THE UNIVERSITY OF HULL

A Modelling and Networking Architecture for Distributed Virtual Environments with Multiple Servers

being a Thesis submitted for the Degree of

Doctor of Philosophy

in the University of Hull

by

Jaewoong Chang

B.E. Naval Academy, Korea

B.S. Yonsei University, Korea

M.S.C. Western Illinois University, USA

December 1999

TEXT BOUND INTO THE SPINE

ABSTRACT

Virtual Environments (VEs) attempt to give people the illusion of immersion that they are in a computer generated world. VEs allow people to actively participate in a synthetic environment. They range from a single-person running on a single computer, to multiple-people running on several computers connected through a network. When VEs are distributed on multiple computers across a network, we call this a Distributed Virtual Environment (DVE). Virtual Environments can benefit greatly from distributed strategies.

A networked VE system based on the Client-Server model is the most commonly used paradigm in constructing DVE systems. In a Client-Server model, data can be distributed on several server computers. The server computers provide services to their own clients via networks. In some client-server models, however, a powerful server is required, or it will become a bottleneck. To reduce the amount of data and traffic maintained by a single server, the servers themselves can be distributed, and the virtual environment can be divided over a network of servers.

The system described in this thesis, therefore, is based on the client-server model with multiple servers. This grouping is called a Distributed Virtual Environment System with Multiple-Servers (DVM). A DVM system shows a new paradigm of distributed virtual environments based on shared 3D synthetic environments. A variety of network elements are required to support large scale DVM systems. The network is currently the most constrained resource of the DVM system. Development of networking architectures is the key to solving the DVM challenge. Therefore, a networking architecture for implementing a DVM model is proposed. Finally, a DVM prototype system is described to demonstrate the validity of the modelling and network architecture of a DVM model.

ACKNOWLEDGEMENTS

Firstly I would like to thank my supervisor, Brian Tompsett, for all of his guidance, advice and support throughout the duration of my thesis work. In addition I would like to thank the Korean Navy for the assistance given to me throughout the course of my studies.

My thanks also goes to James Ward for his valuable proof reading and advice. In addition I would like to thank members of the Distributed Systems Engineering Group for the research discussions that take place.

Finally I would like to thank members of my family especially my parents, my brothers, my wife, my sons Kihyun and Woohyun.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

People have explained that the term Virtual Reality (VR) or Virtual Environment (VE) has many meanings. VR is used to refer to the whole subject area, its hardware, software, applications, etc., and a VE is the thing being partly or wholly simulated by the VR system. In general, VR and VE are used as equivalent words since there is little difference in their meaning. VE is explained as a computer-generated simulation that uses real time three-dimensional (3D) computer graphics. VEs range from a single-user running on a single computer, to multiple-users running on several computers connected by a network.

When VEs are distributed on several computers across a network, they are referred to Distributed Virtual Environments (DVEs). DVEs allow a large number of users to actively participate in a synthetic environment and interact with it in real time. DVEs give the users the illusion of immersion that they are in a computer generated world. The users can move, see, meet and directly manipulate objects within the DVEs. To interact with the DVEs, users use general input devices such as keyboard and mouse, or special devices, such as Head-Mounted Displays and Glove Input Devices.

VE or VR has been the subject of an enormous amount of publicity and academic research. Over the past few years, DVEs have been one of the important topics in VE research. However, very little work has been done on methods for designing and implementing efficient Distributed Virtual Environments [Stytz, 96]. This thesis, therefore, will mainly deal with new methods for designing and implementing Distributed Virtual Environments. This chapter describes the motivation, presents an overview of the problems, the aims of the research, and concludes with a preview of the chapter organisation.

1.1 Motivation

The author became interested in the field of DVE after serving as a naval officer in the Korean Navy. So far, DVEs have been used frequently for military purposes, and then primarily for individual and small-unit training. A large-scale battlefield simulation requires integrating information from various sources, including digital map data, simulation entities such as vehicles, aeroplanes, and humans, and events expected to take place during the interactions with that simulation [Poils, 95].

A large-scale virtual environment for battlefield simulation should also support multiple interactive users within the same environment. However, most virtual environments today usually do not support multiple concurrent users. Instead, they typically immerse one user in a synthetic environment. The users are often restricted to a limited role for interacting with the environment [Mastaglio, 95]. Therefore, the author perceived a need for new DVE technology to support the large-scale simulations.

1.2 Overview of Problems

In 1965 Ivan Sutherland, a pioneer in the field of computer graphics, stated that the user should be able to look in through a window into a virtual world and the world should be realistic. Afterwards, most research has been mainly concentrated on accomplishing this goal. Recently, the cost of hardware has fallen while the performance of hardware such as processors, memory, and graphics display systems have been improving. The capabilities of presenting a realistic virtual environment that users can interact with have also been improving day by day.

The more complex a VE becomes, the more powerful the computing system which will be required to execute it. Some VEs will require very complex processing. A single computer system still can not provide the VE with enough complexity, scale, and fidelity. So we can distribute VEs across computer networks, and gain several advantages in doing so.

By distributing the processing over several computers, the performance can be improved significantly. DVEs also allow multiple users to share the same environment [Pryce, 96]. DVE technology will offer enormous benefits to many different application areas, including education, scientific visualisation, architectural visualisation, entertainment, design, teleoperation and training [Gigante, 93].

However, the design of current DVE systems involves a number of problems which limits their scope and complexity. The problems of current methods will be discussed in Section 2.3. This thesis will pursue a new approach to DVE systems compared to current methods.

1.3 Aims

This thesis cannot deal with all of the areas of DVE systems. The main aim of this thesis is to deal with the modelling of a DVE system, to design the networking architecture for communicating between multiple computers across the network, and to make a prototype for implementation of the modelling and networking architectures. Therefore, this thesis will in particular:

- Design the architecture of a DVE model with multiple servers, where multiple users can simultaneously share virtual environments.

- Allow for real-time interaction between multiple users in the distributed virtual environment, even though they may be geographically dispersed.

- Develop networking architecture to minimise bandwidth consumption for communication between clients and servers on the DVE system. When the DVE system is used over wide area networks, it is made up from many independent, heterogeneous hosts connected by communications links that suffer from low bandwidth and high latency.

- Provide a detailed description of a prototype system based on the proposed solution.

- Evaluate this prototype which is to be implemented on some test platforms and draw conclusions by outlining a number of enhancements which are required to improve the performance of this prototype.

1.4 Thesis Preview

This Chapter has presented a brief introduction, motivation, overview of problems, and aims of the research. The following is a brief summary of the remaining chapters.

Chapter 2 provides an overview of current DVE technologies, the major technical issues for DVEs, and a technical overview of significant DVE systems, including basic techniques and the technical limitations of existing work.

Chapter 3 presents the whole concept for creating a model of a DVE system with Multiple servers (DVM). In this chapter, which is based on the investigation in Chapter 2, the requirements and limitations of this thesis are proposed, and the details of a DVM system, including concepts, organisations and mechanisms, are described. This chapter explains where the model differs from existing distributed virtual environment systems, and concludes with the advantages and disadvantages of a DVM system.

Chapter 4 mainly describes the networking architecture for communication in a DVM system. This chapter not only illustrates the structure of nodes across networks, but also defines the application-layer protocols for communicating between processes in the DVM system.

Chapter 5 describes a prototype of a DVM system that is used as a test-bed to show how the DVM system is implemented. The details of the DVM prototype include a case study, the demonstration, the fundamental techniques for implementation, and the results of the evaluation.

Chapter 6 is the conclusion of the thesis. It reviews the main points of the research, draws conclusions based on the experimental evaluation and includes suggestions for future research.

CHAPTER 2

OVERVIEW OF DISTRIBUTED VIRTUAL ENVIRONMENT (DVE) TECHNOLOGIES

VE technologies make it possible to insert users into virtual environments and allow the users to actively participate in the environments. In the VE systems, user should be able to change the viewpoint to navigate in the virtual world, manipulate the objects in the virtual scene, and become immersed in it. Distributed Virtual Environment (DVE) technology allows users to interact in real time with complex environments based on 3D graphical representations on multiple computers across networks.

To achieve these goals, the development of technologies for DVE systems has become one of the major areas in the VE research field. Powerful hardware and software is required to create realistic DVEs. So far, an enormous amount of specialised hardware and software technology has been introduced. However, the current state of technology still has many problems for generating a situation acceptably close to reality. Current DVEs are limited in their complexity and scope. So this chapter will describe the DVE technologies and technical issues of DVE design, review significant existing DVE systems, investigate the problems of such systems, and propose the aims and objectives of this research.

2.1 Typical VE Technologies

VE technologies supply computer-generated inputs to human sensory systems, monitor their response, and respond to the output. VE technologies use displays, sensors, and effectors to perceive and interact with a virtual world. To achieve these, a typical VE system requires various technologies. However, it has still many technical issues to be improved.

2.1.1 Components of a Typical VE system

Typical VE systems contain five components. As shown in Figure 2.1, the components include visual display systems, tracking systems, computation systems, haptic interfaces, and auditory systems.

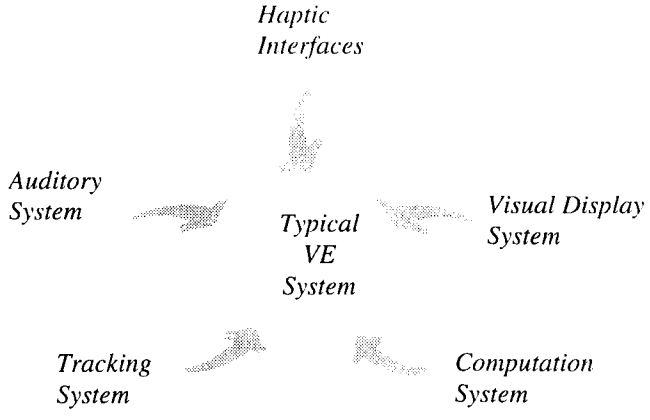


Figure 2.1 Components of Typical VE systems

2.1.1.1 Visual Display Systems

Much of the emphasis in VE research is placed on the visualisation. Visual display systems are the primary means for the visualisation of virtual environments. Human eyes normally have a horizontal field of view of 150° and a vertical field of view of 120°. Kalawsky states that the minimum requirements for a visual display system are that the VEs are greater than 110° for horizontal field of view, greater than 60° for vertical field of view, and greater than 30° of stereo overlap [Kalawsky, 91]. Visual display systems still do not provide this field of view (FOV).

There are several types of visual display systems such as monitors, head mounted displays (HMD), boom-mounted displays, and projection systems. We can choose any of these systems to render the 3D images. Monitors are the cheapest systems but provide for the lowest level of immersion. For immersive display HMDs are the most widely used displays in VE systems. HMDs display directly in front of the user's eyes.

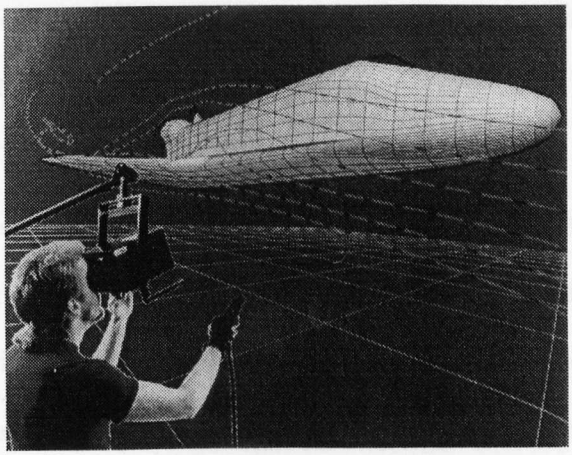


Figure 2.2 NASA Ames BOOM display system

The displays can be Cathode Ray Tubes (CRTs) or Liquid Crystal Displays (LCDs). The CRTs give better image quality and higher resolution but are usually considered too heavy for a HMD. The normal human can see details approximately a half-inch apart from 100 yards away [Sheridan, 93]. The LCDs which are mainly used in HMDs can display only about one-tenth of this resolution. The quality of the image is degraded by the limited resolution of the LCD screen.

2.1.1.2 Tracking Systems

Tracking systems are used to track the position and orientation of the parts of the user's body in space. When users interact with an immersive environment, they should feel natural and unencumbered. This often influences the choice of tracking system. There are many different systems, including electromagnetic, mechanical, acoustic, optical, and inertial systems. Each has some advantages and disadvantages, some of which are described below:

- The electromagnetic system uses a transmitter as a source that radiates electromagnetic fields and sensors that detect these fields. The sensors use the field strength to determine their position and orientation. This system is very sensitive to the presence of metallic objects but does not rely on line-of-sight observation [Bishop, 92].

- The mechanical systems use a rigid framework with several joints. Typically, one end is attached to the object to be tracked, and the other is fixed to a rigid base. It measures the angles between joints and calculates the position and orientation of the object. The accuracy is high, but it can not be freely moved, having a limited workspace.

- The acoustic systems use high frequency ultrasonic pulses to triangulate a source within the environment. A source produces pulses, and microphones receive the pulses. It measures

the time that each pulse arrives at the different microphones, and reports the source position and orientation. Acoustic systems have approximately the same range and latency problem as electromagnetic systems. These systems rely on line-of-sight between the source and the microphones. Accuracy is limited by many factors, such as air density and body parts, between the source and the microphones.

- The optical systems use two common methods. One is to put markers such as infrared LEDs (Light Emitting Diode) or infrared-reflecting dots on the body. Several cameras surround the body so that 3D co-ordinates can be found by using multiple 2D views. Another is to use a single camera to capture an image of tracking targets, analyse the image, and determine the 3D position [Sturman, 94]. These systems need processing time to analyse the image and to determine the position. Accuracy is affected by intervening objects.

- Inertial systems use gyroscopes to measure the three orientation angles. To produce a small, inexpensive inertial system, advances in micro-accelerometers and gyros are required. These devices are commercially available but are still experimental.

2.1.1.3 Computation Systems

Computation systems generate graphics for the display system, compute the state of the environment, control and interpret the input and output devices. The computational requirements are considerable and demand a high performance system. All of these tasks should be synchronised for each frame and require the co-ordination of multiple devices. Computation systems receive input data from tracking sensors and provide output data to visual display systems, force feedback systems, auditory systems, etc. They may use several hardware configurations, including a single workstation with one CPU, a single workstation with multiple

CPUs, a workstation with several graphics engines and multiple CPUs and a workstation connected via a high speed network to a supercomputer that performs the computations while the workstation performs the rendering.

2.1.1.4 Haptic Interfaces

Interaction through touch and manipulation in the virtual environments is known as haptic interaction. Haptic interfaces are devices that support manual interaction with the virtual environments and manipulation of virtual objects. Haptic interfaces allow human users to touch, feel, grasp, and manipulate objects which may be located remotely, exist only in simulated worlds, are too small or too large for normal human interaction, or are too dangerous to touch with human hands. Haptic interfaces not only measure the position of the parts of each user but also provide feedback forces to the user.

These devices employ human tactile, kinaesthetic and motor systems for interaction with the virtual environment. Haptic exploration is a sensory task for identifying properties. The tactile sensory system for getting information uses either tactile sensors or kinaesthetic sensors. Tactile sensors use mechanoreceptors in the finger pad, and the kinaesthetic sensors use receptors in the skin around the joints, joint capsules, tendons, and muscles. There are four categories of haptic devices: devices to measure hand position and orientation, force and torque feedback devices, tactile devices, and devices to produce other stimuli such as hot or cold [Bishop, 92].

2.1.1.5 Auditory Systems

In a virtual environment, some information can be communicated from the computer system to the user via sound. A realistic VE requires an auditory system to generate sounds associated

with objects in the VE. Auditory systems allow for sound output and voice input, and recognise speech for control of the virtual environment. They supply audio feedback from and to the user's virtual environment.

Auditory systems usually use one of two types of sound source. One is a synthesised sound source that reflects the many ways to generate sound from basic waveforms, filters, and envelopes. Synthesised sounds generate very flexible sound and have relatively a compact representation. Another type is sampled sound source that digitally records sounds that are triggered for playback by the interface. Sampled sounds are able to reproduce real world sounds very accurately. Sampled sounds, however, require relatively large amounts of memory. To display sounds auditory systems use either on-head devices (headphones) or off-head devices (loudspeakers).

2.1.1.6 Summary

Since the current technologies are not capable of generating a situation close to reality, more advanced technologies are required. Visual display systems are still not sufficient for immersion. There are latency and accuracy problems for tracking systems. The current computation systems do not provide the ability to render images realistic enough to be convincing to the system user. Haptic interfaces are required to improve a variety of parts related to touch perception and feedback including the development of devices to effect haptic stimulation and software to render the sensation of touch. Auditory systems still require more advanced audio equipment and audio feedback to generate real world sounds very accurately. Development of equipment capable of addressing all of the needs is still currently out of reach.

2.1.2 Technical Issues

There are several important technical issues to be considered when building up large, realistic, rapidly changing VEs. These issues include the provision of high fidelity images, real-time entity management, the real-time collision detection and response, the physically realistic entity modelling, and so on.

2.1.2.1 High Fidelity Images

VEs are aiming to achieve a high degree of visual realism. Users accept the image and act as though it were real. Users do not need the visual accuracy of a photograph, but users will require the most accurate model and best possible image while maintaining interactivity. To support realistic VEs, users must consider the required level of fidelity at reasonable cost in their design and assembly. To achieve the VEs with high fidelity images, users may sacrifice interactivity. There are trade-offs between high fidelity images and interactivity

2.1.2.2 Real-time Management

Complex VEs usually require a few thousand entities. Some VEs contain a large number and variety of active entities such as vehicles, humans, and planes, which are under active control. There are also passive entities, such as buildings, trees and hills. The state of these entities is often changed, so that VEs require real time updates to share the same environment between participants. However, it usually takes a long time to update the data associated with these entities, particularly in real time.

If the state of some entities is changed, VEs need the ability to update the change of entities' state at hosts during the operation. For example, in the battle field simulation, if a fighter attacks buildings or vehicles, the state of the buildings and vehicles must be changed, to reflect the damage. Every participant should recognise this situation in real time. However, a solution to this problem is still an open area of research [Stytz, 96].

2.1.2.3 Real-time Collision Detection and Response

To achieve a realistic VE, it is required to detect and respond to collisions between entities in real time. Without this feature, the realism of a VE will be poor. So this feature is potentially a matter of great importance in a VE. However, real-time collision detection and response is a fundamental problem in a VE. When some entities are managed in a VE, collision detection and response may take a long time to compute.

In a VE, an entity's motion is constrained by collisions with other entities and by other dynamic constraints. The greater the number of entities in the VE, the more complicated collision detection and response becomes. Real-time collision detection and response has important issues, including models with curved surfaces, very large numbers of entities, frame-to-frame spatial coherence, and discrimination between contact and interference [Serrano, 94].

2.1.2.4 Physically Realistic Entity Modelling

Users require realistic, correct and smooth updating in the virtual environment. To achieve accurate models with realistic images, the VE system should consider the state characteristics of the entity being modelled. Ideally collisions would give realistic bending, crumpling and moulding that are expensive to compute. In the case where two moving entities collide with

each other, it requires a powerful computing system, or collision detection and entity deformation may take a long time to compute.

2.2 DVE Technologies

DVE technology is not only based on typical VE technologies, but also require various networking technologies (see Figure 2.3). Current DVE technology is commonly limited to visual cues. To provide appropriate visual cues to a user in a distributed virtual environment, it is necessary that all of these typical VE and communication technologies are efficiently co-ordinated. A variety of communication technologies is required to collectively provide almost unlimited connectivity. The details of communication technologies for DVE will be described in Chapter 4. This section will briefly describe communication models of DVE systems and technical issues to be considered when implementing DVEs.

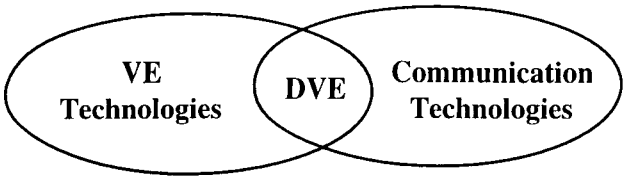


Figure 2.3 DVE Technologies

2.2.1 Communication Models

DVE systems usually use one of three models: a centralised model, a peer-to-peer model, or a broadcast and multicast model.

2.2.1.1 A Centralised Model

In a centralised model (see Figure 2.4), there is one server, attached to centralised resources. All other clients communicate only with the central server. Client-to-client communication is indirect, mediated by the central server. A server computer collects data from the clients, stores the changes of data in its own database, and then sends the results back to the clients. Each client renders the results and handles user input. This model has a simple structure to store and handle the data, but it is not scalable for the more common DVE situation today of a collection of nodes that wish to directly exchange data with other nodes.

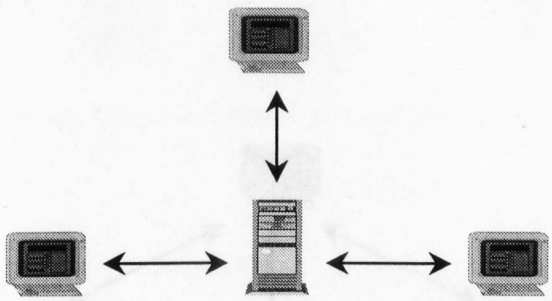


Figure 2.4 Centralised Model

2.2.1.2 A Peer-to-Peer Model

In a peer-to-peer model, each computer maintains the complete or part of the VE data, and performs the rendering and computation of entities. When a node makes changes to its own data, it sends the updated data out, enabling the other nodes to update their individual data.

Basically, in a peer-to-peer model, nodes find each other by broadcasting an initial connect request and waiting until they get a reply from other nodes that are currently running. It therefore requires listening for such connect requests and forming connections between nodes. The more nodes that join the group, the more rapidly the number of connections between nodes and the number of messages being sent is increased. Peer-to-peer communication requires a total connection of (N*(N-1)) / 2 in a network group (see Figure 2.5). Therefore, in the peer-to-peer model, the number of nodes is limited. In spite of this problem, a peer-to-peer model is a more scaleable approach to incorporate than a centralised model. Today, many real-time DVE systems are moving from a centralised to a peer-to-peer model because of flexibility and cost-effectiveness.

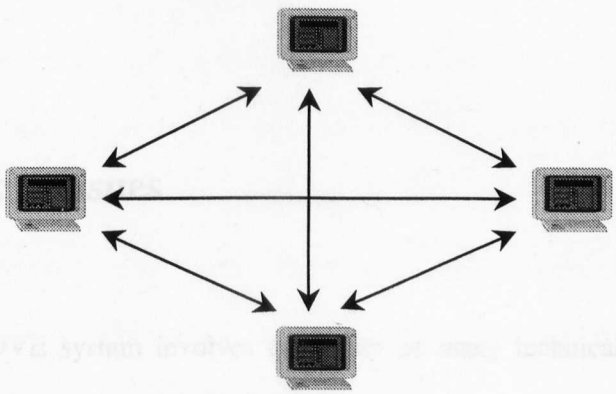


Figure 2.5 Peer-to-Peer Model

2.2.1.3 Broadcast and Multicast Model

To reduce the number of connections and the number of messages being sent, a Broadcast and Multicast Model (see Figure 2.6) is employed. Broadcasting is a protocol which allows a

computer to send one message out, and allows all of the other computers to read that single message. Multicast allows the message to be simultaneously sent to a subset of machines. In the multicast model, the groups of nodes can be established, and when a node multicasts a message, only nodes in the chosen group can receive the message.

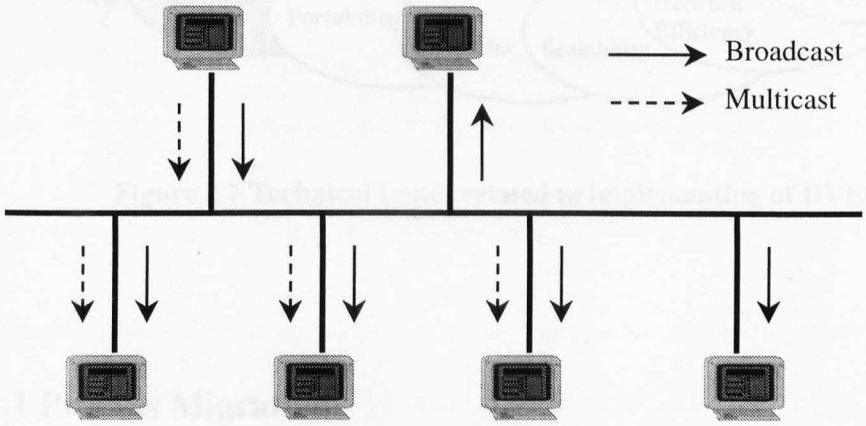


Figure 2.6 Broadcast and Multicast Model

2.2.2 Technical Issues

The design of a DVE system involves a number of many technical issues compared to a traditional VE. Figure 2.7 shows some of the fundamental issues involved when implementing a DVE [Blau 92]. Some of these are given higher priority than others. This is merely because they are relatively important for implementation of the DVEs. Most importantly, a DVE system is an interactive real time system. Therefore, in DVE systems, it is more important for transmitted data to arrive on time than it is for data to arrive uncorrupted. This section will describe the issues that are related to communication during the implementation of DVE system. These issues include process migration and network efficiency.

