide privacy or confidentiality of data, so this is accomplished with another standard header extension that provides end-to-end encryption at the network layer. For this purpose, IPv6 provides another security extension header, namely the Encapsulating Security Payload (ESP) [21]. It is used to encrypt and encapsulate either the transport layer payload or the entire IPv6 packet. ESP provides fields that carry encryption keys and other handshaking information, enabling interoperable encryption of the payloads in IP packets. Data confidentiality and privacy in IPv6 is provided by this header extension for end-to-end encryption at the network layer. Encryption is done on the sender side and decryption on the receiver side.

This mechanism provides integrity and confidentiality to IPv6 datagrams thereby ensuring that only the intended recipient can convert the data to a readable form. The default algorithm used for encryption is DES-CBC [22].

2.2.3.3 Risks Addressed

IPv6 security extensions address the following problems:

- *Packet sniffing:* A major hole in Internet security is the widespread deployment of traffic analyzers and network sniffers which can surreptitiously eavesdrop on network traffic. Due to network topology, IP packets sent from a source to a specific destination can also be read by other nodes, which can get hold of the payload (for example, passwords or other private information). For example, imagine that an intruder eavesdrops on a communication line to capture passwords transmitted unencrypted. Having captured a valid password, the intruder can use it to masquerade as a legitimate user. Using the ESP option, the data traffic may be encrypted so that the intruder will not be able to capture data as clear text.
- *IP spoofing:* Using the Source Routing security hole of IPv4, an intruder can fake his IP address to masquerade as a trusted host when address-based authentication is used. Such source-address masquerading (spoofing) is among the techniques that could be used to obtain valuable financial and corporate data. IP addresses can be very easily spoofed both to attack those services whose authentication is based on the sender's address (as the rlogin service or several WWW servers) and to supply wrong information to subvert the logical organization of the network (for example, by forging false ICMP messages of the type "destination unreachable" or "redirect"). IPv6 has a careful management of Source Routing and use cryptographically strong authentication techniques, instead of address-based authentication, to protect against IP spoofing attacks.
- *Connection hijacking:* After the original source has been strongly authenticated, whole IP packets can be forged to appear as legal packets coming one of the two communicating partners, to insert wrong data in the existing channel. The encryption facilities of Ipv6 protect against these type of connection takeovers since the intruder does not know the session keys required to encrypt or decrypt the data stream.
- *Denial-of-service:* A sequence of connection request messages or big sized packets might overflow the available buffer space. Examples include e-mail bombing and TCP SYN flooding. IP spoofing is known to be one of the most common forms of this kind of attacks; with IPv4 it is typically impossible for a server to determine where packets are being received from the legitimate end

node. If each packet is properly authenticated, as the case for IPv6, a hacker can launch this type of attack only with his true IP address, which would reveal his identity and location.

The security architecture of IPv6 provides a secure network infrastructure that protects data on per-packet basis. The IPv6 security functions embrace extensions for authentication, data integrity, and confidential communication. These features could be exploited in designing a multi-user VE and in constructing a secure communication infrastructure for large-scale VEs over the Internet. IPv6 security headers can be used directly between hosts or in conjunction with a specialized security gateway that adds an additional level of security with its own packet signing and encryption methods. By having security features implemented at the network layer, the use of DVEs for electronic commerce applications and military simulations will spread over the Internet.

2.2.4 Dynamic Load Balancing and Application Robustness

IPv6 introduces a new type of address called "anycast". These addresses are group addresses in which the only member of the group to respond is the "closest" to the source. Conceptually, anycast is a cross between unicast and multicast: Two or more interfaces on an arbitrary number of nodes are designated as an anycast group. A packet addressed to the group's anycast address is delivered to only one of the interfaces in the group, typically the nearest interface in the group, according to current routing protocol metrics. The use of anycast addresses in DVE applications is potentially very interesting because the closest server can be accessed by an anycast address. This feature provides dynamic load balancing among multiple servers. Furthermore, if one of the servers goes down, the next nearest server automatically will receive the traffic, achieving a robust service.

Anycast addresses do not have separate addreessing space; they simply are unicast addresses assigned to more than one interface. When an anycast address is assigned to an interface, it must be explicitly configured to know that it is an anycast address; this information is usually specified by a qualifier at the time of the assignment.

Anycasting is a new service, and its applications have not been fully developed. The only anycast address defined up till now is the Subnet Router anycast address, intended to identify a set of routers connected to a given link [23]. There is little experience with the management of anycast addresses, and most vendors have not implemented use of them yet. As anycast matures, it may become an important method

for allowing endstations to efficiently access well-known services, mirrored databases, Web sites, and message servers. Essentially, this is a highly flexible and cost-effective method of application load balancing. Anycast services could even be used to provide redundancy in the routing process by assigning all the DNS servers in an enterprise the same anycast address. With so many application areas, it is clear that in the future anycast groups would be in extensive use and vital part of the networks.

DVEs that deploy a client-server architecture needs a means of load balancing since the server usually becomes a bottleneck. The most efficient load balancing could be achieved at the network layer, with the minimum latency. This capability is provided with IPv6, by means of anycast addresses. A set of servers can share the same anycast address, and service requests addressed to this anycast address are relayed to the less loaded server. The use of IPv6 anycast addresses for a set of servers minimizes the number of hops as well as the latency to locate and access the server resources.

Anycasting provides a versatile and cost-effective model for enabling application robustness and dynamic load balancing. We have not used this feature in designing VESIR-6 since our IPv6 implementation does not support the management of anycast addresses yet.

2.2.5 Fast Packet Forwarding

IPv6-compliant software has a trick of its own to eliminate a latency-inducing transmission phenomenon called packet fragmentation. Fragmentation is a major source of high latency under IPv4. As WANs become more diverse, the impact of fragmentation will increase. An Ethernet adapter typically sets the maximum payload inside an Ethernet packet at an efficient size of 1,024 bytes. It's likely, however, that somewhere along the transmission path some link will have a smaller payload size. For example, an ATM switch in the network backbone will divide the data carried by an Ethernet packet into 20 ATM cells of 53 bytes each, with a 48-byte payload of program data in each cell. At some time, a cell will likely be dropped or delayed. Since the single delayed ATM payload is part of a much larger packet, the bigger packet cannot be reconstituted until the data lost in the smallest cell arrives. The entire packet is delayed and might even be thrown out because the smallest part didn't arrive on time. Of course, this delay raises the latency.

Under IPv6, though, software in the originating computer checks the path to determine the maximum payload size possible that would not need to be fragmented by

any node through he network. Then, if necessary, the originating host does its own division of data. This ensures that the packet would travel along its path without any intervention, which would lead to increased latency, by intermediate nodes.

2.3 Transition to IPv6

There have been multiple efforts to extend life of IPv4 incrementally with evolutionary changes to the protocol standards. One such example is the introduction of CIDR concept, that allows subnetting IP addresses and assigning partial subnets to small companies, in order to prevent the address space exhaustion [24]. A more recent development is the network address translators (NAT) that preserve address space by intercepting traffic and converting private intra-enterprise addresses into one or a few globally unique Internet addresses [25]. NAT devices, on the other hand, negatively affect the end-to-end viability of emerging Internet applications; they can only handle a limited well-known applications.

We do not know how long IPv4's life can be extended by these techniques. Furthermore, the development of new and innovative Internet applications, such as real-time interactive communication and multi-user VEs, electronic commerce applications that require strict security mechanisms without performance degradation, wi ll make the transition to IPv6 inevitable.

Many of the advanced features of IPv6 bring direct benefits to end-user VE applications. For example, applications will find the mandatory multicasting capability, QoS differentiation, flow-based traffic aggregation, encryption, and authentication services available as an integral part of the IPv6 stack.

2.3.1 Dual Stack Approach

The IETF decided to design a migration strategy based on a "dual-stack" approach. The dual stack approach consists of providing hosts and routers with IPv6 and IPv4 protocol stacks. All IPv6 nodes are required to have an IPv4-based protocol stack to be able to continue communicating with pure IPv4-based nodes. This dual-stack architecture and tunneling techniques that encapsulate IPv6 packets in IPv4 packets and transfers them to the next IPv6 node via IPv4-only routers enables incrementall and simple migration from IPv4 to IPv6 and ensures that IPv4 and IPv6 will, in future, coexist in the Internet.

IPv4 packets have a protocol type equal to 0800H (800 Hexadecimal), and IPv6 packets have a protocol type equal to 86DDH. When a station receives a frame from its local network, the Protocol Type allows it to distinguish whether the frame contains an IPv4 or IPv6 packet.

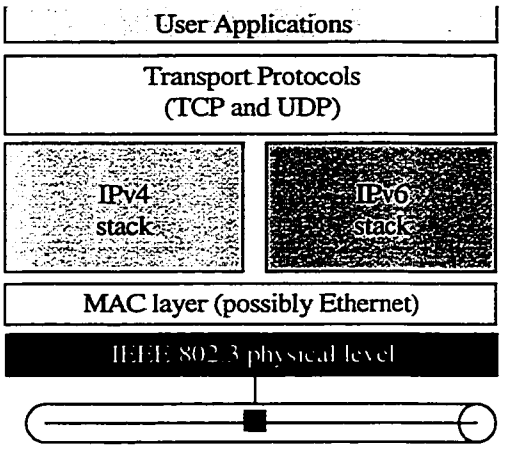

Figure 2: Dual stack approach will make the transition to IPv6 easier.

Dual-stack machines can use totally independent IPv4 and IPv6 addresses, or they can be configured with an IPv6 address that is IPv4-compatible. The following is a simple description of the way the dual stack approach operates:

- If the destination address used by the application is an IPv4 address, then the IPv4 protocol stack is used.
- If the destination address used by the application is an IPv6 address with an embedded IPv4 address, then IPv6 packet is encapsulated inside an IPv4 packet.
- If the destination address is an IPv6 address of any other type, then IPv6 protocol stack is used. First, it determines whether the destination has an IPv6 direct connectivity by looking at the routing table entries. If not, then the packets are encapsulated in IPv4, and sent over the default automatic or configured IPv6/IPv4 tunnel.

2.3.2 Automatic Tunneling

Automatic tunnels use "IPv4-combatible" addresses, which are hybrid IPv4/IPv6 addresses. A compatible address is created by adding leading zeros to a 32-bit IPv4 address to pad it out to 128 bits [26]. When traffic is forwarded with a compatible address, the IPv6 stack automatically addresses encapsulated traffic by simply converting the IPv4-compatible 128-bit addresses to 32-bit IPv4 address. On the other side of the tunnel, the IPv4 header is removed to reveal the original IPv6 packet.

IPv4-combatible addressing means that administrators can add IPv6 functionality to their networks while initially preserving their basic IPv4 address and subnet architecture. Automatic tunneling allows IPv6 hosts to exploit the IPv6 enhancements, including flow labels, authentication, encryption, multicast, and anycast, over the existing IPv4 network infrastructure without the need to IPv6 address space.

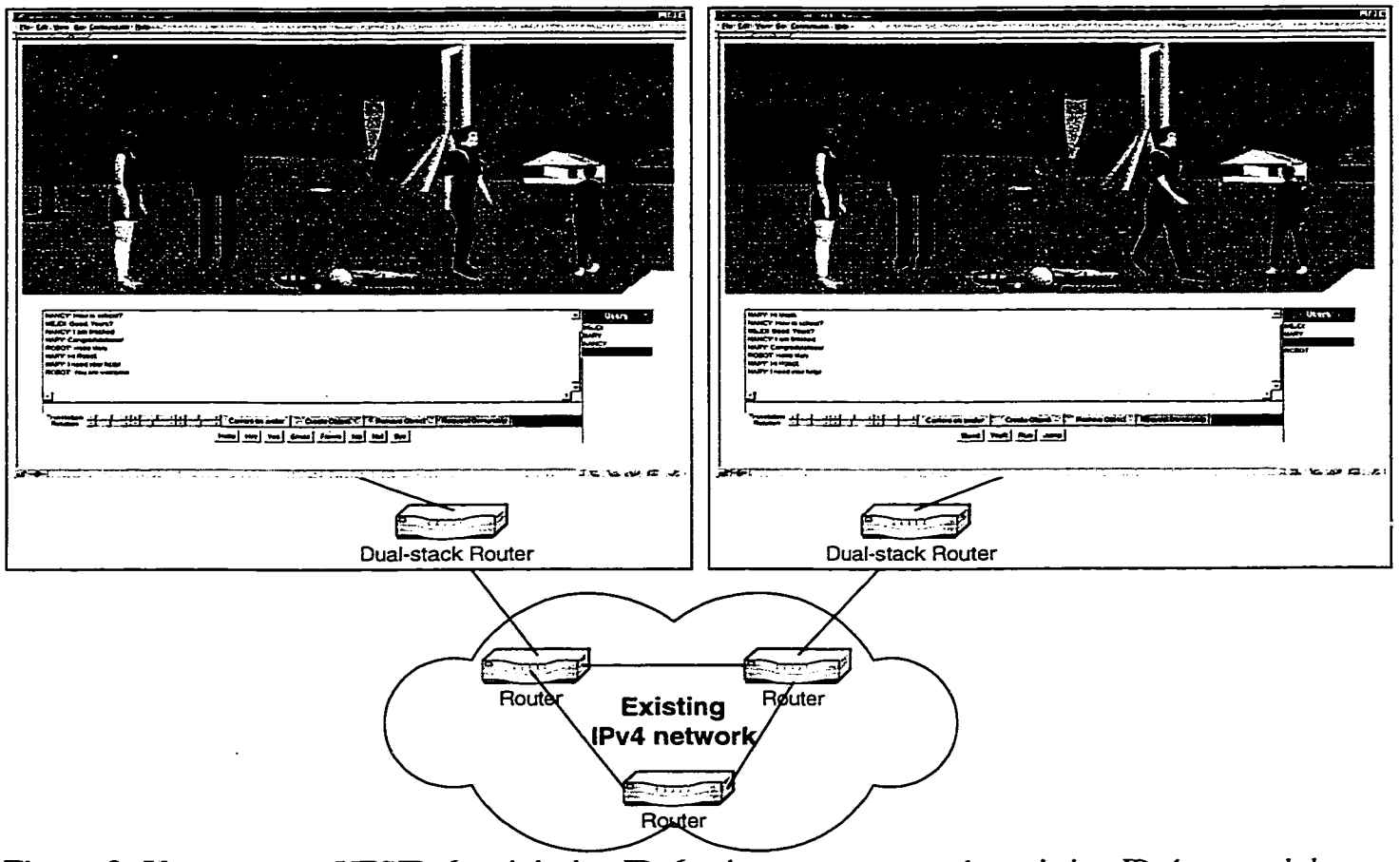

Figure 3: Users can run VESIR-6 and deploy IPv6 enhancements over the existing IPv4 network by means of tunneling.

2.3.3 The 6BONE

Although the IPv6 protocol has been accurately designed and discussed thoroughly, it needs to be tested before being deployed. Until protocols of the IPv6 stack will be widely available and tested, with particular reference to the interoperability of implementations, ISPs and users may not want to migrate to IPv6 in order to avoid risks. For this purpose, IETF initiated a worldwide IPv6 testing and pre-production deployment network, called the 6BONE, was created so that users could acquire experience and test the IPv6 protocol stacks.

The 6BONE project is a spontaneous derivation of the IETF IPng working group and its aim is to implement and test IPv6 protocols with the final goal of replacing IPv4 with IPv6 on the Internet [4].

The goal of 6BONE is to provide an environment in which the transport of IPv6 packets can be tested and users are allowed to gain the required experience. 6BONE is trying to involve as many ISPs and users as it can to spread the experience on IPv6 as much as possible and to create an easy migration to IPv6.

6BONE had already reached approximately 400 sites in 41 countries. There are over 50 IPv6 implementations completed, and over 25 are in test on 6BONE. With the growth of reliability and routing of IPv6 packets on routers, IPv6 will be available by default on new routers and on updated software releases, and 6BONE will be transparently replaced by an IPv6 global connectivity offered by ISPs and by user networks

2.3.4 Ease of Transition

Some characteristics of IPv6 are designed to simplify the migration. For example, IPv6 addresses can be automatically derived from IPv4 addresses, IPv6 tunnels can be built on IPv4 networks, and all IPv6 nodes are required to follow the dual stack approach. This way, users can immediately benefit from many advantages of IPv6 while maintaining the ability of communicating with IPv4 networks.

These IPv6 features could easily be exploited in designing DVEs and provide the necessary means to construct a scalable communication infrastructure for large-scale DVEs over the Internet. Consequently, there is no reason to delay updating to IPv6.

3 Designing a Network Communication Architecture for VESIR-6

A large-scale distributed VE design proves challenging and requires innovations at various layer of the system architecture. The most significant among these are changes required at the network layer, which defines the services exported by a network. For achieving large-scale DVE applications, the available communication bandwidth should be used as efficiently as possible.

Distributed VE applications demand that each user sees a consistent virtual world view, that users are able to interact closely with one another and with other entities in the virtual world, and that maximum realism is provided by hiding the distributed nature of the applications from users. The information flow in distributed virtual environments is handled by the underlying communication infrastructure. The performance of the system depends on the network's response time, which may be highly variable, particularly on the Internet. The real-time interaction feature of distributed multi-user VEs drive network requirements with tight limits on latency between hosts. The simulation must be sufficient to satisfy human perception. The design of network communication infrastructure is important to minimize latency, optimize network resources, and support large-scale virtual environments.

One of the main design goals of VESIR-6 is that it should support user connections over low-bandwidth Internet connections. However, achieving multi-user interactivity in VEs over the Internet is a great challenge. In designing VEs over the Internet using low-level bandwidth modem connections, there might be instances when there is not sufficient capacity to provide all users with their perception of required communication capability. An important issue to be considered in designing the network infrastructure is how to minimize latency over low-bandwidth connections. The current standard of 28.8 or 56kbps modem connections limits the multi-user VEs provide real-time interaction. The network infrastructure is of particular importance in a shared multi-user VE with limited communication resources, low-level bandwidth, and possible degradation of service.

The most difficult requirement for designing the network architecture is the need to examine the complex interactions between the applications in the VEs, the object's

behaviors in the environment that influence the network traffic, and the protocols that connect the members over the network. We will first investigate the types and characteristics of messages that need to be distributed over the Internet to support collaboration among VEs and then look at different network architectures that could be used in VESIR-6. After identifying the network transmission needs and classifying the packets sent over the network, we investigated the possible network architectures to be used in VESIR-6. Because of the complexity of this undertaking, we decided to adopt an incremental approach to the design of our network architecture. Performances issues, in terms of tracker latencies and communication speed, were considered to choose the most suitable network architecture. We then explain how we hide the network from the application by providing an interface for data exchange.

3.1 Identifying the Network Transmission Needs

In order to design the network communication architecture, we have to first analyze the application's needs from the network layer: "What kind of data needs to be transferred, how frequently they occur, what are the reliability or delay constraints?". Then, we can make use of the appropriate protocols considering those requirements.

3.1.1 Characteristics of data sent over the network

The data sent over the network in VESIR-6 have a variety of characteristics. At the initialization phase, i.e., when the user is trying to connect to the system, the virtual world together with its contents should be sent to the new user. These data are very large and contain information about the members in the system and the objects they possess, and the state of these objects. Therefore a reliable connection-oriented transmission is needed during transfer of world contents. This is achieved using TCP/IPv6 during initialization.

After the initialization phase, the other members need to know the new member's presence. The control commands carry this kind of information. These messages are composed of state update and control information. Examples are joining or leaving the group, discovering and adding an object, transferring ownership of an object, etc. Control commands occur once in a while and needs to be transferred reliably over the network. Most protocols, including TCP, achieve reliability by using an acknowledgment scheme. This technique raises many questions in a multicast environment; should all the members reply to each message, is there a need for a server to keep track of the messages and

acknowledging the sender after receiving acknowledgement from all members in the group, and what would be the total latency when the message travels all the way from originator to the server, from server to each member, and the time until every member acknowledges the message, and finally, the time it would take the acknowledgement to travel from server back to the originator of the message. Although there has been some reliable multicast frameworks proposed, they are not widely used due to their complexity, and most of them are designed specifically for non real-time data transfer. For real-time interactive multi-user VEs, the delay introduced by the round-trip time until the acknowledgment arrives, is unacceptable. Reliability for control messages is achieved by exploiting the IPv6 QoS capabilities, as explained later.

During real time operation each user models the behavior of the simulated objects for which it is responsible. Communication between objects is a very important issue in a shared virtual environment. People convey information in the real world through symbolic (such as writing), verbal, and gesture communication. To give a realistic impression to our application, we include both text-based and gesture communication among objects. Whenever specific events occur in the VE or when the the state of the object has changed significantly, the application transmits this new information to the other users running the application on the network. The communication between objects is carried by event update messages of relatively small size. When an object moves in the virtual world, the new position and orientation values are generated as event updates. These updates are distributed to the other members so that they can update the location the object in their screen. These messages are frequent, asynchronous, and do not need much reliability since if one of them is lost the system can pick up the subsequent update and ignore the lost one.

Table 1 summarizes the message types and characteristics used in VESIR-6, and highlights the protocols used for each kind of message and how reliability is achieved.

3.1.2 Message Classification in VESIR-6

In summary, four different transmission needs are identified for VESIR-6. Those are:

- Low latency, best-effort multicast of object attributes that often change continuosly, for example position of moving objects.

- Low-latency, reliable multicast of control information and object attributes that do not change continuously but may change at arbitrary times during the simulation, for example creation of a new object.
- Low-latency, reliable unicast of occasional data among arbitrary numbers of the multicast group. This requirement is for occasional transaction-like exchange of data between two arbitrary hosts in the multicast group, with a low latency that makes TCP connection impractical. Transferring ownership of an object from one user to another.
- Reliable unicast of bulk data between individuals. This requirement is met by TCP. This kind of transmission takes place during initialization, when the virtual world contents are sent to the new user. If users define their own avatars, which could be large in size, this kind of transmission is used too.

Message Types	*Characteristics*	*Frequency*	*Protocol*	*Priority*	*Reliability*
World contents	Large, sent during initialization phase	Once at startup	TCP/IPv6	No priority needed	Reliable since connection-oriented
Control commands	Small, for object discoveries etc.	Once in a while	UDP/IPv6	High-priority	Highly reliable
Event updates	Small, sent when an object changes its location etc.	Frequent	UDP/IPv6	Best-effort	Non-reliable

Table 1: Message classification in VESIR-6.

All of these transmissions take place within the same VE. It is clear that a hybrid of best-effort and reliable multicast is needed to support large-scale VEs, and that low-latency, reliable part of this hybrid is not available in the current Internet. It appears that these requirements will be met by IPv6 deployment.

3.2 Network architectures for VESIR-6

One of the first decisions faced by a multi-user VE designer is to select a suitable communication architecture for the system. This communication architecture defines how data will flow over the network, what possible path it will follow, and how efficiently the scarce resources, such as bandwidth, will be utilized. The choice of communication architecture, therefore, plays a crucial role in the performance and scalability of the system.

We developed and analyzed three different network architectures to support large-scale virtual environments. These prototypes are concerned with the multi user technology abilities of the system to share objects and pass messages between those

objects. Each of these network topologies provides a different level of reliability, functionality, and scalability.

This chapter is incremental in nature, starting with the most basic client-server architecture.

3.2.1 Client-server approach with a central database

Our first protoype used a client-server architecture with unicast TCP/IPv6 communication. There is a central database that holds the virtual world contents and keeps track of the object movements and behaviors. Whenever a new client joins the session, the server adds him to its database. The control or update messages are sent to this server, and then the server forwards them to other members. Figure 4 shows a typical scenario of this architecture.

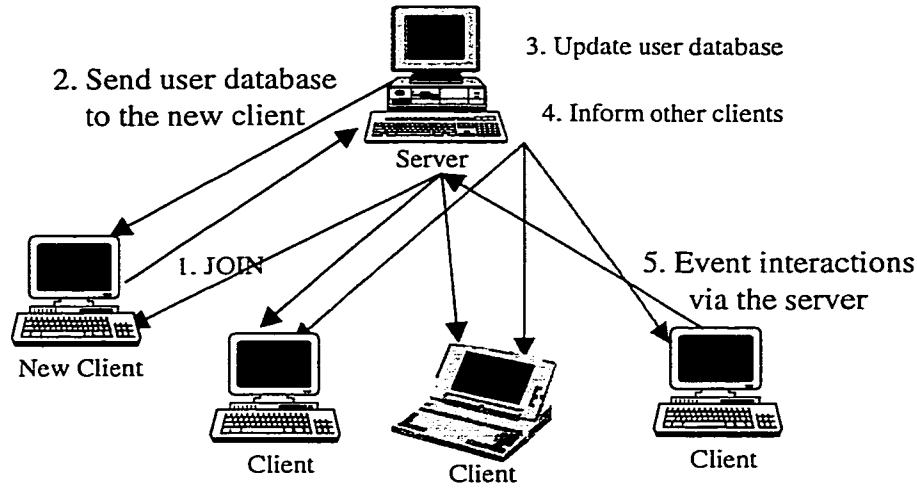

Figure 4: Networking VEs using a unicast client-server architecture.

The main advantage of this architecture is its reliability provided by TCP connections, and the use of a central database to quickly recover from system crashes. There are many other benefits to the central server approach: the design of the system is simpler; management is at a central location and the sponsor of the virtual world has more control over the world.

This architecture, however, three problems. First, the inclusion of a server in each transaction means that a message may not be able to take the shortest path between hosts from source to destination. In addition, the messages must spend time within the server itself, which adds to the total transmission time. This additional time could range from being very short, if the message is simply echoed, to quite lengthy, if database transactions must be performed prior to distributing the message. Interaction speed is

significantly limited, because all messages have to go first to the central server and then to the other users in the group. Second, bandwidth needs are increased somewhat due to the need to send messages to the central server as well as to other users in the group. In comparison to sending messages directly from one user to another, the centralized server approach adds an additional message to the network, and increase the time required for the server to interpret the incoming message, decide what to do with it, and generate an outgoing message. Third, it lacks scalability since the server could easily become a bottleneck both in networking and processing if the number of members and virtual world content increase.

Using a centralized server for message passing within the virtual worlds is obviously limited to tens of participants because of input and output contention. The performance analysis of the system would show the limitations imposed by the network architecture of the system. If all N users in the system update their location and report the change to the server, then a total of N*(N-1) messages per update would be sent by the server.

This architecture can be improved to be scalable over the Internet in two ways. First, more servers could be added to the system and the network load be evenly distributed to them. This design allows dynamic reconfiguration of database and network connections. The major drawbacks of this approach are the consistency of objects among the servers, avoiding more than one server respond to an object query, overhead imposed by continuous information exchange among the servers, and lack of a central site to get information. Another solution is excluding the server from the system and letting multicast communication take care of group membership and state update information. We deployed the latter approach in our next prototypes.

3.2.2 Networking VESIR-6 with IPv6 Multicast

In our second prototype, we deployed UDP multicast over IPv6 to improve the efficiency of the system by avoiding unnecessary duplication of packets. A new member becomes part of a multicast group by sending a "join" message to the multicast address, and a nearby router in turn processes this request by connecting the new member to the group. Members can then multicast a single packet, and it will be replicated as needed and forwarded through the internetwork to the multicast group. This conserves both server and network resources and, hence, is superior to unicast and broadcast solutions. With this method packets are multicasted to all members at once and the delay is reduced

by almost half, compared with our first prototype where messages should first travel to the server that redistributes them to the other members. Users first join a multicast group to become a member of the system, and then all data are sent to this multicast address. This method increases the efficiency of communication and does not restrict us to a particular LAN. Figure 5 shows how easily the collaboration can be established using multicast communication.

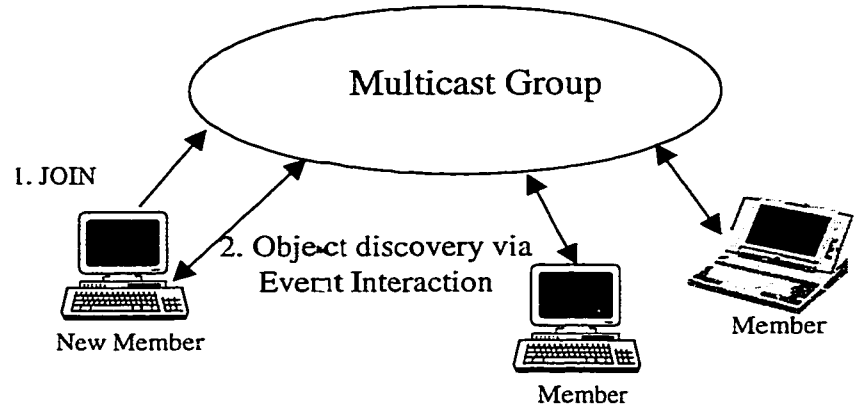

Figure 5: Collaboration using IPv6 multicast communication.

There are many benefits of this architecture. First of all, there is no need for a server, as the group membership issues are taken care by IPv6 routers using ICMPv6 group query, report, and termination messages [27]. All updates of the avatars' movements are sent to the multicast group. This protocol scales to a large number of users since there is no processing burden and database lookups.

Reliability is a major concern with this architecture since IPv6 multicast operates over the UDP protocol, and an upper level architecture is needed to achieve reliability. For event updates reliability might not be a big problem as these updates are generated frequently and if one of them is lost the system can pick up the subsequent update and ignore the lost one, but for the control commands involving state updates some sort of reliability should be achieved. No reliability via retransmissions is provided at the network transport protocol level, as the standard for reliable multicast has yet been adopted by the community [28]. The solution we propose is to integrate IPv6 QoS support to our multicast network architecture as explained later.

In this architecture, since objects are discovered by event interactions, it is not possible to have knowledge of the users connected to the system until all of them perform some movement. When a participant leaves the multicast group, it should inform the other members before disconnecting. This is necessary since when the user disconnects,

his avatar should be removed from the local copies of the world at each member. This method works fine except for the case of system crashes.

This architecture can be improved in two ways. The objects might be required to send some sort of "I am here" messages at periodic time intervals. The other way is to add a group manager to the multicast group and keep a user database as explained in the next section.

3.2.3 Improved multicast communication architecture with a Group Manager

One of the main design goals for VESIR-6 is to support member and object consistency among the virtual worlds. Pure multicast is not enough to make the distributed multi-user VEs consistent, for two reasons. First, members disconnected due to network failures would still appear in the virtual worlds, thinking of them as observers in the system that do not perform any action but watch other users and objects. Second, when a member joins the multicast group, it might not have knowledge of some members and objects in the system until they perform some action and generate an event update. For these reasons, we add a group manager to our architecture to keep track of members and objects they possess. Although there is no need for a database as object discoveries are possible by means of event updates, it will provide us with better information about the users connected to the system during initialization. This is the best method to recover from system crashes quickly. Figure 6 shows the proposed architecture.

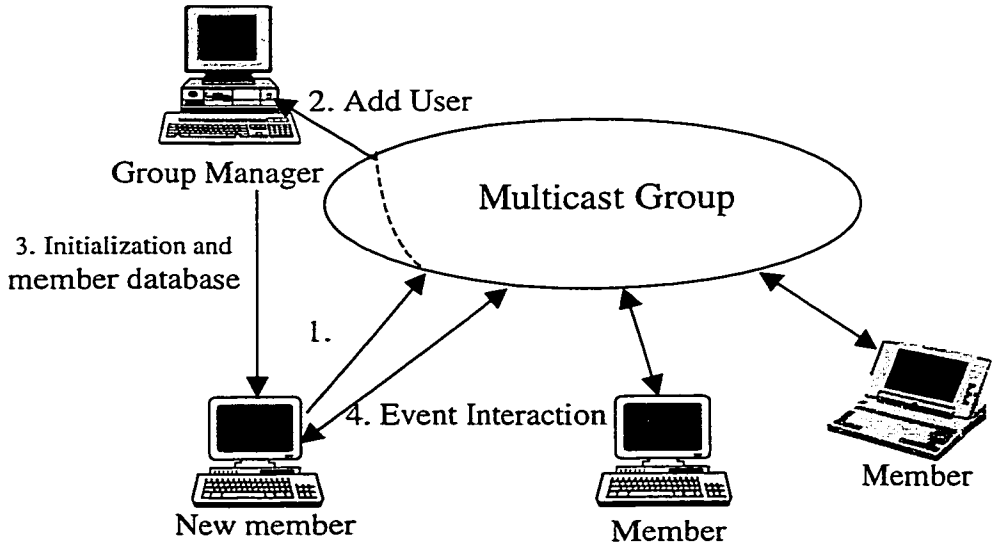

Figure 6: Improved multicast communication architecture with a group manager.

The basic approach of this architecture for achieving rapid and yet reliable communications is to combine the best features of the central server and multicast

approaches discussed before. In this architecture, we use IPv6 multicasting as the communication protocol between all participants of the VE. We added the group manager to the architecture to keep the world content and object database. The reason for including the group manager is that the new members or the members disconnected due to network failures need to restore the contents of the virtual world. This requires a location where the current state of the virtual world is kept or even updated continuously. In our model, this is realized by the group manager. The group manager can be regarded as a place that holds the database and the collection of objects within the system, shares them over the Internet, but has nothing to do with the message distribution. This approach is the most suitable in DVEs, and is deployed in VESIR-6.

This architecture exploits the IPv6 QoS capabilities to discriminate between traffic flows and assigning different priorities to these flows. Object attributes such as event updates of an object, that often change continuously, are multicasted with best-effort. Object attributes that do not change continuously but may change at arbitrary times during the simulation or as a control command from user interface, for example object discovery, transferring ownership of an object etc., have to be multicasted reliably. To accomplish this, we first classified the traffic into two seperate flows and then assigned high-priority to the flow consisting of state updates and control commands, as explained later.

3.3 Performance Analysis

The resource utilization of a networked VE is directly related to the amount of information that must be sent and received by each host and how quickly that information must be delivered by the network. The number of messages exchanged per second and the number of destination hosts for each message greatly places a burden on network resources. The network topology, therefore, is of great importance to efficiently utilize the resources and achieve scalability. The traffic generated and forwarded to the network by VESIR-6 application can be given by the formula (note that text-based chat messages are generally infrequent enough to be neglected in this estimation):

Traffic generated (bits/sec) = (packet size) * (update frequency) * (# of packets)

In this formula, the packet size refers to the total size of the network packet. This includes any headers associated with the protocols used. All of the three architectures described require considerable communications overhead. The update message is 22 bytes, combined with the TCP/IPv6 header, the network packet becomes of size 82 bytes for the first architecture, whereas the multicast architectures requires UDP/IPv6 header, and the network packet becomes of size 70 bytes. The more overhead associated with the first case, is the sacrifice for reliability provided by TCP connections. For speed, we prefer peer-to-peer UDP/IPv6 multicast, with one message in each packet. (Because there is a one to one correspondance between network packets and updates, the terms "packet", "message", or "update" are used interchangeably in this document.)

The recommended practice for communications architecture [29] indicates that the underlying communications structure should provide 100ms or less latency for packet exchange for closely coupled interactions between simulated objects. This requirement is based on human reaction times that have been the basis for real-time operation. The update frequency is determined by this human perceptual limitation and should be at least 10 updates per second.

We can improve resource usage, by applying simple optimizations that reduce the size of each network packet and reduce the number of network packets that are actually transmitted. Although the first two quantities are almost fixed, the number of packets sent per update depends on the underlying network topology. For unicast case, it is N*(N-1), whereas for multicast it is only N. Therefore, deploying multicast communication is essential in a VE to scale to a large number of users.

The performance analysis of the system would show the limitations imposed by the network architecture of the system. If all users update their location and report the change to the server, then a total of N*(N-1) messages per update would be sent by the server. Let us assume that all users are in the same LAN and are connected to the network over IPv6 by 10Mbps Ethernet cards, and each user generates an update message of 22 bytes every 100ms in heavily interactive conditions. Let us calculate the maximum number of users, N, supported by the system. In doing this, we have to consider the behavior of the network under heavy load conditions. The performance of Ethernet networks, which uses CSMA/CD algorithm, suffers from higher load because of the increased likelihood of collision or no transmission. Maximum possible utilization of our CSMA/CD network

can be approximated as 67% [30]. The update messages are 82 bytes[1] including the 40-byte IPv6 header and 20-byte TCP header, and 10 of them are generated within a second. The number of bits/second at the egress point of the server would be 10*N*(N-1)*82*8, and limiting this quantity to (67%)*10Mbps would yield that N should be no more than 32. Hence, this architecture supports at least 32 highly interactive users in worst-case conditions in the local network. Considering the fact that some of the users will remain stationary while others move, we see that the system can support more than 100 users of which only 11 are moving at a time. If we try to extend the network to a WAN such as the Internet, assuming a 56k modem connection, the system is physically limited to 4 users.

For our second and third prototypes, the performance is increased significantly by exploiting multicast communication. Not only the message duplication is avoided but also the UDP header is 8 bytes and smaller than the TCP header, imposing less overhead to the packets. If we assume the same system as before, we have packets of 22+40+8=70bytes that are generated periodically every 100ms. The updates are directly multicasted to the members instead of passing them through the server, hence the network traffic generated for each update is proportional to N. The number of bits/second at the network interface of each member would be 10*N*70*8. Doing the same calculation for worst-case scenario yields that this architecture can support up to 1196 users within the LAN. This is a significant improvement achieved by the multicast technology. Even with users distributed over the Internet, and assuming 56k modem connections, the system supports at least 11 users in heavily interactive conditions.

There are other benefits of deploying multicast network architectures. IPv6 multicast provides scalability to large groups with dynamic membership, and robustness in the presence of topological changes. By reducing the amount of duplicate traffic in the network, it makes applications that uses in one-to-many and many-to-many communication operate in a more efficient manner. This in turn allows the developers of these applications to add more functionality without significantly impacting the network.

Overall, multicast is rapidly emerging as the recommended way to build large-scale VEs over the Internet. It provides desirable network efficiency while also allowing the network partition different types of data by using multiple multicast addresses. Multicast is also an appropriate technique for discovering resource availability. Using a well-known

[1] Although the IPv6 specification favors packets greater than 1500 bytes in size over Ethernet [31], this limitation can be overcome by aggregating the messages and synchronizing them as explained in [32].

multicast address, shared VE participants can announce their presence and learn about the presence of other participants. These features make multicasting preferable even for LAN-based multi-user VEs. Distributed VE applications will be hindered without native multicast support. Distributed virtual reality applications are impossible to build without multicast due to the shear volume of data that needs to be transmitted.

For these reasons, we prefer to use the third prototype in our multi-user VE; but any of them could be integrated to the architecture by means of a well-defined API, as explained in the next section.

3.4 Hiding the Network

In order to be able to choose the most suitable network topology according to the circumstances we abstracted the network function calls within a well-defined API library. The main purpose of this API is to hide the network specific issues from the higher modules. It is implementation-independent, that is, by abstracting the network functions with a well-defined API, we can deploy any kind of topology at the underlying network level. We created a class that implements these network function calls, called "networking module", in our design.

3.4.1 Networking Module API

The networking module provides an interface such that higher level modules can access the network by calling the appropriate functions. This API does not expose the network to the application developer. In fact, the application code will work exactly the same way whether the entire world is residing on a single machine or distributed over an arbitrary number of users on the network. The network function calls needed to share the state of the DVE are encapsulated in the networking module API, and listed below:

- `Join(localIPv6Address, groupManagerAddress, port, username, avatarURL)`

  This function is called when the user wants to join the networked VE. The network module then calls its internal AddMembership() and AddObject() functions.

- `Leave()`

  When the user quits the VE application, this method is called to free the resources associated with the user and the objects she owns.

- `AddMembership(localIPv6Address, groupManagerAddress, port)`

This method is called when the user joins the VE. According to the underlying network topology, the groupManagerAddress could be the server address when using our first network prototype, or the IPv6 multicast address when using the second prototype, and the GroupManager address or the multicast address, since GroupManager is a member of the IPv6 multicast group as well. In the case of many multicast groups, the GroupManager address must be specified. In this case, GroupManager assigns the multicast address to the user.

- `DropMembership(localIPv6Address, groupManagerAddress, port)`
  This method is called if there are many multicast groups, and the user could subscribe to more than one. When moving from a region to another, it might be necessary to change the multicast group from the old region to the new one.
- `AddObject(username, avatarURL)`
  This method is necessary for dynamic object creation. Upon receiving this request, the GroupManager assigns a unique object ID and sends it to the user.
- `RemoveObject(objectId)`
  The objects might be destroyed dynamically. Any user can create new objects, but only the owner of an object can remove it.
- `SetPriority(flowType, trafficClassValue)`
  Sets the priority of the specified traffic flow to the given traffic class value. The `flowType` could be one of the `EVENT_UPDATE` or `CONTROL_COMMAND`.
- `GetPriority(flowType)`
  Returns the traffic class field associated with the specified `flowType`.
- `EventUpdate(objectId, field, value)`
  This method is usually the most frequent one called in an interactive VE. It encapsulates the update type in the field parameter.
- `SendString(objectId, message)`
  This method enables text-based communication among participants.
- `SendPrivateString(objectId, message)`
  This method enables private text-based communication between two users.
- `SendGesture(objectId, gestureIndex, value)`
  If the virtual object has some predefined behaviors, then the owner of the object can set these object behaviors or gestures by using this method. The gestureIndex

determines which behavior of the object is to be set, and cannot be higher than the maximum number of behaviors defined for the object.

- `RequestOwnership(objectId)`
  If the ownership transfer mechanism is used to interact with non-owned objects, then this method has to be implemented. It contacts the owner of the object and asks for ownership trasfer.
- `GrantOwnership(objectId, requestingUserId)`
  This method is issued by the owner of an object that has received an ownership request message, and decided to grant the ownership. After calling this method, the user has no access to modify any field of the object until it requests and gets the ownership back.

The potential benefits of hiding low-level network aspects from application developers include provision of high-level programming abstractions for real-time interactive communications within the shared VE, including QoS specification and ownership management. The networking module can easily be extended by integrating the additional network function calls to the API, hence, making them available to the application.

4 System Architecture

The system architecture of a multi-user VE is related to a variety of areas, including user interface design, graphics and rendering, database development, network protocol design and implementation, and asynchronous systems design. Developing an effective multi-user VE involves managing the interactions among various system components, including the network, processor, graphics display, user interface, databases, and other external information servers. At the same time, the developer must support the desirable goals of heterogeneity, scalability, and easy deployment.