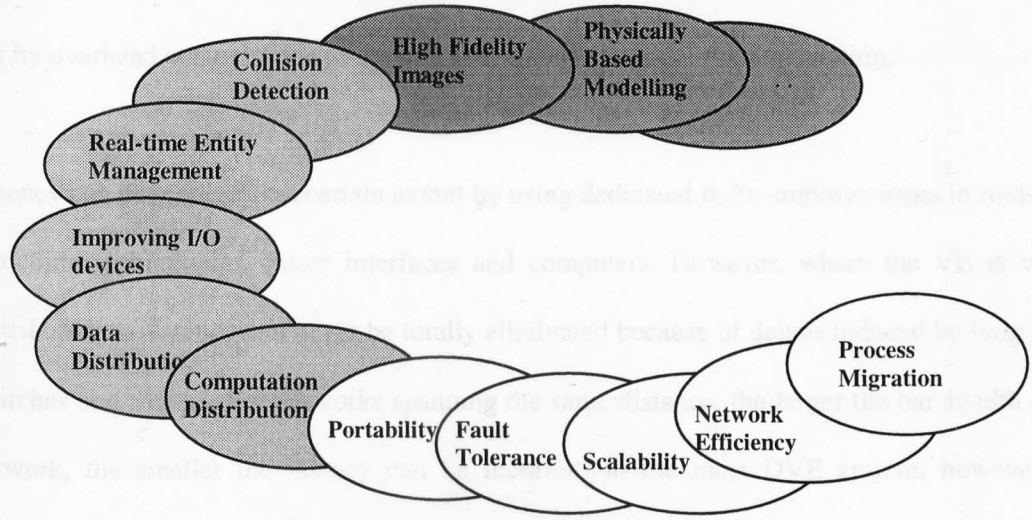


Figure 2.7 Technical Issues related to implementing of DVEs

2.2.2.1 Process Migration

DVE technology can use parallelism to improve performance and reliability. In some cases, if a computer on the network either fails, or becomes overloaded with too many processes, one can migrate processes to other processors which are not as heavily loaded. This involves migrating the entire state of the process, and either suspending its activity during the movement, or being able to have it continue to process messages during the transition. This is not just a simulation problem, it is related to many distributed applications.

2.2.2.2 Network Efficiency

The DVE systems often work on a wide area network. Such systems usually suffer from low bandwidth and high latency. Network efficiency usually means both bandwidth and latency.

The effective bandwidth is determined not only by the raw bandwidth of the network interface, but by overhead occurred due to the protocol and the protocol implementation.

Latency can be reduced to a certain extent by using dedicated links, improvements in router and switching technologies, faster interfaces and computers. However, where the VE is widely distributed, the latency can never be totally eliminated because of delays induced by long paths, switches and routers. For networks spanning the same distance, the larger the bandwidth of the network, the smaller the latency can be incurred. In the most DVE system, however, the bandwidth can not be increased unlimitedly, the bandwidth of the network is limited. The latency, therefore, usually depends on the protocol overhead.

To maintain consistency across the DVE it is necessary to reduce the latency within the limited bandwidth of the network. There are trade-offs between reducing latency and increasing reliability. DVE applications usually require low latency, because speed is important. However, most techniques that can be used to decrease latency will be less reliable. Many DVE applications, therefore, use less reliable protocols. For example, in the Internet protocol suite, the UDP data packet, which uses an unreliable protocol, is smaller and faster than TCP, a more reliable protocol. UDP is usually used when speed is more important than accuracy. TCP is usually used when accuracy is more important then speed. In general, increasing network efficiency is a difficult engineering problem.

2.2.3 Summary of DVE Technologies

This section and the previous one have presented a number of technologies and technical issues that must be considered to produce realistic VE and DVE systems. These technologies and

issues are interdependent, and a DVE system must consider all these issues in its design and assembly. The solution to these issues is important to increase the realism of DVE systems

As diversity and detail of distributed virtual environments increase without bound, network requirements have become the primary bottleneck. This requires changes in the hosts, protocols, bandwidth, software, and hardware architectures. In order to achieve a high-performance and realism, an advanced networking technology is required.

2.3 Technical overview of Significant DVE Systems

There are many possible VE models, and each model has its own particular set of laws and technologies. This section reviews the basic techniques and technical limitations of the current well known DVE systems.

2.3.1 Distributed Interactive Simulation (DIS)

DIS has been developed in the Department of Defence (DoD) community. This system has evolved from the earlier SIMNET, which was developed with DARPA funds in the late 1980s. The DIS protocol has become an IEEE standard (IEEE 1278.1) for logical communication among entities in a real-time distributed fashion with interactive users in the loop. DIS is only a protocol definition and does not specify the structure of implementation.

2.3.1.1 Basic Architecture

DIS is a virtual environment within which humans may interact through simulation at multiple sites. DIS exercises involve the interconnection of a number of simulators. The simulators may

be present in one location or be distributed geographically and the communications are conducted over the network.

DIS is based on a peer-to-peer structure. There is no central computer, event scheduler, clock, or conflict arbitration system. DIS was designed to link distributed, autonomous, heterogeneous hosts over a network into a real-time DVE [Katz, 94]. The DIS protocol provides a standard mechanism for communication between nodes, facilitating interaction between multiple entities in multiple virtual worlds. Heterogeneous nodes, which may have different architectures, can interact with each other using the DIS protocol.

Data may be sent using broadcast, multicast or point-to-point. Sending a multicast message is more efficient than having to send point-to-point messages to every host in the simulation. Hosts that are not in a particular multicast group will ignore the packet at a very low level[1], with minimal overhead.

The DIS system also uses the dead reckoning algorithm to reduce the amount of network traffic. A host sends a message that includes the entity's location, a timestamp, and a velocity vector. Using that information, the location of the entity in each host is calculated and updated without additional messages. The position and orientation of the entity are set using the entity state PDU (refer to Section 2.3.1.2) which arrived most recently. When an entity exceeds the dead reckoning threshold (refer to Section 4.4.5), it sends out another Entity State PDU.

This architecture provides very flexible trade-offs between computational loading, positional error and network bandwidth. If a simulation model requires a highly accurate position, the

[1] The network interface card typically handles this process, so that no consideration is necessary in software.

error threshold can be reduced, which will result in more network broadcasts, clarifying the new position.

The DIS protocol is self-healing. When a new entity enters the virtual world, it begins to broadcast entity state packets. If recipients have never heard from this entity before, they can add it to their remote entity database. If an entity is not heard from within five seconds, recipients will remove it from the remote entity database. Users can enter and leave at will without disturbing other participants, and dropped packets do not cause a failure of the system. Since there is no central server, there is no single point of failure.

2.3.1.2 Packet Data Unit

DIS defines a standard message format for interchanging information between simulation hosts. This standard format is called a PDU, for Protocol Data Unit. It consists of some 27 different network packets for communication, and is defined by the IEEE 1278 DIS standard [Steven, 94]. The DIS protocol uses an open format Message PDU that affords user-defined extensions. This provides flexibility for extension of the message-passing paradigm to a large-scale distributed system. The PDUs are encapsulated in UDP packets in Ethernet frames. The packets can be sent over any network medium, from telephone lines to ATM switches. More bandwidth allows more entities to be supported [Zyda, 95].

The principal PDU type is the Entity State PDU (ESP) (see Table 2.1). The ESP encapsulates the position and posture of a given entity at a given time, and is broadcast to all nodes. Additional PDU types include sensor or weapon interaction, signals, radio communications, collision detection and logistics support.

Field Size (Bytes)	Entity State PDU Fields	
12	PDU Header	Protocol version, Exercise ID PDU Type, Padding, Time Stamp Length in bytes
6	Entity ID	Site, Application, Entity
1	Force ID	
1	Name of Articulation parameters	N
8	Entity Type	Entity kind, Domain, Country, Category, Subcategory, Specific, Extra
8	Alternative Entity Type	Same type of information as above
12	Linear Velocity	X, Y, and Z (32 bit components)
24	Location	X, Y, and Z (64 bit components)
12	Orientation	Psi, Theta, Phi (32 bit components)
4	Appearance	
40	Dead Reckoning Parameters	Algorithm, other parameters Entity Linear Acceleration Entity Angular Velocity
12	Entity Marking	
4	Capabilities	32 Boolean Fields
N*16	Articulation Parameters	Change, ID Parameter Type, Value

Table 2.1 Entity State PDU

2.3.1.3 Limitations and Problems

NPSNET Research Group describes a number of practical problems with DIS in their paper [Steven, 94]. DIS does not provide an efficient method of reducing the bandwidth and computational resources in a large-scale simulation. Large-scale simulation usually requires enormous bandwidth and computational resources. For example, in a simulation with about 100,000 users, the required bandwidth will be about 375 Mbps to each computer [Loral, 92]. It requires a more powerful network medium than FDDI (Fibre Distributed Data Interface).

However, this requirement is unrealistic in the near future. What is required is to use efficient methods of bandwidth reduction rather than increased computational or network resources.

The underlying problem is the DIS paradigm. Entities that are physically far apart do not need to know what each one is doing until they are in closer proximity to one another. Entities, however, broadcast "keep-alive" PDUs to every entity at least every 15 seconds.

Models and world databases must be replicated at each simulator. Complete replication of the database is grossly inefficient and some means of partitioning information is required. At present the DIS has no mechanism for distributing data on demand, which is necessary for large-scale simulations. Besides reducing the required bandwidth, distribution of data increases protection against malfunctioning devices or software in the simulation.

Although the simulation usually contains a large number of static objects, DIS does not provide an efficient method for handling them. Static objects such as buildings and trees must repeatedly broadcast their state, even though they have not changed, so that new players can pick up their state if they missed earlier changes. If a node lost the last message, the entire simulation database must be replicated at each node since there is no method of partitioning the database. This wastes both bandwidth and computational time.

Different types of real-time data (e.g. simulation packets, video, and audio) are currently multiplexed and demultiplexed at the application layer rather than the network or transport layer. This means that video and audio must be treated in the same way, causing multiplexing and demultiplexing overheads in the application.

There is no software layer to mediate between the simulation and the network. A network using routers is limited only by the address space. DIS, however, must use bridges for large-scale simulation. The number of nodes is limited to tens of thousands.

DIS has problems of limited scale and inflexibility caused by its architecture, inherited from SIMNET [Kanarick, 91], which was originally constructed for small unit training. DIS is only used for small-scale environments suitable for distribution over a single LAN. For this reason simulations do not scale well and are not currently suitable for large scale VEs [Macedonia, 95].

The broadcast approach is another obstacle to scaling of DIS. As the number of entities grows larger, the slower-speed links become saturated. The processing of updates for a large number of entities begins to exceed the capacity of the slower hosts, especially since they must also perform tasks such as rendering, input device processing, etc. The broadcast can flood the network with unwanted traffic, and cause routing loops. It can also incur a performance penalty by forcing packets to be checked at the operating system, or application level.

Although the DIS provides multicast transport, it is insufficiently broad and adaptable to meet general virtual environment requirements. The most important limitation is that it is not universally implemented, and is available only for a restricted number of platforms.

DIS was designed for a very specific application and most of the PDUs are therefore inappropriate for general VR use. The Entity State PDU is too specific, and cannot easily be applied in a general way. It also includes a great deal of superfluous information. This makes the individual PDUs very big, which greatly increases the bandwidth requirements. A leaner, simpler format is needed.

2.3.2 Naval Postgraduate School Networked Vehicle Simulator IV (NPSNET-IV)

NPSNET started as a test-bed for a research project by faculty and students at the Computer Science Department of the Naval Postgraduate School in February of 1990. The focus of NPSNET is to design and implement a very large scale, real-time distributed virtual environment based on a ground combat world using inexpensive, commercial, off-the-shelf components [NPSNET, 95].

2.3.2.1 Basic Architecture

NPSNET is a workstation-based, 3D visual simulator. It utilises SIMNET databases with hierarchical data structures and DIS (Distributed Interactive Simulation) protocol for application level communication among independently developed simulators.

NPSNET is a real-time, 3D visual simulation system capable of displaying object movement over the ground or in the air. All nodes connected to the same network simulate the same VE. The architectures use spatial, temporal, and functional relationships to partition the dissemination of information in the DVE. NPSNET has a number of key functional components (see Figure 2.8).

NPSNET-IV uses the Ethernet network (10Mbps) and TCP/IP multicast packets, and uses unreliable (UDP) messaging. NPSNET-IV is capable of playing across the MBONE of the wide area network, bandwidth is generally limited to T1 (1.5Mbps).

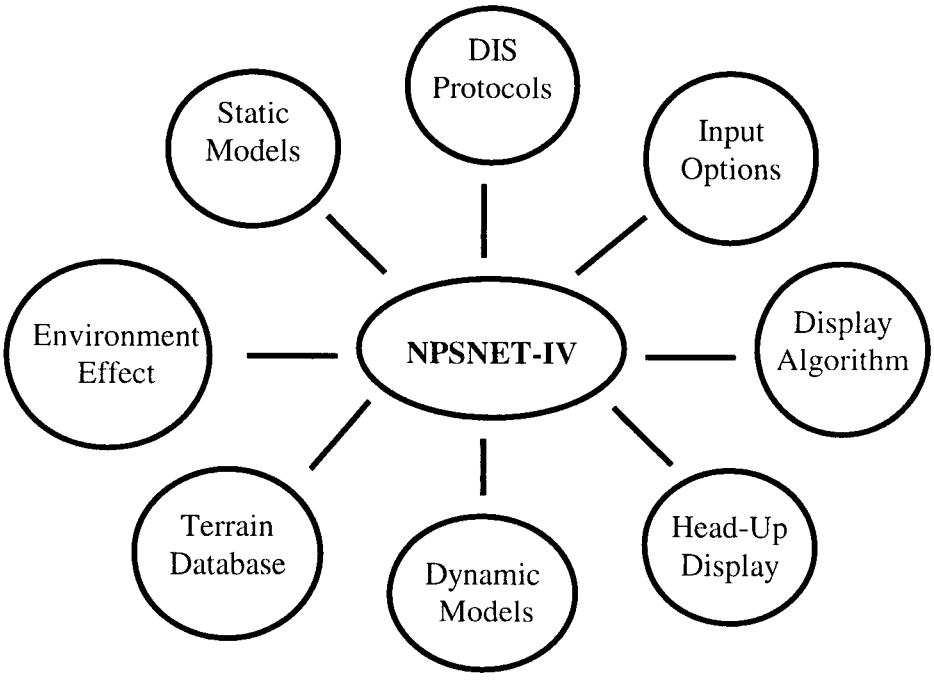


Figure 2.8 NPSNET-IV key functional components

The NPSNET Research Group has devoted itself to improving several functional issues of DIS. The issues being implemented are as follows:

- Large scale, wide area, multi-platform virtual environments (over 1,000 players) using DIS communication protocols.

- Support for IP multicasting across the multicast backbone (MBONE) of the wide area network.

- Heterogeneous parallelism to minimise system latency. The parallelism includes multi-processor / multi-machine interprocess synchronisation and communication using reflective memory.

- Real-time expert systems to provide "intelligent" autonomous agents in virtual world simulations.

- Real-time, 3D computer graphics simulators utilising texture mapping, efficient polygon culling, multiple LOD (Level of Detail) models, lighting models, etc.

- Real-time atmospheric affects, including wind, smoke, clouds, haze, rain and snow.

- Integration of video and audio.

- Physically based motion and physical reactions.

- Integration of dynamic terrain.

2.3.2.2 Limitations and Problems

NPSNET-IV was designed for supporting more than 1,000 users. However, at present NPSNET-IV can only support about 250-300 players using currently available technology. Since NPSNET-IV implements the DIS communication protocols, it also has the same limitations and problems as the DIS.

NPSNET-IV provides mainly mechanisms for communication between nodes, and solutions for collision detection, terrain rendering, and so on. Like DIS, the NPSNET-IV does not support technologies such as tracking, haptic interfaces, and sound feedback.

NPSNET Research Group describes also several practical limitations to the NPSNET-IV approach in its paper [Steven, 94].

- The implementation is based only on the SGI workstations.

- Extra data copying can cause more contention for memory, bus bandwidth, and the CPU.

- DIS handles the traffic at the application level. It will likely become more complicated as the protocol matures, demanding more processing resources.

2.3.3 Distributed Interactive Virtual Environment (DIVE)

DIVE is a system designed and built by the Swedish Institute of Computer Science (SICS). DIVE is a toolkit for building a DVE application in a heterogeneous networked environment based on UNIX and Internet networking protocols within local and wide area networks [Hagsand, 96].

2.3.3.1 Basic Architecture

DIVE is based on shared 3D synthetic environments. It supports multi-user applications. In a DIVE world, a user is called an actor and is represented by a body-icon. A user can see a virtual world through a rendering application that renders a scene from the actor's viewpoint. Users can navigate in a shared synthetic 3D environment. Several networked users can see, meet and interact with other users over an Internet. The DIVE system is a truly distributed system that uses a concurrent runtime model, based on a peer-to-peer approach with no centralised server. Peers communicate by reliable multicast, based on IP. DIVE uses the concept of a shared

memory to describe its model. A set of processes interacts by making concurrent accesses to the shared memory and by sending messages to each other.

A DIVE process is either a human or an application process. A process group in DIVE is a set of processes, which can be addressed as one entity. The process group consists of all processes that are members of a virtual world. A process can join, leave or travel across groups at any time.

The database of a DIVE is partitioned into worlds. Each database of DIVE contains 3D graphical objects (polygons, spheres, boxes, etc.) that make up a virtual world. The world represents a specific set of objects and parameters. Each world has an associated process group. Each member of the process group has a copy of the world database and there is no main server. When a process joins a group, it receives a copy of the world data from one of the other processes. Each world is a separate DIVE process. A DIVE process can access one of the databases, which they can update concurrently.

DIVE uses three mechanisms to ensure the consistency of the replicated database. The first is mutual exclusive locks to prevent concurrent modifications of the local database. The second is sequential ordering. Messages sent from a single process always reach all recipients in the order they were sent. The last message is a distributed lock on an object. By using a distributed lock, a process can manage an object exclusively in the local database, and then distribute the changes.

2.3.3.2 Basic Modules

The basic modules of DIVE include the following : Threads, Sid, dive core, dive aux, Graphics, Audio and Video libraries. The Threads library contains a slimmed and extended threads

interface with a multiplexing I/O and timer module. The Sid library is a basic communication library based on IP multicast and Scalable Reliable Multicast (SRM). The dive core library contains basic DIVE functionality. The dive aux contains a variety of DIVE modules adding extra functionality useful for an application builder, including the implementation of the Dive/Tcl interface. The Graphics library is a variety of graphic libraries for rendering 3D modules. The Audio library is a real-time audio library for 3D conference and object-based audio. Finally, the Video library is a real-time video library for texture-based video communication.

2.3.3.3 DIVE Implementation

The DIVE platform is basically implemented in C. The core DIVE source consists of C files, tcl files and yacc/lex files. The source taken from these files is compiled into libraries. An application file needs to be compiled and linked with the DIVE libraries and external libraries so that it can be used to create an executable application. A dynamic entity's behaviour and user interface are implemented in Tcl/Tk.

2.3.3.4 Limitations and Problems

DIVE manipulates the object's visual properties, spatial translation / orientation changes, generates a sound or triggers a behaviour in another object. However, current DIVE (version 3.2) can support only simple, fixed behaviours in distributed environments. For example, an object can be made to move or rotate when clicked on. More complex actor motion, such as general human motion, generates more continuous network traffic.

Over the Internet, messages might have to be delivered among many participants over high latency paths. However, as the number of peers increases in the DIVE system, the number of messages needed to communicate between these peers increases. The amount of traffic between peers will be rapidly increased. This increases the load on sending and receiving peers, as well as on the network. The latencies will be also increased.

Peers of an advanced distributed application need to exchange large amounts of information, including object and world definitions, navigation commands, audio, and bitmaps. This is essentially limited by network bandwidth.

DIVE uses a reliable multicast protocol. While the reliance on a reliable protocol eases the distribution of the world databases, it counteracts scalability. The scalability of a DIVE is limited by the available network bandwidth. As the number of users increases so do the bandwidth requirements. Higher levels of reliability require more bandwidth, and restrict scalability.

DIVE has focused on multi-user aspects and 3D interaction, it uses a multi-user environment where every user can see what other users see. So an object cannot have a state associated with one user alone. For example, a user has to send the information of an object to other users, although the others do not need to receive the information, a network may become flooded with the unwanted messages' traffic and incur a performance penalty, interrupting operations in order to perform this task.

2.3.4 The Minimal Reality (MR) Toolkit

The MR Toolkit was developed by University of Alberta in Canada. The MR Toolkit is a low-level toolkit supporting the development of VR applications. It is a set of software tools for the production of virtual reality systems and other forms of 3D user interfaces.

2.3.4.1 Basic Architecture

The goal of the MR Toolkit is to provide a software foundation on which a programmer can easily build single-user virtual environments. It focuses on providing software that supports real-time interaction with a virtual environment using a head-mounted display.

MR is based on a bottom-up development philosophy [Green, 99]. MR (Version 1.5) supports the following features:

- Device drivers are provided for a wide range of standard VR devices.

- MR provides a standard facility for recording each site's hardware configuration and room geometry. Applications are independent of devices and their location within the room.

- MR provides efficient and convenient mechanisms for dividing an application into multiple processes and handling communication between these processes.

- MR provides a set of tools for monitoring the real-time performance of VR applications.

- MR provides a set of standard 2D and 3D interaction techniques.

- The OML (Object Modelling Language) language can be used to define the geometry and behaviour of objects. An interactive 3D modeller can be used to produce object geometry and behaviour.

The MR toolkit is designed for the production of VR applications that run at more than one site using different hardware configurations. MR provides a portable, device-independent platform for the development of VR applications. MR applications can easily be ported to other sites with little or no change to the source code. The MR Toolkit can easily integrate the driver of any new device into its system, so that it is versatile and can support many VR devices.

The MR toolkit was designed to be easy to extend. Frameworks are provided for adding new device drivers, interaction techniques and tools. The MR Toolkit (version 1.5) provides support for common VR devices, such as 3D trackers, Gloves and Head-mounted displays. The MR Toolkit also includes support for SGI Videosplitter, SGI Multi-channel, and sound synthesis equipment.

2.3.4.2 Software Structure

The MR Toolkit consists of three software levels (see Figure 2.9). The lowest level consists of the routines that interact with the hardware devices. The second level of the MR Toolkit consists of a collection of packages. Each package handles one aspect of a VR user interface. This level also changes the messages it has received from the device drivers into a more usable format, and provides data sharing services between workstations. The third level is an application level. A MR application consists of one or more processes, with one designated as the master process and others as slave or computation process.

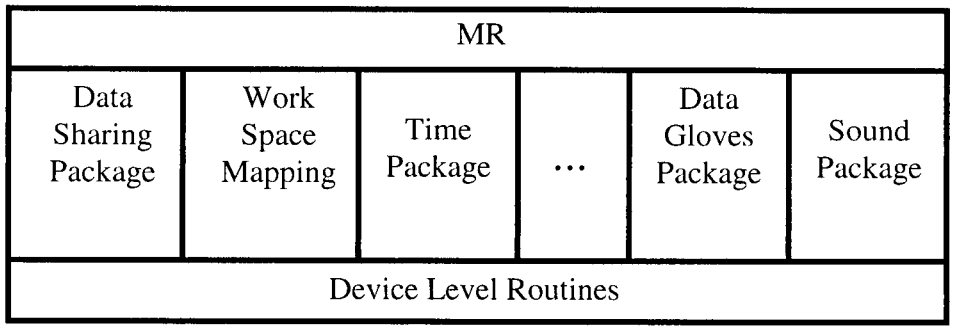


Figure 2.9 Three-level software structure of the MR Toolkit

2.3.4.3 Relationship between the MR Tool Components

MR tools for developing virtual environments include Object Modelling Language (OML), 3D modeller (JDCAD+), and Environment Manager (EM) for running multi-user networked applications. Figure 2.10 shows the architecture of the MR tool package and the relationship between its components [Wang, 95]. EM is responsible for constructing the VE, and provides facilities for monitoring the execution of the VE. OML is used to describe the geometry and behaviour of 3D objects.

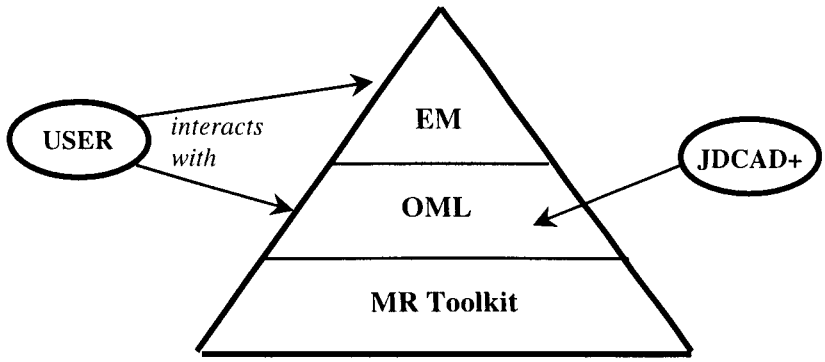


Figure 2.10 MR Toolkit Architecture and its Component Relationships

2.3.4.4 Implementation of MR Applications

MR is based on a peer-to-peer approach. One MR application runs on one machine at a time. The MR application can start up the peer package at any time, and may initiate and quit communications with other processes. Application-specific information may be shared between machines using UDP to send messages to specific addresses [Shaw, 93].

The MR peer mechanism supports communications between multiple applications running at different sites. The peer package allows a master process to communicate with other master processes on other machines. This allows multiple users to share the same 3D environment [Green, 96].