Multi-user VE design is made more complicated because of the services they must provide. Among these services, the most important are ownership management in order to avoid conflicts, and consistency management to provide a coherent view of the virtual world to all participants. DVEs typically must integrate with database systems that store persistent information about the virtual environment. These databases include the world model and detailed information about the user objects in the environment. Multi-user VEs pose several problems such as managing consistent distributed information, guaranteeing real-time interactivity, and contending with limited network bandwidth, processing and rendering resources.

When designing the system, we must consider the overall behavior of the multi-user VE. Rarely can a single aspect of the DVE be changed without affecting the rest of the system in subtle ways. Understanding these interactions among the various system components is vital for effective DVE development. The components of a multi-user VE system interact in many ways, and the designer must regard the application as a unified system. Invariably, trying to optimize one element of the system can adversely impact the behavior of other components. The complexity of the system theoretically increases with its size because of the number of possible interactions among entities. In effect, DVE system architecture development is a difficult balancing act of tradeoffs. This chapter describes how we architect the VESIR-6 system to support multi-user interactions, ensure real-time response, and how to organize the communication infrastructure.

4.1 Design Considerations

The system design of multi-user VEs is an art balancing the performance requirements against available resources. Designing an effective multi-user VE requires

innovations at various levels. The challenges arise in the areas of system architecture and design, and interfacing the system to the underlying network in order to support distribution. There are many considerations that should be taken into account while designing the system, in order for the application to be well-deployed and scalable. The main design considerations can be listed as follows:

- Platform independence and easy integration to the web
- Users should be able to join VESIR-6 from any place at any time
- Supporting low-bandwidth modem connections
- Scalability to a large number of users
- Dynamic object creation, removal, or modification
- Implementation of an ownership mechanism in order to avoid conflicts
- A user friendly interface for controlling objects in the environment
- Updates and changes should be witnessed in real time by remote users
- Text-based communication could be useful for several purposes
- Design generic networking modules in order to deploy any kind of communication architecture in the underlying network.

When designing the system, these goals should be met, although success is challenging and requires innovations at various levels of the system architecture. In the next section, we will see how the system is designed having these considerations in mind, and present the proposed system architecture of VESIR-6.

4.2 Overall system architecture

Multi-user virtual environments involve physically distributed users interacting with each other and with distributed objects via complex highly graphical user interfaces. These factors can result in interaction that suffers from serious and unpredictable delays in system response times. Meticulous system interaction design can alleviate the problems resulting from such delays. We look at several areas of shared virtual world design, including object interaction, avatars and scene rendering, investigating ways of dealing with communicating information, preserving world coherence and providing users with effective real-time interaction. In this section, we will discuss the fundamental system architecture of VESIR-6, and give a high-level view of how it is constructed in a multi-threaded fashion.

There is the virtual world running on the web browser, and the user in front of it. How would we architect the system such that we will end with a multi-user VE? The first thing we need is an effective interface to the system in order to attract the user. This user interface lets the user join the system, create objects, interact with the system, and chat with other users. The second thing we need is a representation of the user in the VE. We refer to those representations and other entities in the world as virtual objects. Thirdly, we need an intelligent agent that will be aware of ownership issues for interaction purposes among participants in the system. Now, the user might approach the computer, and navigate through the virtual environment. But s/he still cannot see the other users in the system, because there is no functionality of informing the user about other participants yet. We have to include another component to the system for message distribution over the network to support interactivity among participants. Now the user can truly interact in real-time with other users in VESIR-6. Therefore, we can conclude that VESIR-6 system is mainly composed of four components: a user interface, virtual objects, interaction agents, and a networking module. The question, now, is how to develop those components and achieve the communication among them, taking the design goals discussed before into consideration.

The main issue in designing the system architecture is to make it efficient by choosing the right set of interactions among the modules. The complexity of the modules and software architecture should be abstracted and a high level description of the interconnectivity should be presented in a form of well-defined APIs. It is important that the system architecture be based on flexible architectures that do not require developers to understand all the complexities inherent in a system. Object-oriented frameworks provide an important enabling technology for reusing both the architecture and the functionality of software components. For this purpose, we divided the functionality of the system into four modules: User interface, virtual objects, interaction agents, networking module. Each module constitutes a separate library with minimal dependencies. As discussed in Chapter 3, the system also needs an external information source that will provide the complete and coherent state of the virtual world content to the users, especially to the new members. We included a group manager in VESIR-6 for this purpose. The proposed system architecture of VESIR-6 is shown in Figure 7.

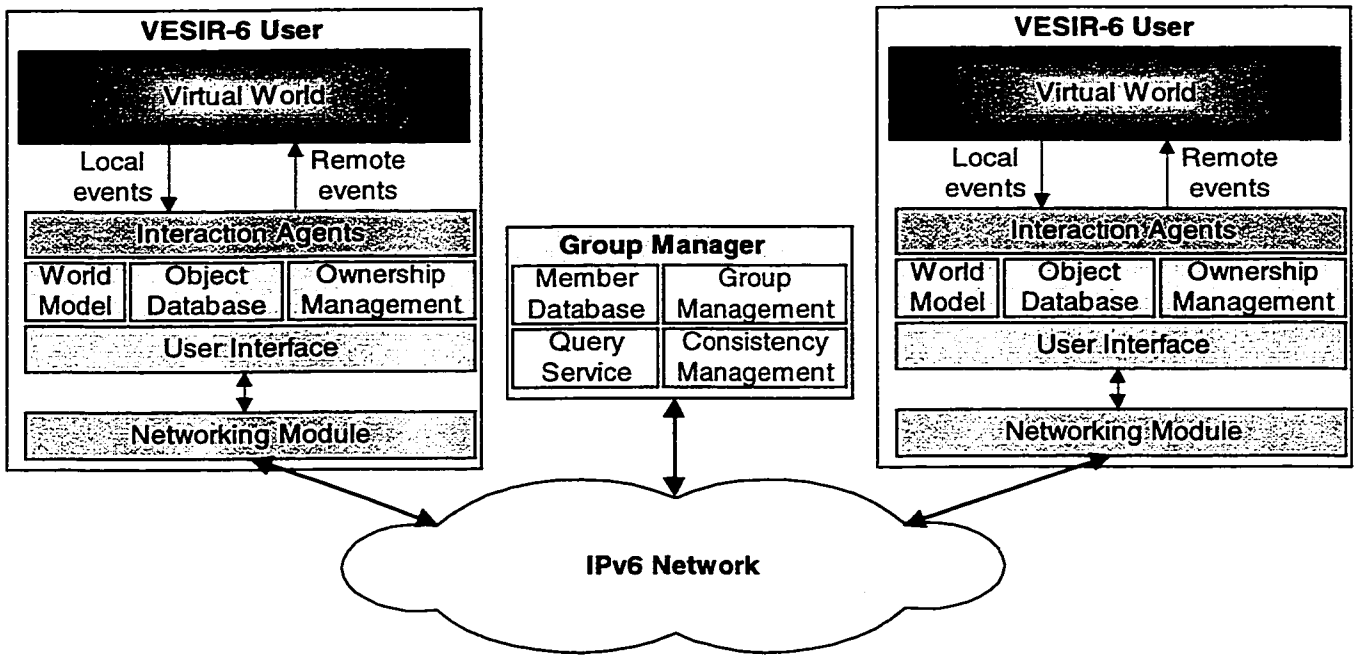

Figure 7: Overall system architecture of VESIR-6.

Each user has a local copy of the data they share, which we call the world model. This data embody the virtual world contents including the objects and users within it. The users can modify the content of the environment by importing new objects into the shared space. Interaction agents control the actions of the objects. To avoid modification conflicts, each object in the virtual world has one user as its owner and only the owner user can modify the object. However, the ownership of an object can be transferred from one user to another. The ownership management part is responsible for these tasks. The ownership management layer keeps track of objects deposited by various users with their access rights and updates. When a user tries to modify an object, this layer sends the update if access rights permit. The local events are detected by interaction agents and forwarded to the networking modules. Before sending the local update, the corresponding interaction agent applies message filtering based on the ownership mechanism. The interaction agents are also responsible for receiving any remote updates and modifying the specific objects in the local replica of the environment accordingly.

The update messages are sent whenever an object updates its attributes or an interaction takes place. The keep-alive messages, which might be useful to maintain consistency, are not used as they waste a lot of bandwidth. To ensure that the object updates sent by a user can be decoded by the receivers, the system must provide the receivers with the knowledge of the objects in the environment. It is possible to include

the relevant objects in each update, but the update messages would then be tremendously large and bandwidth resources would be wasted. This can be done more efficiently, by ensuring that users contain the objects in their local database. Hence, in VESIR-6, the users keep a local replica of the object database. Each object has a unique descriptor, and only the descriptor is attached to the update message in order to identify the particular object. The virtual world is fully replicated at each user side, therefore, during each state update, only the changing part of the database is sent to the clients.

The data sharing is supported via the networking module. The update packets and other information are sent out over the multicast network. Each user maintains a database of objects whose state is updated using the received data. The system relies on the networking module for reliable and real-time data transfer. Networking module separates the communication management from the application design, and thereby reducing the effort required to built a networked VE, and can be used independently from VESIR-6 as well. The generic architecture of the system allows VESIR-6 to use any networking module that deploys any kind of underlying network communication architecture and protocol.

The content of the environment is the same for all users. Individual users may be looking at different part of the virtual space from different viewpoints but they all have the same set of objects loaded in their worlds at a given time. The object consistency is provided by the group manager. Once running, the group manager receives commands from users and in turn updates its world database accordingly. When a new user joins the system, the group manager sends him information about the environment and other users. It might be desirable to divide the world model into regions for scalability. The group manager is designed to provide a query service and ease this kind of partitioning.

The users are able to run VESIR-6 over any web browser with a 3-D viewer plugin. This assures platform independence and easy integration to the Internet. They need a dual-stack machine since the networking is done over native IPv6 communication. There are many IPv6 implementations for Windows NT, Solaris, and Linux operating systems, and most of them are distributed freely [33]. By installing IPv6 to their computers, the users will not sacrifice their IPv4 connectivity, since IPv6 implementations support the dual-stack approach. Hence, users can run VESIR-6 from anywhere and exploit many advantages of IPv6 at no extra cost.

4.3 Virtual objects

The system consists of a static scene, which is populated by some number of dynamic objects. The term object is used to refer to some entity in the virtual environment whose state can change over time. An avatar is a particular type of object that happens to be under control of a human being; in effect, an avatar is the "virtual body" the user inhabits while s/he is in simulation. Avatar objects are either included within the virtual world during initialization or dynamically created at a later time.

New objects, or changes to existing objects at any site, are distributed among all sites. After creating a new object, it is distributed among all users. Each user site provides a database of objects. Each object in the database can be controlled by exactly one user at any given instant.

In order to represent the users in a realistic way inside the VE, the avatar object definitions might include some gestures or animations. For example, if the user is represented by a robot, then the movement of the user in the VE should not be merely consisting of incrementally setting the position of the robot in the specified direction, but also synchronizing the arm and feet behavior of the robot object. This requires access to the joints of the robot object and complex kinematics knowledge. Virtual objects in VESIR-6 have the ability to hide this complexity by providing user predefined gestures or animations. The objects obey a prototype definition, which is explained in detail in Chapter 7. This feature allows reuse of object definitions and an object-oriented approach to the design of complex objects. Based on this feature, we also developed anthropomorphic avatar objects that provide animation of human figures including walking and facial expressions to be used in VESIR-6.

Virtual objects in VESIR-6 are organized in an object-oriented approach, and there is a software agent called VIA for each of them. These virtual object agents stand behind the scene and detect the object movements. They handle the interaction between user interface and avatar objects in the virtual world.

4.4 VESIR-6 Interaction Agents (VIAs)

VESIR-6 objects use intelligent agents, called VIAs, for interaction purposes. VIAs are mainly responsible for receiving events from the user interface and computing how those inputs change the user's appearance within the VE. They also determine how

and when to notify the networking module of these changes initiated by the local user. A VIA holds the profile of its object, based on observing their behavior and attributes.

The objects in the local virtual world should be consistent with those in the virtual worlds of other users. The actions that change the state of the objects have significant role in the maintenance of consistency. For example, information about transferring ownership of an object, adding or removing an object should be propagated to all participants in a reliable way. The interaction agents provide consistency by using an object ownership mechanism, encapsulating the object behaviors, classifying the interactions as state data and event updates and pass them to the user interface to be transferred to the underlying network module and distributed in the appropriate way. Figure 8 shows how agents interact with objects in the virtual world.

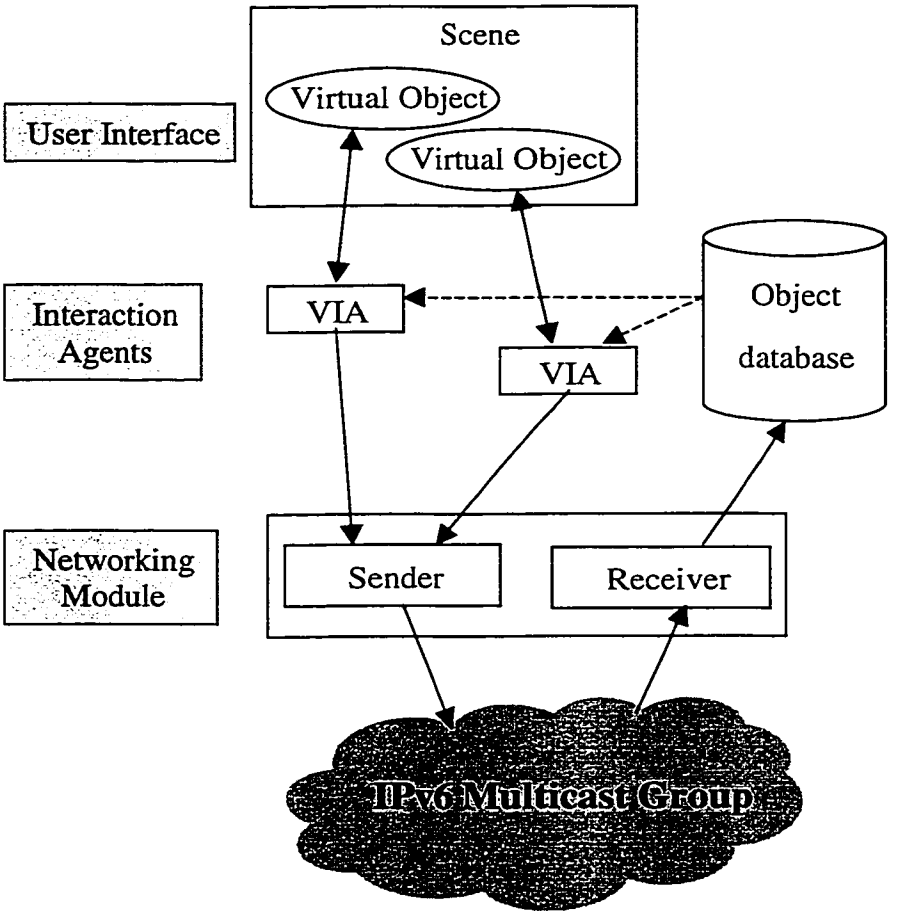

Figure 8: VIAs are responsible for interacting with the objects in the virtual world.

VIAs support local interaction with the objects. VIAs observe the changes on object attributes and perform the appropriate communication actions. They accommodate quick detection of user action and immediate response. To support the most immediate reaction to user input, VIAs employ an "event driven" model for processing user actions.

Where users are interacting in a distributed virtual environment, the actions of each user must be observed by peers with sufficient consistency and within a limited delay so as not to be detrimental to the interaction. The most difficult part of the collaboration in DVEs is how to offer a truly shared environment where objects can be created and modified dynamically in a synchronized manner among participants. To minimize communication processing and bandwidth requirements, VIAs transmit only changes to object attributes instead of the whole object during the interaction period.

VIAs receive events and filter them before passing them to the networking modules. They detect if the user is trying to modify an object she does not own, and if so, simply discards the action and issues a warning message to the user. This filtering greatly eliminates the contention in the case where two users try to modify the same object, and reduces the communication overhead over the network. VIAs use their own protocol, called VIAP, when interacting with the virtual world. This protocol is explained later in the implementation part of the thesis.

4.5 User interface

Developing an interactive DVE system requires a user-friendly interface for controlling objects in the VE. This user interface allows the person controlling the object to carry out various activities. Users need to be able to move about, pick up and manipulate objects, and communicate with other participants in the virtual environment. In order to create a good environment for exploration of virtual worlds, maximum flexibility has to be provided to the user, offering a wide range of possible interactions. These tasks are accomplished through the user interface, which allows objects navigate through the virtual world and to interact with other objects. Object movements and interactions are controlled by the user interface. User interface is also responsible for encapsulating object behaviors and event updates and passing them to the underlying networking module. The communication among objects is achieved via this networking module.

The user interface of VESIR-6 is designed to be generic and platform independent to allow any form of collaboration from its framework. Java applets are supported by most of the Internet browsers and are well suited for creating portable user interfaces.

4.6 Collaboration Aspects

Participants in the distributed VE rely on the network for exchanging information. For example, as the user moves within the virtual environment, s/he must transmit updates over the network so that other users can visualize him/her in the correct location. In a multi-user DVE, users interact by transferring information through a network. The network is also used to synchronize the shared state in the environment. Delays in transfer of information over the network cannot be totally eliminated. Our aim is to keep these delays within the limits of human perception, especially for long haul networks. We try to maximize concurrency and scalability in a consistent way in VESIR-6 system, while minimizing the effect of network delays.

Networking technologies enable large number of participants to join VESIR-6 regardless of their physical locations The network must be able to accommodate the volume of messages among participants in a timely fashion with a minimum amount of latency. The performance of the system is influenced by both the architecture of the network communication and the protocol used for transmitting information. Protocols are needed to minimize message traffic across the network. The VESIR-6 system provides service differentiation that distinguishes between time-critical interactions and lower-priority messages that are less sensitive to time delays.

Networking considerations define the basic part of the real-time collaboration among diverse users in a distributed VE application. One goal in the development of VESIR-6 is to create a system such that users would be unaware of the distribution, i.e. the distribution would be transparent. We, therefore, separated the network part from the system, and build another flexible component, called networking module, to handle low-level aspects of collaboration. The networking module of the application provides mechanisms for handling packet loss, receiving data out-of-order, consistency, synchronization, and connection management. It manages the connections among the participants, and handles the possibility that the data might get delayed in transit.

4.6.1 Networking module

Distributed multi-user VE applications require some form of communication to propagate the state and event updates to other participants, and to access the database of objects for virtual world contents during initialization phase. The networking module is a high-level communication abstraction used for this purpose. It is basically responsible for

handling data transfers to and from the network. It is mostly accessed by the user interface, when a user joins or leaves the system, and interaction agents, in order to distribute update events during modification of object attributes. VIAs do not call for redundant transmission of object attributes; they rely on the networking module to transfer data reliably.

There are two major concerns in designing the networking module for VESIR-6 systems. Firstly, it should optimize usage of network resources, efficiently manage the communication among the participants at run time, and minimize the latency associated with communication. In addition, the traffic required to maintain consistency among the virtual world replicas must be minimized. This issue is related to the communication architecture deployed in the underlying network. Since it is a generic interface, any topology could be deployed with the networking module. It is up to the user to choose the most efficient data distribution mechanism over the network. We have presented three communication architectures to the user, and preferred the multicast distribution architecture with presence of a group manager to provide a consistent view of the world contents. The user, however, is free to integrate any kind of distribution mechanism to VESIR-6, by means of the networking module. This first issue is addressed by deploying a multicast communication architecture in the underlying network. Secondly, the distributed system should provide the DVE designer with a high-level API. The application developers hence can be shielded from the mechanics of communications as much as possible so that their creativity can focus on graphics and behavior. Providing an abstract interface for network functionality lets the DVE application designer to integrate any type of communication architecture. Network communication details and consistency maintenance of virtual world replicas are made completely transparent to the application developers. For example, application developers can simply transform each user action into the corresponding database update, and rely on networking modules to reflect the update to the related participants over the network in an efficient manner. The networking module provides an interface for the expression of high-level database updates so that low-level communication details can be made completely transparent to the user. This second issue is addressed by the presence of the networking module API.

The networking module is mainly responsible for passing data between the application and the communication link over IPv6. The state transition diagram of the networking module is shown in Figure 9. The entire network communication is made

transparent to the user by means of the networking module. When the user runs VESIR-6, the networking module sends the necessary query messages to the group manager. The group manager then responds with the list of available groups and the networking module presents this group list to the user. Once she chooses which group to join, the networking module does the rest to log the user into the system. During navigation, the networking module forwards the updates generated by the local user to the network, and listens for remote updates transmitted by diverse users connected to the system over the network. Since reading from the network is a blocking function call, i.e. the application would be blocked until there are some data to be read over the network, the networking module creates a separate thread called "Receiver" for this purpose. After the user is finished with the application, s/he terminates it, and the networking module informs other members and the group manager of its disconnection from the system, and then frees the system resources.

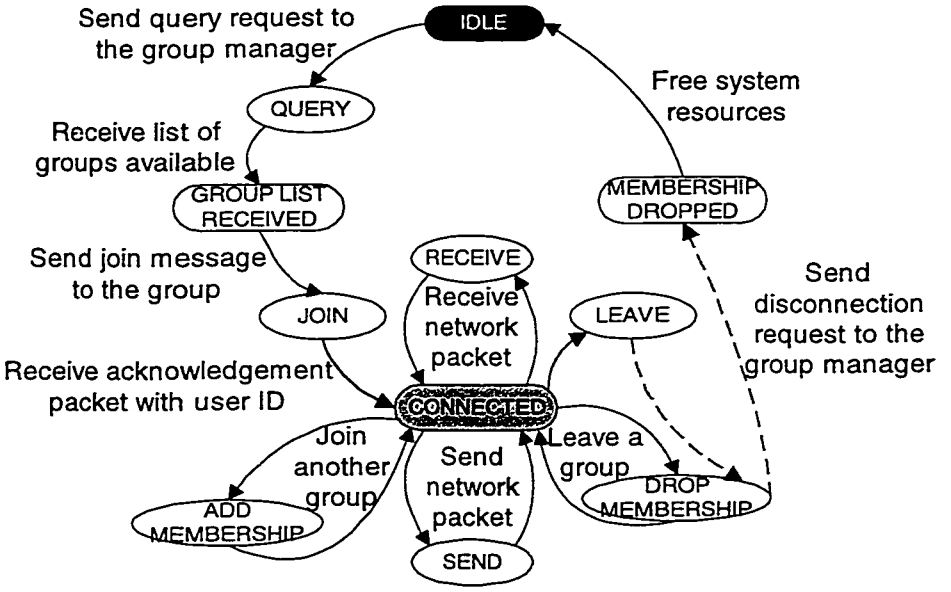

Figure 9: State transition diagram of the networking module.

The networking module is a wrapper interface for the application. It hides network-specific issues from the application developer. Networking module could be integrated on top of any network topology that provides its API and implements the interfaces needed to support collaboration among users in the networked VE.

4.6.1.1 Networking Module API

The networking module provides an interface to the underlying network, so that higher level modules can access the network by calling the appropriate functions. The

networking module API is a middleware that builds high level Java abstractions on top of IPv6 C++ network calls. The key challenge is providing useful abstractions without compromising the performance of the system. The basic abstractions that networking module API must implement were outlined in the last chapter.

The networking module API does not expose the network to the application developer. In fact, the application code will work exactly the same way whether the entire world is residing on a single machine or distributed over an arbitrary number of users on the network. The main purpose of this API is to hide the network specific issues from the higher modules. It is implementation-independent, that is, by abstracting the network functions with a well-defined API, we can deploy any kind of topology in the underlying network level. This provides the system a great flexibility to choose the most suitable network topology according to the circumstances.

4.6.2 Group Manager

Connectivity within the VESIR-6 is achieved over IPv6 by using a hybrid of peer to peer networking and multicast technologies with a group manager as described before. This method of networking infrastructure has been adopted to support large-scale VEs and minimize network latency over low-bandwidth Internet connections.

The group manager has a significant role for the networking module as the primary access point to the database of objects that exist in the virtual world. As shown in Figure 7, the group manager provides the users with a member database, query service, group and consistency management. The query service is essential in order to accommodate several multicast groups and allow the user to choose which group to join. The current implementation of VESIR-6 has only one group, but the structure of the group manager makes it easy to accommodate several groups simultaneously.

The existence of the group manager helps avoiding the overhead of periodic update messages associated with static objects. In addition, it provides an up-to-date state of the users and objects in the system. The role of the group manager in VESIR-6 is explained in more detail in the next chapter.

4.6.3 VESIR-6 Network Communication Protocol (VNCP)

Network capacity is a limited resource, so the system must carefully determine how to allocate this capacity to the various types of information that VESIR-6 users must exchange. For example, when a user connects to VESIR-6 through a modem connection,

s/he might not receive the frequent updates by the remote users continuously if the network packets containing the update messages are relatively large. The network protocol should be designed to impose minimal overhead. For this purpose, we implemented our own protocol, VNCP, as middleware to provide the system with transparent, low-overhead communication and message distribution in VESIR-6. Above VNCP lies the application itself, and the networking module API exists at the interface.

The VNCP is used to exchange information about a VE among users in a distributed system that is simulating the behaviors of objects in the environment. A VNCP real-time flow consists of packets of length 176 bits at rates up to 10 per second per simulator.

Changes to object behaviors, called event updates, are transmitted with a best-effort transport protocol. At relatively rare intervals, events which require reliable transmission of some data, on a unicast basis, to any other host in the system, may occur. These events are referred to as control commands, and generated when a new user joins the system, a user transfers ownership of an object, or when a member leaves the group. Control commands are assigned higher priority in order to provide reliability and consistency.

4.6.3.1 Packet format used in VNCP

We define a simple and extensible message format that provides enough information for filtering of event updates to be used in VNCP. Our objective is to distribute network packets with minimum latency while maintaining low overhead.

In order to optimize network communication across low bandwidth dial-up links we designed an efficient protocol for encapsulating the frequent event updates. This protocol is aimed at supporting 3-D interaction transformations with minimal data exchange. VNCP is compatible with VIAP except for that the latter is used for interactions between the user interface and the VE whereas VNCP is used for message transfer over the IPv6 network. The packet format according to our protocol specification is shown in Figure 10 below.

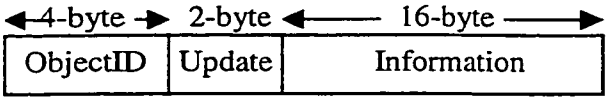

Figure 10: The packet format used in state and event updates.

The VNCP specifies that the packets are composed of three parts: The object id that uniquely identifies the object in the distributed VE, the update field that specifies the type of update information encapsulated in this packet, and finally the data field that contains the value of the update. The packet size is 22 bytes in total. Keeping the packet size small reduces the overhead and improves the performance of the system since most of the overhead is generated by the frequent event updates of the objects within VESIR-6.

5 Managing the Shared State of VESIR-6

Maintaining a consistent state of the virtual world and its contents among hosts in a multi-user VE is a complex issue. A multi-user VE system does not necessarily require a server to mediate communication among clients, especially if the underlying network uses multicasting, but it does require a central repository to keep track of users and objects with their associated ownership and location data. In this chapter, we will see what the group manager provides to the system, and how consistency is maintained in VESIR-6.

5.1 The role of the Group Manager

The ideal VESIR-6 system consists of a cluster of group managers, multicast groups, and users. Group managers are members of an anycast group and have identical IPv6 anycast addresses. The users are divided into several multiple groups, and they may be subscribed to multiple groups simultaneously. Data distribution and connection paths among the group managers stay hidden from the user. The use of anycasting allows dynamic load balancing and reduces latency by connecting users to group managers geographically close to them. This architecture has advantages in data sharing, reliability, performance, and system growth, and is shown in Figure 11. The availability of multiple group managers has the advantage of splitting the processor and network load between several machines. However, not all envisioned features in the VESIR-6 system architecture could be implemented. Especially, IPv6 is still in its infancy, and not all the features proposed in the standards are exploited in the implementations yet. For example, anycast has not been deployed in IPv6 implementations yet, and the research on how to manage anycast addresses is still going on, therefore our group manager is now a member of a "well-known" multicast group, as shown in Figure 6 previously. This technical issue made it necessary to redefine the group manager somewhat, but the core features of the multi-user environment have been realized well over IPv6. The users simply send their requests to this multicast group, and the group manager listens for user requests on this multicast channel. The communication architecture of VESIR-6 given in Figure 11 could easily be deployed after anycast address management is implemented on IPv6 nodes.

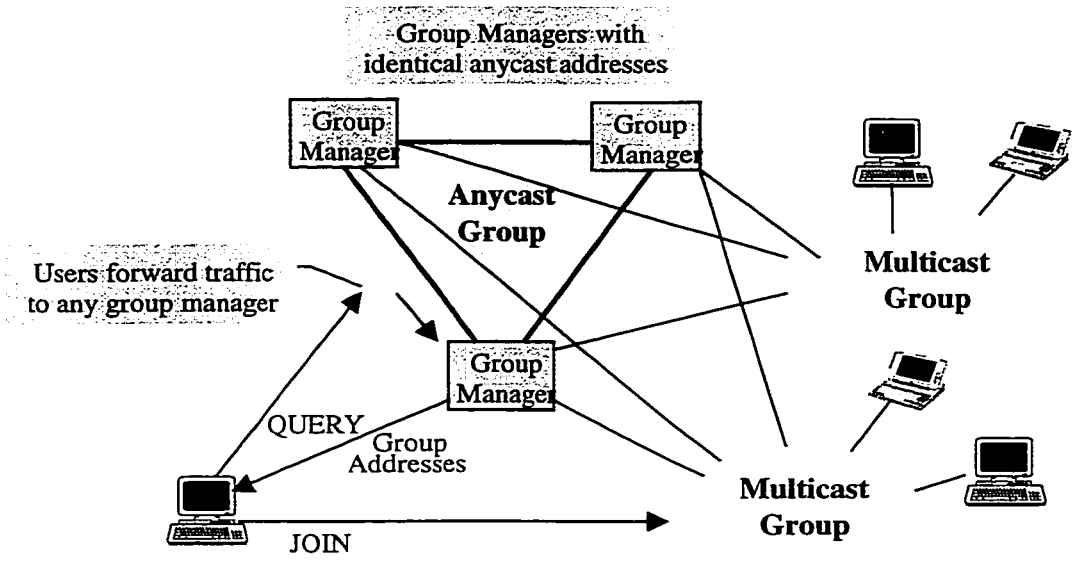

Figure 11: The role of the group managers in VESIR-6.

The group manager basically provides two enhancements to VESIR-6. First, it hides the network topology from the user, and uses the same protocol, VNCP, as the networking module. Second, new members to the system, or the members disconnected for a certain time, possibly due to a network failure, need a certain point to connect to the virtual world. This requires a location where the current state of the virtual world is kept and updated continuously. The group manager keeps the member database, and the users can use the query service provided by the group manager to get information about other participants.

The group manager also generates a unique identifier for each object in the system to uniquely identify itself to the shared virtual environment. It is mainly responsible for maintaining a consistent state for the virtual world. It does this by implementing a kind of database system in which objects can be added, removed, or modified.

5.1.1 Providing Topological Flexibility to VESIR-6

The group manager eliminates the effect of the communication architecture on the system. In Chapter 3, we developed three possible communication architectures that could be used for message distribution in VESIR-6. The group manager has diverse functionality according to the underlying network topology.

In the first architecture, the underlying topology has a unicast communication architecture, and the group manager plays a pivotal role both in maintaining the consistency and passing the messages to related participants. Each user node transmits its

updates to the group manager, which in turn distributes them to other users and updates its database. The disadvantage of this architecture is that as messages have to pass through the server first, the network latency increases. This approach is fundamentally limited in scalability because the central server inevitably becomes the bottleneck as the number of clients increases.

As we have seen in our second architecture, multicast technology makes it possible to architect a DVE without a server. In the multicast communication environment, each user knows how to communicate with all other peers. It is the router's responsibility to keep track of group members and to handle the message distribution. The members communicate directly with one another without going through a central server. Our second communication architecture has a completely distributed multicast topology that is very efficient in terms of scalability and response time. However, it is neither rapid in joining, since objects are discovered merely by event interactions, nor robust in case of network crashes. We included the group manager in our final architecture in order to overcome these limitations.

Although we prefer to use the third communication architecture for message distribution over the network, the user is free to choose any of them, and even more, to integrate his own design to VESIR-6. Together with the networking module, the group manager shields the user from the complexity of the network communication, and provides topological flexibility to the system.

5.1.2 Query Service

The group manager is built around a user and object database. It provides a unique identifier to the user when s/he joins the system. It saves the world contents in a hierarchically linked list database for persistence as shown in Figure 12. There are several groups in the database, and the user may join any of them. Under each group, there are the members belonging to that multicast group. If a user is member of more than one group, than each of those groups has a link to the member, although there is only one entry for that particular member in the database. The object entries are created under their owner users. The objects in VESIR-6 cannot have more than one owner, so there is only one link for each object in the database, and that link points to the owner of the object. This hierarchical linked list storage mechanism drastically decreases the table lookup and processing time of the group manager.

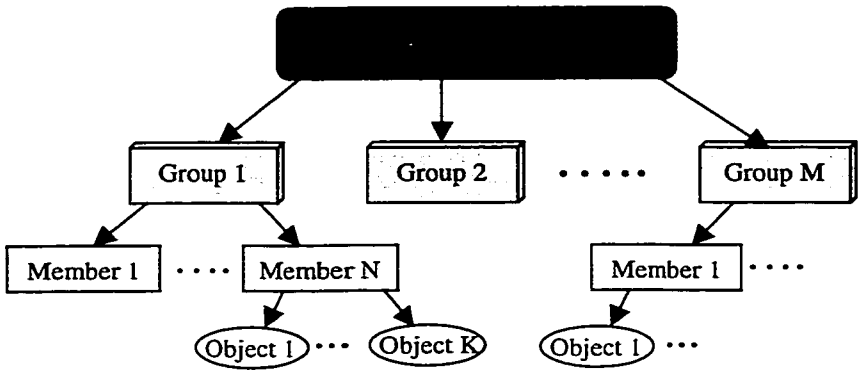

Figure 12: Structure of the database stored in group manager.

The group manager presents a single object database image to the user and provides transparent data access. It sends the complete database to the user during initialization reliably. During navigation, the user might inquire the group manager about the presence of other groups. The group manager listens for query messages and replies to them attaching the necessary information. As changes occur, the group manager updates its database and preserves consistency.

Group manager's object structure is the key to sharing dynamic object creation, removal, or ownership transfer. When the ownership of an object is transferred from one user to another, the object is simply unlinked from the old owner, and then linked to the new owner.

Group manager does a good job of allowing rapid joining, because a new user can receive an immediate download from the group manager of everything it is supposed to know, including presence of other participants and database of the objects in the environment. This is also useful in recovering participants from temporary disconnection and helping them synchronize with the system.

5.2 Joining and Leaving the World

The VESIR-6 system supports dynamic connection of the users to the shared environment. In this section, we will outline the steps taken by VESIR-6 when a user joins or leaves the system, or modifies the virtual world contents.

After the browser has downloaded the virtual world from the web server, the user types in a login name and chooses his avatar. The application calls the `AddMemership()` function of the networking module, which, in turn, sends a join message directly to the group manager, which is a member of a well-known multicast group. Since anycast has not been deployed in IPv6 implementations yet, and the research on how to manage

anycast addresses is still going on, our group manager now listens for user queries on a multicast group. The users simply send their requests to this multicast group, and the group manager listens for user requests on this multicast channel. When the group manager receives a join message, it generates a unique identifier for the new user, adds her to its database, and responds to her with an acknowledgement message that includes the object ID of the user. When a user joins the system, the group manager creates a new member entry in the database and sends a notification message to all other users. Each entry in the database is assigned a unique index value, which is also the identifier for the user. The users communicate with the group manager and other participants by means of the networking module. However, since events are directly multicasted, the users must first subscribe to the corresponding multicast group in order to receive the updates to the environment.

If the user wishes to create an object, s/he sends a request message to the group manager. The group manager then assigns a unique identifier for that object, adds a new record for this object to its database, and propagates this information to all members. Upon receiving this information, the users add the new object to their local copy of the environment. If the user decides to delete an object, s/he removes the object from the scene, and informs other users and the group manager. In response, the group manager performs a delete on that entry in the database. Any changes to the appearance of the object directly reflects to the database as well.

When the user wants to exit the VE, s/he would call the `DropMembership()` method of the networking module, which sends a leave message to notify other members and the group manager of the user's disconnection from the system. When other users receive this message, they will remove the corresponding avatar information together with the objects owned by that user from the virtual world and user list, and the group manager will set the resources associated with that user free.

5.2.1 Spatial Partitioning

When modeling large virtual environments that accommodate many users, it is often helpful to split up the virtual space into a number of smaller worlds. Thus network load can be bounded, because not every participant has to receive position updates of all the others. Another reason to break up large worlds is to accelerate rendering. It is clearly not practical to download an entire world every time you enter to the VE. In large-scale

DVE applications a particular object is typically interested in a subset of all other objects in the virtual world. For example, assume that you are at a virtual mall. While you are travelling along the pathway at one section of the mall, you cannot see and therefore would not be interested in whatever is happening at the other sections of the mall. When you are buying a shoe, for example, you only need to receive the interactions within the shoe store or those that occur within your vision of sight. The virtual world can be broken up into smaller regions, and that region-to-region visibility can provide a basis for filtering updates in the multi-user VE. Without some filtering mechanism, each simulation must process the data sent by all other object. In large-scale DVEs this processing alone can overwhelm the capabilities of the users.

The group manager recognizes the importance of spatial partitioning and provides explicit support for regions. It stores objects hierarchically; each object has a parent member, which is the owner user of the object, and each member is located under one group, which is the spatial partition the user is residing in the virtual world. Each group is associated with a different multicast address, and the group manager is a member of all those multicast channels. Further improvements in scalability could be achieved using a network of group managers that cooperates by taking on responsibility for different regions within the simulated environment. When a user move from one region to another, the group manager transfers the user to the corresponding group.

5.3 Object Consistency & Persistency

The virtual world consists of objects that are under control of users. Objects in our architecture have system-allocated, unique component identifiers. The group manager registers each object and assigns them a unique object identifier, ensuring that all components created by a user's environment have a unique address in the system.

Since DVEs are highly interactive and dynamic, objects and their attributes have to be modified quite frequently. Objects can be created or destroyed dynamically. Each user can control one or multiple objects. The user who creates the object is the default owner of that particular object, although others have the chance to ask for the ownership of the object later. The system architecture must support concurrency control for managing conflicting user actions. If you try to open a door while I am trying to close it, what happens? VESIR-6 solves this issue by associating each object with only one owner.

This locking mechanism is necessary to restrict more than one user modifying an object simultaneously, to prevent contention, and to maintain consistency in the virtual world.

5.3.1 Explicit Object Ownership

In most of the shared VE systems, any host could modify information about any object in the environment. Treating the world as a shared database forces us to deal with the issues of conflicting user actions and concurrent access to the shared objects in the database. We must ensure that multiple hosts do not attempt to update an object simultaneously because we cannot guarantee how those updates will be received and processed at each host. For example, suppose that two hosts are trying to move a chair at the same time, or one of them is trying to raise the volume level of a television object while the other is trying to set it mute? Each host would transmit an update for the required interaction. Now, some users might receive the first host's update before the others, and some users might receive the updates in the opposite order. Consequently, some users will display the chair in the position set by the first host, while others will display it in the position set by the second host.

In order to eliminate this conflict, we must ensure that each piece of shared state can only be updated by one host at a time. We accomplished this in VESIR-6 by assigning the ownership of an individual object to exactly one user. The user that owns an object takes responsibility for continually multicasting the changes to that particular object. A user cannot send updates to an object's state unless s/he owns the object. The interaction agents used in VESIR-6 filter the messages by checking the ownership field of the object, and hence tremendously reducing the network traffic by letting only the appropriate messages passed to the network module.

The ownership mechanism ensures that only the owner user has the ability to update object attributes and delete object instances. When acquiring the ownership of an object, the user sends a `REQUEST_OWNERSHIP` message to the group. The owner of the object will then either grant the ownership and reply with a `GRANT_OWNERSHIP` message, or ignore the request. If there is no reply after a certain timeout, the requester knows the ownership was not granted. This mechanism requires fewer messages to be transferred over the network.

Object ownership is similar to the locks one acquires upon files in a shared file system. Typically, one user can open a file with read-write permissions, while the others

can only read the files already opened, and wait for the file closed before opening it with write permission. Similarly, the group manager in VESIR-6 prevents multiple users from simultaneously modifying the same data. Users must explicitly request and obtain the ownership of the specific object before updating it. The group manager assigns locks that enable a host to update the state value of an object until ownership is transferred.

5.3.2 Interacting with non-owned objects

Suppose that a customer in a virtual auto-shop wants to drive a car for test purposes before buying it. The car belongs to the shop, and nobody else is allowed to access or modify the car object. However, in order for the user to test drive the car, some means should be provided to give the user temporary access to get into the car and start the engine. We implemented two mechanisms for interacting with non-owned objects in VESIR-6: Source forwarding and ownership transfer.

5.3.2.1 Source Forwarding

In the first approach, only the owner of the object has the right to modify the object. Other users, however, might indirectly interact with the object by sending the update request to the owner of the object. This might be accomplished by encapsulating the required update into an `ENCAPSULATED_UPDATE_REQUEST` message and sending this encapsulated update packet to the owner of the object. Upon receiving the `ENCAPSULATED_UPDATE_REQUEST` packets, the owner of the object decides whether to accept the update, and if she does, she decapsulates the packet, then interacts with the object and multicasts the new state of the object to other users. In the source forwarding approach, the owner of the object serves as a proxy for handling all changes to the object's state, even though the updates may originate at any host. This approach incurs an extra message cost on each non-owner update. It is most suitable when non-owner state updates are rare or when many hosts desire to update the state of an object.

5.3.2.2 Ownership Transfer

Ownership transfer method refers to the actual transfer of object ownership from one host to another. The user who is interested in modifying the object sends a private message to the owner of the object requesting the ownership of that object. The owner of the object then responds with a message either conforming or denying the request. If the ownership transfer is granted, the old owner notifies the group manager of this change. From that point on, the new owner is free to update the state of the object, and it is the

only host that can destroy the object or transfer its ownership to another user. The ownership transfer incurs an extra message round-trip to acquire ownership of the object. Therefore, it is most appropriate when the user is going to make a series of updates to the object, and there is little contention among users wishing to make updates to that particular object.