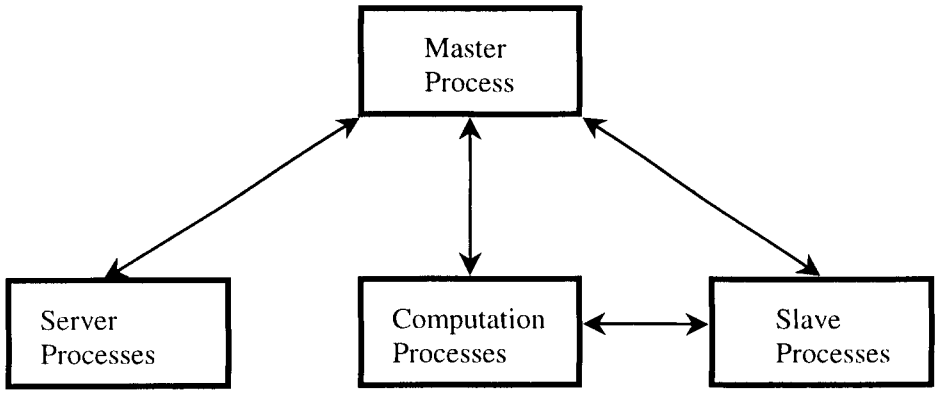


Figure 2.11 Process Structure of MR application

Figure 2.11 shows the relations of processes in a MR application. Each application has a master process that initiates the execution of other programs. The master process maintains all the state variables and geometric model of the virtual environment. When an application starts, the master process first initialises all existing resources and summons up the slave and computation processes. Each master program keeps a list of other peers that it is connected to.

Slave processes are usually in charge of the input and output of all the devices. The slaves receive data from their peers via their master, slaves do not communicate directly with other slaves or computation processes. At the beginning of the loop, the computation process reads the state variables from the master process. Then, it computes the new scene according to the input and sends the result to the master process at the end of each loop.

The MR Toolkit implements a hybrid runtime model which combines the simulation loop model and event-driven model together. The device input and display refreshing commands are obtained through an event-driven mechanism, while the update of the scene data is done by the simulation loop in the computation process.

2.3.4.5 Limitations and Problems

All peers are connected directly to one another which requires a lot of network traffic to maintain and, as a result, the use of more than five networked machines is not recommended [Green, 96].

MR Toolkit's process structure can cause a bottleneck problem. Everything must go to the master process first. If the master process breaks down, the whole system is likely to break down.

The MR Toolkit peers package does not currently use multicasting, and relies on unicast for communications among the applications. This scheme is bandwidth inefficient for large groups.

2.3.5 Networked PC Games

There are three basic levels of networked PC game that include *Play by mail* (Imperial Nomic...), *Turn based games* (Go, Chess, Bridge, etc..), and Real time (Quake, Half-Life, Unreal, etc...) [Flipcode, 99]. Play by mail does not require a server. Turn based and play by mail games allow relatively long delays between turns, and are not subject to real time constraints typical of VE applications. Real time games are the main concern of the game industry today. Therefore, this section reviews the basic techniques and the technical limitations of real time games.

2.3.5.1 Basic Architecture

The complexity or size of the game should be considered. The greater the level of interactivity with the game, the bigger it becomes. As the game becomes more complex, more data must be transmitted. The more data that has to be transmitted, the more bandwidth that is needed. These problems tend to restrict the number of players that can be supported, or result in lag. Based on these considerations, a client/server or peer-to-peer model is chosen. There are two types that are generally chosen in current systems.

The first is a peer-to-peer model. In the distributed model, every client is connected to every other. Currently, this model is not recommended for a large scale multi-player game because the number of connections is O (n^2) which almost guarantees that users will lose some crucial data and fall out of synchronisation. However, a distributed model can have some advantages. Although one computer logs off, the game does not end. Theoretically the game data could be maintained forever, as long as there always were two or more players in the system.

The second is a centralised model. In the centralised model, all clients are connected to a central server. There are O(n) connections at any time, meaning much less work on the part of the client computers. The advantage of this model is that the server can keep everyone in synchronisation, which is very important for real time games. However, there are some disadvantages. If the server fails, the game is over. If the server lags, everyone lags. Furthermore, the server has to be O (n) times more powerful than any client computer in terms of bandwidth in order to be able to handle the transmitted data.

2.3.5.2 Limitations and Problems

Network games use the TCP/IP protocol suite including TCP and UDP. TCP is a simple and effective way of transmitting data. It is very good at ensuring that client and server can communicate reliably. However, it carries with it a lot of overhead and extra network lag, so that games usually use connectionless unreliable transport (UDP) for performance reasons. UDP unfortunately brings some serious problems. For example, some data may not reach the other computer, or data might arrive in the wrong order. The problems of UDP communications will not be covered in any more detail here, as they are well documented [RFC, 87].

Synchronisation is extremely important because it makes sure that what people see on their screen is the same as everybody else. To solve this problem, the server keeps a copy of everything going on in the game and updates players as necessary. However, as shown in Figure 2.12, this method results in a latency problem. At time t1 the source client copy starts to move forward. The message is sent through the network. A little later at time t2 it arrives at the server. At this point the server copy starts moving forward. It is already behind the actual position because it was late receiving the message. Now the server broadcasts to all clients. At time t3 every target client receives the message. All clients are now behind the server. Everyone see the object at a slightly 'old' position.

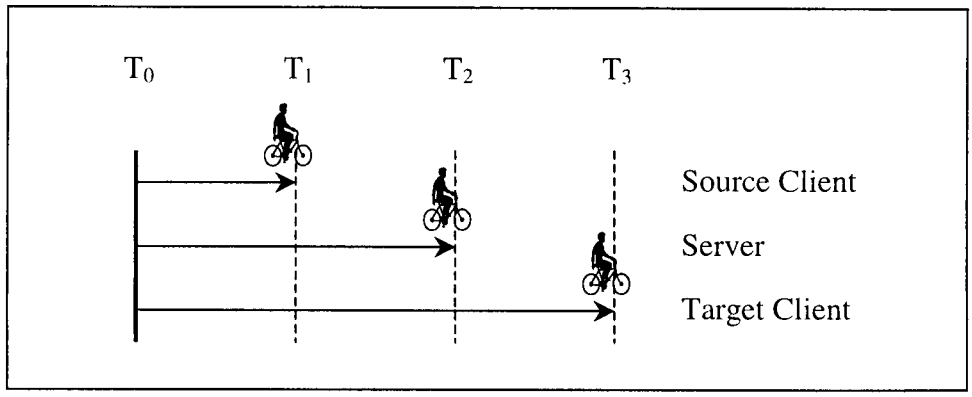


Figure 2.12 Synchronisation between clients and server

2.3.6 Summary of the Technical Overview

In the above section, some of the major DVE systems have been described. A summary of the major DVE systems is shown in Table 2.2. All these systems have their own advantages and disadvantages. From the analysis presented above, a new architecture for implementing DVE systems can be derived to effectively utilise the technologies and solve the technical issues that have been described in Sections 2.1 and 2.2. This new architecture will be described in the next Chapter.

2.4 Summary

In the future there will be practically no limit to the diversity and detail which DVEs can provide. DVE construction can include concepts and components from nearly any subject area. The scope of distributed virtual environment development is so broad that it can be seen as an inclusive superset of all other global information infrastructure applications.

However, the development of DVE applications has been so far limited because of several factors. Because graphics workstations require specialised real-time performance, they are currently very expensive. The development of graphics databases and software has been progressing slowly. Large-scale VEs may require the use of a WAN (Wide Area Network) to enlarge geographic scope and number of participants. However, present network technology has usually implemented distributed VEs on Ethernet LAN (Local Area Network). These problems are being overcome little by little through the rapid growth of high speed inter-networks, the recent availability of low-cost graphics workstations, and the development of graphics software including tools and libraries [Durlach, 95].

Especially, in a DVE system, the major advance in performance will largely depend on advances in modelling, computer architecture, operating systems and networks. The modelling issue is really at the core of every DVE system. The feasibility of realistic rendering in real time largely depends on the improvement of modelling. Network requirements are likely to become the primary bottleneck. However, little work has been done on the efficient modelling of large DVE system and networking methods between DVE processes over a network. So this research will mainly concentrate on the modelling of DVE systems with multiple servers and the design of a networking architecture that inter-operates with the applications in the DVE.

Feature	DIS	NPSNET	DIVE	MR Toolkit	Networked PC games
Specification	Protocol Definition	Implement DIS Protocol	Toolkit	Toolkit	Application
Running Platform	Unspecified	Workstation	Workstation	PC & Workstation	PC
Operating System	Unspecified	UNIX	UNIX	DOS/ UNIX	Windows
Communication Protocols	UDP	UDP	TCP	UDP	TCP or UDP
Communication Scheme	Point-to-Point, Broadcast, or Multicast	Multicast	Multicast	Point-to-Point or Broadcast	Point-to-Point, or Broadcast
Bandwidth	Unspecified	10 Mbps+	10 Mbps+	10 Mbps+	14.4 Kbps+
Interaction Model	Peer-to-peer	Peer-to-Peer	Peer-to-Peer	Peer-to-Peer	Peer-to-Peer Client/ Server
Servers	NO	NO	NO	NO	Single
Distributed Execution	YES	YES	YES	NO	YES
Multi-Users Scene Sharing	YES	YES	YES	NO	YES
Data Replication	Total	Total	Total	Partial	Total
Computation Replication	Partial	Partial	Total	Total/ Partial	Total
Interactive behaviours	YES	YES	YES	YES	YES
Scalability	Small	Small	Small	Small	Small
Visual Display System	Unspecified	W/S Monitor	W/S Monitor & HMD	HMD	PC Monitor
Tracking System	NO	NO	NO	YES	NO
Haptic Interfaces	NO	NO	YES	YES	NO
Auditory System	NO	YES	NO	YES	YES

Table 2.2 Summary of Current Major DVE systems

CHAPTER 3

CREATING DISTRIBUTED VIRTUAL ENVIRONMENT SYSTEMS WITH MULTIPLE SERVERS (DVM)

Traditional simulations differ from VEs in several respects. They operate with a fixed time quantum of minutes or even hours between successive states in the simulation, while VEs operate in continuous time. Especially, a virtual environment has to allow users the same type of interaction that they experience in the real world. Realistic rendering of the VE requires 3D computer graphics. The virtual environment is a set of data elements that describe a geographical region, entities such as vehicles and humans, and that are events expected to take place during the interactions with that environment.

Constructing 3D graphical virtual worlds requires integrating information from various standard data sources, including digital map data, aerial imagery, detailed line drawing, terrain data, etc. (Figure 3.1). A complex virtual environment requires a high degree of collaboration and co-operative engineering in hardware, software, human factors, visual systems, and usability.

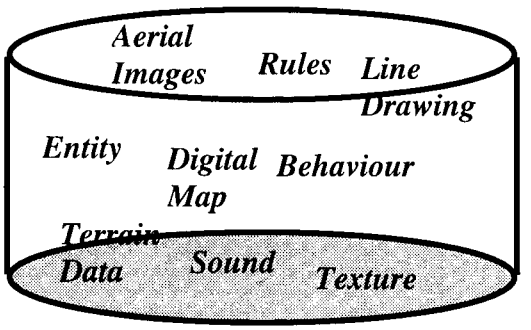


Figure 3.1 The diversity of information in the virtual world

To achieve this effectively, it will be necessary to distribute the VE between a number of computers. Virtual Environments can benefit greatly from distribution strategies. Such systems are called Distributed Virtual Environments (DVEs). Not only can a VE be spread between many computers but information from across the network can also be combined within a single environment as shown in Figure 3.2. This chapter will describe a new model of DVE systems with Multiple-Servers (DVM) which run on multiple computers across a network.

3.1 Concepts of a DVM system

In Section 2.3, some of the other DVE approaches have been described, and compared. These approaches have some advantages and disadvantages. From an analysis of these approaches, the concepts for an effective new DVE system can be derived. This section will describe the DVM concepts based on this analysis.

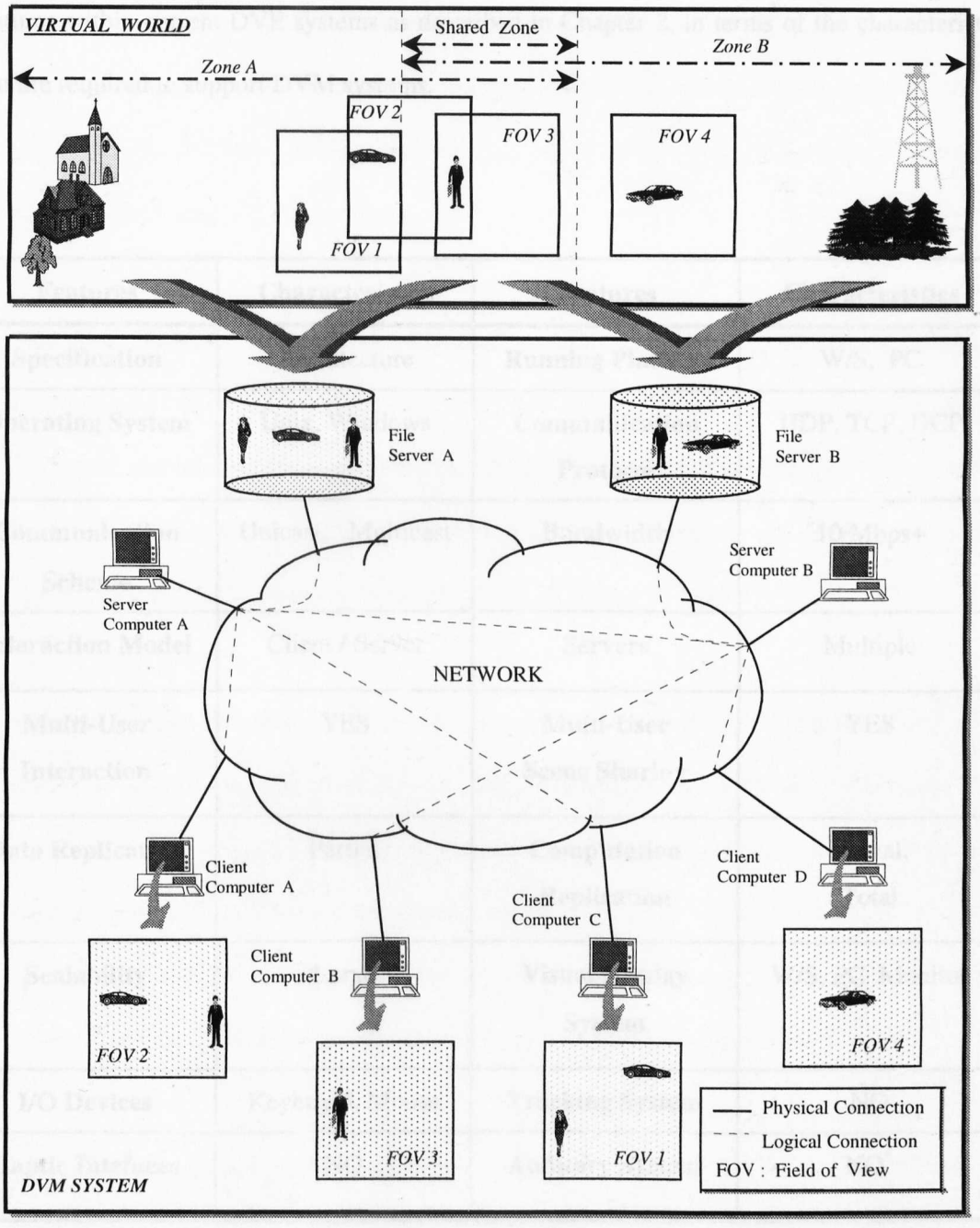


Figure 3.2 Basic Context Diagram of DVM system

3.1.1 Basic Characteristics of a DVM system

This section will describe the characteristics that a DVM system should include, and explain where the new model differs from current DVE systems. Table 3.1 identifies the common

features within current DVE systems as described in Chapter 2, in terms of the characteristics that are required to support DVM systems.

Features	Characteristics	Features	Characteristics
Specification	Architecture	**Running Platform**	W/S, PC
Operating System	Unix, Windows	**Communication Protocol**	UDP, TCP, DCP
Communication Scheme	Unicast, Multicast	**Bandwidth**	10 Mbps+
Interaction Model	Client / Server	**Servers**	Multiple
Multi-User Interaction	YES	**Multi-User Scene Sharing**	YES
Data Replication	Partial	**Computation Replication**	Partial, Total
Scalability	Large	**Visual Display System**	W/S, PC Monitor
I/O Devices	Keyboard, Mouse	**Tracking System**	NO
Haptic Intefaces	NO	**Auditory System**	NO*

* refer to section 6.3.

Table 3.1 Summary of Characteristics of a DVM system

- Specification : A DVM is a working system. It provides an architecture that defines a model of DVE systems with Multiple-Servers (DVM) which run on multiple computers across a network. The DVM system specifies how the implementation is structured, and also provides a protocol definition with associated guidelines.

- Running Platform : A DVM should be portable to a number of different computing platforms due to the cost of replacing existing systems. It should be also possible to use commercially available computers, including workstations and PCs, without having to purchase dedicated VE workstations.

- Operating System : A DVM running over a wide area network should be used in a heterogeneous environment. That is, it should be able to inter-work to provide a seamless DVM on various operating systems. However, most workstations and PCs are currently based on the Unix and Windows operating system. Therefore, the DVM system will support to run on workstations under UNIX and PC under Windows.

- Communication Protocol : TCP/IP currently provides better reliability than any other protocol suite for interworking in the heterogeneous systems. Therefore, the communication of a DVM system is based on TCP/IP, including a reliable connection-oriented protocol (TCP) and a connectionless and unreliable protocol (UDP). In the DVM system, to enable a client and a server to communicate with each other, an application-layer protocol will be used. Section 4.3.3 examines protocol issues in more detail.

- Communication Scheme : To implement a DVM system, two communication schemes are used in the DVM. One is one-to-one communication (Unicast) between two nodes, another is a one-to-many communication (multicast) that communicates with several other nodes simultaneously. More detailed explanation can be found in Section 4.4.1.

- Bandwidth : The DVM can require enormous bandwidth to implement complex environments in real time. The bandwidth available in the DVM will vary depending on the communication protocol used across networking mediums and the amount of traffic.

However, a DVM basically requires over 10 Mbps bandwidth for example, that could be based on an Ethernet (10 – 100Mbps) LAN. Please refer to Section 4.2.1.

- Interaction Model : The DVM system is based on a client-server model with multiple servers. In a DVM model, to reduce the amount of data and traffic maintained by a single server, the servers are to be distributed across a network, and the DVM is divided over a network of servers. The detail will be described in section 3.1.2.

- Multi-User Interaction : It should be possible for multiple users to interact within the same DVM. It should be also possible to support real-time interaction. Especially, it should be tuned for multi-user applications, where several networked users interact in real time over the Internet, even though they may be geographically dispersed. Please refer to Section 4.3.5.

- Multi-user Scene Sharing : Every user in the DVM should recognise what other users are doing. The state of the entities should be exchanged to share the same environment between users in real time. Please refer to Section 3.3.6.

- Data Replication : Data should be distributed across more than one server node. To replicate the data, the DVM will use a partial data replication methods. Please refer to Section 3.2.3.2.

- Computation Replication : Rendering of the complex DVM can be a large computational burden. To reduce the computational load, a DVM uses two computation methods, including partial and total replication. Please refer to Section 3.2.4.

- Scalability : The DVM system should have unlimited servers and clients connected in a heterogeneous network. It should be possible to support both small and large scale DVMs with many computers. A scalable DVM system is one that can easily cope with the addition of users and nodes, and whose growth involves minimal expense, performance degradation, and administrative complexity. Please refer to Section 3.1.2.

- Visual Display System : A DVM can use several types of visual display systems such as monitors, head mounted displays (HMD), boom-mounted displays, and projection systems. However, currently a DVM typically uses monitors as the primary means for display. Please refer to Section 5.1.6.4.

3.1.2 Structure of a DVM system

Current VE systems usually use one of four VE computing architectures: PC based VE, workstation based VE, parallel VE and distributed VE. The first three architectures run a VE on a single computer. The last distributes a VE among several computers over networks [Burdea, 94].

A networked VE system is currently the most popular Distributed VE system. A networked VE system is a typical application of distributed graphics. It consists of a set of workstations / PCs and servers interconnected with a network. Networked VE systems are a popular and powerful computing paradigm for DVE systems. Networked VE systems allow the sharing of information and resources over a wide geographic and organisational spread. However, it is far more difficult to design distributed graphics rendering programs than to design programs running on a single processor. The reason is that several problems derived from a distributed

environment must be considered: task partition and allocation, communication, synchronisation, protection of shared resources, consistency controlling and load balancing.

The networked VE systems are mainly based on the Client-Server model. In the Client-Server model, the data is stored on a server computer. The server computer in a DVE system communicates with its client or other server computers via networks. In some client-server models, however, a powerful server is required, or it will become a bottleneck. To reduce the amount of data and traffic maintained by a single server, the servers themselves can be distributed, and the virtual environment can be divided over a network of servers, each of which is responsible for a local region. Each server co-operates with the VE in a different region.

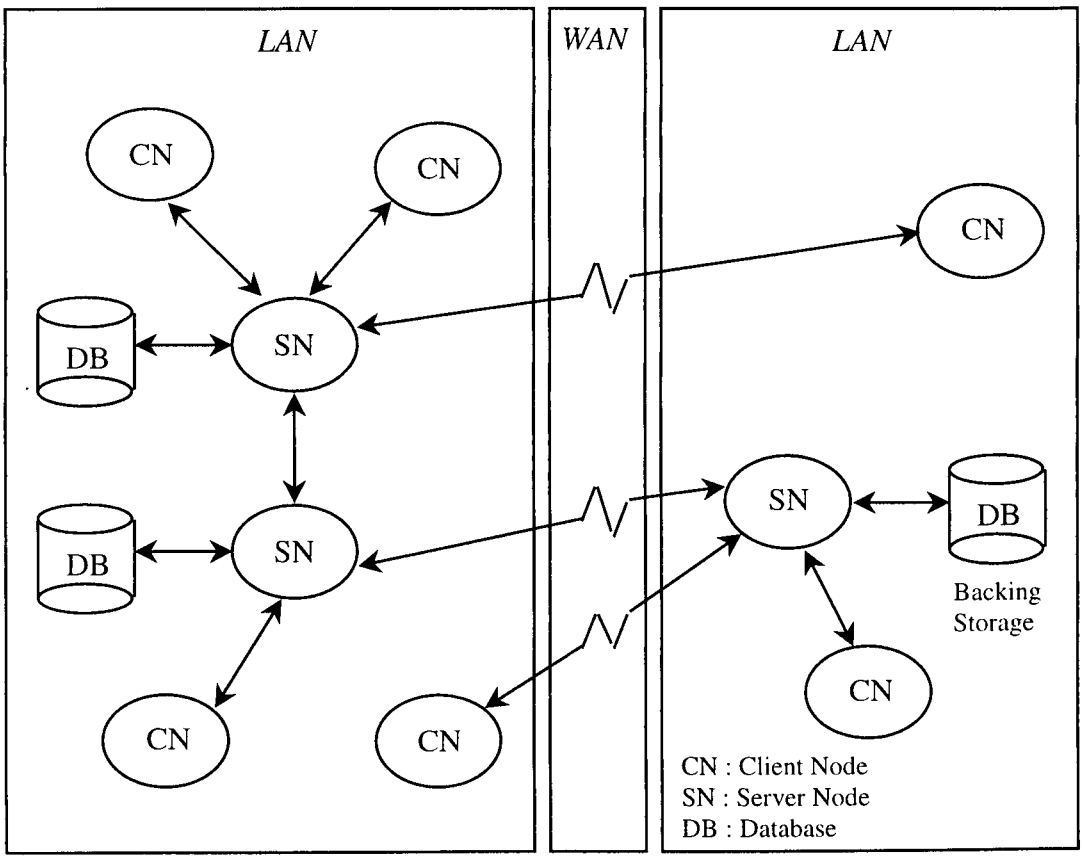


Figure 3.3 The logical routing structure of DVM

The system described in this thesis, therefore, is based on the client-server model with multiple servers as shown in Figure 3.3. This grouping is called a Distributed Virtual Environment System with Multiple-Servers (DVM). A DVM System is a new experimental platform for the development of distributed virtual environments, user interfaces and applications based on shared 3D synthetic environments.

3.1.3 Limitations of a Real-time DVM system

To design DVE systems, the most important issue is "How can we best exploit the potential of a large number of networked computers to meet real-time requirements of multiple interacting users and applications?" [Bryson, 94]. To design real-time DVM systems, it is necessary to consider several limiting factors that have to do with the very nature of hardware and software architectures.

3.1.3.1 Memory

Most computers have a finite amount of physical or virtual memory. In the future, memory prices may fall. However, memory is often the expensive component of any system these days and therefore physical memory may be seen as a precious resource [Hawke, 96]. A current (1999) Pentium PC usually has on average 16-64 Mbytes of memory. Silicon Graphics workstations usually have 64 Mbytes as standard.

3.1.3.2 Processor

On faster networks or in large simulation, when the amount of computation that is involved in a networked VE exceeds what is currently available from a single CPU, or when the CPU fails to

keep up with the traffic, this will become a bottleneck. To reduce the amount of computation, we can consider the use of parallel processing and spread the computational load among several processors within one or several computers. However, these systems are currently more expensive and thus less common.

3.1.3.3 Bandwidth and Protocol

As the number of users increases the higher the bandwidth required. The network bandwidth has an effect on the size and richness of a DVE. The actual bandwidth available will vary depending on the protocol used across the mediums and the amount of traffic. The effective bandwidth is determined not only by the raw bandwidth of the network interface, but also by the overhead of the protocol.

The performance of the network will have serious effects on the performance of the DVE system. When the DVE is running over a WAN, the round trip time of the package will probably be too long to meet real-time requirements. The improvement of a DVE model will largely depend on advances in network technology (refer to Section 4.2.1) such as in bandwidth and protocols.

3.1.3.4 Operating Systems

Current Operating Systems usually provide only minimal support for the transmission of real-time data. At the operating system level, most DVE applications are built on commercial versions of UNIX which are not designed for real-time performance. The transparency and the simplification of DVE operations will depend on the improvement of the operating system software.

3.1.4 Target Applications of a DVM system

Many target applications may be possible in the future, it may depend on the environment in which they are performed. If the above characteristics are adequately supported in the DVM system, the feasible target applications that are being explored may be as follows:

- Military training: Small or large numbers of live networked participants in real-world exercises are feasible. It has lower operating costs and is safer to use than real training.

- Distributed design: The DVM application can support interaction among networked remote participants, for the purpose of industrial design. The advantages of distributed design stations are numerous, but an essential point is economy and time-to-market.

- Education: Education can benefit with regard to long distance learning. No more travel time to study, no more expensive building environments. For this to be possible, very high bandwidth networks will be required.

- Entertainment: Networked entertainment allows for the simultaneous support of multiple players within the same immersive environments, joined over long distance. Networked multi-player VE entertainment is likely to become popular in the next few years.

3.2 Proposed Architecture of a DVM system

The problems of distributing a VE over a number of nodes are many and are compounded by increasing the distance between nodes. These problems are slightly different depending on the combination of hardware used and the geographical distance

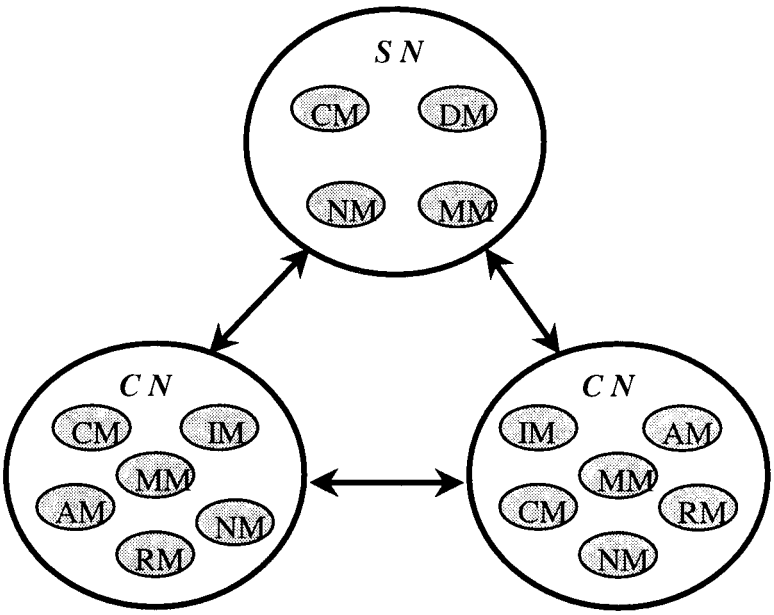
covered. There is no one-level solution that can be applied at all levels of DVE that will address all of the problems posed. A DVE needs suitable multi-level solution. Therefore this section describes the architecture that refers to the operational components and their interconnections among the techniques for a DVM system.

3.2.1 The components and connections

In a distributed VE system, the processors have different sizes and functions. The processors include small microcomputers, workstations, minicomputers, large general-purpose computer systems, and so on. The processor is referred to by a number of different names, such as *sites*, *nodes*, and *computers*. It is mainly referred to by the term *node* in the DVM system, in order to emphasise the distribution of these systems. The DVM architecture is divided into server and client nodes. Each node includes a number of processes. Distributing processes to multiple nodes increases the aggregate computing power associated with a simulation. The DVM system can use this not only to provide the capability to distribute views but also to handle a variety of input devices.

A DVM system assumes the logical connections of its components that can be viewed as shown in Figure 3.4. A DVM system contains seven components: a master manager module, rendering manager module, interaction manager module, computation manager module, network I/O manager module, network I/O manager module, and audio manager module.

A DVM system also assumes that the same kinds of processes are running on each server node or client node that has the same function. This is a logical view and does not consider the physical path for communications between the components. This section describes the components shown and the connections between them.



SN	Server Node	MM	Master Manager	CM	Computation Manager
CN	Client Node	RM	Rendering Manager	NM	Network I/O Manager
AM	Audio Manager	IM	Interaction Manager	DM	Database I/O Manager

Figure 3.4 Components of DVM system

3.2.1.1 Server Node (SN)

The major responsibility of a server node is to provide a bridge between its client nodes and databases. The server node is also responsible for maintaining the data consistency between its client nodes. It is usually the central workstation co-ordinating most of the VE activities. The server node loads data from databases, checks the requests from its clients or other servers, transmits the requested data to its clients or other servers, monitors its clients' activity, performs the maintenance of updated data, and manages network traffic. It has the responsibility of maintaining the state of all of the virtual entities in the local area of the DVM.

The server node runs four important process modules: Master Manager, Database Input/Output Manager, Computation Manager, and Network Input/Output Manager. Services provided by the Server nodes include

- Co-ordination of the client node when joining and departing from the DVM.

- Co-ordination of global VE time.

- Loading the initial data from the database and transmitting it to the client node upon request.

- Maintaining the data updated from its client nodes to keep the data consistent.

3.2.1.2 Client Node (CN)

The users use the client node to interact with the DVM system. Each client node manages the local VE, interacts with I/O devices and perform graphics rendering. The client node establishes communications with the server node and through it or directly, transmits the data to the server or the other client nodes. Most client nodes must operate alone in other operations. The management routines required by a client node can be divided up into six processes: Master Manager, Interaction Manager, Computation Manager, Rendering Manager, Network Input/Output Manager, and Audio Manager.

3.2.1.3 Master Manager (MM)

When a server or a client application starts, the master process first sets all resources and summons up the Network, Computation, Database, Rendering, Interaction and Audio

Managers. The MM maintains all the state variables of entities and the geometric model of the virtual environment. It is also responsible for routing messages appropriately.

3.2.1.4 Network Input/Output Manager (NM)

The NM is responsible for dealing with the network. It is a process that makes it possible for users using the same VE to communicate with each other. The NM keeps a list of other nodes that it is connected to. When an operation takes place, the NM is in charge of the data exchange between the nodes. The client NM requests initial data from the server NM. The server NM checks the requests from the client NM. It transmits initial data to the client NM. The client NM receives the initial data from the server NM. It retransmits updated data to the server NM again. The server NM receives the updated data from the client NM. If the data in the shared area is updated, then the server retransmits to the other server NMs. The other server NMs retransmit the updated data to its own client NMs.

3.2.1.5 Database Input/Output Manager (DM)

A DM is a process that runs on the server nodes. The DM handles a collection of data forming the virtual environment. Its primary function is to load the initial data, e.g. images, entities, sounds, etc., from the databases. It provides the initial data to its own clients. It also restores the updated data, which is sent from the client node to the database.

3.2.1.6 Computation Manager (CM)

The CM is a process that runs on the server and client nodes. It is in charge of the computation of all events. When an operation takes place in the client node, a CM reads the initial state

variables from NM or the changes of state variables from RM via MM. Then the CM computes the new scene according to the input, and sends the result to RM (defined in Section 3.2.1.7) for display. In the server node, CM computes the new state according to the input, and sends the result to DM via MM for saving.

3.2.1.7 Rendering Manager (RM)

The RM is a process that runs on the client node. It provides services for rendering and managing the visual representation, including shaded, texture-mapped surfaces, and coloured lighting. Each CM is actually an indefinite simulation loop. At the beginning of the loop, the RM reads the input from CM via MM. Then, it displays the new scene according to the input. Entities can be controlled directly by the user on the screen. RM also keeps track of the client's AOI (Area of Interest) and transfers updates on entities to MM as necessary.

3.2.1.8 Interaction Manager (IM)

An IM is a process that provides services related to interaction devices. IM is usually in charge of the input and output of all the devices (such as 3D trackers, data gloves, and visual display, etc.).

3.2.1.9 Audio Manager (AM)

An AM is a process that provides services related to the audio display. AM derives the sound source for auditory display and deals with the processing of the sound for placement, including stereo sound, enhanced stereo processing, or true 3D spatial placement. AM displays the final sound to the client user, usually via speakers or earphones.

3.2.1.10 Connections between components

Communication between components of the DVM system is divided into two methods: within a same node and between two nodes, according to the placement of nodes in which processes reside. Figure 3.5 shows the connections between these components.

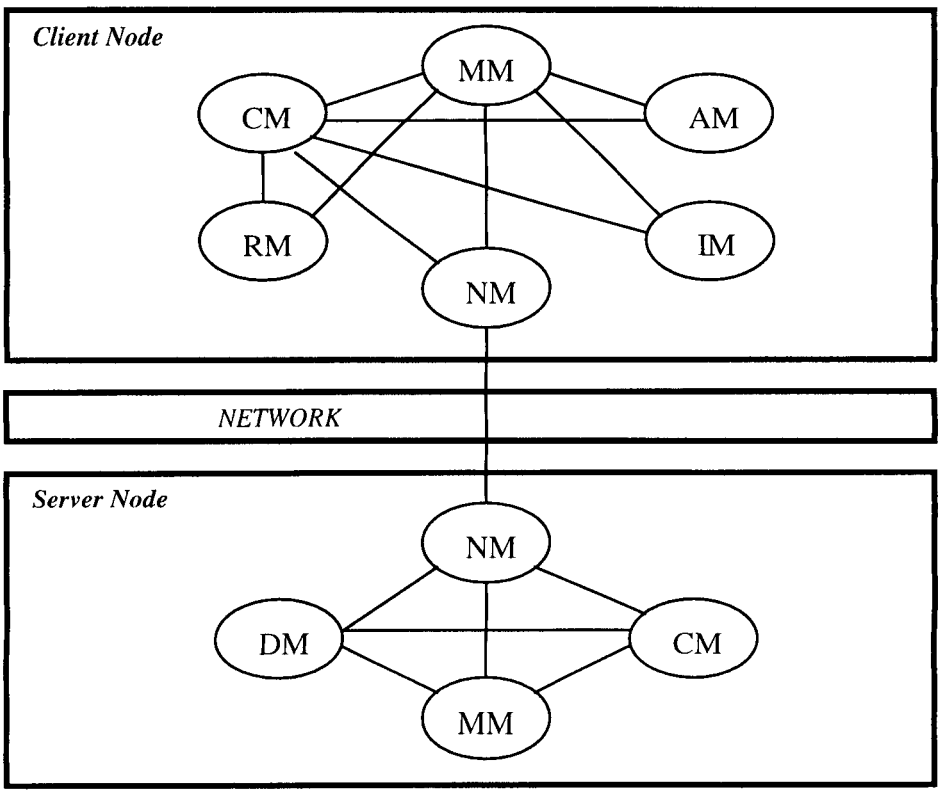


Figure 3.5 Connections between Components

3.2.2 Co-ordinate Systems

We cannot directly use a real-world co-ordinate system in the DVE (refer to Section 3.2.2.3). We need a conversion from the real-world co-ordinate to the DVE co-ordinate. This section presents the co-ordinate systems of a DVM system.

3.2.2.1 Real-world Co-ordinate System

The most familiar location reference system of a real world is the spherical co-ordinate system measured in latitude and longitude. Longitude and latitude are angles measured from the Earth's centre to a point on the Earth's surface. Longitude is measured East and West, while latitude is measured North and South. Latitude and longitude are measured in degrees, minutes, and seconds (DMS). Latitude values range from 0 at the equator to +90° at the North Pole and -90° at the South Pole. Longitude ranges from 0° at the Prime Meridian to 180° when moving east and from 0° to -180° moving west from the Prime Meridian.

3.2.2.2 Map Projection

A map projection is a mathematical conversion from spherical to planar co-ordinates. Most maps display co-ordinate data by conforming to a recognised global co-ordinate system. Map projections are used to represent the earth's three-dimensional features on a flat surface (Figure 3.6).

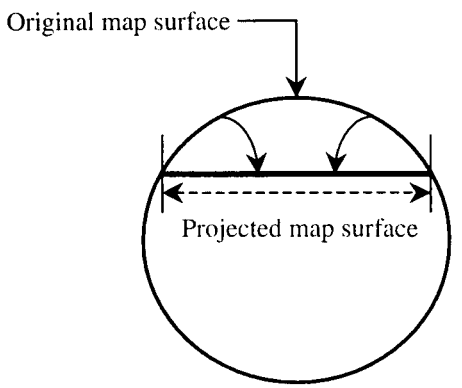


Figure 3.6 Map Projection

Any representation of the Earth's surface in two dimensions always distorts shape, area, distance, or direction. Map projections ensure a known relationship between locations on a map and their true locations on the Earth. The co-ordinate values in the database are measured in a real-world co-ordinate system. A mathematical conversion must be used to create a flat map sheet from the three dimensional surface.

3.2.2.3 Map Distortion

Latitude-Longitude is a geographical reference system. The length of one degree of longitude varies depending upon the latitude at which it is measured (Figure 3.7). For example, one degree of longitude at the equator is 111 km in length, but the length of one degree of longitude converges to zero at the poles. This reference system measures angles from the centre of the Earth, rather than distances on the Earth's surface. Degrees cannot be used as an accurate measure of distance or area because they are not associated with a standard length.

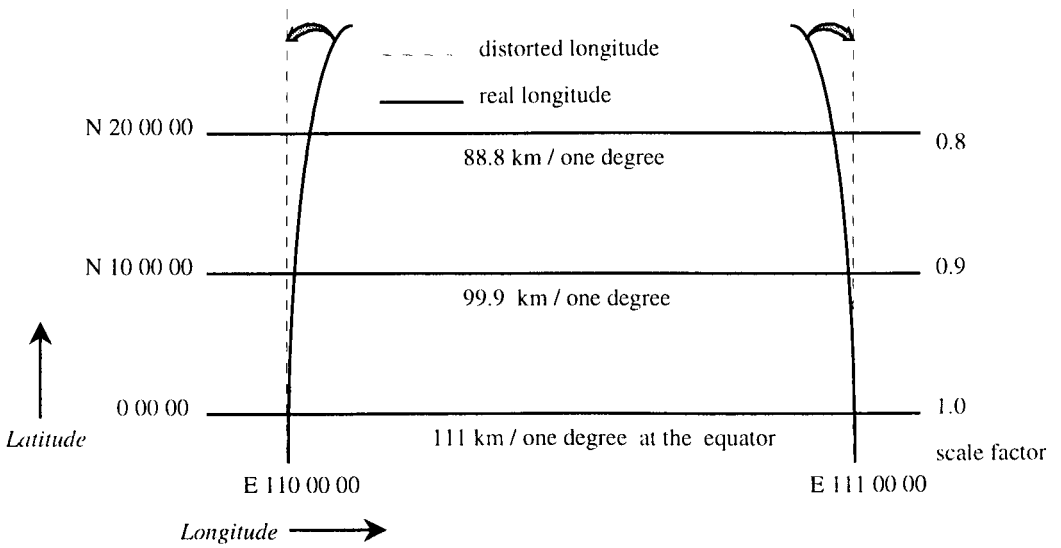


Figure 3.7 Map distortion

3.2.2.4 Planar Co-ordinate Systems

Planar co-ordinate systems have several properties that make them useful for representing real-world co-ordinates on maps. There are two dimensions: *X* measures distance in a horizontal direction, and *Y* measures distance in a vertical direction. Measures of length, angle, and area are constant across the two dimensions.

Various mathematical formulas exist to project the Earth's spherical surface onto a flat, two-dimensional surface. Geographical information systems, like flat maps, use various planar co-ordinate systems to map the Earth's surface. Each co-ordinate system used is based on a particular map projection.

3.2.2.5 Transformation of Co-ordinate System

Co-ordinate system transformation for display and sensor analysis is one of the major issues that affect 3D graphical models and terrain modelling. The real terrain model and 3D graphical models used in the DVM must be transformed into the DVM co-ordinate system for display. Co-ordinate system transformation for display affects 3D graphical models and terrain modelling. Co-ordinate data must be transformed from the real-world co-ordinate system into the DVE co-ordinate system. After loading and before proceeding with simulation, real-world co-ordinates must be converted from degrees, minutes, and seconds (DMS) into decimal degrees (DD). Decimal degrees can be computed using the following equation:

Decimal Degrees (DD) = Degrees + Minutes/60 + seconds/3600 (DMS)

When an integral co-ordinate value of DMS type is converted to floating value of DD type, the value is in the representable range. The range of values will be usually represented in a 32-bit word. The precision of value uses a rounding scheme. The decimal value of the co-ordinate is rounded up from the fifth place of decimal expansion of floating point (This value provides a guide only; this value depends on the resolution of terrain model). Table 3.2 shows an example of the real world conversion

DMS		DD	
X	Y	X	Y
85 00 00	40 03 45	85.0000	40.0625
85 59 15	40 03 45	85.9875	40.0625
85 58 30	40 03 00	85.9750	40.0500

Table 3.2 Real World Conversion

The size of VE passed to the client has a major impact on the performance of the DVM system. The entire VE can not be passed to a client because of the limitations and capabilities of the hardware. In order to minimise the size of VE passed to the client for rendering, the VE should be segmented. Figure 3.8 shows an example of DVM co-ordinate units used to segment the VE. A global VE represents a world with a length of one degree of longitude and latitude. To segment the VE, a global VE is first subdivided into sixteen sub-VEs. Each one of the sub-VEs is subdivided into twenty five sub-VEs. The sub-VE is subdivided until some minimum sub-VE is reached.

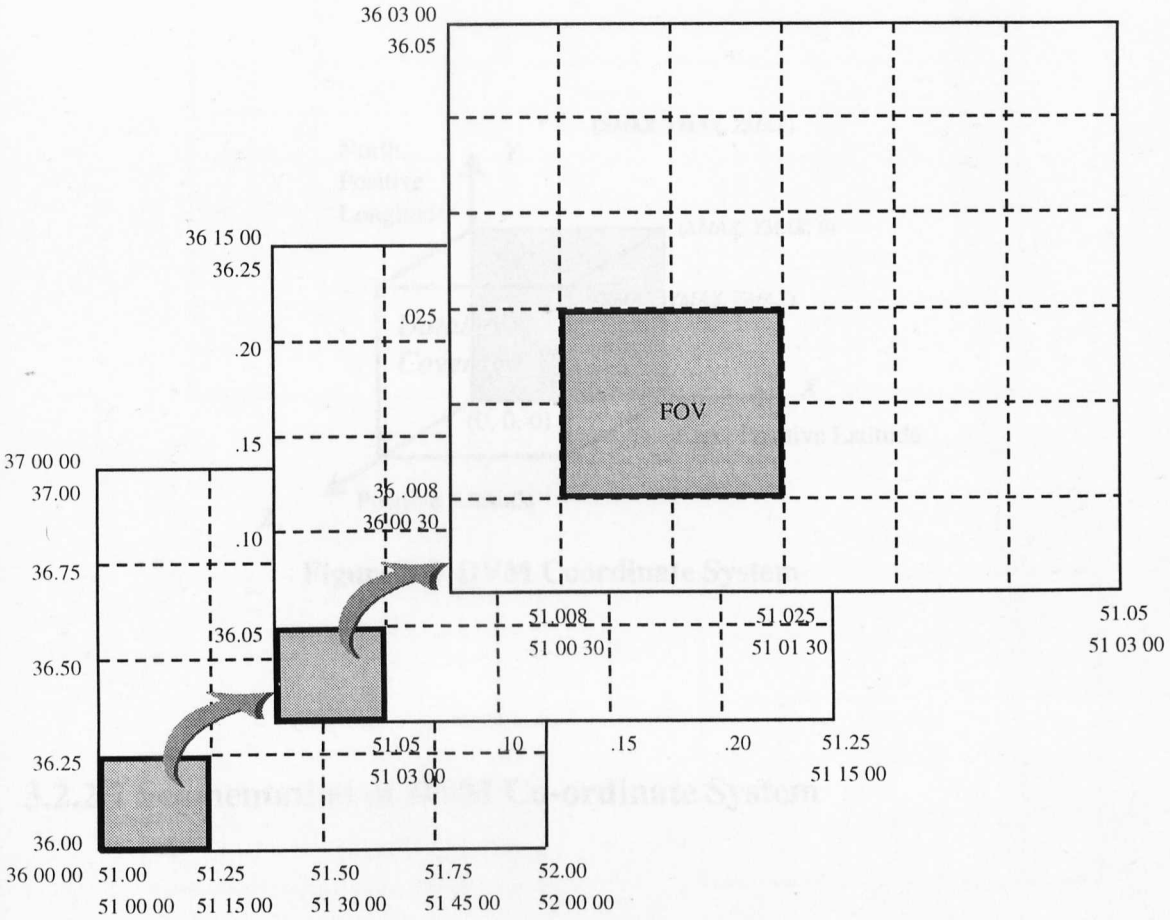


Figure 3.8 DVM Co-ordinate Units

3.2.2.6 DVM Co-ordinate System

A DVM system uses three dimensions that are useful for representing virtual-world co-ordinates on the screen. As shown in Figure 3.9, the co-ordinate system has (0,0,0) located at the left under corner of the virtual world. The X-axis represents the latitude of the world; the positive is heading east. The Y-axis represents the longitude of the world; the positive is heading north. The Z-axis represents the altitude of the world; the positive is the altitude above sea level.

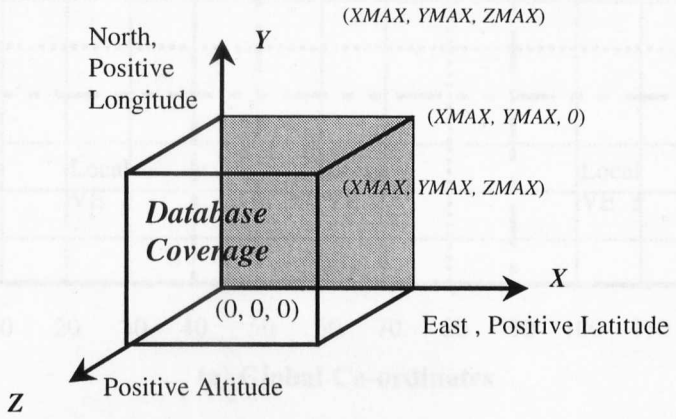


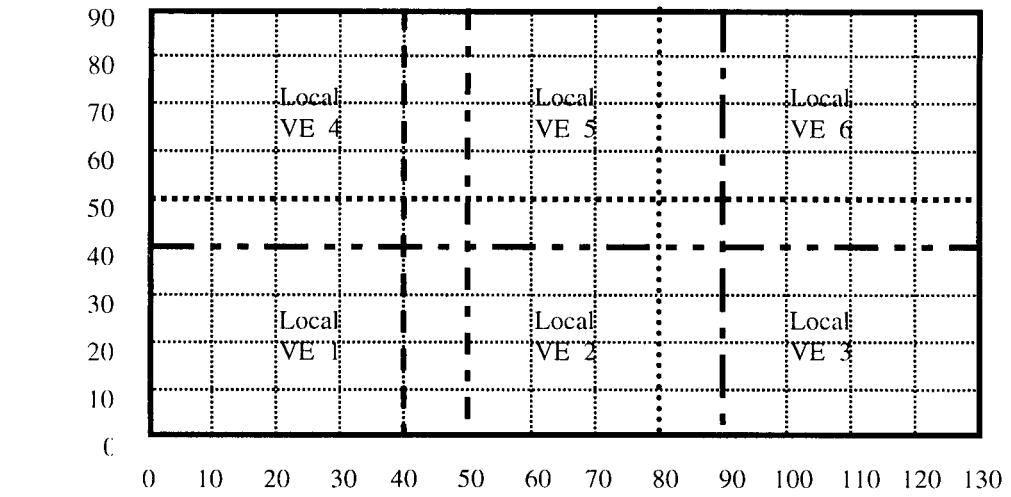
Figure 3.9 DVM Coordinate System

3.2.2.7 Segmentation of DVM Co-ordinate System

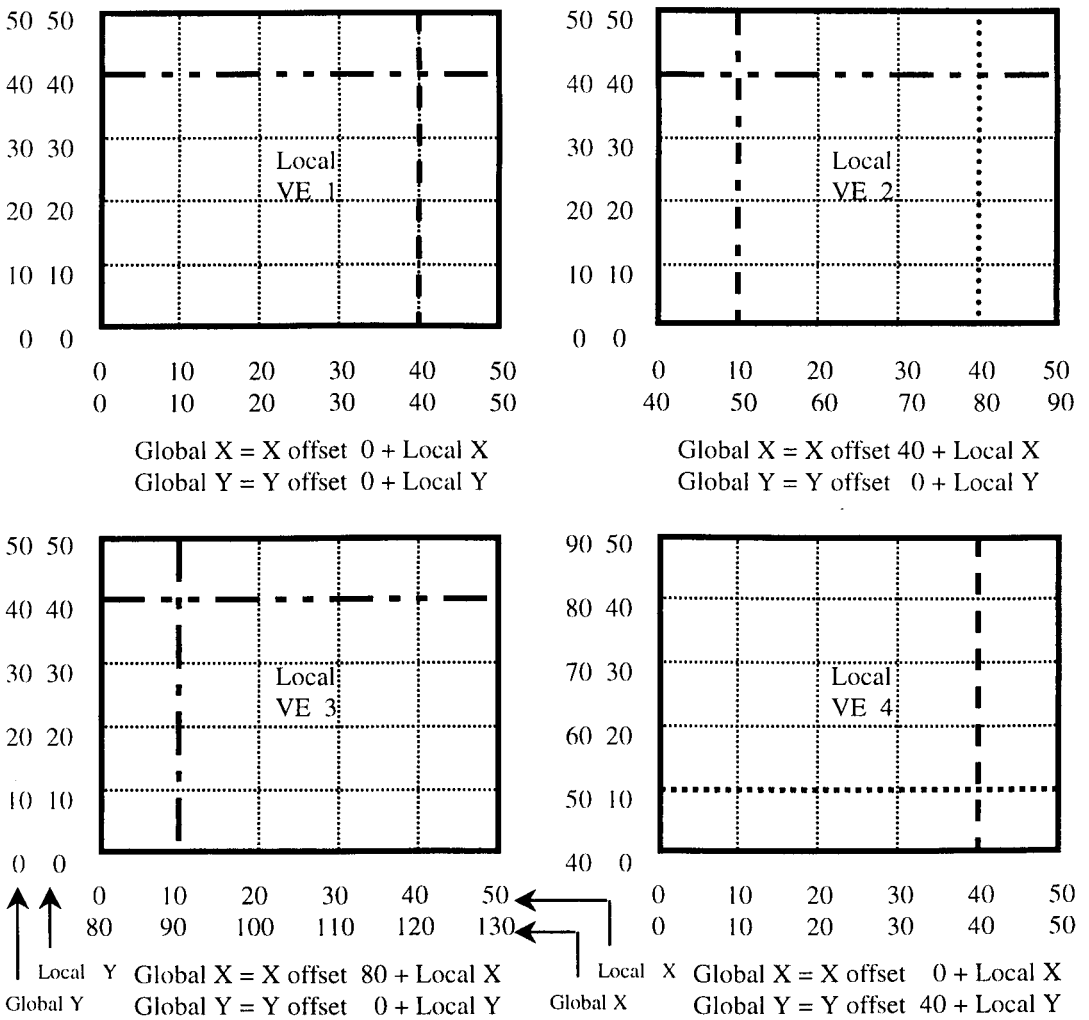
DVM uses two co-ordinate systems (Figure 3.10), one for the local co-ordinates and the second for the global co-ordinates, to represent the location of grid square. The local co-ordinates are used for the rendering of the local virtual world passed to each client computer. The global co-ordinates are used to represent the relative position of the local world in the global world.