6 End-to-End QoS in VESIR-6

DVE networks are carrying more data in the form of bandwidth-intensive, real-time interactive traffic, which stretch network capability and resources. Traffic management in such network is therefore an important issue and involves managing allocated resources at data transfer to ensure continuous maintenance of required QoS levels. If not managed efficiently, the network could easily become a bottleneck. As more users participate in the distributed VE, the aggregate amount of information generated by the application increases. However, the network capacity is a limited resource, so that the system must carefully designed to determine how to allocate the network capacity to the various types of information that the diverse users in the VE must exchange. For example, when a user connects to the multi-user VE through a modem connection that provides minimal network resources, it is desirable to differentiate among the various traffic flows in order to ensure consistency within the VE.

Our research objective in this chapter is to develop and evaluate an end-to-end QoS architecture for VESIR-6. We first outline the need for QoS, and then provide an overview of the two standard IP QoS architectures, and how they fulfill our objective. Finally, we propose our QoS architectural framework to be used in VESIR-6, explain how QoS is achieved throughout the network using this architecture, and how the proposed architecture helps to get the most from scarce network resources.

6.1 The Need for Network-level QoS

QoS is a way of providing differentiated service classes and priority levels for data flows. Its aim is to ensure that the mission critical traffic has acceptable performance. In the real world where bandwidth is finite and distributed VEs and other multimedia applications such as videoconferencing vie for scarce resources, QoS becomes a vital tool to ensure that all applications can coexist and function at acceptable levels of performance.

Network-level QoS expedites the handling of mission-critical applications, while sharing network resources with noncritical applications. It also gives network managers control over network applications, improves cost-efficiency of WAN connections, and enables advanced differentiated services. Network-level QoS technologies enable application developers to balance the service priority levels for user satisfaction with

efficient backbone and access utilization to minimize network latency. They efficiently prioritize a variety of traffic types and requirements for mission-critical and time-sensitive DVE application traffic. Benefits of network-level QoS can be summarized as better control over resources, more efficient use of network resources, coexistence of mission-critical applications, and foundation for a fully integrated network in the future.

One of the strongest motivations to use IPv6 as the networking protocol for VESIR-6 is that it inherently supports better QoS than IPv4 by the means of flow label and traffic class fields in the IPv6 header. IPv6 networks are also able to export multiple service classes and align the services they provide with application needs. This is basically achieved by setting the traffic class field in IPv6 header, and getting the service corresponding to this class, as defined by the Service Level Agreements (SLAs) provided by the ISP networks.

6.1.1 Achieving Reliable Multicast Transport

VESIR-6 network communication architecture is based on IPv6 multicast. Network layer multicast capability is an essential element for achieving QoS because it enables more efficient use of network bandwidth than traditional point-to-point operations. IP Multicast prevents traffic from going to nodes or users that have no need for it, as happens with the broadcasts on many multimedia applications. Multicasts only require that a single copy of data, which could represent an event update of a moving object in the DVE, issued to the network regardless of how many end- to-end points must receive the data. All replication of the data is done within the network, on an as-needed basis, contingent upon end-user requests for the data.

The problem with IPv6 multicast is that it runs over connectionless UDP, therefore it is neither reliable nor order preserving. The packets might get lost, be duplicated, or arrive in different orders. The ordering of the messages might be achieved by simple acknowledgement schemes. Most reliability mechanisms attempt to retransmit the original data to correct for packet loss. This approach may be required for convention applications such as file transfer, but in distributed real-time simulation such recovery is of little use. A general issue with reliable transport for multicast is the congestion problem associated with delivery acknowledgements, which has made real-time reliable multicast transport infeasible to date [28]. VEs require high interactivity which cannot be provided if the messages have to be acknowledged by all participants. We exploited the

IPv6 QoS mechanism to increase reliability in multicast communication, as explained later in this chapter.

6.2 IP QoS Standards

IP QoS refers to the performance of IP packet flow through the network. It is characterized by a set of metrics, including service availability, delay, jitter, throughput, and packet loss rate. The researchers are focusing on two architectures developed by the Internet Engineering Task Force (IETF): The Integrated Services architecture (often referred to as IntServ) [34], and the Differentiated Services architecture (often referred to as DiffServ) [15].

6.2.1 Integrated Services Architecture

IntServ extends the existing IP architectural model to support both real-time and best-effort traffic flows. IntServ suggested that for a flow to receive a desired level of service in terms of quantifiable bandwidth or delay, it is necessary to install and maintain flow-specific state in the network. IntServ defines three classes of service [34]:

- Guaranteed: This service supports real-time traffic flows that require bandwidth, bounded delay, and no-loss guarantees.
- Controlled load: This class approximates best–effort service over a lightly loaded network.
- Best-effort: Also known as lack of QoS, best-effort service is basic connectivity with no guarantees. The Internet today is a good example of best-effort service.

The original IntServ specification proposed Resource Reservation Protocol (RSVP) as a working protocol for signaling the IntServ architecture. This protocol assumes that network resources are reserved for every flow requiring QoS at every router hop in the path between receiver and transmitter, using end-to-end signaling. Using a method similar to ATM's SVCs, IntServ uses RSVP between senders and receivers for per-flow signaling. RSVP messages traverse the network to request resources. Routers along the path must maintain soft states for RSVP flows.

True end-to-end QoS requires that every element in the network path, including switches, routers, firewalls, and hosts, deliver its part of QoS, and it all must be coordinated with QoS signaling. QoS signaling is the means to deliver a QoS service requirement across the network. However, the challenge is finding a robust QoS signaling

solution that can operate end-to-end over heterogeneous network infrastructures. RSVP signaling is useful for coordinating the traffic handling techniques and has a key role in configuring successful end-to-end QoS service at the edge of enterprise networks where user flows can be managed at the desktop user level. Most QoS signaling solutions, including RSVP, have limited scope across the network.

While RSVP was introduced as the solution to IP's QoS shortcomings, its applicability and scalability over large networks, in particular the Internet, are limited [35]. The routers with finite amount of buffers and CPU may not be able to maintain state information on thousands of RSVP flows while processing frequent RSVP refresh messages.

6.2.2 Differentiated Services Architecture

DiffServ is a relatively new IETF working group that has defined a more scalable way to apply IP QoS. DiffServ minimizes signaling and concentrates on aggregated flows and per hop behaviors (PHBs) applied to a network set of traffic classes. It focuses on the use of the traffic class field in IPv6 header and the ToS field in IPv4 header as a QoS signaling mechanism, and provides definitions appropriate for aggregated flows for any level of aggregation.

The approach taken by DiffServ is to classify individual flows at the edge of the network into one of several unique service classes, and then apply a per-class service to the flow through the network. The classification is performed at the edge router based on analysis of some fields in the packet. The packet is then marked as belonging to a particular service class and then injected to the network. The routers that forward the packet examine the code points in the packet header to determine how the packet should be treated.

DiffServ architecture defines several components:

- DS-field: DiffServ renames the 8-bit IPv4 type of service (ToS) and the traffic class field in IPv6 as the DS-fields. The 8-bit DS-field contains six bits for DS code points (DSCP) and two bits that are currently undefined [36].
- Per-hop-behavior (PHB): PHBs define the service the packet receives at each hop along its path through the network.
- Behavior aggregate (BA): A PHB is applied to each BA inside the network.

- Edge router: This device is responsible for packet classification, metering, packet marking, and traffic scheduling.
- Core routers: Those are interior nodes that provide the PHB based on the DSCP bits contained in the DS-field. These devices typically employ queue management and scheduling discipline to provide the PHB.

Within the six available bits, codepoints 'xxx000' are standardized as the Class Selector codepoints, and interior nodes must implement PHBs selected by these codepoints. This is done to preserve a backward compatibility with current uses of the IP Precedence field in the Internet. RFC 791 assigns the following meaning to the eight possible values of Precedence field:

111 - Network Control

110 - Internetwork Control

101 - CRITIC/ECP

100 - Flash Override

011 - Flash

010 - Immediate

001 - Priority

000 – Routine

The above values can be interpreted to describe the increasing importance of an IPv4 packet from the value 000 to value 111. The mapping of Precedence field to class selector codepoints in DiffServ is straightforward: '000000' is used as the default PHB for the Internet while the remaining seven non-zero Class Selector codepoints are configurable, and the codepoint with larger numerical value has a higher relative priority. The codepoints '11x000' preserved for routing traffic, as done in IP Precedence. In summary, DiffServ provides five classes of service to the application in addition to default best-effort service. It is possible to partition the traffic up to six classes of service using DiffServ.

There is ongoing research on extending DiffServ to the data link layer by providing a mapping from IP Precedence signaling to the IEEE's 802.1p frame prioritization standard [37]. This allows differentiated services to be mapped seamlessly across the data-link layer technology on the campus and onto the WAN network to provide end-end QoS services.

6.3 Proposed End-to-End QoS Architecture

Though IP QoS is in its infancy, it is quite clear that it will be an absolute requirement in next-generation distributed applications. Its evolution with IPv6 will be rapid, exciting, and rewarding. IPv6 router vendors have already integrated the IP QoS standards in their products, and IPv6 networks will take full advantage of the network-layer QoS.

Since the size of the users in the DVE may be very large, system resources may not be sufficient to serve all the traffic flows simultaneously. As a result, resources must be allocated and distributed among processes. Our QoS attention is being focused at the WAN level, since in the local network, in general, there is adequate bandwidth. We propose a technique brings per-packet based QoS at the WAN level by using IPv6 features and DiffServ standards. This technique lets the network handle the difficult task of utilizing an expensive wide-area network connection in the most efficient way for bandwidth intensive DVE applications.

Our proposed technique distinguishes between messages which are causally significant and those which are merely contingent. We maintain causal consistency only for the former. This architecture achieves a further optimization in state consistency by anticipating the effect of interactions allowing control commands and ownership exchange to be carried out in advance. We defined two service levels over the network. Service levels refer to the actual end-to-end QoS capabilities, meaning the ability of a network to deliver service needed by specific network traffic from end-to-end. We then selectively inject the mission-critical packets to the flow associated with higher service level.

Selecting the mission-critical packets is of significant importance to ensure that they get the service they require even in times of congestion. VESIR-6 basically generates two types of network packets: Event updates, and less frequent control commands. The event updates in VEs are frequent and if some are lost or arrived out of order, the next updates will correct these errors. Event updates occur when the user is moving within the virtual world. As this user is moving, his or her system sends out a series of positional messages. If one of those positional messages is lost, the system can pick up subsequent messages and ignore the dropped message. For example, a series of positional events are distributed by multicasting when an avatar is moving. If one of these positional multicast

messages is dropped, the position will be updated in the next message and other users will simply see the avatar jump to the new position. To some extent event updates can compensate for excessive latency by extrapolating continuously changing values, as used in dead reckoning of object movement based on initial position plus velocity and acceleration. However, the control commands involve state changes and are of higher importance. We separated the event update and control command traffic into different traffic flows, and assigned higher priority to the flow associated with the control commands. This is shown in Figure 13.

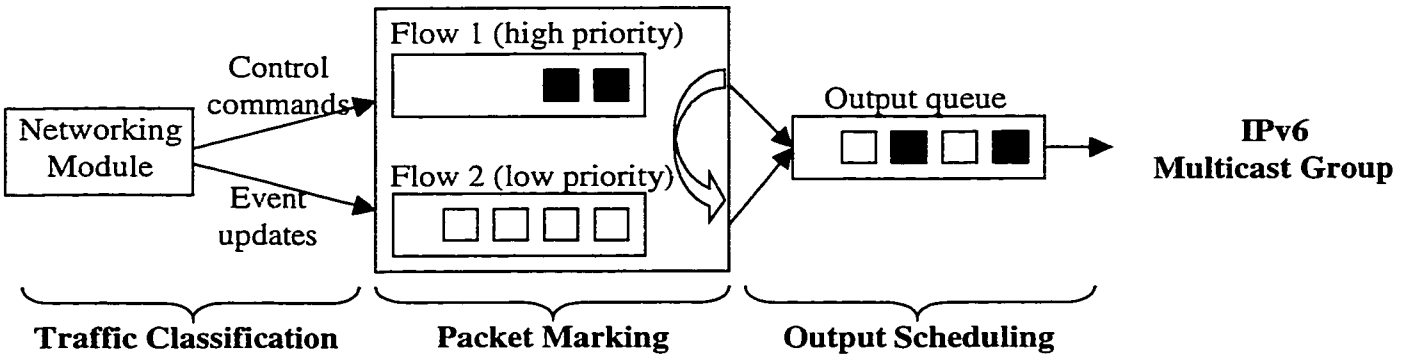

Figure 13: Separating multicast traffic flows and assigning priorities to achieve reliability.

Packets are classified into flows by the protocol and DSCP codepoint fields in the packet header. Each flow corresponds to a separate output queue. When a packet is assigned to a flow, it is placed in the queue for that flow. During periods of congestion, IPv6 node allocates a portion of the available bandwidth to each active queue.

We used the DiffServ QoS signaling mechanism, namely DSCP, to achieve end-to-end QoS in VESIR-6. DiffServ is more scalable than IntServ because it handles flow aggregates and minimizes signaling, thus avoiding the complexity of per-flow state at each node. DiffServ allows network traffic to receive premium treatment at the expense of other less-critical traffic on the same WAN link. This idea is similar to what we find in airlines where a first-class passenger may receive better treatment or service than an economy class passenger while they both physically reside on the same airplane.

This architecture provides better treatment, including faster forwarding by intermediate IPv6 nodes and less probability of the packets being dropped due to lack of buffering resources, for mission-critical data. It does not provide more bandwidth, but introduces a way to supply adequate resources to data flows, assuming those resources are fundamentally available. It prioritizes control commands to ensure that mission-critical packets get the service they require, while simultaneously servicing event update packets.

By eliminating the need for acknowledgement traffic, our proposed architecture reduces the overhead of maintaining the consistency among the replicas of the shared virtual world at each player's site. It optimizes networking efficiency further by relaxing the reliability requirements of event updates and exploiting IPv6 multicast and QoS capabilities.

It is possible to extend this architecture to provide dynamic QoS to the application. The design, in this case, must take into account from the beginning that QoS cannot be guaranteed in particular circumstances; therefore, the coding of the information must be designed to always provide a minimum service, even if it is low-quality. For example, it is possible to achieve some QoS for the event updates in VESIR-6 by assigning a set of priorities to them periodically. For example, assume that the event updates occur at 100ms intervals. Let us assign a low priority to the first packet, and a higher priority to the second one, and a low priority to the third one, and so on. When the network is congested, the low priority packets will be dropped first. In this case, the event updates are still transmitted, but at 200ms intervals. The result is a dynamic QoS architecture, incrementally reducing the service level in response to network congestion. This is better than a burst error in which all of the sequential packets will be lost, and the user might get disconnected from the system for a while. With slight modifications, this architecture can provide expedited handling appropriate for a variety of multimedia applications.

6.3.1 Implementing End-to-End QoS in VESIR-6

End-to-end QoS is achieved by classifying, marking, scheduling, and possibly shaping the packets sent to the network by the VESIR-6 application. A variety of queuing, traffic shaping, and filtering technologies are needed for implementing traffic priority and controlling congestion end-to-end across the network. Network nodes handle an overflow of arriving traffic by using a queuing algorithm to sort the traffic, then determining some method of prioritizing it onto the output line. The primary QoS building blocks for managing differentiated traffic are packet classification, marking, and scheduling, as shown in Figure 14 below.

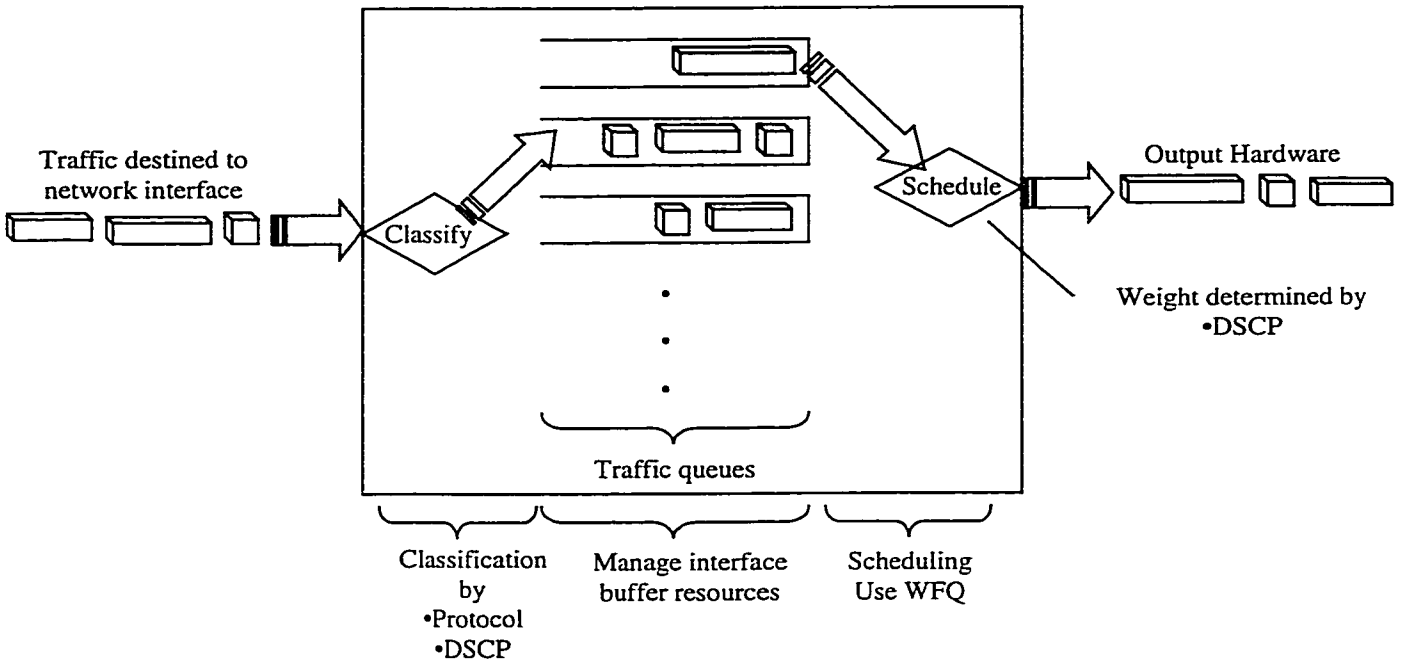

Figure 14: QoS building blocks for managing differentiated traffic flows.

6.3.1.1 Packet Classification

Packet classification means that traffic gets grouped or aggregated into a small number of classes, with each class receiving a particular QoS in the network. The aim of packet classification is to group packets based on predefined criteria so that the resulting groups of packets can then be subjected to specific packet treatments. The classification is made by analyzing protocol, flow label, and DSCP fields of the packet. IPv6 reduces the elaborate overhead and marks packets with a flow label inserted in the IPv6 header. This identifier can be cached in routers and used for a quick classification of packets. This technique simplifies the classification when the source station differentiates flows by marking them with different flow labels. In this way, it is possible to manage the QoS for already existing applications, without modifying them, but trying to make decisions on the basis of the header content. In fact, finding the information on ports and on application entails processing the whole chain of headers, with a considerable computing burden, and this process can be quite complicated if the payload is encrypted. This disadvantage brings about the introduction of the flow label in IPv6.

```
Send_Output_Packet(int objectId, short field, struct data value,
                    struct sock_addr_in6 stDstAddr)
{
    OutputBuffer.PacketData.objectId = objectId;
    OutputBuffer.PacketData.field = field;
    OutputBuffer.PacketData.value = value;
    if (field && EVENT_UPDATE)
        SetPriority(EVENT_TRAFFIC, LOW_PRIORITY);
    else if (field && CONTROL_COMMAND)
        SetPriority(CONTROL_TRAFFIC, HIGH_PRIORITY);
    Sendto(hSocket, OutputBuffer.cBuffer,
            sizeof(OutputBuffer.cBuffer), 0,
            (struct sockaddr*) &stDstAddr, sizeof(stDstAddr));
}
```

Listing 1: Separating the event and control traffic in VESIR-6.

6.3.1.2 Packet Marking

Once we classify packets, the next step is to "mark" packets with a unique identification to ensure that this classification is respected end to end. The simplest way of doing this is the IPv6 traffic class field in the header of an IPv6 datagram. Being compliant with the standards, we used the DSCP as the classification criterion of choice.

```
SetPriority(int flowlabel, int priority)
{
    unsigned int flowinfo;
    flowinfo = (IP_VERSION >> 28) && (priority >> 20) && flowlabel;
    hSocket.sin6_flowinfo = htonl(flowinfo);
}
```

Listing 2: Marking IPv6 flow label and traffic class fields to required priority level.

The purpose behind this kind of marking of packets is to ensure that downstream QoS features such as scheduling and queuing may accord the right treatment for packets thus marked. Classification allows leveraging the services, and provides a means of differentiation among the multiple traffic flows within the same application.