3.2.3 Data Management

When the VE is distributed across more than one server node then the VE data is moved around from one node to other nodes. It is desirable to manage the data in some logical method. There are several methods for data management in a DVE system. DVM uses data distribution, and data segmentation.



(a) Global Co-ordinates



(b) Local Co-ordinates

Figure 3.10 The Segmentation of DVM Co-ordinate System

3.2.3.1 Data Distribution

In the case where all VE data is stored on a single database, managed by a single central server, this database can become very large. This prohibits the exploration of large, continuous data spaces. Furthermore, in distributed VE applications, potential disadvantages of a centralised server are that it could become a bottleneck and that it is potentially a single point of failure. A solution to these problems is data distribution. An approach to data distribution is the distributed server.

With the distributed server, the VE data may be present in one location or be distributed geographically. Each location has a local server, which responds to client requests for that location's data. The VE data that is collected at several locations is stored at those locations. The VE data can be managed by multiple servers, which can be distributed across multiple databases.

Even though the distributed database construction issue currently obstructs the development of very large scale VE, and distributing the virtual world over several databases can lead to problems with consistency and lag, there are several advantages for data distribution, including reliability, availability, and speed-up of processing.

3.2.3.1.1 Complete Distribution

In the complete distribution, there is no duplication of data between nodes across the networks. Each node controls its own data. Each node has different data from others. The state of the VE is completely distributed in several nodes across the networks. If a node wants to update some VE state under the control of another node, a node should make a request to the node managing that VE state. A given VE state is only held in one place. Sometimes, in the DVM system,

several nodes should share some VE state at a time, and should change part of the VE state. Therefore, it is not adequate for the DVM system.

3.2.3.1.2 Data Replication

With data replication, the same data is duplicated in more than one location in order to speed up access to the data. It is of course important to ensure that procedures exist for updating the local copies of the replicated data so that the data does not get out of date. Two common approaches for data replication between nodes are to either use a server to hold all the data and partially replicate this in the client node or to replicate the whole environment at every client node in the network. The former burdens the server and the latter burdens the clients.

By replicating data, we can improve the overall throughput by placing server replicas at nodes where the data is needed. To optimise transmission of data in a client-server system, the server stores the data for a virtual environment, composed of entities that are arranged spatially. The client allows the user to display and navigate this VE database. For this purpose, the client needs only those data items that are actually being displayed.

The DVM system uses partial replication in one form or another. Instead of replicating entities in each node, requests are sent to a server over the network. There is no need to transmit the whole data from the server and store it in the client node. Clients only receive the data they need, thus reducing bandwidth consumption. In particular, a DVM system uses active replication. Most data is modified at the clients locally and the updated data is sent back to the server and the other clients. The complete replication of the database is also grossly inefficient and some means of partitioning data is required. If the whole scene is stored locally, it must contain all data, even if some data is never actually used. The DVM system does not use the complete replication method.

3.2.3.2 Data Segmentation

In order to display some VE scenes, all VE data needed to be displayed should be stored locally; so the data must fit into memory. The number of polygons passed to the client node has a major impact on the frame rate of the system. The texturing of the polygons, depth complexity, and vertex transformation also has an impact on the system's performance as well.

Therefore, in order to increase the performance, the virtual world can be segmented to reduce the amount of data passed to the rendering process on the client node. So the VE area is divided into a number of areas, the whole VE data is divided into small groups. That is, the global virtual environment is divided into a number of local virtual environments. Each local virtual environment consists of grid squares (Figure 3.11).

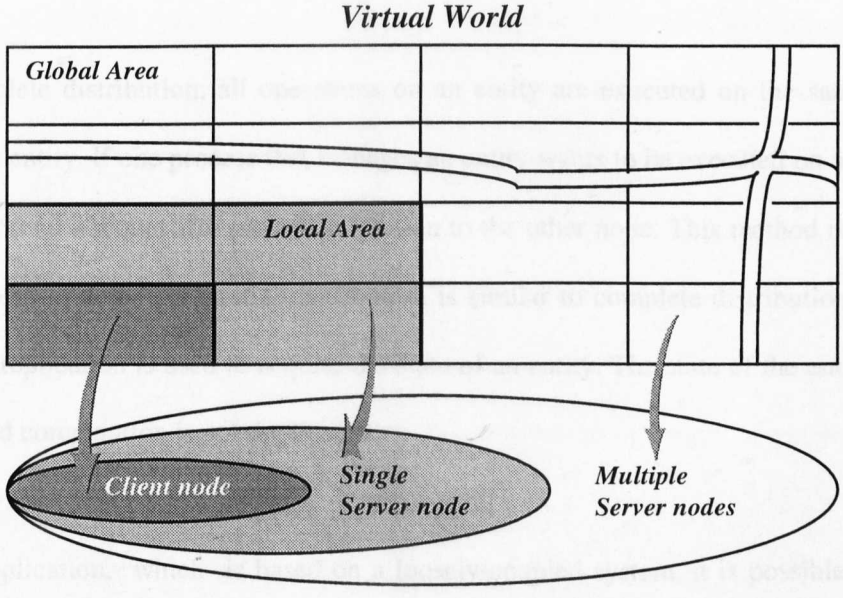


Figure 3.11 Logical Data Inclusion

3.2.4 Computation Management

The computations required to realise a VE can be performed within a single node or through a set of nodes communicating via networks. Rendering the virtual environment on a single node requires a large computational burden. The computation needed for rendering complex scenes affects the real-time property of DVE systems. Computational load distribution is required owing to the characteristics of real-time DVE systems. The distribution of computation is very important for a DVE with a large number of entities.

A distributed computation describes the execution of a distributed application by a collection of processes on multiple computers. It is important to increase computation capability, to broaden disk bandwidth, and to extend physical memory capacity. The distribution of computational load could greatly improve the performance of a DVE system. There are several methods for computation management in a DVE system, including Complete Distribution, Partial Distribution, Partial Replication and Total Replication.

In the complete distribution, all operations on an entity are executed on the same node that includes the entity. If one process that manages an entity wants to be executed on another node, then it must send a request for process migration to the other node. This method is useful for a tightly-coupled system. The partial distribution is similar to complete distribution except that partial data replication is used to acquire the state of an entity. The state of the entity is locally changed, and computation is not duplicated.

In partial replication, which is based on a loosely-coupled system, it is possible to replicate some of the entity state computation on the client nodes. Ghost or proxy processes are usually used to approximate the entity's behaviour using a dead-reckoning method (refer to Section

4.4.5). This method is useful for reducing the amount of bandwidth required. In total replication, also based on a loosely-coupled system, if the DVE is running over a very large distance, partial replication that receives periodic updates is not efficient. Therefore, all computation is performed locally, and only information that changes the state of the entity is sent. Every node can have the same state of entities. However, this method requires duplication of computation.

A DVM system is mainly based on a loosely-coupled system that operates over a large distance. Therefore, a DVM system uses both partial and total replication. When the state of an entity using an algorithm such as dead-reckoning is updated, to reduce network traffic, it uses partial replication to update some of the entity state. Otherwise, total replication is usually used during the implementation of the DVM system.

3.2.5 Fault Tolerance

As networked systems gain increasing importance, system down-time becomes less and less tolerable. The down-time of a network system, even for a short time, will affect the whole system significantly. The purpose of a fault-tolerant system is to overcome system faults and continue the execution of user programs. In other words, fault tolerance means that a system will continue to process even when a component has failed. There are many reasons for the failure of the system, including software and hardware component failure, node failure, and failure of a communication path. To overcome these problems, a variety of systems have been used for implementing fault-tolerance. Fault tolerance is a very important issue for a DVM system as well. It may be considered as a future work. This is currently out of the scope of this thesis.

Although a DVM system does not support complete fault tolerance, it can be a fault-tolerant system. If a server or client node fails, a whole DVM system is never disabled by a single node failure. In a DVM system, all server and client nodes in the same VE group keep the same copy of entities that is currently running, if a node fails then it can be recovered by restarting the node and receiving the appropriate data from a client or server node. The node can be recoverable, it can therefore be fault tolerant.

3.2.6 Time Management

In the DVM system, there is a need to synchronise the clocks between nodes since the execution of the schedule for a node is done globally. VE time must be co-ordinated so event causality is maintained for all VEs. A DVM system co-ordinates VE time among all client and server nodes.

Figure 3.12 shows the time sequences for common VE time synchronisation of a DVM system. When a server node starts up, a server node sets the global VE time. When a client node starts up, a client node sets the local VE time. To perform time synchronisation and establish a common VE time, a client node requests the global VE time from the server nod. The server node then transmits the global VE time to the client node. The client node calculates the round trip time to check the amount of time, as latency and lag, that elapses between the time at which the global VE time was requested and the time of receipt.

The round trip time is calculated by sending a message to request the global VE time from the server and waiting for the global VE time be returned by the server. After calculating the round trip time, values that are too small will be ignored. Finally, the client node replaces the local VE time with the global VE time. All messages in the DVM system are time stamped; this is to

identify messages that have been processed. Each node resets the global VE time every fixed time period (for example, 100 or 200 seconds).

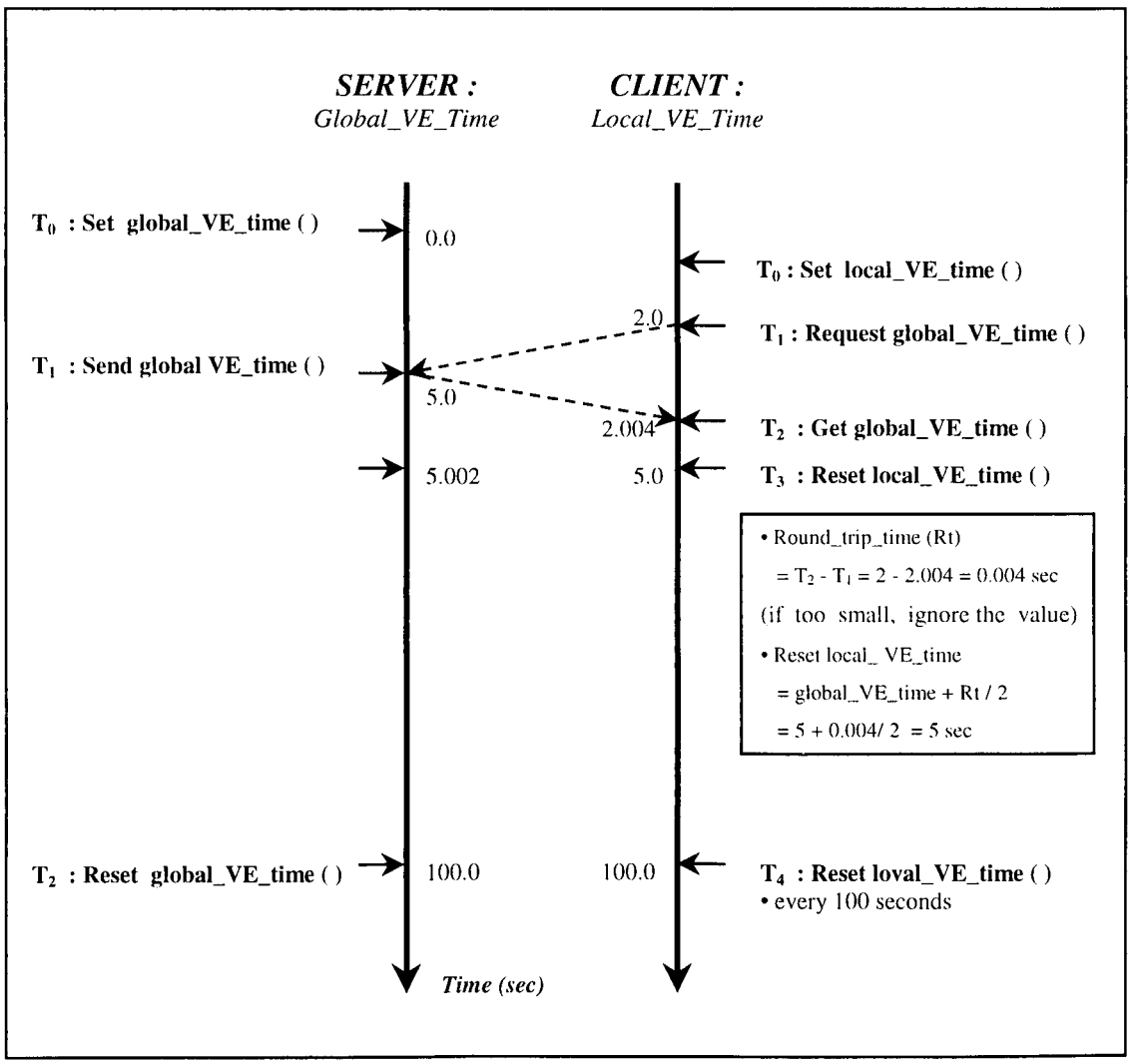


Figure 3.12 Time Sequences for Common VE Time Synchronisation

3.2.7 Security

DVE systems have multiple security domains and thus exhibit the set of security properties. The trusted computing base is scattered among many components that operate in environments with varying degrees of physical security, differing security policies, and possibly under different authorities. The interconnections among the computers are physically insecure. It is hard to know what is being trusted and what can be trusted. But, because the system contains many computers, exploiting a security flaw in the software or environment of one computer does not automatically compromise the entire system. In the DVM system, current efforts in this area is out of the scope of this thesis.

3.2.8 Communication Management

A DVM uses a network to distribute entity-state information and graphical images. A DVM has its own local virtual environment, and it is based on the client-server model. A server node communicates with its clients or other servers using process-to-process communication. Each server node keeps a list of its client nodes. The client node exchanges data with the server process. Each client node can also communicate directly with other client nodes.

The DVM protocol can be symmetric or asymmetric. In a symmetric protocol, either side can play the master or slave role. In an asymmetric protocol, one side is always the master, and the other is the slave. The DVM protocol uses reliable point-to-point or multicast, based on IP using Berkeley Sockets. The DVM system is a loosely coupled heterogeneous distributed system based on UNIX and Internet networking protocols within local and wide area networks. The DVM uses UDP and TCP sockets for handling the network connections between clients and server, which are bi-directional. This section contains only short explanations about the

communication of the DVM system. The details of communication management will be covered in Chapter 4.

3.3 System Operations

All of the components in a DVM system are independent processes, but individual processes can not properly run without the interactions with other processes. Each process has some functional relationships with other processes. In this section, we will describe the major operations that are performed by processes and the relationships between them.

3.3.1 Initial Join

In the DVM system there are several server nodes, each server node operates independently. There is no central server node among the server nodes. The various server nodes are aware of one another. The execution of the global VE will require communications via networks between the server nodes and client nodes. Each server node provides an environment for processing both local and global VEs. When a server node starts, it can connect to the other server nodes in the same local VE. Each server supports its own client nodes. The server node must normally be able to support requests from multiple clients.

Most client nodes operate independently. Each client node is able to access data in a server node to process a local VE. A client node can also participate in the execution of global VEs that access data in several server nodes across the WAN. A client node can freely access, connect, and communicate directly with one of the server nodes in the same global VE. A client node would be permitted to join and leave the server node at

will without disturbing other client nodes. A fault in the client node does not lead to the breakdown of the server node. A client node application can be turned off at any time without fear of inconveniencing other client nodes.

Figure 3.13 shows a routing sequence for the initial join between client and server node. When a DVM application in the client node starts, the client node always requests the initial join to a server node. The Master Manager (MM) first starts and summons up the Network I/O Manager (NM) in the same node. The NM of the client node requests the initial join to the NM of a server node. The server node normally listens at a well-known address for service requests. At such a time the server node responds to the request for the initial join. That is, the NM of the server node checks the requests from the NM of a client node. The NM of the server node transfers the initial requests to the MM of the same node. The MM of the server node decides whether to permit the initial join or not. The MM of the server node transfers the result to the NM. The NM of server node transmits the result to the NM of the client node.

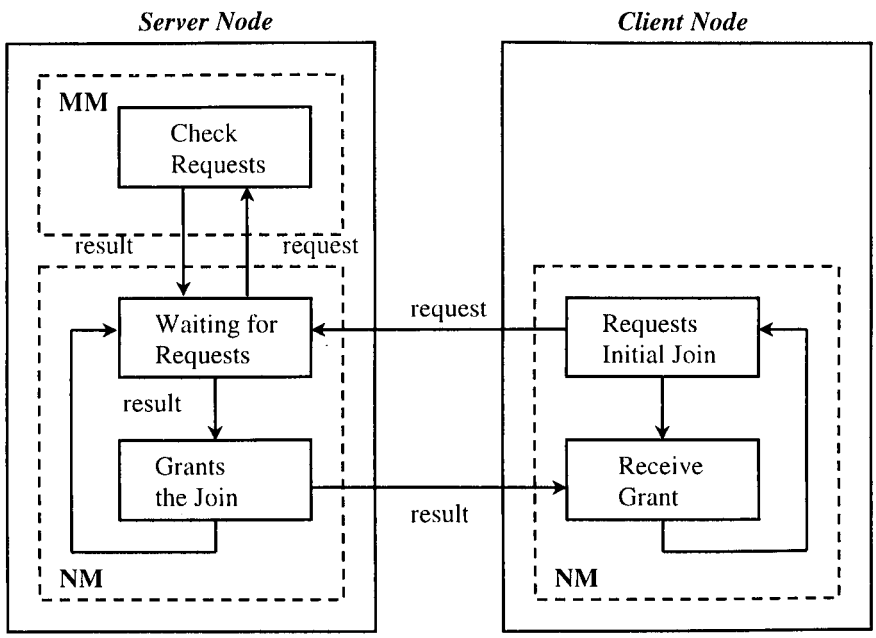


Figure 3.13 Routing Sequence for Initial Join

3.3.2 Initial Data Transmission

Each server node has its own databases. Client nodes can access any databases via a server node in the same DVM system. That is, the client nodes can load initial data from the databases via a server node, no matter whether they are on the same or different networks.

As shown in Figure 3.14, the NM (Network I/O Manager) of a client node sends a request to the NM of the server node to load the initial data. The NM of the server node sends the request to the MM (Master Manager) to load data from the databases. The MM gives commands to the DM (Database I/O Manager). The DM of the server node directly interfaces with the databases. The DM loads the initial data from the databases, and transfers to the NM of the same node. The NM of the server node transmits the initial data to the NM of the client node. The client node does not need to remember the databases on which data are currently located or where they were created.

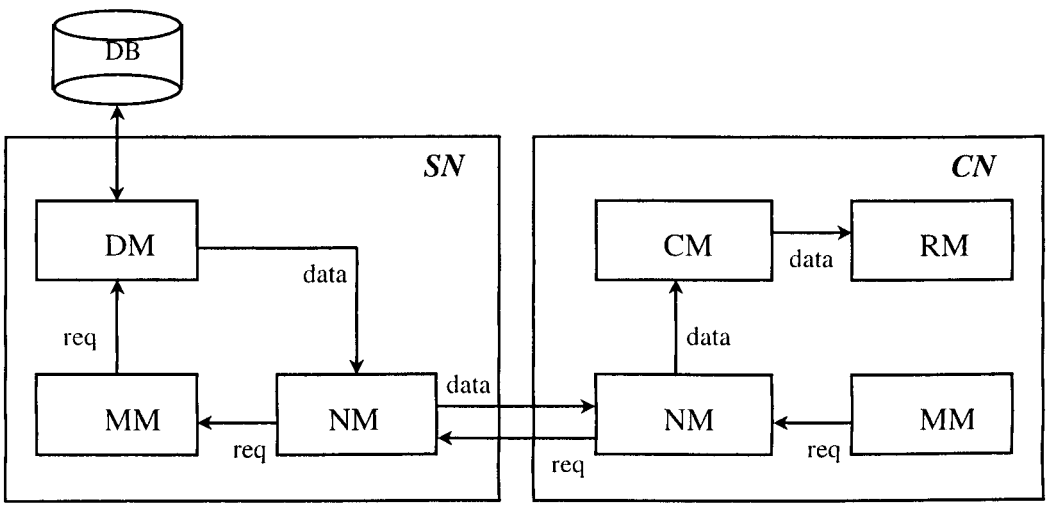


Figure 3.14 Routing Sequence for Transmission of Initial Data

3.3.3 Data Loading Hierarchy

In a DVM system, a client node allows the user to display and navigate the virtual world. The virtual world usually contains large amounts of data, e.g. buildings, trees, bridges, cars, planes, people, etc. The data of the virtual world is loaded from a number of databases over networks. However, access to the databases is slow; fetching the data from the databases is the difficult part of the process. It is one of the major problems in implementing the VE. If the required data can be delivered over the network just in time for the rendering process, the distribution of data is very useful. So we attempt to optimise transmission of VE data in a client-server system, and to gain significant savings in local memory requirements and network bandwidth.

For this purpose, there is no need to transmit all of the data from the databases and store it at the client, the client needs only those data items that are actually being displayed. Therefore, the client node loads the part of the VE data required for rendering the VE scene from the databases. The selection of data is up to the client node, so that various strategies for data management are possible.

Figure 3.15 shows a strategy for loading of VE data for the DVM system. The VE data is distributed on two databases A and B. A DVM system uses a hierarchical structure to load the data from the databases. Each database stores the data for a local virtual environment, composed of entities that are arranged spatially. Users invoke an individual client node to interact with the environment. The client node can request data from the server node. If a client node wants to access the local VE 2, the client node requests the data of local VE 2 from the server node A. The server node A loads the data from the database A, the data of the local VE 2 is stored on the server node A locally for access by the client node. The data is replicated in the

memory X of the client node. The data replication means that the VE scene must be stored locally.

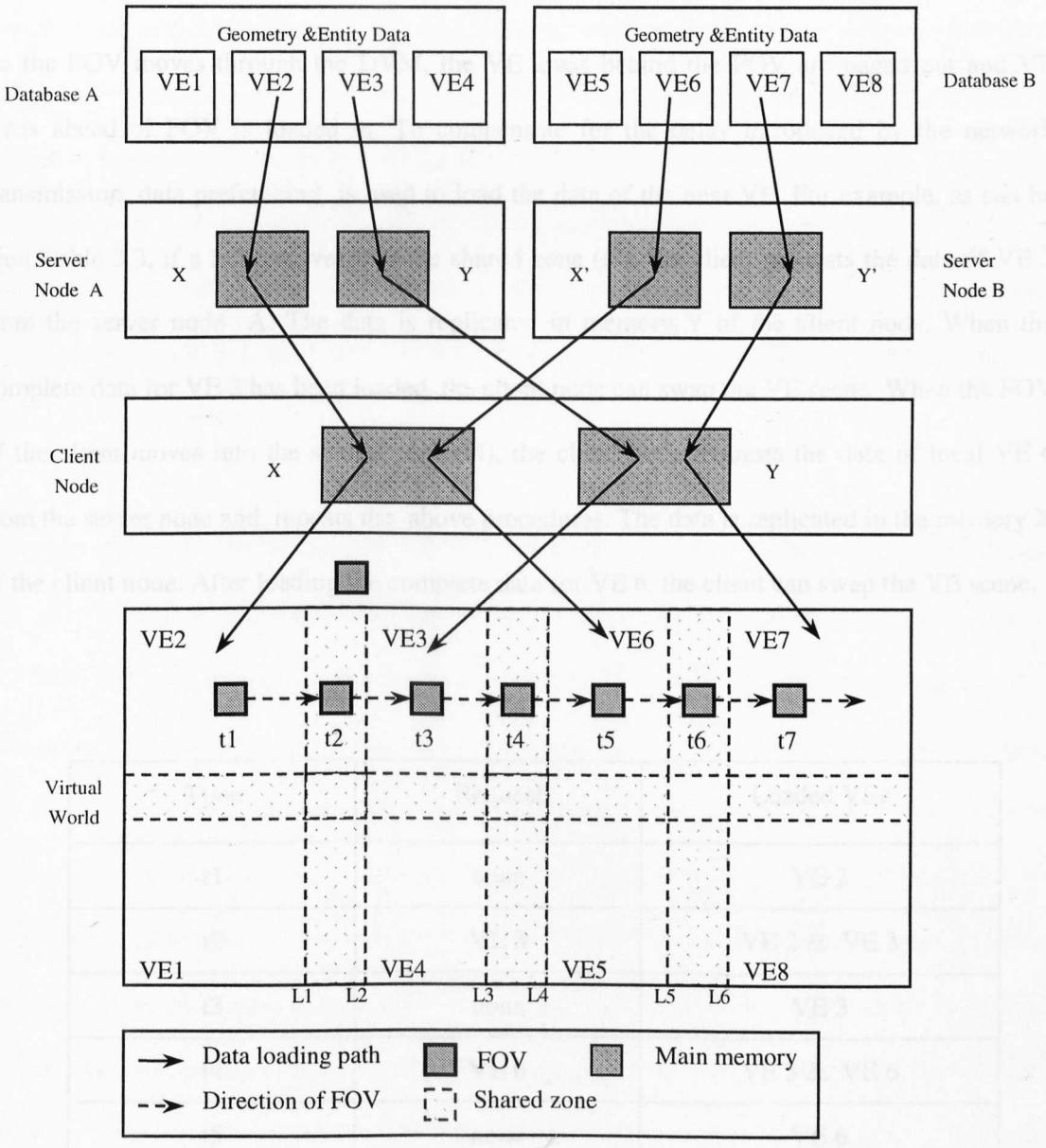


Figure 3.15 Data Loading Hierarchy

In order to be displayed at the client node, the data of the local VE 2 must be replicated from server node A to the client node. The data of the whole local VE 2 does not usually fit into main memory of the client node. This prohibits the exploration of large, continuous data spaces. Therefore, a part of the local VE 2 is replicated at the client node.

As the FOV moves through the DVM, the VE areas behind the FOV are paged out and VE areas ahead of FOV is loaded in. To compensate for the delay introduced by the network transmission, data prefetching is used to load the data of the next VE. For example, as can be seen Table 3.3, if a FOV moves into the shared zone (t2), the client requests the data of VE 3 from the server node A. The data is replicated in memory Y of the client node. When the complete data for VE 3 has been loaded, the client node can swap the VE scene. When the FOV of the client moves into the shared zone (t4), the client node requests the data of local VE 6 from the server node and repeats the above procedures. The data is replicated in the memory X of the client node. After loading the complete data for VE 6, the client can swap the VE scene.

Time	Request	Loaded VEs
t1	none	VE 2
t2	VE 3	VE 2 & VE 3
t3	none	VE 3
t4	VE 6	VE 3 & VE 6
t5	none	VE 6
t6	VE 7	VE 6 & VE 7

Table 3.3 Prefetching of VEs

3.3.4 Screen Generation

The available memory works like a cache. Naturally, its size is limited, so the active data set must exceed neither the processing capacity of the renderer nor the cache size. As typical workstations have ample memory, the graphics processing power is usually the more restrictive factor. The image presented to the user should be smoothly animated and updated at a sufficiently high and constant frame rate. Both the user and simulated entities can change their position, but if the user moves too fast, or entities move too fast, the data needed for display at the appropriate resolution cannot be made available in time. In some situations the client may be able to store and display a scene at high resolution, but the network is so slow that the data cannot be transmitted in time.

Polygons are used to draw and depict the virtual world viewed by the user. By decreasing the size of the polygons, more detailed images can be created. Using levels of detail (LOD), users can keep the total polygon count in their view fairly constant and thus allow the scene to maintain an adequate frame rate as they move through the environment. While moving closer towards an entity will increase the number of polygons for that particular entity, the total number of polygons that have to be rendered in the scene may actually go down because other entities may move out of view and no longer need to be displayed. Modern high performance Computer Graphic Workstations can render as many as 350,000 independent facets (polygons) per second when giving medium quality shaded images. This includes hidden surface removal and some lighting effects [Gigante, 93]. Approximately thirty frames per second are a minimum for providing acceptable performance in a virtual environment. It has been estimated that rendering of a realistic image consists of approximately 80 million polygons per second [Rheingold, 91]. However, current systems might only produce such detailed pictures at a rate of five or six frames per second.

3.3.5 Screen Control

All client nodes connected to the same server node simulate the same local VE. The server node monitors the movement of entities for every connected client node. Two geographically distant client nodes do not need to be aware of each others activities until they connect to the same server node. Figure 3.16 shows the movement of the FOV (Field Of View) and entities in a VE. The FOV of the client node can move in any direction within the local VE. The entities can also move in any direction within the VE. If entity-a moves into the shared area between L1 and L2, the entity a has to be newly introduced to the server node B that cannot otherwise know about this entity. If entity **a** moves out of the shared area, it is removed from VE A. The FOV-B in the VE A and FOV-C in VE B can both share the entity **d**.

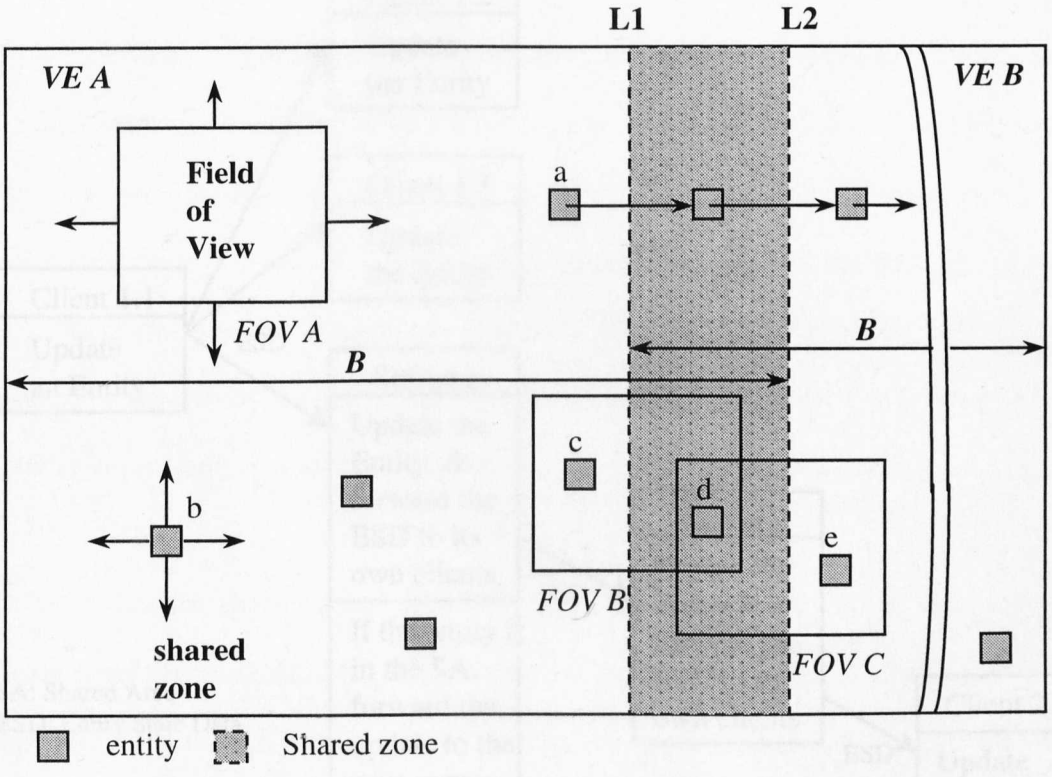


Figure 3.16 The movement of FOV and Entities in the virtual world

3.3.6 Update of Entity State

The changes of entity state can be transferred between the nodes using multicast or unicast, after establishing a logical connection with the destination node (Figure 3.17). If the state of entities in a client node is updated, the data is sent to the server and client nodes. Changes in the state of entities can be directly sent to all of clients within the same local VE. Changes in the state of entities within the shared area can be sent to other server nodes in the different local VEs. The other server node in a different local VE retransmits the data to its own client nodes. Backup is needed only on the server node running the DM, since the client node not running the DM is merely used for caching.

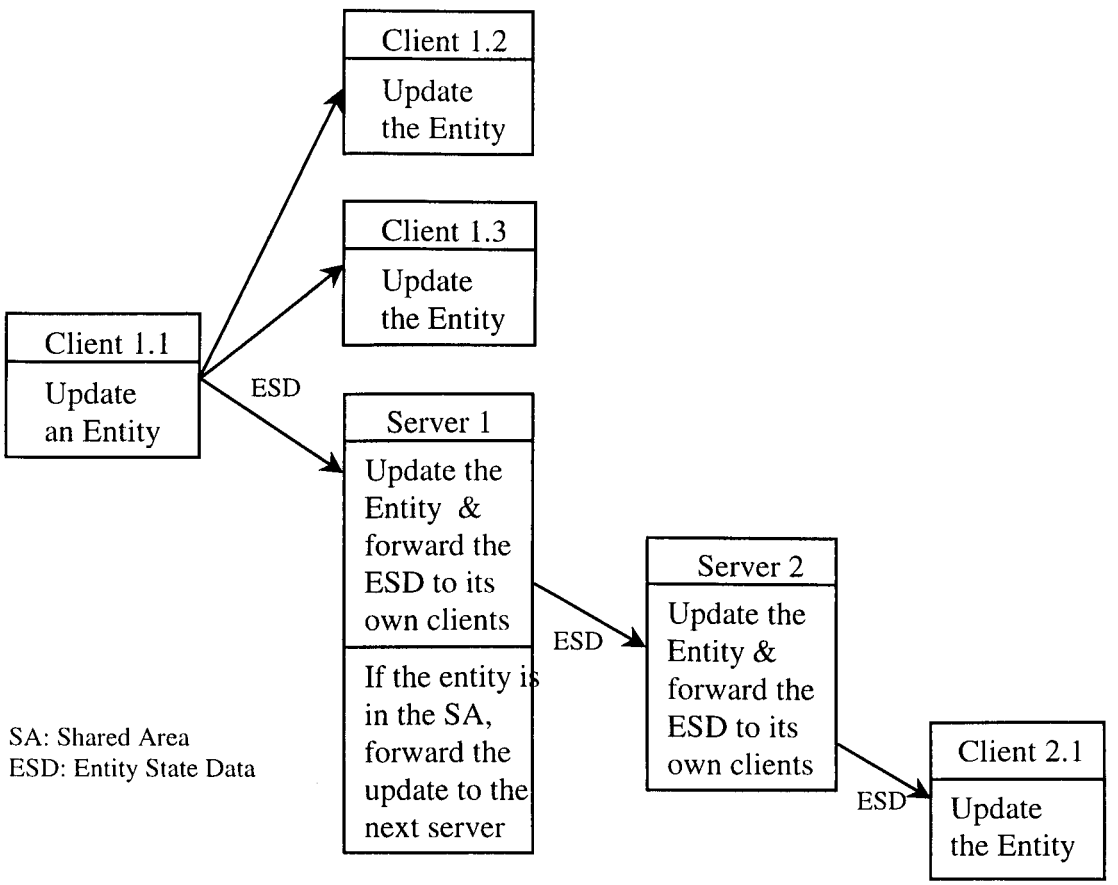


Figure 3.17 Order of events for an entity update

In order to prevent an inconsistent instance state of the entity when VE worlds simultaneously try to update it, the concurrency control scheme can be used. It maintains the consistency of the distributed virtual world database by restricting manipulation in such a way that only one site can perform operations to alter the status of the virtual world at a time.

Furthermore, when a client node updates some data, it has to transmit the updated data to the server node. The server keeps the current states of the entities in the DVM. Therefore, if a client node fails, then the client node can try to recover the current DVM environment, by restarting the client node and receiving the data from the server node. Backup operation is needed only on the server node running the DM.

3.3.7 Time-Based versus Frame-Based VE simulation

On a particular computer, the VE is calculating the position of the entities using the last reported velocity. The process may draw the entity in one position, then clear the image and draw it again in another position. These drawings are called frames. When the velocity of an entity is specified in frames, if the entity moves one centimetre per frame, the entity may end up at different locations on different machines. For fast machines drawing at fifty frames per second, in one second the entity will have moved over half a metre. For a slower machine, drawing at ten frames per second, the entity will only move a tenth of a metre.

To solve this, the entity's speed must be specified in distance per unit of time, rather than distance per frame. Instead of 1 cm/frame, we specify velocity as 1 cm/second. On fast machines the animation would be smoother than on slower machines, but both would present the entity at the same position.

3.4 Summary

This chapter has described where a proposed DVM system differs from traditional VEs and DVEs. First, the concepts of a DVM system were described. The concepts include a basic structure, characteristics, requirements, limitations, and target applications of a DVM system. Next, the proposed architecture capable of implementing a DVM was described. The architecture presents the components and their connections, co-ordinates, data management, computation management, and so on. Finally, the operations of the DVM system are described. The operations show how the DVM system works, how components interact each other, how data is loaded, how data is updated, how the screen is controlled, and so on. The issues discussed show what the DVM actually does.

To implement this DVM system, much effort is required. Especially, a DVM system requires fast data communication, with the overall goal of minimising bandwidth consumption and latency on the network. It is reasonable to assume that the data communication is the most constrained functionality of the DVM system. A new networking architecture is therefore required for implementation of a DVM system. The technical details of a networking architecture are presented in Chapter 4.

CHAPTER 4

A NETWORKING ARCHITECTURE FOR THE DVM SYSTEM

The general architecture for implementing a DVM model was described in Chapter 3. This chapter will concentrate on the networking aspects of a DVM based on the architecture described in the previous chapter. The DVM system, which is made up from nodes (computing resources) connected to Local Area Networks (LANs) and Wide Area Networks (WANs), will host several independent programs, called processes, which will be working concurrently. The DVM system attempts to simultaneously connect many nodes in order to create the functionality of a realistic world in meaningful ways. The interconnection of all nodes is the best way of supporting the global interaction of users and processes in the DVM. Each individual node will wish to communicate over the network to other nodes. To accomplish this, the network becomes more than one connection between nodes.

A variety of network elements is required to scale up the DVM to arbitrarily large size, simultaneously connecting thousands of interacting users and all kinds of information entities. The DVM systems require the distribution of ongoing events and user actions. This requirement quickly exceeds network capacity even for a moderately sized network group. So it is important

to design new methods for the distribution and transmission of VE data, with the overall goal of minimising bandwidth consumption on the network. The networking is the most constrained resource of the DVM system. The networking architecture determines not only the performance of the network module, but also the capabilities of a DVM application. Development of networking architectures is the key to solving DVM challenges.

4.1 Goals of DVM Networking

As already described in Section 3.1, a DVM is based on the client/server mechanism with multiple servers. A DVM system therefore consists of a collection of nodes that include clients and servers. As shown in Figure 4.1, a DVM system establishes connections between a client and a serve, between servers, or between clients. A global VE is divided into several local VEs. The nodes are grouped into a local VE group based on a server node. That is, the servers are distributed across multiple locations. A server connects physically to a local VE database. A client can be connected logically to one of the servers. The client nodes can be distributed physically either over a large geographical area or over a small geographical area. The former type of network is referred to as a wide area network (WAN); the latter is referred to as a local area network (LAN).

If a client in one network wishes to load data stored in another database in the wide area network, it is possible to load packets from the database via a server node on another local network. A long distance telecommunication link and a pair of boxes called IP Routers attach the networks. The forwarding of packets requires routers that are attached to networks. They forward packets based upon their destination IP address. Remote router-linked networks are physically distant from each other and use a pair of routers to connect two networks over a telecommunications link, to make the two nets

operate as one. Packets between the server nodes on the networks will have to go through the paired IP Routers in order to reach their destination.

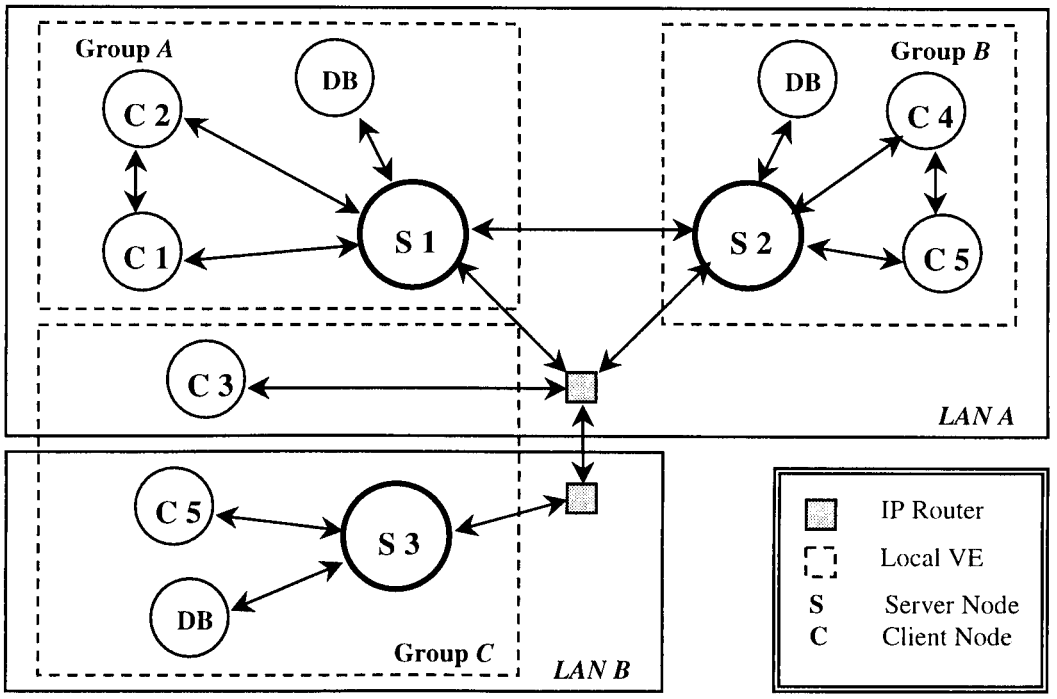


Figure 4.1 Networking Configuration of a DVM system

To implement a DVM system based on above configuration, a networking architecture for exchanging messages between the nodes on the networks is required. The network performance and correct co-operation between nodes are very important as well. This section, therefore, describes the major goals for the design and implementation of the DVM networking architecture. These are summarised below:

- Allow for low latency interaction between nodes in the DVM.
- Allow for implementation even in low-bandwidth environments.
- Each server to support simultaneous handling of multiple clients

- VE operations should be distributed on the multiple nodes, and these nodes should correctly inter-operate in real time.
- A server node controls and monitors all its own client nodes
- Allow the DVM to run on multiple processors.

To achieve the above goals, the following sections will mainly concentrate on the networking subsystem and the major techniques for implementation of a DVM based on a client/server structure over the Internet and LAN. These sections will describe the basic restrictions of DVM communication, the layered structures of DVM networking, and DVM operations. It also details how users interact with the DVM and its networking, what users can and cannot do, how messages are transmitted, what messages are exchanged, and what techniques are useful.

4.2 Basic Restrictions of DVM Communication

There are several restricting factors for communication in a DVM. The major concerns are *limited bandwidth* and *network latency*, and there is a relationship between the two. As already described in Section 2.2.2.2, network efficiency is usually expressed as bandwidth. For a DVM system, bandwidth and latency are equally important. The greater the bandwidth of a transmission system, the smaller the network latency between nodes. Every DVM transmission system has a limited bandwidth that can not be increased unlimitedly.

4.2.1 Limited Bandwidth

A DVM can require enormous bandwidth to include multiple users, video, audio and the exchange of 3D graphic primitives and models in real time. If a DVM application violates the bandwidth limits of inter-networks, it can cause congestion and loss of packets, or overrun of

the receiving hosts. As the number of participants increases so the bandwidth requirements do. The larger the geographical distance covered by a network, the smaller the effective bandwidth available to each node. So the available network bandwidth determines the size and richness of a distributed virtual environment. The bandwidth available in the DVM will also vary depending on the communication protocol used across networking mediums, and on the amount of traffic.

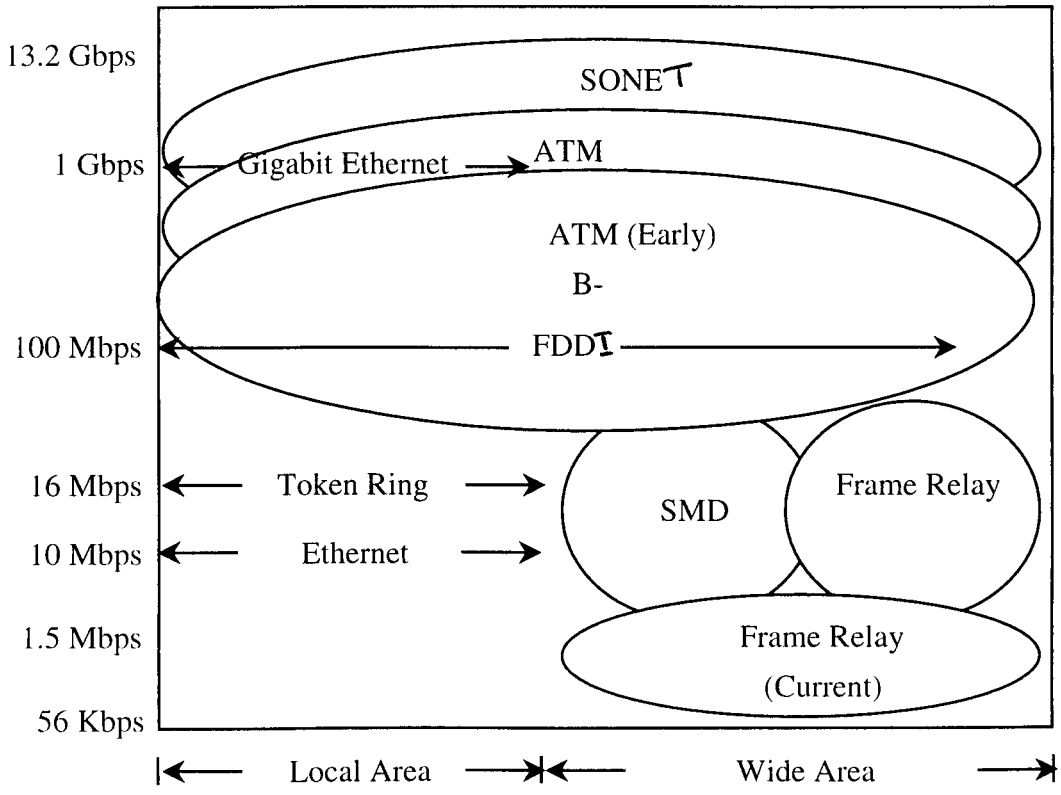


Figure 4.2 The Technology Scenario

Figure 4.2 shows the bandwidth technology scenario for networking services [Nees, 94]. On the Local Area Networks (LAN) like Ethernet (10 – 100Mbps), this is not a major issue, LAN-based DVEs have a relatively limited number of users. However, on Wide Area Networks (WAN), maximum bandwidths have been currently limited to Asynchronous Transfer Mode (up

to 2.5 Gbps) but the potential user base is much larger because of the Internet. In the future, the bandwidth requirements will be a less important problem for communication in the DVM.

4.2.2 Network Latency

The interactive and dynamic nature of the DVM is controlled by network latency. If a DVM system is to mirror the real world, the DVM system should deliver packets with minimal network latency and generate textured 3D graphics at 30-60 Hz to guarantee the illusion of reality. Network latency will range from under a millisecond on local area networks up to several hundred milliseconds on a wide area network. In general, the distributed systems that use wide area networks, long paths, switches and routers induce latency. Latency is a problem for network cue correlation including the delay of an individual cue and variation in the length of the delay [Sawler, 91]. Although lag can never be totally eliminated for environments where the DVM is widely distributed, network latency can be reduced to a certain extent by using dedicated links, improvements in router and switching technologies, faster interfaces and computers, better communication protocols, and so on.

4.2.3 Summary of the Communication Restrictions

To reduce latency and the amount of bandwidth required, trade-offs obviously have to be made. A common approach is to sacrifice exact consistency at all times in favour of approximate consistency. Another common technique is to subdivide the environment into zones, pieces that can be treated more independently [Barrus, 96]. The DVM networking, therefore, is based on these approaches.

4.3 DVM Networking Subsystem

As described in Section 3.2, a DVM system is implemented by exchanging messages between nodes on the network. The Network I/O Manager (NM) at each node is in charge of the communication of all messages between nodes. The DVM Subsystem is implemented on the TCP/IP Protocol Suite. Figure 4.3 shows the DVM Network Subsystem in the TCP/IP Protocol Suite. The messages are transmitted to the targets, using TCP/IP and the DVM network subsystem.