6.3.1.3 Packet Scheduling

Once traffic has been classified the next step is to ensure that it receives special treatment in the nodes. This brings into focus scheduling and queuing. The packet-scheduling disciplines arbitrate access to link bandwidth and provide performance guarantees to applications. The packet scheduler determines the order in which each

packet is served (transmitted). The simplest scheduling algorithm consists of ordering packets as a function of their priority. In this way, packets with higher priority are transmitted first. This method of transmission can cause an indefinite waiting period for lower priority packets if the traffic of higher priority data is very heavy. To avoid this kind of problem, usually, multiple queues are used, with a minimum service rate associated to each queue.

Currently, the commonly used algorithm for the management of real-time traffic is Weighted Fair Queuing (WFQ) [38]. It is a flow-based queuing algorithm that does two things simultaneously: It schedules interactive traffic to the front of the queue to reduce response time, and it fairly shares the remaining bandwidth among traffic flows with different weights. Each flow queue is associated with a weight proportional to the frequency it must be served. Weights are derived from DSCP in IPv6 header. With Weighted Fair Queuing, interactive applications are not pushed out of the way by other applications that require bulk transfers, and high-volume applications are guaranteed to be transmitted (possibly with some delay), even if they are in competition with other similar capacity-demanding applications.

The WFQ algorithm alternates the transmission of packets belonging to several flows. During transmission, the algorithm gives higher priority queues absolute preferential treatment over low-priority queues. WFQ is useful for making sure that mission-critical traffic traversing various WAN links gets priority treatment. It is also efficient in that it will use whatever bandwidth is available to forward traffic from lower priority flows if no traffic from higher priority flows is present. This is different from Time Division Multiplexing (TDM) which simply carves up the bandwidth and lets go unused if no traffic is present for a particular traffic type.

Packet scheduling ensures that queues do not starve for bandwidth, and that traffic gets predictable service. The queuing algorithm ensures that delay-sensitive, interactive traffic receives quick service, while actually providing most of the network bandwidth to the important, high-volume traffic.

The packet scheduler is DSCP-aware, that is, it is able to detect higher priority packets marked with precedence by the application and can schedule them faster, providing superior response time for this traffic. As the priority value increases, the algorithm allocates more bandwidth to that flow to make sure that it gets served more quickly when congestion occurs.

IPv6 allows deployment of different schedulers and congestion control algorithms for different applications. Fair scheduling algorithms such as WFQ can handle arbitrating access to network bandwidth.

6.3.2 What happens through the network?

According to the DiffServ, the ISPs provide Service Level Agreements (SLAs) to their customers [39]. An SLA basically specifies the service classes supported and the amount of traffic allowed in each class. Users mark the DS field of individual packets to indicate the desired service. At the ingress of the network, packets are classified, marked, scheduled, and possibly shaped. When a packet enters one domain from another domain inside the network, the DS field may be re-marked, as determined by the SLA between the two domains.

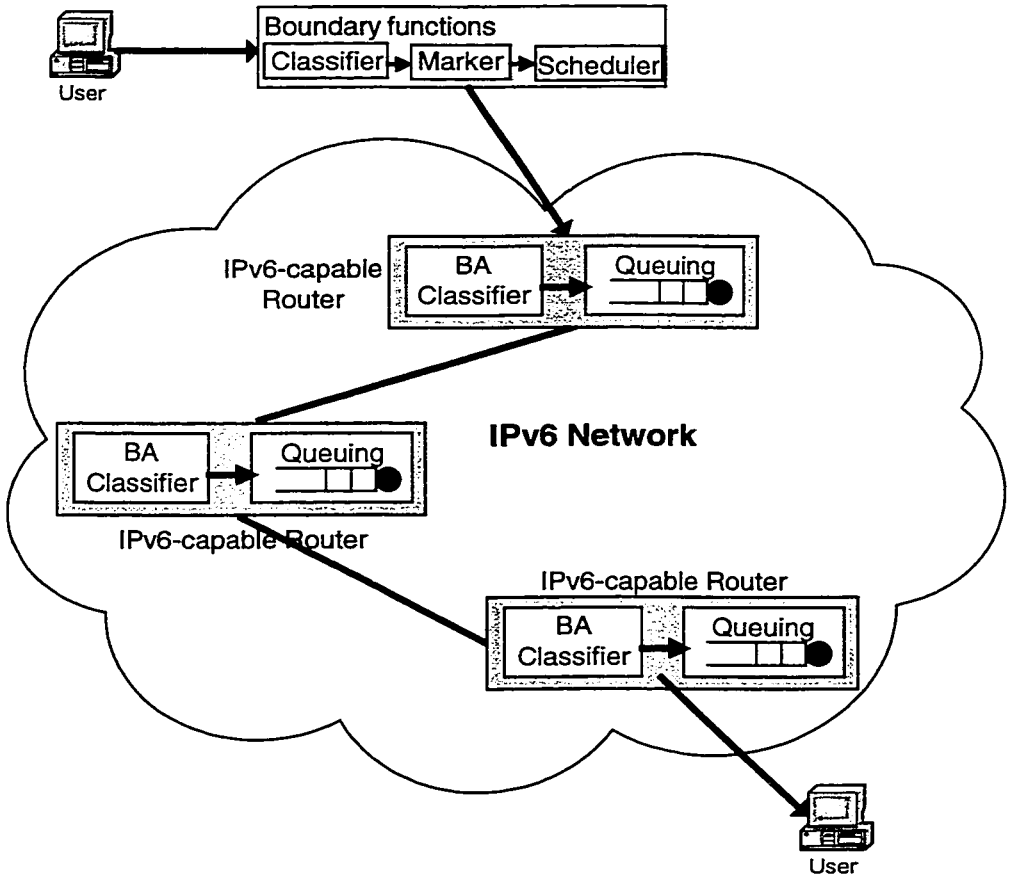

Figure 15: Achieving End-to-End QoS in VESIR-6.

Core routers inside the network implement BA classification, as required by DiffServ standard, and sophisticated classification, marking, scheduling operations are only needed at boundary of the networks. This allows core routers forward packets very fast, and boundary routers that links slow customer connections spend more time on

classification, policing, and shaping. Figure 15 shows what happens to the traffic through the intermediate network nodes and how end-to-end QoS is achieved in VESIR-6.

IP QoS and IPv6 standards are evolving and being tested simultaneously, and IPv6 vendors are implementing those QoS architectures on their products. Hence, by implementing VESIR-6 on top of IPv6, we are guaranteed that most of the network nodes would be DS-aware. Our architecture improves the overall performance of high priority packets, since they are less likely to be dropped by DS-capable IPv6 routers, and achieves the end-to-end QoS in VESIR-6.

7 Implementation

We have used the ideas and innovations described so far in the thesis to implement our distributed virtual environment, VESIR-6. In this chapter, we will follow a bottom-up approach, starting with low level network aspects, then explaining the implementation of individual system modules and achieving communication among them, and finally how we implemented the system to ease its integration to the Internet by means of ubiquitous web browsers.

The software that runs on the client side is actually composed of several different and largely independent pieces of code, which are combined using standardized interfaces. The VRML browser presents the 3-D world to the user. The networking module is implemented as a DLL library that access IPv6 stack on the user machine, and hide the low-level network protocols from the user. This means that changes can be made to the networking module without affecting any of the higher-level application code. Similarly, the API can be redesigned if necessary, with the minimal change in the interface only.

7.1 MCRLab IPv6 Testbed

The IPv6 implementation MUSICA-IP [40], from Dassault Electronique, was installed on three Windows NT machines in the lab as shown in Figure 8.

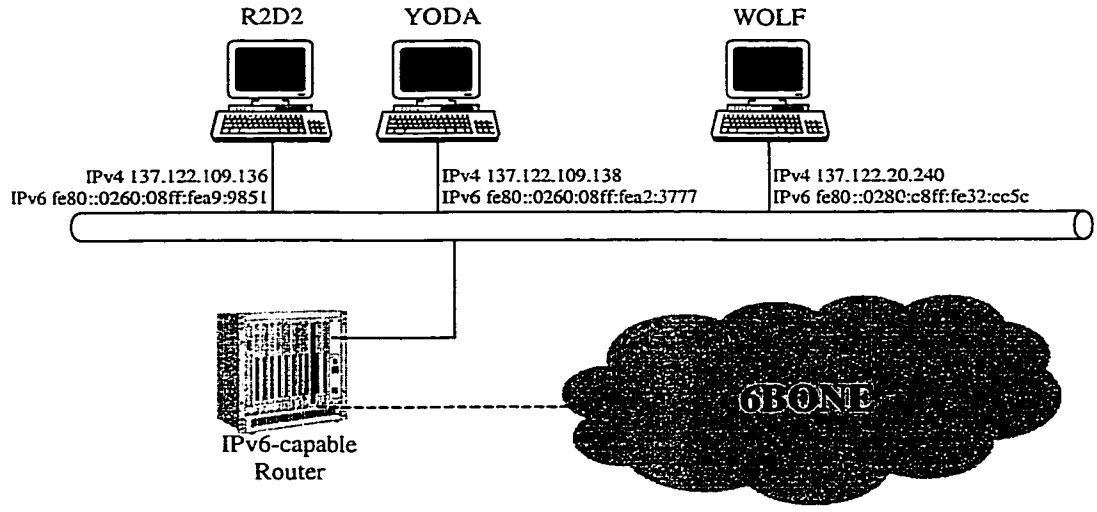

Figure 16: MCRLab IPv6 testbed.

Although the university router is able to support IPv6 routing, the system administrator is not ready to install the necessary upgrade software since the software is a

beta version and not totally stable yet. Without an appropriate router, IPv6 multicast communication is not possible, since it is the router's responsibility to manage the multicast groups. For this reason, we deployed the first communication architecture described in Chapter 3 for IPv6 message distribution in VESIR-6. We also implemented and tested the other architectures over IPv4 multicast.

MUSICA-IP operates on a dual IPv4/IPv6 stack architecture, as required by the IPv6 standard. It includes a module that performs some local resource reservations by using the QoS features of IPv6 in order to obtain service differentiation on the network [41]. The following diagram describes the modular architecture of the MUSICA-IP product.

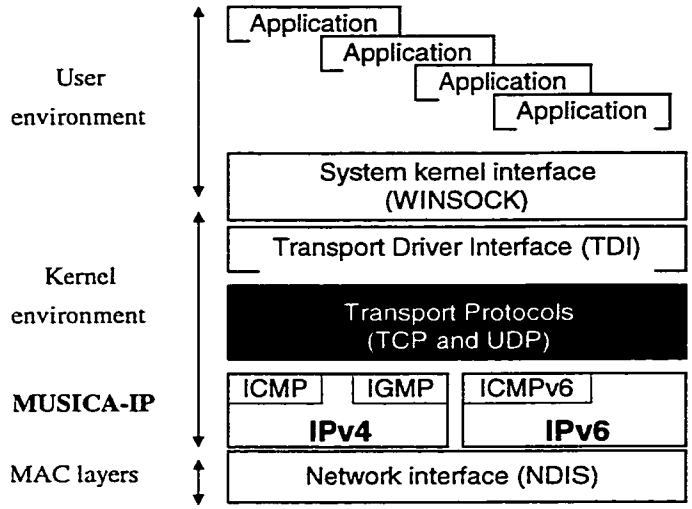

Figure 17: MUSICA-IP architecture for Windows NT.

The Dassault Electronique IPv6 stack can coexist with another IPv4 stack provided that they are not bound to the same IP addresses.

7.2 Networking Module

The networking module resides at the user computer. It is primarily used to encapsulate network calls and to access the IPv6 stack on the host machine. The necessary IPv6 network calls are integrated into a Dynamic Link Library (DLL) file called "member.dll", which is loaded by the application dynamically at runtime. The network packet types and the interaction between networking module and group manager are specified by VNCP.

The networking module is composed of two independent threads: One for forwarding the data to be transmitted to the network, and the other for receiving data from

the network. Each user's environment has its own receiver and sender components, to communicate with other users' environments. Both sender and receiver run asynchronously with events and virtual agent processes. This ensures fast response time and good performance over low bandwidth networks. The data exchange in both cases is achieved by means of IPv6 socket connections.

7.2.1 Sender Thread

The sender thread is entirely responsible for posting the state changes to the network. The VIAs call this module if there are any changes to the shared VE that need to be posted. The ability of this thread is to minimize packets actually sent on the network, by applying a filtering mechanism. The packet filtering mechanism deployed in VESIR-6 is based on the ownership of the objects. Since VRML has no access control to the objects, all users are able to modify the objects within the VE. Thesender thread checks if the user has the right to modify the object before sending any event, hence greatly reducing the number of packets sent over the network.

The sender thread is one of the key components towards achieving scalability for the networked VE. The simplest implementation has this thread sending packets, if there are any, to the network as fast as the thread can run. With our improved implementation, the sender thread has the ability to reduce the network traffic by aggregating the update messages that arrived within a time limit, of no more than 100ms in order to satisfy human perception, into one network packet.

7.2.2 Receiver Thread

The receiver thread listens to the network and picks up the messages that are destined to the user. This thread runs as fast as it can read packets, process the packets, call the appropriate function and return.

A benefit of putting the network read processing into a separate thread is that, in this way, a blocking read of the network can be performed rather than a no-wait, non-blocking IO call. This reduces calls through the operating system's kernel and allows the thread to be put to sleep when there are no packets to read, hence freeing up the processor.

7.3 Group Manager

The group manager is a stand-alone IPv6 server, which is implemented in C++. It implements a hierarchical linked list for saving the virtual world contents. It accommodates the system users inside the groups they belong, and places the objects under their owner members. This storage mechanism is very efficient in the sense that lookups and modifications are fast. Separating the users into multiple groups allow the environment be divided into spatial partitions, so that each region in the virtual world would correspond to an individual group. We have included all necessary network functions in order for the users to move from a group to another. The applications using spatial partitioning need this ability in case the user moves from one region in the VE to another.

The group manager can be run on one of the machines that runs VESIR-6 application or on a separate machine. The only requirement is that the user machine must have an IPv6 implementation installed prior to running the application.

If compiled with DEBUG preprocessor definition, the group manager is able to log the interactions with the members to a text file. This might be useful for accounting purposes and user statistics.

7.4 The User Interface

We built our worlds using VRML, and coded the user interface in Java. VRML is a file format for describing interactive 3D objects and worlds. Java is a platform-independent programming language that suits Internet applications. An applet is a compiled Java program embedded into a Hypertext Markup Language (HTML) document, transported across a network, and executed at the machine that downloads it. Our interface to the VRML browser is via EAI. EAI allows us to easily build and access the virtual world. We took advantage of Java's multithreading capabilities. We coded low-level network communication part of the system in C++ and kept the high-level functionality in Java. JNI was used as a bridge to IPv6 network calls encapsulated in the networking module (DLL file).

We present our own interface to the user, as shown in Figure 18, for interaction, communication, and navigation. We created this interface using Java AWT. When the user is ready to enter the multi-user system, he has to activate the button labeled *Join*. When the browser is initiated, the user is asked to type her name and choose or specify

her avatar to log into VE. After the user has successfully logged in, all the users in the VE are sent by the group manager, and displayed in the virtual world, with their names listed in the user list window. When the user logs in, the names of all users already inside the world are displayed in the user list window. The lower-right window is the chat output window that displays the text messages communicated among distributed players; the user types in a message from the chat input box.

The users are represented in the virtual environment by avatar objects that mediate interaction. Each object needs a unique identifier in order to support communication among different worlds. These avatar objects not only represent the physical appearance of the user, but also define some gestures or simulations. In the snapshot of the user interface of VESIR-6 shown in Figure 18, the avatar object has "laugh", "smile", "frown", "yes", "no" predefined gestures loaded dynamically from the VRML file during object creation.

The VRML file embedded to the application includes the initial world model. The user defines or chooses his avatar and joins the system. The applet sends a join message to JNI, which in turn multicasts this message over IPv6. The group manager issues a unique object identifier for this user and sends it directly to him/her. The group manager first sends the avatars of the members in the group and then the avatar of the new member is multicasted to the group. The user's avatar is then represented in the system.

Communication among users takes place in two ways: Avatar object movements, when a user changes its location, this is reflected in all members of the world. The other means of communication between users is text-based; a user can send a message to any user in private or multicast it to all.

When the user moves inside the world, the applet is informed by means of the VRML plugin. The applet then has to notify other participants about the new position and orientation of the user.

7.4.1 Software Components

The user interface is divided into several almost independent components. This allows minimal dependency among the various components, and easy extension of the system. The software components used are **Scene**, **UserPanel**, **UserList**, **ObjectDatabase**, **Sender**, **Receiver**. **VRMLObject**, **VIA**, and a set of wrapper classes for VRML fields.

Scene is the main applet that is called when running VESIR-6. It creates all other classes and maintains the states of the environmental objects in the region and interaction agents. Updates to environmental objects are propagated to other participants using the networking modules. The applet is responsible for parsing the world model and extracting the information inside it as it is delivered during initialization, determining a beginning viewpoint, rendering the scene based on that viewpoint. It creates an IA for each object in the world. VIAs are basically responsible for user's interaction with the scene. Location changes of the objects, avatar behaviors, or any other kind of change is triggered by VIAs, and sent to the networking modules. The messages coming over the network are also decapsulated and the world is updated accordingly.

UserPanel provides an interface for the user to perform several actions. Among those actions, the most important one is navigating in the virtual world. This panel offers the user to attach the camera to his avatar, or let the avatar stay while the user navigates the world. There is a list of the users connected to the system. The user might click on any avatar in the virtual world or select any user in the list to locate her in the VE, or send her a message. The user panel also provides scrollbars to modify the selected objects appearance, including position, orientation, color, and scale. The user can modify the objects they own by setting those fields. The users might ask for the ownership of the objects that they have no right to modify. The user might choose to grant or reject the right when other users ask for the ownership of an object controlled by her.

UserList is an abstraction for name to object ID conversions. It has two hash tables: One for name to ID mapping, and the other for ID to name mapping. This list is integrated to the user panel, and allows users select or locate particular users within the VE.

VRMLObject is an abstraction for the objects in the world. Each object has an identifier, a name, a URL address of its VRML definition, an owner, and possibly some gestures.

VIA is the implementation of interaction agents. It inherits all the fields in the **VRMLObject** class, and defines extra fields in order to track the object actions. It adds the event routes necessary in order for the object to be able to move with the user viewpoint. It can show the user name on top of the avatar object in the virtual world, by slightly modifying the VRML definition of the object, if the user prefers so. It parses the VRML file and loads the gestures associated with the object, if there is any. It implements a

callback function for the object; hence, if the user clicks on the object in the VE, an event is generated. This event, in VESIR-6, is used to provide information about the object. This ability could be useful in many applications, for example, in a combat simulation, this callback mechanism could be used to fire on the specified object. In this case, the user in a jet craft might click on a target tank object to fire a missile and destroy the tank object.

ObjectDatabase is the place where objects are stored locally. All objects are of type **VRMLObject** and stored in a hash table. This class implements the necessary functions to get objects from the database, to add new objects to the database, to remove some of them, to modify their ownership, and to update their fields.

Sender class encapsulates the update messages and passes them to the networking module. It is the primary access point to the underlying network.

Receiver class listens to the network, reads the packets destined to the user, decapsulates them, and forwards them to appropriate modules. Since reading packets from the network is a blocking system call, i.e. the application is locked or suspended until there is something to read, this class creates another thread to do that job.

7.4.2 Object Management

The users are able to grab objects they own and move them within the shared VE. Object management is a joint effort of the user interface and VIAs. The main issues in object management are the consistency and ownership issues. The consistency is achieved by separating the object state updates from event updates by VIAs, and assigning higher priority to them by the networking module. The ownership management is achieved by the user interface, where users can transfer ownership of the objects if requested. Our user interface provides the user the necessary means to create, delete, or ask for ownership of an object.