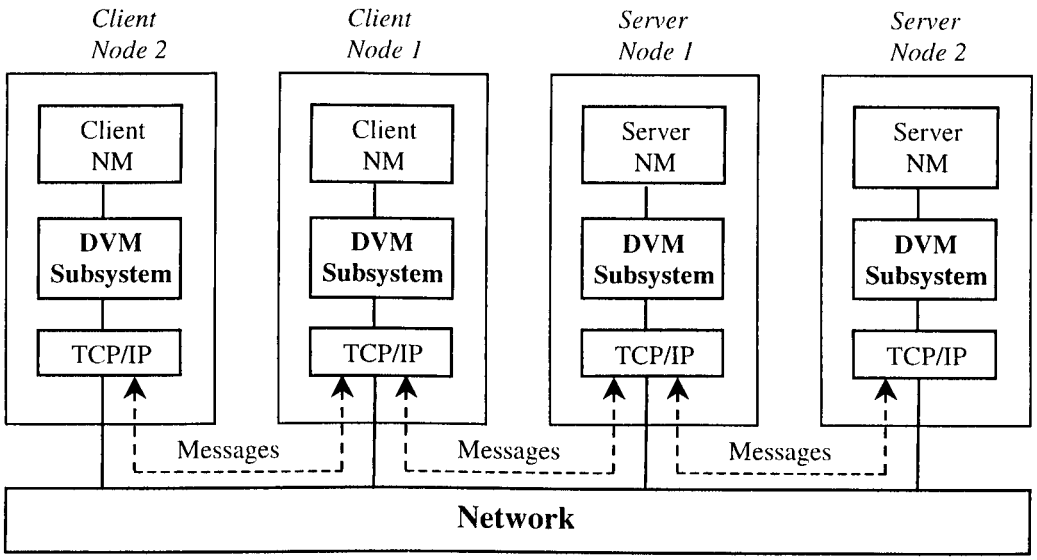


Figure 4.3 DVM Network Subsystem in the TCP/IP Protocol Suite

4.3.1 Requirements for DVM communication

In a DVM environment, when several client and server nodes connect to the network, each individual node can try to communicate over the network to other nodes. This requires pre-agreed communication mechanisms. The communication between the nodes usually requires reliability and high speed. Reliability means that data being sent is always received correctly, thus excluding the need to periodically re-send information. However, to guarantee delivery, the communication should

use acknowledgement and error recovery schemes that can involve long delays. Higher levels of reliability involve more overheads, slowing communication. These two issues are often at odds. A DVM, therefore, should consider these two potential problems of communication. The communication reliability between nodes negatively affects the real-time response of the entire system and is also prohibitively expensive in terms of bandwidth. Deciding whether to use a reliable message delivery service or not is a key decision in the networking design of a DVM system.

4.3.2 Subsystem Structure

The DVM networking subsytem basically consists of four distinct layers. These layers contain *a Packet Interface Layer, a Socket Interface layer, a DCP (Communication Protocol for a DVM system) Protocol Layer, and a Process Interface Layer.* Figure 4.4 shows the DVM Networking subsystem structure containing these layers. The incoming packet is passed up to the NM via these layers in sequence.

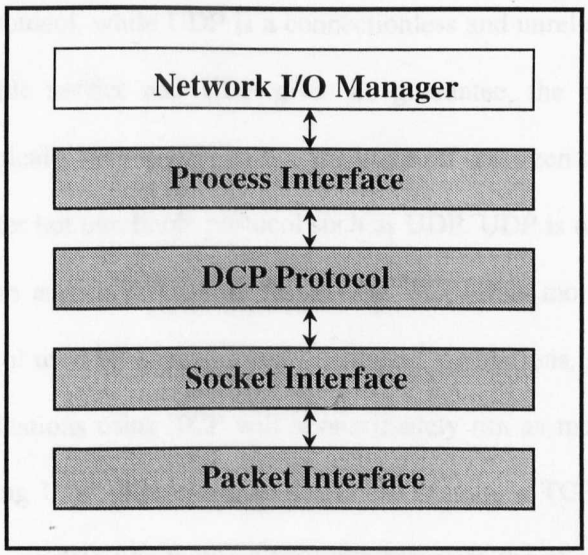


Figure 4.4 DVM Networking subsystem structure

4.3.3 Packet Interface Layer

There are many different kinds of communication protocols running on networks. The current implementation of the DVM system uses TCP/IP to let clients and server communicate with each other. This was chosen because a reliable protocol is preferred, and the interworking of multi-vendor equipment is implemented better with TCP/IP than by any other current protocol suite. TCP/IP can provide more than one protocol, as can be seen in Figure 4.5. There are two very popular protocols, TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). TCP and UDP provide services at the Transport Layer and IP protocols offer services at the Network Layer. The IP layer handles routing through an inter-network. It is also responsible for breaking up large datagrams into smaller ones for transmission and reassembling them at the other end. The IP layer provides a connectionless and unreliable delivery system.

The Packet Interface layer is responsible for sending packets over the network, using UDP or TCP. This layer encapsulates the networking protocol (transport protocol) provided by the host system. This layer also encapsulates the physical network address format. The TCP is a reliable connection-oriented protocol, while UDP is a connectionless and unreliable protocol. Although TCP provides a reliable service and UDP gives no guarantee, the probability of a given message arriving is typically very high. There is the trade-off between using a reliable protocol such as TCP and a faster but unreliable protocol such as UDP. UDP is usually used when speed is more important than accuracy. UDP is faster than TCP, it is more representative of the communication protocol used by sophisticated distributed simulations, and the communication code is simpler. Applications using TCP will approximately run as much as ten times slower than those written using UDP. Therefore, in the DVM system, a TCP connection is used to transmit large amounts of data like textures from the server to the client, and UDP is used to exchange smaller objects between nodes, like the entity state data. TCP and UDP both have Port

Numbers, to distinguish among different data exchange participants. The same Port Number can involve two different processes, if one uses the number for TCP and the other uses it for UDP.

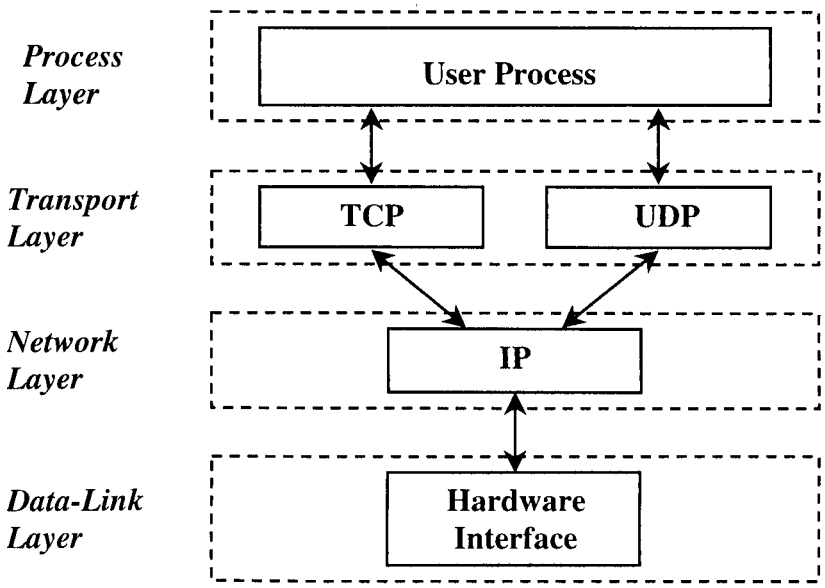


Figure 4.5 TCP/IP protocol suite using 4-layer model

4.3.4 Socket Interface Layer

For two processes to communicate with each other, they must both agree to it, and the operating system must provide *application program interfaces* (APIs) for the *Interprocess Communication* (IPC). The API is the interface available to a programmer. The API is the interface between a user process and a networking protocol. The availability of an API depends on both the operating system being used and the programming language. Examples of APIs include the Unix sockets, WinSock and Java sockets. These APIs use sockets to exchange data. A socket is a data structure maintained by the operating system to handle network connections. Sockets are represented by integers, and can be treated much like file descriptors. A connection

is defined by the combination of the two processes' socket numbers. A socket is an endpoint of communication which can be named. Each socket in use has a type and one or more associated processes. Once a socket is created, a user process will read from it, and write to it, and finally close it. It is the basic building block for socket-based IPC.

In general, there is one protocol for each socket type within each domain. The socket interface supports the TCP/IP protocol. To create a socket, the communication protocol to use (e.g. UDP or TCP) should be indicated. The code that implements a protocol keeps track of the names that are bounded to sockets, sets up connections, and transfers data between sockets, sending the data across a network. It is also possible for several protocols, differing only in low-level details, to implement the same style of communication within a particular domain.

Another important input parameter is the port number. For a process in a client host to communicate with a process in a server host, each process within a host must have its own unique address within the host. This allows the data to be delivered ultimately to the proper process. This is a 16 bit number. To send a message to a server, it should be sent to the port for that service of the host that it is running on. These addresses are known as port numbers. The lower numbers are often reserved, or "well-known" port numbers used for telnet and other communication software. Ports in the region 1-255 are reserved by TCP/IP. The system may reserve more. User processes may have their own ports above 1023.

Sockets are either connection-oriented or connectionless. Connection-oriented sockets allow for data to flow back and forth as needed, while connectionless sockets allow only one message at a time to be transmitted, without an open connection. There are different socket families. The two most common are AF_INET for Internet connections, and AF_UNIX for Unix IPC (interprocess communication), but the DVM deals only with AF_INET sockets.

The DVM uses two types of sockets: *stream socket and datagram socket.* A stream socket provides a bi-directional, reliable, sequenced, and unduplicated flow of data without record boundaries. A datagram socket supports bi-directional data flow, but doesn't guarantee that the message data is sequenced, reliable, or unduplicated. Datagram sockets provide routing and fragmentation. Routing is used to forward messages from one local network to another nearby or distant network. Fragmentation divides large messages into pieces small enough to fit the local medium. Table 4.1 shows a summary of stream sockets and datagram sockets.

Role	Steps	Stream Sockets	Datagram Sockets
Server	create endpoint	socket ( )	socket ( )
	bind address	bind ( )	bind ( )
	specify queue	listen ( )	
	wait for connection	accept ( )	
	transfer data	read ( ) write( ) recv ( ) send ( )	recvfrom ( ) sendto ( )
Client	create endpoint	socket ( )	socket ( )
	bind address	bind ( )	bind ( )
	connect to server	connect ( )	
	transfer data	read ( ) write ( ) recv ( ) send ( )	recvfrom ( ) sendto ( )
Both	terminate	close ( ) shutdown ( )	close ( ) shutdown ( )

Table 4.1 A Summary of Stream Sockets and Datagram Sockets

4.3.5 DCP Protocol Layer

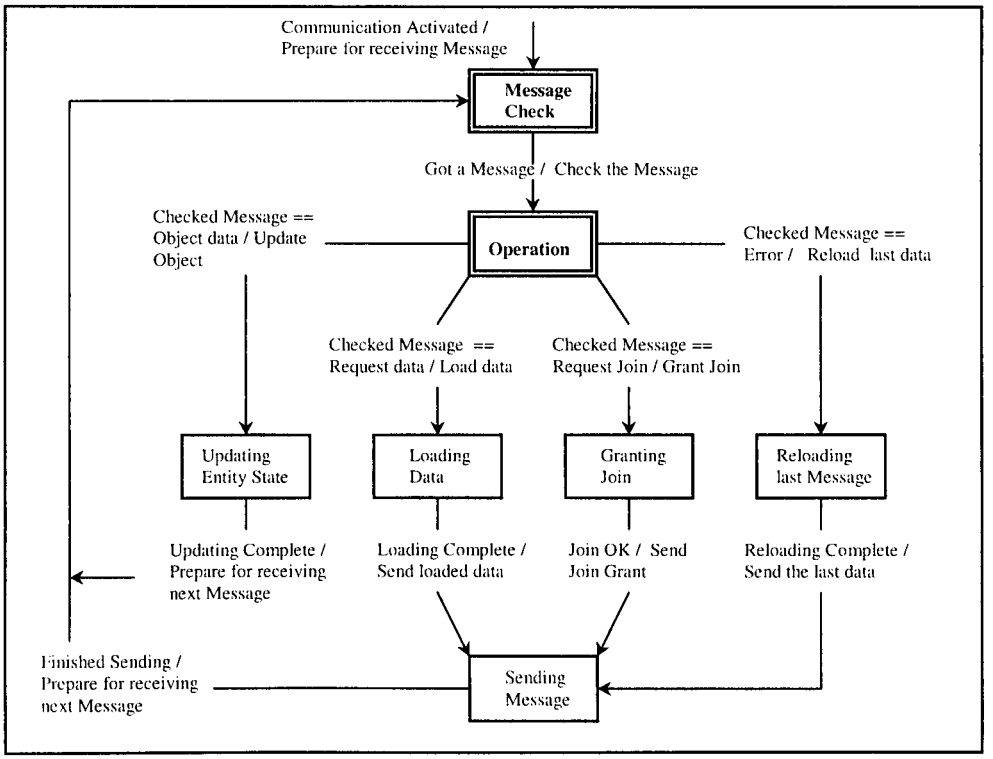
The DCP Protocol Layer has nothing to do with the networking protocols provided by the host system. The important functions of this layer are to establish and break a connection to a server,

and to join and leave a server. These functions are interfaced to the NM via the process interface layer. DCP also relates to the way clients communicate with each other. The DCP Protocol layer employs a Client / Server model, which contains Client DCP and Server DCP components. The client DCP provides services to enable a client to connect to the remote server node, and then to interact with the Server DCP.

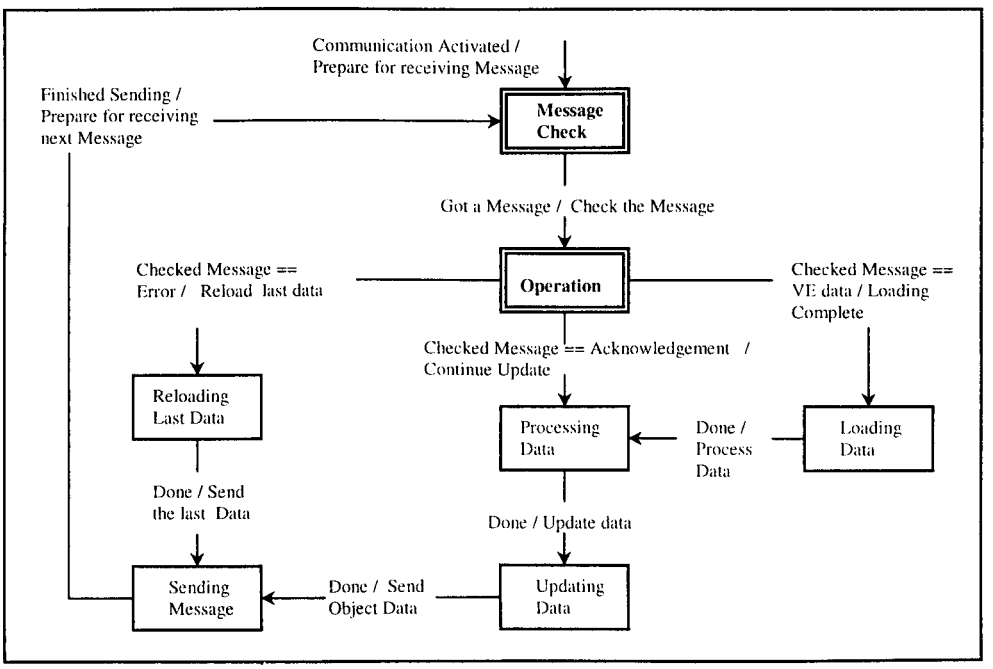
4.3.5.1 DCP Protocol Specification

In general, it is very complex to define the operation of a protocol allowing for all the possible events and error conditions that can arise. Protocols are usually specified using one of a number of more precise methods and formalisms. A protocol is modelled as a *finite state machine* or *automaton*. The most common method used for specifying protocols is the *state transition diagram* (STD) [Halsall, 92]. The protocol can be in just one of a finite number of defined *states* at any instant. Transitions between states take place as a result of an incoming event. As a result of an incoming event, an associated outgoing event is normally generated.

The STD notation is used to document the time-dependent behaviour of the system. For real-time systems, the STD would usually address the whole system and its changes, and to model the inside of all control bubbles. Figure 4.6 shows the *State Transition Diagrams* for communication between server and client in the DVM system. These state diagrams illustrate the protocol using pseudo-code to describe what actions are taken in each state. All possible states are shown in the diagram together with the reasons for changes of state. The STDs show the observable states in which the system resides subsequent to an event and whilst waiting for the next event. These STDs show two major loops that are of importance to both client and server and are part of the STD representation at each node. The loops are the message-check loop and the operation loop. After starting up, each node enters the message-check loop, and also enters the operation loop. The message-check loop and the operation loop run in parallel.



(a) Part of the STD representation at a server node



(b) Part of the STD representation at a client node

Figure 4.6 The State-Transition Diagram for a DVM system

The message-check loop checks all messages that arrive from other nodes, and transfers the received messages to the operation loop. The operation loop performs actual VE operations, including controlling, rendering, etc. These STDs consist of three major message management groups. These message groups contain *a Joining/Detach group, an Initial Data Transmission group,* and *an Entity State Updating group.* More detailed explanation about these message groups can be found in Section 4.3.5.2.

The DVM communication protocol proposed in these STDs is called the *Communication Protocol for a DVM system* (DCP). The DCP is an application layer protocol to support DVM systems on local area networks as well as on the Internet. The DCP is a protocol that permits integration between nodes in the DVM system. It describes the format used to send and receive messages between nodes. The DCP is mainly used to exchange the visual data. It is a protocol definition with associated guidelines and specifies how the implementation is structured. The DCP is used for simulating large scale distributed virtual environments, and will allow a DVM system to run over wide area networks.

4.3.5.1.1 Joining / Detach

In DVM environments, every client can join any server at any time, therefore joining a server is a fundamental requirement of the clients. The join operation is started at the request of the clients. The server should be running before the clients start. Each client has an initial list of servers. Whenever a client starts up it chooses a server from the initial list of servers, then sends a request message to join. When a server starts, it does not have a complete list of clients. When a client joins to the server, the client is added to the list of clients. When a client releases from the server, the client is removed from the list of clients. The list of clients is updated at each server. When a client attempts to join a server, it will generally specify the server to join by

name, rather than Internet address. The client has to translate that name to an address before it can do anything.

The server decides how many clients are currently connected and how many clients are allowed as a maximum. The server checks the request, and permits a join as long as the permission to the client is granted, and the limit to the number of joining clients is not exceeded. If the number of clients is already over the limit, the new client cannot join the server. Each server also maintains a list of currently running servers in the DVM, allowing another server to replicate the list of servers. Whenever a server is added to the list or removed from it, the list gets updated at each other server, so that every server knows the state of all other servers at any time.

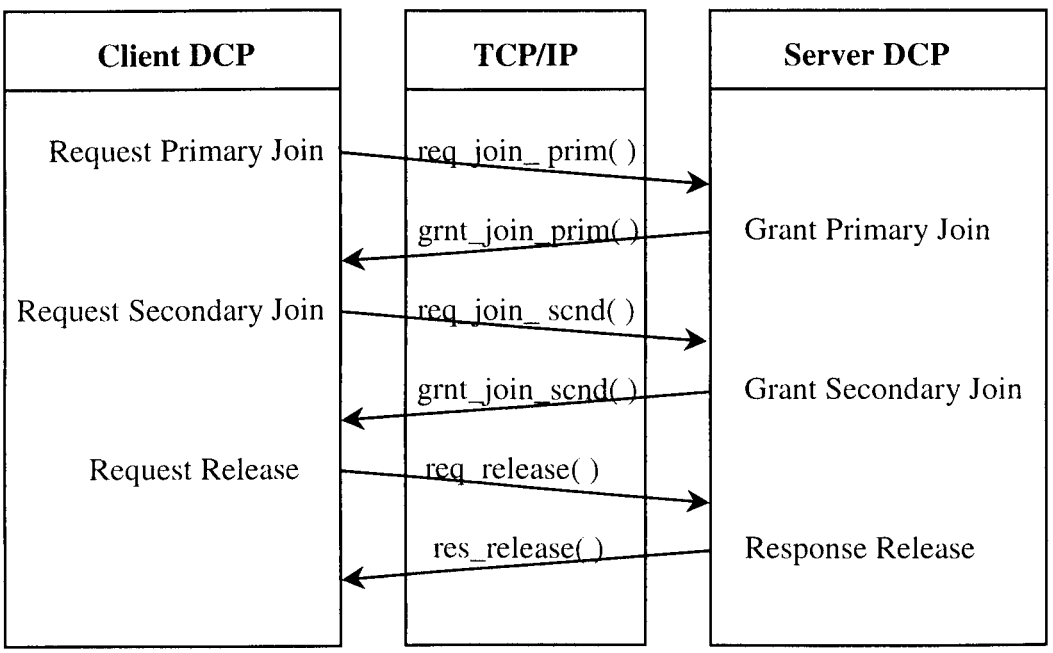


Figure 4.7 Message Exchange Sequence for Joining / Detach

Figure 4.7 shows the message exchange sequence for Joining/Detach using pseudo-code. When a client starts, it tries an initial join with one of the servers. It is called the " primary join". To start a primary join operation, a client sends a req_join_prim ( ) message to the server.

For example, the client might display the following message (This message is used only as an aid to understanding; it does not need to be displayed if not required) :

"Requested a Primary Join to the Server : *Primary_Server_Name*"

When the server has successfully received the *req_join_prim ( )* message, the server responds with a *grnt_join_prim ( )* message to grant the primary join to the client and acknowledge the receipt of the *req_join_prim ( )* message. This is the acknowledgement for the client that the joining request was accepted from the server.

For example, the server might reply with one of the following messages:

"Join accepted"

"You're rejected from this server"

"You sent an invalid request"

A client needs to wait until it receives a message from the server allocating a join for the client. After a successful join, the server will send the client an acknowledgement message. If after a certain amount of time no reply has been received from a server, the client retransmits a join message to the node.

As described in Section 3.3.3, if the FOV of a client moves into the shared zone, the client tries to prefetch the data of next VE zone from an adjacent server sharing the shared zone. To prefetch the next VE data, the client tries to join the adjacent server. This is called the "secondary join". In this case, the client sends a req_join_scnd ( ) message to the adjacent server.

For example, the client could proceed like this:

"Requested Secondary Join to the Server"

The server sends a *grnt _join_scnd ( )* message to grant the secondary join. The server will pass the next server name to the client.

For example, the server might then reply with one of the following messages:

"Secondary Join accepted. The Secondary Server Name : *Secondary_Server_Name*"

"You're rejected from the secondary server"

"You sent an invalid request"

Finally, if a client wants to leave the server, it simply sends a *req_release ( )* message to the server.

For example, the client might display the following message:

"Requested a Release from the Server : *Primary_Server_Name*"

The server simply responds with a *res_release ( )* message to the client as well.

For example, the server might simply reply with the following message:

"Releasing You"

A server can respond to requests from multiple clients concurrently. After receiving a message from clients, if messages arrive approximately at the same time, the conflicts can be resolved by comparing the clients' node addresses. That is, the higher address will be served first.

4.3.5.1.2 Initial Data Transmission

Once a client has joined a server, the initial data transmission phase is started. Figure 4.8 shows the message exchange sequence for the loading of initial data using pseudo-code. First, the client sends a *req_data_init ( )* message to the server to request the initial data for the local VE area.

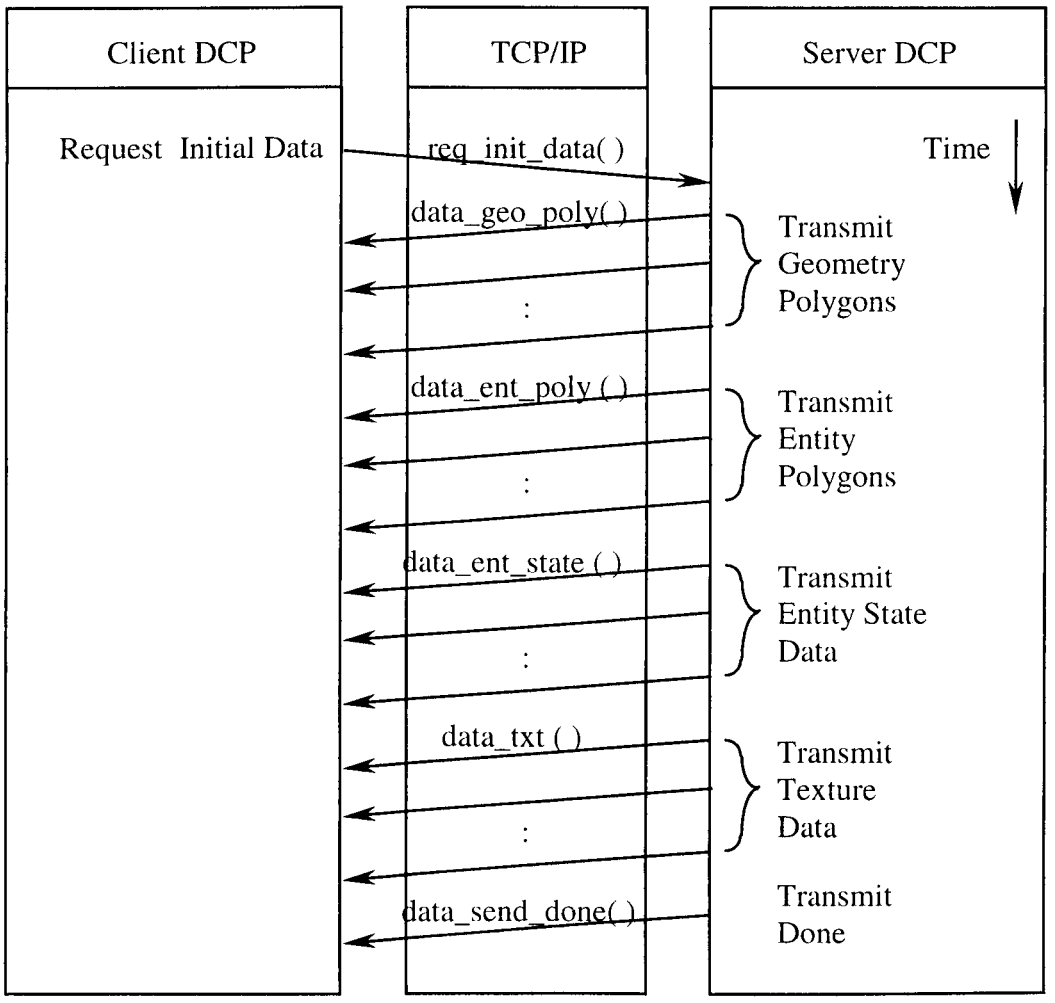


Figure 4.8 Message Exchange Sequence for transmission of Initial Data

For example, the client might display the following message:

"Requested Initial Data to the Server : *Local_VE_Area*"

On receiving this, the server transmits the initial data to the client in sequence. Messages are sent by a *nonblocking* send operation. UDP is used for transmitting of the initial data, except the texture data.