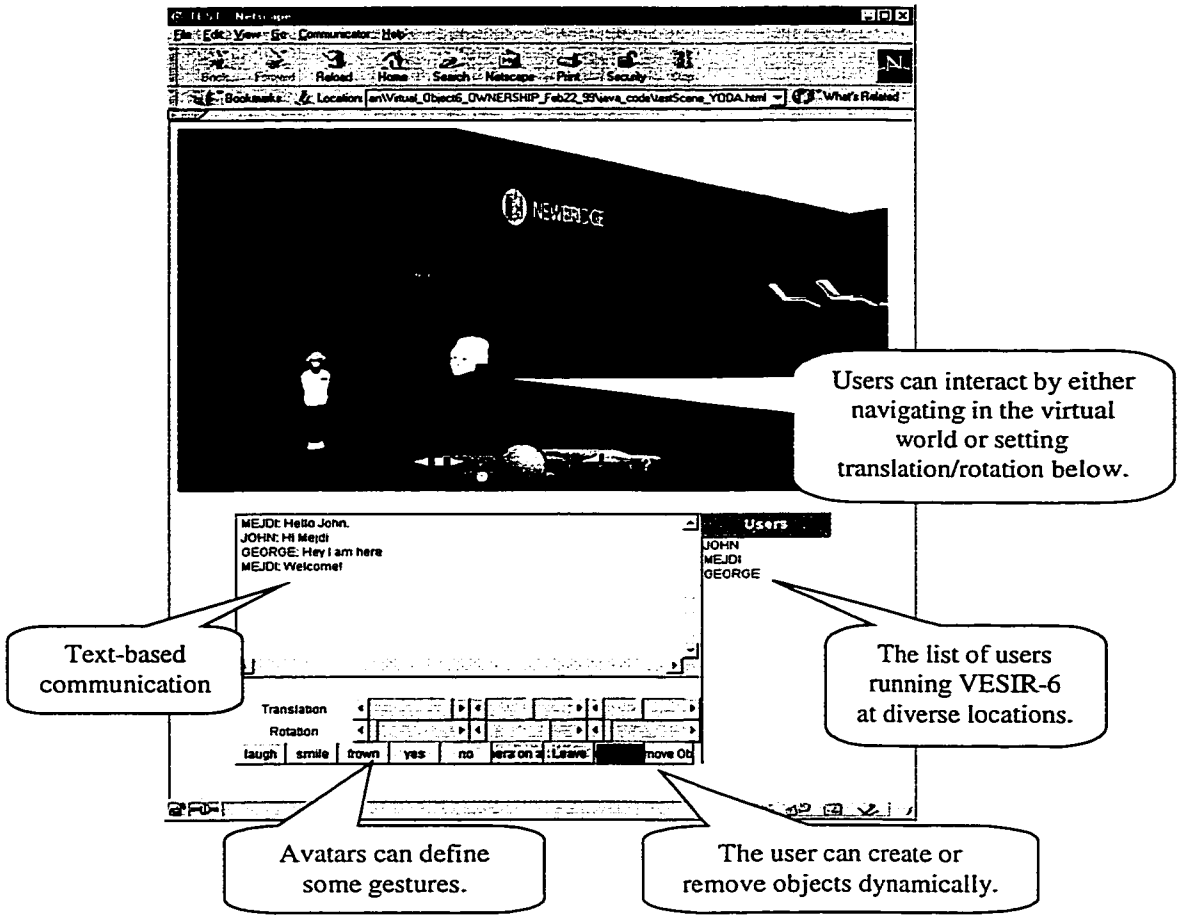

Figure 18: A snapshot of VESIR-6 application running over IPv6.

When users exit the application, the networking module sends a message to the multicast group to disconnect from the system, and then the active threads and sockets are closed, and the system resources are freed.

7.5 Interconnectivity of software modules

The software components should interact with each other by means of appropriate interfaces and the overall system has to be platform independent by the use of standardized protocols. The system is run via browsers that support VRML [1] and External Authoring Interface (EAI) [42] powered Java to ensure platform independence. A VRML plugin is a software module that is capable of processing 3-D type of data and rendering geometry, and is installed separately from the browser. The plugin renders the scene and responds to user input. The EAI is an API that allows Java applets to access the VRML scene graph in a browser window on the same HTML page. It enables developers to create applets that are able to change, add and delete VRML components dynamically.

The IPv6 networking component is accessed from the user interface via the Java Native Interface (JNI) [43]. Figure 7 shows the interconnectivity of the various software modules used in the system.

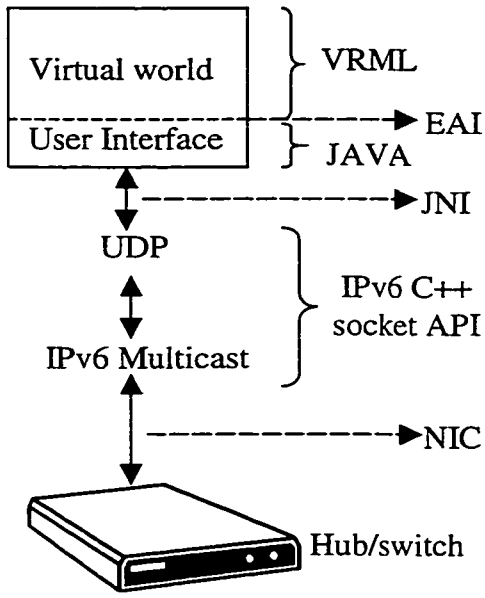

Figure 19: Interconnectivity of software modules.

7.5.1 Interfacing Java to IPv6 C++ stack

MUSICA-IP, like many other IPv6 implementations, is written in C++, therefore the socket calls to IPv6 network functions have to be made using C++. One of the design considerations for VESIR-6 was, however, that it has to be platform-independent and easily integrated to the web. For this purpose, we programmed the user interface and most of the system in Java. For collaboration purposes, we need to call IPv6 C++ socket calls. The Java Native Interface (JNI) allows Java code that runs within a Java Virtual Machine (VM) to operate with applications and libraries written in other languages, such as C, C++, and assembly.

The JNI framework lets the native method utilize Java objects in the same way that Java code uses these objects. A native method can create Java objects, including arrays and strings, and then inspect and use these objects to perform its tasks. A native method can also inspect and use objects created by Java application code. A native method can even update Java objects that it created or that were passed to it, and these updated objects are available to the Java application. Thus, both the native language side and the Java side of an application can create, update, and access Java objects and then share these objects between them.

The native method, using the JNI framework, can call the existing Java method, pass it the required parameters, and get the results back when the method completes. It is easy to see that the JNI serves as the glue between Java and native applications. The following diagram shows how the JNI ties the C side of an application to the Java side.

The JNI is the native programming interface for Java that is part of the JDK. By writing programs using the JNI, you ensure that your code is completely portable across all platforms.

7.5.2 Interfacing VRML to Java

VRML is a powerful language to deliver compelling animated, interactive 3D content on the World Wide Web. For communication between a VRML world and its external environment an interface between the two is needed. This interface is called an External Authoring Interface and it defines the set of functions on the VRML browser that the external environment can perform to affect the VRML world. With the advent of the EAI, many of VRML's behavioral aspects can be moved to an external Java applet. The EAI allows the user to control the contents of a VRML browser window embedded in a web page from a Java applet on the same page. It does this with a browser plugin interface that allows embedded objects on web page to communicate with each other.

The External Authoring Interface comprises a set of functions which include accessing nodes and their fields, sending and getting messages to and from `eventIn` and `eventOut` fields in the world, and adding nodes to, or removing nodes from the world.

7.5.3 Protocols used in VESIR-6

The applet sends events using VIAP to the VRML scene to control it, and the scene sends events to the applet through a callback mechanism. The Java applet can exert so much control over the VRML world that it can literally construct a 3-D scene from scratch, then send it to VRML for rendering.

Avatar objects are first encapsulated as high-level object prototypes in VRML, then instantiated by Java. These objects are then controlled from Java by sending events to their interaction agents, which are specifically designed for the control task.

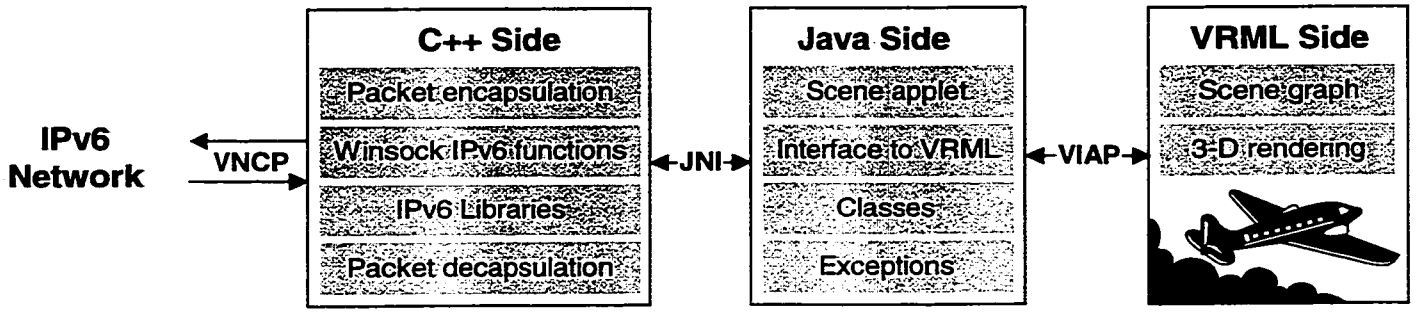

Figure 20: Protocols used in VESIR-6.

7.6 Event Handling via VESIR-6 Interaction Agents (VIAs)

The interaction with the virtual world in VESIR-6 could be done in two ways: By means of the VRML plugin panel that is provided in most VRML viewers, or by controlling the objects from the user interface. Users can navigate the virtual world by clicking and dragging the control panel provided by the VRML-plugin (this could be any VRML compliant browser, including CosmoPlayer [44] and Blaxxun Contact [45]), while other users observing the movement of user's avatar. The VRML plugins provide a navigation mode such that users can walk, fly, or examine the world, using the keyboard or the mouse, or the control panels provided with the plugin.

The event interaction between objects is achieved by the VIAs. The VIAs provide a powerful mechanism to interact between the VRML world and the Java applet. VRML defines an event or message-passing mechanism by which nodes in the scene graph can communicate with each other. Each node type defined the names and types of events it generates or receives. ROUTE statements define event paths between event generators and receivers. Routing creates reactive behaviors by using sensor nodes or scripts to initiate changes in the scene graph.

We use a Java applet to create and control instantiated prototypes defined in VRML, through the VIAs. Creating effective VRML prototypes is essential for allowing large worlds to be composed of many smaller, separate pieces. By encapsulating the models inside of prototypes, it is possible to hide the details of implementation of the model and its behavior, while exposing the attributes of the model that vary on an instance by instance basis. The creation of prototypes with built-in behaviors and

interactions provides a powerful way to create components that are reusable from one scene to another.

The VIAs provide a means of communication between a Java applet and a VRML world. While the EAI interface can be used to give Java complete control over the creation and manipulation of the scene, this does not take advantage of the powerful features of VRML. Instead it duplicates them in Java with a loss of performance and programming simplicity. VRML has the ability to create and manage scenes that include behaviors, animations, and the relationships between objects. With its ability to package functions into prototypes, an easy to maintain and extend object system can be created. The Java applet is better used controlling these objects at a high level, instantiating them as needed. A prototyped object system, controlled by Java, allows the author to create a consistent and easy to maintain multimedia system, which is able to take advantage of the many facilities built into VRML for animation and user interaction. VIAs use their own protocol to exploit those advantages.

7.6.1 VESIR-6 Interaction Agent Protocol (VIAP)

The VIAP is a simple, compact protocol for sending VRML fields to the networking modules to allow multi-user participation in a virtual world. The protocol is mainly used by the user interface and VIAs. VIAP is a toolkit to implement VEs in which the user is able to navigate and interact with objects. The user is able to associate pre-defined object behaviors with objects in the scene as well as defining special gestures for objects. Several instances of VIA may be used across a network to create a shared VE with several users. The user is able to call VIAP API in Java.

VIAP is a protocol for encapsulating the interactions between a virtual environment and Java applet. Each VIA behaves as the controlling agent for the object it owns. VIAP is used to communicate the state of objects from the virtual world to the Java applet.

All basic data types used by virtual agents (integer, boolean, float, double, string) are encoded using the same format. VIAP specifies encoding for each of the field type used in VRML. The field encoding used for event updates is shown in Table 2.

Field	*Type*	*Encoding*	*Description*
POSITION	SFVec3f	float, float, float	The position of the object
ROTATION	SFRotation	float, float, float, float	The orientation of the object
SCALE	SFVec3f	float, float, float	The position of the object
COLOR	SFVec3f	float, float, float	The position of the object
NAME	MFString	string	The name of the object.
GESTURE	SFBoolean	boolean	Turn on/off the gesture, if any defined, of the object

Table 2: VIAP field encoding for event updates.

The packet type used by VIAP is the same as VNCP in order to achieve interoperability. All messages consist of three parts: an object identifier, type of the message, and value. The Object identifier is a 32-bit integer whose value represents the object whose field is to be changed. This identifiers are assigned by the group manager when an object is created, and are integrated to object definitions. Thus, all members refer to a specific object by the same object identifier. The type field is a signed 16-bit integer that represents which field of the target object should be set. Currently, five fields are reserved with specific values as shown in Table 2. The messages that do not confirm to those fields in virtual objects are sent by direct method calls. For example, to send geometry changes to the scene, such as "add the object named myObject with a URL myAvatar.wrl", the user interface calls the `AddObject(myObject, myAvatar)`, and the networking module does the necessary packet encapsulation to VNCP format.

7.7 VESIR-6 and WWW Integration

VESIR-6 is designed to run with any web browser that has a VRML plugin and supports Java 1.1 or later versions. VESIR-6 enables 3-D, interactive and multi-user web interfaces be provided over the Internet. It addresses multi-user and interaction issues. It supports dynamic object creation, and ownership mechanism to prevent contention for accessing shared objects. It might use any network topology providing the networking module API, according to the resource availability.

Multi-user VE applications face a challenge in deploying their software to potential participants. For example, if the client software is large and monolithic, then it is inappropriate for downloading. In order to solve this problem, VESIR-6 was designed alternatively with a small core and components that may be dynamically downloaded depending on the changing need of the executing environment. The core package is slightly larger then 300kb, and easy to download using a modem connection. However,

the virtual world, depending on its complexity, could be very large. In that case, the initialization might take several seconds, but once the world is downloaded the user can start interacting with the system.

To run VESIR-6 over the Internet, the user needs the address of the web site that contains the necessary information to get connected to the system. This information includes the virtual world model, the address of the group manager, the local interface (IPv6 allows many interfaces per machine), and the applet code. On startup, the web browser loads the HTML page containing an embedded VRML document and a Java applet. The format of this file is shown in Listing 3. The VRML document is displayed by a VRML plugin.The applet code then is downloaded to the client machine by the browser. The browser passes the information in the PARAM tags in the HTML file to the applet. The applet then runs on the client machine, and calls the networking module, which is a DLL library loaded at runtime. The networking module sets connection to the IPv6 stack on the user machine.

```
<HTML>
<EMBED SRC="AnyVirtualWorld.wrl" BORDER=0 HEIGHT="75%" WIDTH="100%">
<APPLET CODE="Scene.class" WIDTH="100%" HEIGHT="20%" MAYSCRIPT>
<PARAM NAME="LOCAL_IPv6_INTERFACE" VALUE="yodav6">
<PARAM NAME="GROUPMANAGER_IPv6_ADDRESS" VALUE="r2d2v6">
<PARAM NAME="PORT" VALUE="2910">
</APPLET>
</HTML>
```

Listing 3: The HTML file to run VESIR-6 over an IPv6 web browser.

Using VESIR-6 with HTML pages and Java applets achieves platform independence, and is very effective.

7.7.1 The World Model

Modeling of the environment is an independent component of the system. The world model is specified in the HTML file, and any world model can be deployed in VESIR-6, as long as they obey the VRML standard. The VRML standard defines the world model and stores it in a simple ASCII or UTF8 text-based format. VRML extracts the geometry, appearance, location, orientation, and scale of each object defined in the text file, and then puts the objects together to make a 3D scene. The virtual worlds are typically created using 3D modeling tools. The model is created visually using a computer program with a special graphical user interface that makes it easy to point and

click on points in 3D space to insert objects and define their geometry appearance, location, orientation, and scale. Modeling tools could also be used to create the avatar objects, which enables non-verbal communications within the multi-user world. Once it is created, the virtual world is delivered to each participant in order that all participants collaborate in the same space.

The 3D Render Engine module provides a visual window for the user to explore the virtual world. When a browser retrieves a scene file from the Web server, it is an ASCII file. The scene file is then translated into a virtual world database recognized by the 3D Render Engine Module.

The virtual world is embedded to the application. It includes a set of nodes (avatar prototype, sensors, and geometry) to support interaction with the Java applet. A standard web browser and VRML plugin is used to navigate within the virtual world and control is achieved by the Java applet running on the browser. On initiation, the user is presented the virtual world embedded into the browser together with the applet. The applet controls the user interface, the interactions to the objects in the world, to create new objects, to delete objects, to send text-based messages to other members, etc. Based on initial choices, the user joins the multicast group. Once loaded, the user may start to interact with the environment. This is either by moving the shared objects, or sending some text messages. As new users connect to the system, their avatars become visible within the virtual world. After initialization, a hash table is created to map the object names to unique object IDs in the system. The user list is then presented to the user.

7.7.2 Using VRML Prototypes to Encapsulate Avatar Behaviors

Users in VESIR-6 can communicate with one another via humanoid multi-user avatars. The members may choose one of the avatars the group manager provides to represent themselves in the virtual world, or can define their avatars to the system. Avatars have the ability to encapsulate some behaviors inside their VRML definition. VRML includes a prototyping mechanism for encapsulating and reusing a scene graph. This is achieved by using a PROTO statement to build a generic set of objects and embed geometry, properties, animations, and dynamics inside them. The PROTO node allows an author to encapsulate all characteristics and behaviors of an object [46].

```
#VRML V2.0 utf8
EXTERNPROTO Avatar [
    exposedField  MFString behaviors [ "laugh" "smile" "frown" "yes" "no" ]
    eventIn       SFBool      laugh
    eventIn       SFBool      smile
    eventIn       SFBool      frown
    eventIn       SFBool      yes
    eventIn       SFBool      no
] "http://yodav6.mcrlab.uottawa.ca/Avatar.wrl"
Avatar { }  # Instances an Avatar
```

Listing 4: Encapsulating avatar object behaviors in VESIR-6.

The EXTERNPROTO statement permits fetching new PROTO definitions from anywhere on the Web. This serves as the basic extensibility mechanism for VESIR-6 where users can choose one of the avatars defined by the group manager. Creating effective VRML prototypes is essential for allowing large worlds to be composed of many smaller, separate pieces. This allows the object prototype to be used in the virtual world, by simply instantiating the prototype, and placing it in the desired location in the scene. The VIA constructed for the object can now be controlled from the Java applet to modify its behavior. Hence, the internal definition of the prototypes used to implement the animation inside VRML can change without changing the external interface.

By encapsulating the models inside of prototypes, it is possible to hide the details of the implementation of the model and its behavior, while exposing the attributes of the model that vary on an instance by instance basis. The creation of prototypes with built-in behaviors and interactions provides a powerful way to create components that are reusable from one scene to another.

This combination of prototypes, internal and external scripting puts a great deal of power into the author's hands. The internal definition of the prototypes used to implement the animation inside VRML can change without changing the external interface. Also, the way in which the user interacts with the HTML page to start the animation can be modified without changing the implementation inside VRML. This will allow large systems to be authored by teams of artists and engineers, whose pieces can be fit together without a large amount of overlap. With the proper separation of VRML implemented components and Java implemented components, truly powerful and compelling multimedia content can be produced.

VIAP does not only send the hardcoded set of location messages, but also the gestures that objects might have in their VRML definition. These behaviors are extracted from the avatar VRML file dynamically, enumerated by the client and given unique field identifiers to distinguish them from other kind of messages used in VIAP. For each behavior the object inherits, a button would be added to the user interface panel to allow the user to trigger them. The user can modify the avatar objects, which include realistic motions and facial expressions, with the user interface. When the owner of the object clicks one of these buttons and turns a behavior on, the corresponding VIA sends a simple `SFBool` message with the index of the behavior, since each object might define several behaviors, and the unique object identifier of the avatar to the networking module. The networking module, in turn, would do necessary protocol conversions and forward this message to IPv6 multicast group. The other members of the multicast group, would then receive the message, and pick the appropriate behavior field from the object `PROTO` definition and trigger it, causing an animation of the object in the virtual world. More complex interactions could be built using the other VRML data types. Objects themselves could initiate asynchronous events by listing such fields in the avatar `PROTO` in a similar way.

Encapsulating avatar behaviors in a `PROTO` type and extracting them dynamically by means of VIAs separates the design issues of graphics animation and user interface. Hence, application developers have the chance to improve the graphics animation in the avatars and VRML world, without worrying how to integrate them to the system, as VAIP extracts the necessary information from the VRML file and acts like a bridge between the object in the virtual world and user interface.

7.8 VESIR-6 Trials over CA*net II

VESIR-6 has been used in a project called "Distributed Interactive Virtual Environment (DIVE) over CA*net II: QoS Management, IPv6 and Performance Tools" [47] and funded by CANARIE [48]. CANARIE's mission is to facilitate the development of Canada's communications infrastructure, stimulate next generation products and applications for the marketplace. As part of its commitment to advancing the information highway in Canada, CANARIE has undertaken the funding and deployment of a national Internet network, called CA*net II [49]. The CA*net II initiative is very similar to the Next Generation Internet [50] and Internet 2 [51] initiatives in the US. The purpose of

CA*net II is to test and deploy next generation Internet technologies and services and to support advanced meritorious research between leading research institutions and universities, particularly those applications and research that cannot be carried out on today's existing commercial Internet.

In order to show the capabilities and possible applications of VESIR-6, we have tested it by launching a 3-D telelearning application among three IPv6 islands linked through CA*net II. The participating nodes at University of Western Ontario (UWO), Communications Research Center (CRC), and University of Ottawa (UofO) ran the application on their browsers by simply specifying the IPv6 address of the group manager which could be any IPv6 node with global connectivity, and the local IPv6 interface (if no interface defined then the default is used). The application over CA*net II is shown in Figure 21 below.

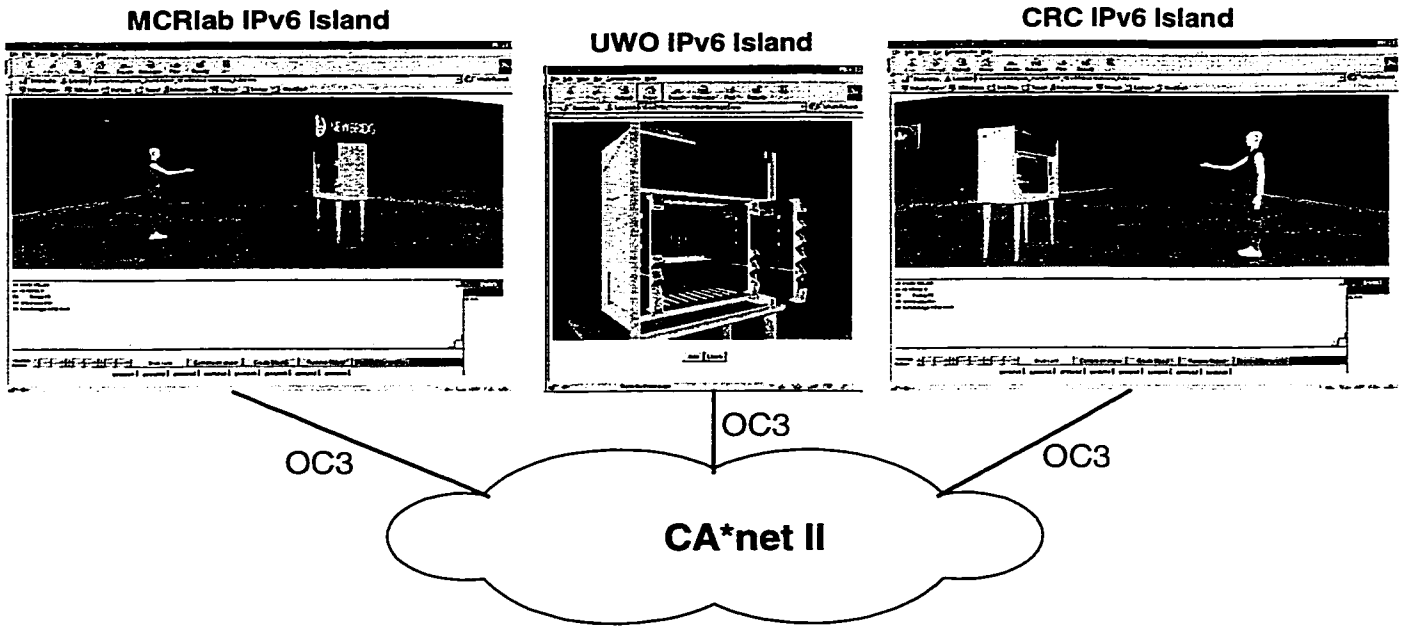

Figure 21: VESIR-6 trials over CA*net II.

Since not all the participating sites had a dual stack router, we had to tunnel IPv6 packets over IPv4. In doing this, we had a firewall problem as the encapsulated packets had their protocol type field value equal to 41, which is the required type for IPv6/IPv4 tunneled packets and was not allowed to go through by the filtering mechanism. We contacted the system administrators and fixed the problem. In the early stages of IPv6 deployment, we believe that there would be similar problems, since IPv6 packets, even when they are tunneled over IPv4, have different protocol types than IPv4 packets, and most of the existing firewalls in the current Internet do not let them pass through.

This trial provided a demonstration of the way that an important application, distance training, can be implemented in a more powerful way, through a distributed interactive virtual environment, over the next generation Internet protocol.

8 Related Work

There is much research done on IPv6 protocol, there is also lots of effort in developing scalable multi-user VEs, however, there is no effort yet trying to combine those two research areas. IPv6 is still in its infancy, the entire IPv6 network is only a small backbone that connects mostly research sites over the world, and applications critical for IPv6's deployment such as IPv6-capable web browsers have not been in use widely yet, all of which restrain the interest in developing applications on top of IPv6. This chapter, therefore, gives the related work in developing large-scale VEs over the existing IPv4 network infrastructure. The related work to VESIR-6 can be divided into two research areas: One that enables the communication over the network in an efficient manner and the other part is the multi-user VE applications themselves.

8.1 Communication Protocols for DVEs

Communication protocols developed for DVEs are considered as "middleware" because they separate the network and operating system considerations from the application design. The development of large-scale virtual reality and simulation systems has been mostly driven by the DIS and HLA standards community. The three standardized DVE communication protocols defined so far are described below.

8.1.1 Distributed Interactive Simulation (DIS)

The DIS protocol convey messages about entities and events, via a network, among various simulation nodes that are responsible for maintaining the status of the entities in the virtual world [52]. It is the approved IEEE 1278 standard for logical communication among entities in distributed simulations. DIS was initially developed for military users and inherited properties of the Simulator Networking (SIMNET) protocol [53] that was designed for use in Ethernet LANs and in which communication was achieved using broadcasting. DIS started using IP multicast in 1995 but the keep-alive messages that each host should send periodically introduces burden in the network, especially when an object is stationery. A second problem with DIS is that, joining is not rapid, because the users become aware of other participants by means of keep-alive messages only. Consequently, it takes more than five seconds for a new user to get the complete information about the environment. This problem is fixed in VESIR-6 by

including a central database, called group manager, in the system architecture to support rapid joining.

8.1.2 High Level Architecture (HLA)

A new approach being developed by the Defense Modeling and Simulation Office (DMSO), called the High Level Architecture (HLA) [54], is aimed to replace the DIS standard on the long-term. The HLA moves the interface from the messages to an API that resides between the simulation applications and a layer of standard services called the Runtime Infrastructure (RTI). The HLA supports the capabilities of DIS but in a more general way. In the DIS architecture, a simulation joins a distributed exercise by connecting to a communications network, and members interoperate by transfer of Protocol Data Units (PDUs). In contrast, in the HLA, each simulation operates within the virtual world via a common software module known as the RTI. While each federate (exercise member) has only one connection to the RTI, the RTI can be distributed throughout the "federation". The HLA RTI utilizes a standardized set of services to allow each federate to interoperate with the other federates in the exercise. Among these services are starting, stopping and pausing an exercise, intelligent publication and subscription of state data, time synchronization and event ordering, and exchange of the responsibility for modeling object attributes. The RTI relies on UDP/IP services to move the data. In December 1997, HLA was accepted as a draft IEEE standard to be supported by the Simulation Interoperability Standards Organization [55]

8.1.3 Interactive Sharing Transfer Protocol (ISTP)

The ISTP supports the sharing of information about a virtual world among a group of users. It breaks the virtual world down to regions called locales [56]. The ISTP provides scalability by associating each locale with a separate multicast communication group. Object updates are transferred by using UDP/IP multicast, and not to waste bandwidth reliability constraints are relaxed for event updates, as done in VESIR-6.

8.2 Applications

Multi-user VE applications have originally evolved from distributed interactive simulation experiments for military purposes. While applications in military area follow the communication standards described before, most of applications developed by

research institutes or entertainment industry tended to develop their own proprietary protocols. Now we will give a brief overview of the multi-user DVE applications.

NPSNET is a distributed VE for large-scale military simulations that deploys IP multicast and supports few hundred users in the LAN and over dedicated T-1 links [57]. It implements the DIS protocol.

The Scalable Platform for Large Interactive Networked Environments (SPLINE), by Mitsubishi Electronic Research Lab, provides a communication protocol that achieves interoperability by use of a common interface [58]. SPLINE communicates with a world model, an object-oriented database, rather than directly with each other. The world model may be distributed across a network or could be local, and interaction with that world model is through one API. The world model process has methods executable via calls across the net, with some of those methods just reading/writing instance variables of world-model objects. SPLINE also has server processes. The server processes provide compression of network information for low-speed clients in the world and provide information for objects that move into new locales. SPLINE defines its own protocol, called Interactive Sharing Transport Protocol (ISTP) for the packets sent to the network. Using ISTP, SPLINE performs all the processing necessary to maintain a distributed, modifiable, and extendable model of a virtual world that is shared among the participants.

There are other distributed VE applications used for non-military purposes. Among those, DIVE was developed by the Swedish Institute of Computer Science, originally in 1991. The more recent version of DIVE associates multicast groups with object hierarchies to allow the structure of the virtual world to limit communication. Using IP multicast and reducing the communication cost by partitioning the virtual world into zones and exploiting spatial proximity to derive communication relationships, the DIVE architecture supported up to 10 users in the LAN [59].

Model, Architecture, and System for Spatial Interaction in Virtual Environments (MASSIVE) [60], and MASSIVE-2 systems use, in a similar sense to DIVE, a spatial model for data partitioning among clients. They developed the concept of levels of awareness between objects in a large scale virtual environment. Object based filtering is accomplished via subscription agents. The client-server architecture used in MASSIVE increases the latency and network load and therefore is not suitable for large-scale VEs. MASSIVE-2 changes this and pursues the use of multicast communication and weaker data consistency.

Real-time Interactive Networked Graphics (RING) [61], developed by Bell Laboratories, supports interaction among a large number of users in a VE via multiple communicating servers. It uses precomputed line-of-sight calculations to determine which users can perceive changes in a particular entity's state, and then sends updates only to the relevant users.

AVIARY has an architecture comprised of a collection of communicating autonomous objects for which performance is a secondary concern [62]. AVIARY provides a set of standard heavyweight objects to provide the necessary features for a usable VE such as collision detection and name-serving.

VNET [63] is another distributed VE that aims to bring many users into a virtual world. Using a client-server model and TCP connections, VNET provides reliability and consistency but lacks scalability since communication among users is established by means of the server, and messages have to travel through the server, thereby increasing the latency and network load as in MASSIVE.

8.3 Comparison with VESIR-6

There are plenty of DVE applications as we described in the last section, but they all have their limitations. All of these architectures work over IPv4, and they all suffer from low level of quality, high latency, unreliability, unsecured transactions, and complex system design. These distributed VE systems did not carefully manage how they used the network resources, and since networks have limited capacity, deploying them over the Internet would not scale to more than a dozen simultaneous users. There has been no work yet in providing a distributed multi-user VE over the next generation IPv6 protocol and exploiting new capabilities introduced with IPv6. Our work is therefore of great importance in addressing the multi-user VE network service requirements.

The security issues make it hard to execute most of the DVEs within web browsers over the Internet. Especially the systems that use Java network communication are imposed to the security restrictions present in the existing web browsers. VESIR-6 fixes this problem by using native IPv6 communication in the underlying network. It uses the networking module for message distribution over the network, which hides the low-level communication aspects from the application developer. The packets are encapsulated in the VNCP protocol format to minimize the overhead.

Centralized servers have been extensively used to deliver 3D virtual worlds and provide communications between users in most of the current DVE applications, since multicast is optional with IPv4 and not supported by all the routers on the Internet. VESIR-6 deploys IPv6, which requires all network nodes be multicast capable.

Another issue that makes it painful to run most of the DVE applications is the huge size of the files and dependencies on other compilers. For example, the DIVE system needs a Tcl/TK compiler to be installed in the computer before you run the application. VESIR-6 is completely platform independent and has no external dependencies except for the IPv6 stack and proper VRML-capable browser.

In summary, VESIR-6 is a very flexible system that provides a secure, high quality multi-user virtual environment over the next generation Internet. It addresses most of the issues current DVE applications are faced with, and exploits many features of IPv6 in order to provide a secure, low-latency, and highly reliable network infrastructure to the application.

9 Conclusion

Designing a large scale multi-user VE system is a challenging task. In this thesis, we have identified the steps to architect VESIR-6, starting with the low-level network communication aspects, ending with integrating the system to the web. The VESIR-6 system is designed to be so robust and efficient that hardly any new concepts have to be introduced for enhancing the system architecture. It solves most of the problems faced in designing a DVE by meticulous efforts and by exploiting the powerful features of IPv6.

The ability to construct a large-scale distributed VE depends on the underlying network infrastructure. When we analyze the network service requirements for distributed multi-user VEs, we observe that the current Internet protocol, IPv4, is far away from satisfying those requirements. IPv6 comes to address these issues with its capabilities such as multicast, per-packet level QoS support without a need for additional signaling mechanism, authentication extensions, and data privacy and confidentiality. The authentication extensions of IPv6 provide security at network level and will encourage people to do electronic transactions without worrying about any kind of eavesdropping or theft in electronic commerce applications over VESIR-6.

In designing VESIR-6, we first developed the communication architecture for message distribution over the network. We considered three network architectures that provides more efficient packet delivery in large-scale systems. The basic client-server approach encounters scalability problems because all traffic must pass through the central server. As the number of participants increases, the server becomes the bottleneck component in the system. This architecture is expanded to a multicast architecture including a cluster of group servers. Deploying multicast have reduced the number of duplications for each packet and provided a better utilization of scarce network resources such as bandwidth. The users, hence, communicate in a peer-to-peer manner, reducing the latency and avoiding unnecessary packet duplication over the network.

We provided useful abstractions of the services provided by the network layer in the networking module API. In fact, this API could be deployed in other DVE applications to provide the developer easy and transparent access to the underlying IPv6 network.

We have used the features of IPv6 to create a large scale multi-user VE, called VESIR-6, that is capable of working over wide area networks such as the Internet via low bandwidth dialup lines. The network infrastructure deployed IPv6 multicast to transfer state and event updates of the objects within the virtual world. A packet protocol has been specified to encapsulate state data and event updates to transfer them to the IPv6 multicast group efficiently over low-bandwidth dialup links.

We deployed IPv6 service differentiation capabilities to achieve end-to-end QoS and reliability in VESIR-6 over IPv6 multicast that works over connectionless UDP. We proposed a dynamic QoS architecture that incrementally adjusts the service level of the application according to the network conditions. Characterizing the multicast network traffic into different flows and assigning higher priority to the state updates and control commands helped maintaining world consistency.

VESIR-6 makes use of an external information source in order to provide users with query services, fast connection to the system, and although not fully implemented yet, spatial partitioning of the virtual environment to smaller regions in order to achieve scalability. Thus, VESIR-6 combines the best features of centralized unicast and distributed multicast systems.

Finally, VESIR-6 shows the possibilities of using IPv6 capabilities to provide a high-quality network infrastructure for distributed virtual environments over the Internet and can be used for conducting online distributed training, interactive simulations, or electronic commerce applications.

REFERENCES

[1] Information technology, Computer graphics and image processing, "The Virtual Reality Modeling Language (VRML)", International Standard ISO/IEC 14772-1:1997 (also referred to as VRML97).

[2] http://java.sun.com

[3] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[4] http://www.6bone.net

[5] Communication Architecture for Distributed Interactive Simulation (CADIS), Institute for Simulation and Training, Orlando, Florida, 28 June 1993.

[6] Miller, D., Distributed Interactive Simulation Networking Issues, Briefing presented to the ST/IP Peer Review Panel, Massachusetts Institute of Technology, Lincoln Laboratory, 15 December 1993.

[7] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[8] Deering, S., "Host Extensions for IP Multicasting", RFC 1112, August 1989.

[9] http://www.mbone.com

[10] "ISO/IEC 10038 Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Common specifications - Part 3: Media Access Control (MAC) Bridges: Revision (Incorporating IEEE P802.1p: Traffic Class Expediting and Dynamic Multicast Filtering", ISO/IEC Final CD 15802-3 IEEE P802.1D/D16, March 1998.

[11] "IEEE Standards for Local and Metropolitan Area Networks: for Virtual Bridged Local Area Networks", IEEE Draft Standard P802.1Q/D11, July 1998.

[12] Braden, R. Ed. et al, "Resource ReserVation Protocol -- Version 1 Functional Specification", RFC 2205, September 1997.

[13] ATM Forum Technical Committee, "ATM User-Network Interface (UNI) Signaling Specification Version 4.0", af-sig-0061.000, July 1996.

[14] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

[15] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss. "An Architecture for Differentiated Services", RFC 2475, December 1998.

[16] Schmid, S., Dunmore, M., Race, N., and A. Scott, "RSVP Extensions for IPv6 Flow Label Support", draft-schmid-rsvp-fl-01.txt, August 1998 (work in progress).

[17] Partridge, C., "Using the Flow Label Field in IPv6", RFC 1809, June 1995.

[18] R. Atkinson, "Security Architecture for the Internet Protocol", RFC 1825, August 1995.

[19] R. Atkinson, "IP Authentication Header", RFC 1826, August 1995.

[20] R. Rivest, "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[21] R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 1827, August 1995.

[22] Karn, P., Metzger P., and W. Simpson, "The ESP DES-CBC Transform", RFC 1829, August 1995.

[23] Johnson, D. and S. Deering, "Reserved IPv6 Subnet Anycast Addresses", draft-ietf-ipngwg-resv-anycast-00.txt, August 1998 (work in progress).

[24] Rekhter, Y. and T. Li, "An Architecture for IP Address Allocation with CIDR", RFC 1518, September 1993.

[25] Egevang, K. and P. Francis, "The IP Network Address Translator (NAT)", RFC 1631, May 1994.

[26] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, July 1998.

[27] Conta, A. and S. Deering, "ICMP for the Internet Protocol Version 6 (IPv6)", RFC 2463, December 1998.

[28] http://www.tascnets.com/mist/doc/mcpCompare.html

[29] "IEEE Standard for Distributed Interactive Simulation--Communication Services and Profiles", IEEE Standard 1278.2, 1995.

[30] Stallings, W., "Data and Computer Communications", 5th edition, Prentice Hall Inc., 1997, pp. 458-464.

[31] Crawford, M., "Transmission of IPv6 Packets over Ethernet Networks", RFC 2464, December 1998.

[32] Chandy K., Mani. and Misra, Jayadev (1979). Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. IEEE Trans. on Software Engineering, Vol. SE-5, No.5, Sept. 1979, pp. 440-452.

[33] http://playground.sun.com/pub/ipng/html/ipng-implementations.html

[34] Braden, R., Clark, D., and S. Shenker, "Integrated Services in the Internet Architecture: an Overview", RFC 1633, S. Shenker.

[35] Mankin, A., Baker, F., Braden, B., Bradner, S., O'Dell, M., Romanow, A., Weinrib, A., and L. Zhang, "Resource ReSerVation Protocol (RSVP) -- Version 1

Applicability Statement Some Guidelines on Deployment", RFC 2208, September 1997.

[36] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2475, December 1998.

[37] http://www.cisco.com/warp/public/732/net_enabled/qosio_wp.htm

[38] Demers, A., Keshav, S., and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm", Internetwork: Research and Experiences, vol.1, no.1, John Wiley & Sons, pp.3-26, 1990.

[39] Biegi, M., Jennings, R., Rao, S., and D. Verma, "Supporting Service Level Agreements using Differentiated Services", draft-verma-diffserv-ntimplem-00.txt, November 1998 (work in progress).

[40] http://playground.sun.com/pub/ipng/html/ipng-implementations.html#Dassault

[41] Dassault Electronique, "MUSICA-IP - Evaluation Guide", NE857845/AN Iss. 1, 13 October, 1998.

[42] http://www.vrml.org/WorkingGroups/vrml-eai/Specification/

[43] http://java.sun.com/docs/books/tutorial/native1.1/ index.html

[44] http://www.cosmosoftware.com/download/player.html

[45] http://www.blaxxun.com/products/contact/index.html

[46] http://www.vrml.org/technicalinfo/specifications/vrml97/part1/concepts.html#4.9

[47] http://www.mcrlab.uottawa.ca/research/DIVE.htm

[48] http://www.canarie.ca/

[49] http://www.canet2.net/

[50] http://www.ngi.gov/

[51] http://www.internet2.edu/

[52] Institute of Electrical and Electronics Engineers, International Standard, ANSI/IEEE Std 1278-1993, Standard for Information Technology, Protocols for Distributed Interactive Simulation, March 1993.

[53] Garvey, Richard E. and Monday, Paul, "SIMNET SIMulator NETwork," BBN Technical Note, Ft. Knox, KY, July 1988.

[54] http://hla.dmso.mil/hla/

[55] http://siso.sc.ist.ucf.edu/

[56] Waters R.C., Anderson D.B., & Schwenke D.L., "Design of the Interactive Sharing Transfer Protocol", Postproc. WET ICE '97 — IEEE Sixth Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, (MIT, June 1997), IEEE Computer Society Press, Los Alamitos CA, 1997 (also available at http://www.merl.com/reports/TR97-10/index.html).

[57] Macedonia, M.R., Brutzman, D.P., Zyda, M.J., et al., "NPSNET: A Multi-Player 3D Virtual Environment over the Internet", Proceedings, Symposium on Interactive 3D Graphics, New York: ACM, 1995, pp.93–94.

[58] Anderson D. B., Barrus J. W., Howard J. H., Rich C., Shen C., and R. C. Waters, "Building Multi-User Interactive Multimedia Environments at MERL", IEEE MultiMedia, 2(4):77-82, Winter 1995 (also available at http://www.merl.com/reports/TR95-17/index.html).

[59] Carlsson and Hagsand, "DIVE - A Multi User Virtual Reality System", IEEE VRAIS, Seattle, Washington, September 18-22 1993.

[60] C. Greenhalgh and S. Benford, "MASSIVE: A Collaborative Virtual Environment for Teleconferencing", ACM Transactions on Computer-Human Interaction, Sep. 1995.

[61] Funkhouser, Thomas A., "RING: A Client-Server System for Multi-User Virtual Environments.", ACM SIGGRAPH Special Issue on 1995 Symposium on Interactive 3D Graphics, (Monterey, CA), 1995, pp.85-92.

[62] Snowdon, D. and A. West "AVIARY: Design issues for future large scale virtual environments.", Presence. Vol. 3 No. 4, Fall 1994, MIT press, pp.288-308.

[63] http://www.csclub.uwaterloo.ca/u/sfwhite/vnet/