For example, the client might display the following messages:

"Receiving GPD ....."

"Got all of GPD"

"Receiving EPD ....."

"Got all of EPD"

"Receiving ESD ....."

"Got all of ESD"

Whenever a client loads the texture data from the server, a TCP connection is used. The client does not need to notify acknowledgement of the received message because this DVM communication is based on the TCP protocol. The client first has to establish a connection with the server. This means an explicit connection in the sense of a client-server connection.

For example, if the connection is successful, the client might display the following messages:

"Trying to connect to the Server"

"Accepted connection to 224.1.1.123"

After the connection has been successfully accepted from the server, the server sends an acknowledgement to the client. The server then starts to send the texture data to the client. Because DVM communication uses a TCP window (see Section 4.4.6) to control congestion, a client node allocates buffer space for receiving data, and a server is allowed to send data without waiting for an acknowledgement response providing that it will fit within the buffer. The flow-control mechanism is used to send data in sequence.

For example, the client might display the following messages:

"Receiving TXD ....."

"Got all of TXD"

The server monitors the transmission of texture data until the connection is broken. After the transmission of the texture data has been completed, the TCP link is disconnected.

For example, the client and the server might display one of the following messages:

"TCP socket closed"

"TCP socket close failed"

Finally, the *data_send_done( )* command is signalled by the server. The server terminates the initial data transmission phase.

For example, the client might display the following message:

"Got all of Initial Data"

4.3.5.1.3 Entity State Updating

The information created at each client can be categorised as entity state information. The entity state information is the current state of every entity that all other clients always need to know about. The client is allowed to inform the server about its actions, and to obtain information from the server about other clients. The client informs the server about events, the server then forwards this information to the other servers who then replicate the events. All this entity state information has to be replicated over the network at all the other nodes.

In a DVM system, latency can increase in some nodes. The time between a user performing an operation and when it is seen and can be reacted to by all users in the system, can make a difference. There are many reasons for this latency, including delays from sending and receiving information on the network, from processing and computation, and from displaying the graphical results to the user. In general, reducing latency is a difficult system engineering problem, as it has so many contributory factors. One popular technique is to use protocols that do not require acknowledgements for network messages. These can run more quickly, but provide no guarantees that the message actually arrived.

A DVM usually uses less reliable protocols to reduce latency. First, the DVM uses UDP to reduce the amount of message passing, to minimise network load, and to increase scalability, although it still has unreliable communication problems because the current UDP communication does not have flow control over data transfers. Second, the DVM uses a multicast scheme in order to guarantee that a packet is simultaneously received at every node in the group.

Figure 4.9 shows the message exchange sequence for updating of the entity state between nodes. When a client updates the state of an entity, it multicasts an *entity_state( )* message to

update the entity state to the server and client nodes within the same local VE group, and the nodes update the corresponding entity (e.g., aircraft).

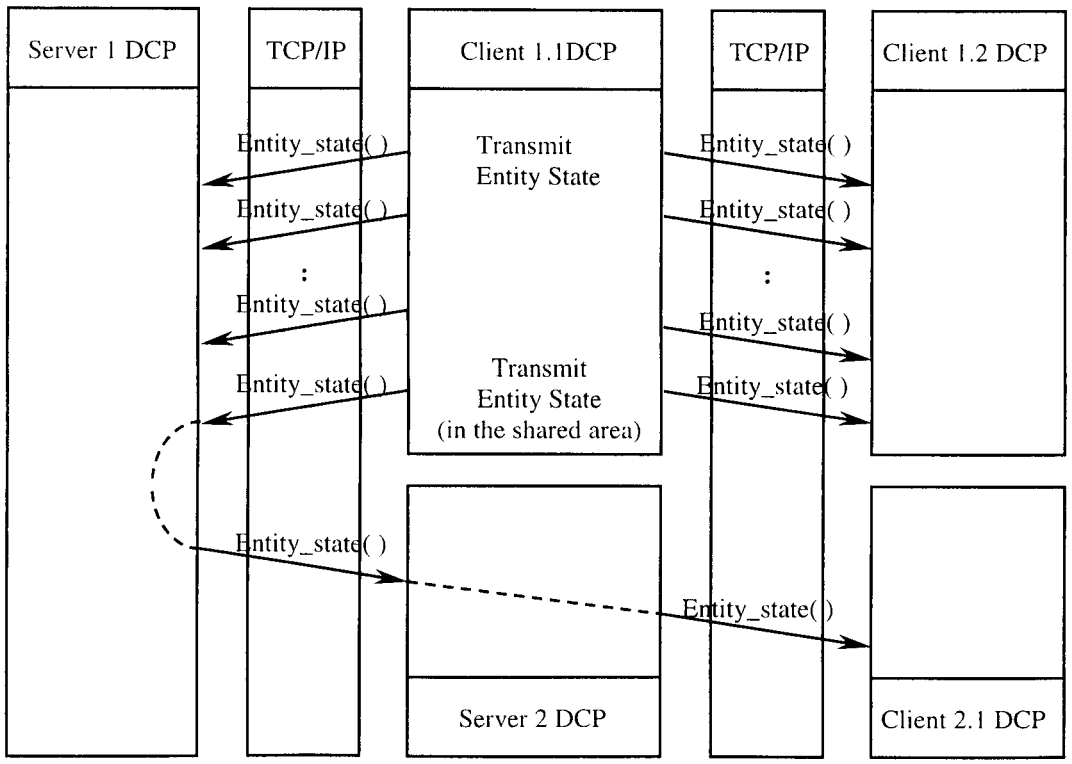


Figure 4.9 Message Exchange Sequence for Updating of Entity State

When an *entity_state( )* message is received, as described above, the clients and the server don't send back an acknowledgement message to the sender. When an entity state message is lost or undelivered, the state of the entity will be recovered quickly because the state of entities usually changes continuously.

The server keeps track of the entities in the local VE. As described in Section 3.3.6, for example, if an entity moves into the area that is shared between Server-1 and Server-2, a client

transmits an *entity_state( )* message to Server-1 in the same local VE group, and Server-1 retransmits the *entity_state( )* message to Server-2 that is sharing the entity. On receiving this, Server-2 transmits the *entity_state( )* message to its own clients. After the entity moves out of the shared area, and the two servers no longer share the entity, then no unnecessary packets are sent.

4.3.5.2 DCP Protocol Units

The format of the messages exchanged between the two nodes - DCP protocol units - must be in an agreed syntax to ensure they have same meaning in both computers. Within the DVM system, communication between nodes is for two purposes. The first of these is control, which can be defined as transfer of details about the sender's state between a sender and another node. Control messages are not generally visible to the users of the DVM system. The second is data itself. The data is received, processed, stored and distributed to and from nodes in the DVM system.

As can be seen in Table 4.2, the Protocol Units of DCP involve three message groups for *Joining/Detach*, *Initial Data Transmission*, and *Entity State Updating*. The *Joining/Detach* message group is used to establish or release the initial connection between server and client node, or between server and server node. The *Initial Data Transmission* group is used to transmit the initial data from the server node to client node. The *Entity State Updating* group manages the exchange of entity information between server and client nodes, or between server and server node. The direction of each message is distinguished by whether the client or server node sends the message. Each message involves the combination of information, including operation code, operation time, etc. To achieve operation of these groups, the DCP packet types (describe in section 4.3.5.3.2) are used.

Management	Message	Direction	Pseudo-code
Joining/Detach	Request_Primary_Join	C→S, S→S	req_join_prim( )
	Request_Secondary_Join	C→S	req_join_scnd( )
	Request_Release	C→S	req_release( )
	ResponseRelease	S→C	res_release( )
	Grant_Join	S→C, S→S	grnt_join_prim( )
		S→C	grnt_join_scnd( )
Initial Data Transmission	Request_Data	C→S	req_data_init( )
	Transmit_Data	S→C	data_geo_poly( )
			data_ent_poly( )
			data_ent_state( )
			data_txt ( )
			data_send_done( )
Entity State Updating	Update_Entity_State	C→S, S→S	entity_state( )
		C→C	

Table 4.2 Protocol Units

4.3.5.3 DCP Packets

The DCP is designed to be implemented on top of TCP and UDP. Since TCP and UDP are underpinned by IP, packets will have an IP header, a TCP or UDP header, and a DCP header. Additionally, the packets will have a header (eg. an Ethernet header) to allow them through the local transport medium. The UDP header and the TCP header both contain the source port number and the destination port number. The TCP ports are independent of the UDP ports, since the IP header specifies the protocol.

As shown in Figure 4.10, the order of the contents of a packet will be : local medium header, IP header, TCP/UDP header, DCP header, followed by the remainder of the DCP packet. The DCP uses the source and destination port fields of the IP header and the length field reflects the size of the DCP packet. The DCP header consists of a 2 byte opcode field that indicates the packet's type (e.g., ERROR, DATA etc.). These opcodes and the formats of the various types of packets are discussed further in the section 4.3.5.3.2 on DCP packets.

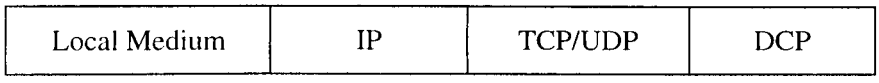
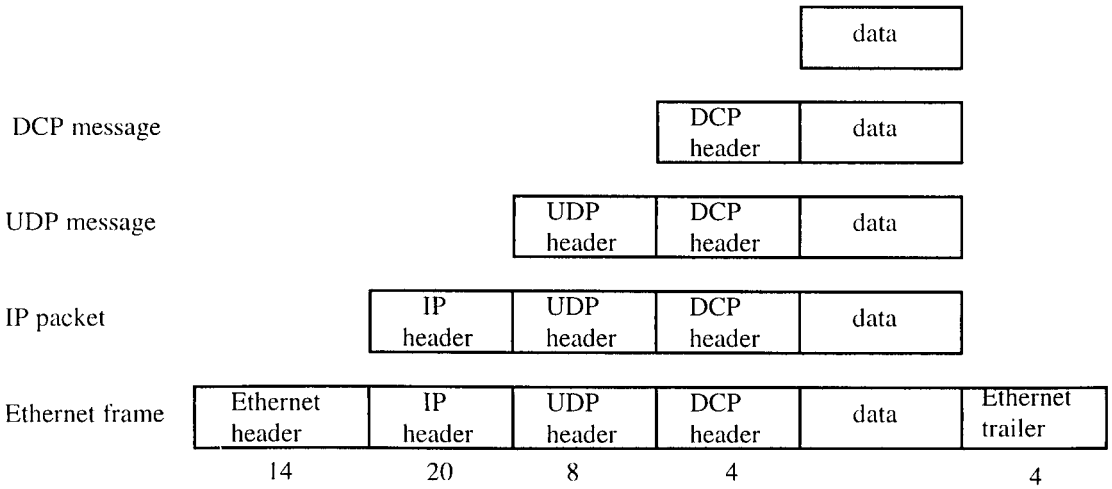


Figure 4.10 Order of Headers

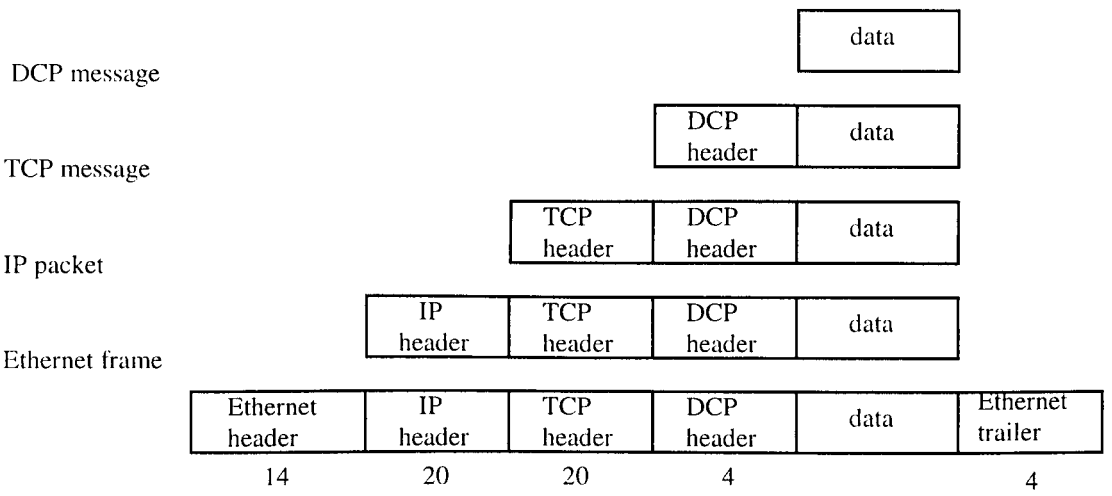
4.3.5.3.1 Data Encapsulation

The addition of control information to data is called *encapsulation*. When we assume the DCP applications, using TCP/IP or UDP/IP protocols, between two systems that are connected with an Ethernet, Figure 4.11 shows the encapsulation by each layer. If there are 100 bytes for the DCP client process to transfer to the DCP server process, the DCP client process adds 4 bytes of control information to the beginning of the data buffer, before passing the data down one layer to the UDP layer. The UDP layer does not interpret the 4-byte DCP header at all. The task of the UDP layer is to transfer 104 bytes of data to the other UDP layer. The UDP layer then adds its own 8-byte header and passes the 112 byte buffer to the IP layer. The IP layer adds its 20 byte header and passes the 132 byte buffer to the data link layer for the Ethernet. At this layer a 14 byte header and a 4 byte trailer are added to the buffer of information. The final diagram of Figure 4.11 (a) shows the Ethernet frame that is physically transmitted across the Ethernet, along with the sizes of each of the headers and trailers, in bytes. We call the unit of interchange

at the network layer *packets*. At the data-link layer we call them *frames*, and at the lowest layer, the physical layer, *bits* are exchanged.



(a) over UDP/IP on an Ethernet



(b) over TCP/IP on an Ethernet

Figure 4.11 Encapsulation of DCP data

4.3.5.3.2 DCP Packet Types

DCP supports twelve types of packets. The DCP header of a packet contains the opcode associated with that packet. The types of packets are as follows:

Opcode	*Operation*
1	Primary Join Request (PJR)
2	Primary Join Grant (PJG)
3	Secondary Join Request (SJR)
4	Secondary Join Grant (SJG)
5	Release Request (RRQ)
6	Release Response (RRS)
7	Data Request (DRQ)
8	Data Response (DRS)
9	Geometry Polygon Data (GPD)
10	Entity Polygon Data (EPD)
11	Entity State Data (ESD)
12	Texture Data (TXD)

A client node can join one of servers in a DVM system at any time. Whenever a client node wants to join a server, it first has to transmit a PJR packet to a server node to get a join permission. PJR packets have the format shown in Figure 4.12. PJR packets (opcode 1) contains Client_Identification (*Client_ID*) field. Every client has a unique *Client_ID*. The *Client ID* is a sequence of bytes in *netascii* terminated by a zero byte. A node that receives netascii mode data must translate the data to its own format.

2 bytes	string	1 byte
Opcode	Client_ID	0

Figure 4.12 PJR packet

When a server node receives a PJR packet from a client node, it transmits a PJG packet to a client node as long as the permission to the client is not denied. PJG packets have the format shown in Figure 4.13. PJG packets (opcode 2) contains Operation_Time (*Optime*) field.

2 bytes	8 bytes
Opcode	Optime

Figure 4.13 PJG packet

As described in Section 3.3.3, if a client node wants to access an adjacent VE area that another server maintains, then it transmits a SJR packet to the current server node to get the adjacent server's address. SJR packets have the format shown in Figure 4.14. SJR packets (opcode 3) contains the Current_Area_Number (*Curr_Area_#*) and Direction (*DIR*) fields.

2 bytes	8 bytes	3 bytes	2 bytes
Opcode	Optime	Curr_Area _#	DIR

Figure 4.14 SJR packet

When a server node receives a SJR packet from a client node, it transmits a SJG packet to a client node to inform the adjacent server's name. SJG packets have the format shown in Figure 4.15. SJG packets (opcode 4) contains Next_Server_Name (*Nxt_Sv_Name*) field.

2 bytes	8 bytes	string
Opcode	Optime	Nxt_Sv_Name

Figure 4.15 SJG packet

A client node can leave from a server in a DVM system at any time. Whenever a client node wants to release from the server, it transmits a RRQ packet to a server node. RRQ packets have the format shown in Figure 4.16. RRQ packets (opcode 5) contains just *client_ID* field.

2 bytes	8 bytes	string
Opcode	Optime	Client_ID

Figure 4.16 RRQ packet

When a server node receives a RRQ packet from a client node, it transmits a RRS packet to a client node to inform the acknowledgement of RRQ. When a client receives the RRS, it leaves from the server and logs out of the corresponding DVM. RRS packets have the format shown in Figure 4.17. RRS messages (opcode 6) contains just *server_ID* field.

2 bytes	8 bytes	string
Opcode	Optime	Server_ID

Figure 4.17 RRS packet

Whenever a client node wants to load some local VE data from a server, it has to transmit a DRQ packet to a server node. DRQ packets have the format shown in Figure 4.18. DRQ messages (opcode 7) contains just *Data_Type* (*Dtype*) field.

2 bytes	8 bytes	2 bytes
Opcode	Optime	Dtype

Figure 4.18 DRQ packet

When a client node completely receives the requested VE data from a server, it transmits a DRS packet to a server node. DRS packets have the format shown in Figure 4.19. The DRS packet (opcode = 8) has a data_number (*Data_#*) field.

2 bytes	8 bytes	2 bytes	8 bytes
Opcode	Optime	Data_#	MsgOptime

Figure 4.19 DRS packet

When a server receives a DRQ packet from a client, it transmits the local VE data to the client node. All transmitted packets are not acknowledged by the client. The local VE data includes geometry polygon data, entity polygon data, entity state data, and texture data.

Geometry polygon data is transferred in GPD packets depicted in Figure 4.20. Each GPD packet (opcode = 9) has an *Area_Number* (*Area_#*), *Polygon_Number* (*Poly_#*), *Offset*, and *Vertex* fields.

2 bytes	8 bytes	2 bytes	2 bytes	2 bytes	2 bytes		2 bytes
Opcode	Optime	Area_#	Offset_X	Offset_Y	Poly_#[*0*]	..	Poly_ #[*n-1*]

2 bytes		2 bytes	2 bytes		2 bytes		2 bytes	
Txt_#[*0*]	..	Txt _#[*n-1*]	Vt_X[*0*][*0*]	..	Vt_X[*n-1*][*0*]	..	Vt_X[*0*][*3*]	..

2 bytes	2 bytes		2 bytes		2 bytes		2 bytes
Vt_X[*n-1*][*3*]	Vt_Y[*0*][*0*]	..	Vt_Y[*n-1*][*0*]	..	Vt_Y[*0*][*3*]	..	Vt_Y[*n-1*][*3*]

2 bytes		2 bytes		2 bytes		2 bytes
Vt_Z[0][0]	..	Vt_Z[n-1][0]	..	Vt_Z[0][3]	..	Vt_Z[n-1][3]

Figure 4.20 GPD packet

Entity polygon data is transferred in EPD packets depicted in Figure 4.21. The EPD packet (opcode = 10) has *Polygon_Number* (*Poly_#*), *Patch_Number* (*Patch_#*), *Vertex* (*Vt*), *Normal* (*Nor*), and *Colour* (*Col*) fields.

2 bytes	8 bytes	2 bytes		2 bytes	2 bytes		2 bytes
Opcode	Optime	Poly_#[0]	..	Poly_#[n-1]	Patch_#[0]	..	Patch_#[n-1]

2 bytes		2 bytes	2 bytes		2 bytes	2 bytes		2 bytes
Vt[0][0]	..	Vt[0][2]	Vt[n-1][0]	..	Vt[n-1][2]	Nor[0][0]	..	Nor[0][2]

	2 bytes		2 bytes	2 bytes		2 bytes
..	Nor[n-1][0]	..	Nor[n-1][2]	Col [0]	..	Col [2]

Figure 4.21 EPD packet

Entity state data is transferred in ESD packets depicted in Figure 4.22. The ESD packet (opcode = 11) has Data_Type (*Dtype*), Area_Number (*Area_#*), Entity_Type (*Otype*), Entity_ID (*Obj_ID*), Offset, Position (*POS*), Direction (*DIR*), Speed (*SP*) and Dead-Reckoning_flag (*Dflag*) fields.

2 bytes	8 bytes	2 bytes	4 bytes	4 bytes	8 bytes	8 bytes
Opcode	Optime	Dtype	Area_ #	Etype	Ent_ID	Offset_X

8 bytes	8 bytes	8 bytes	8 bytes	6 bytes	8 bytes	2bytes
Offset_Y	POS_X	POS_Y	POS_Z	DIR	SP	Dflag

Figure 4.22 ESD packet

Texture data is transferred in TXD packets depicted in Figure 4.23. The TXD packet (opcode = 12) has Texture Number (*Txt_#*) and Imagefile (*ImgF*) fields.

2 bytes	8 bytes	2 bytes	n bytes
Opcode	Optime	Txt_#	ImgF

Figure 4.23 TXD packet

The format of DCP packets are shown in detail in Appendix A.

4.3.6 Process Interface Layer

When developing a DVM application there is the problem of how the actual VE processes, including client and server, interface to the network. In DVM this is done through the process interface layer. This interface mostly is a collection of functions, each corresponding to a specific event. The actual VE process interfaces to the DCP protocol layer via the process interface layer. The process interface layer couples the actual VE process with all functions that are related to the networking.

Every time an event occurs inside the actual VE process, the VE process calls the corresponding function to inform the networking process. If the actual VE process causes an event, for instance, it calls the corresponding function of the networking process. The networking process is also invoked to establish or break a connection and to join and leave a DVM. Everything else the DCP protocol layer provides is exclusively used by the process interface layer.

4.4 Major Techniques for Networking of a DVM

There are many important techniques to be considered to improve the communication efficiency when implementing a networked for a DVM system. These techniques include communication schemes, message communication methods, data portability, data compression, dead-reckoning algorithm, and so on. This section describes these major techniques.

4.4.1 Communication Schemes of the DVM

There are two communication schemes that have been used in the DVM (Figure 4.24). These communication schemes are *unicast* and *multicast*. A DVM system usually employs one form of these communication schemes. This section describes these two methods.

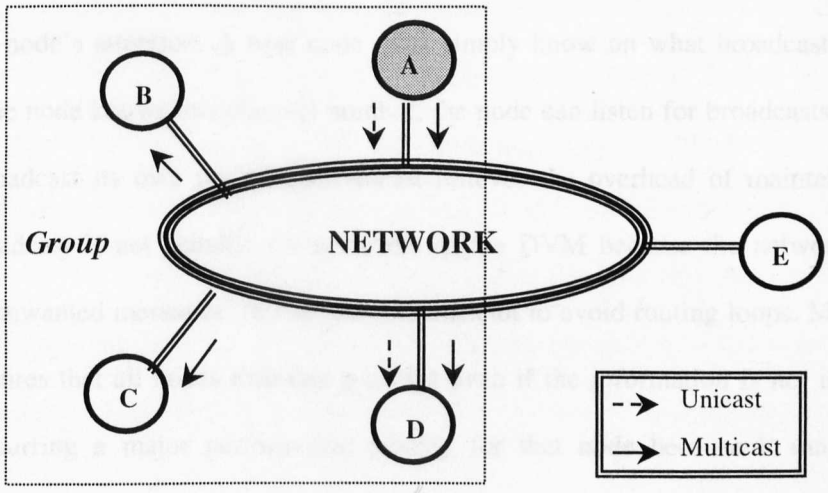


Figure 4.24 Communication Schemes

4.4.1.1 Unicast

Unicast establishes communication between two nodes. Unicast requires the establishment of a connection or path from each node to every other node in the network. A link must be established between every node to communicate with each other. If there are n nodes in a DVE, this will require links for a total of $n(n-1)$ virtual connections in a group. If one node makes a change, that node must be connected to $n-1$ other nodes and send $n-1$ messages. If each node updates its individual entity's position before rendering and informs every node, this results in $n(n-1)$ messages per rendering frame. Unicast can place a burden on each node that can quickly become a performance bottleneck. So in a real-time environment, unicast communications between nodes should be avoided.

4.4.1.2 Multicast

Broadcast allows a node to send data to every node. Each node sends data with an address that catches every node's attention. A new node must simply know on what broadcast channel to listen. Once the node knows this channel number, the node can listen for broadcasts from other nodes and broadcast its own updates. Broadcast relieves the overhead of maintaining links. However, broadcast is not suitable for networks of the DVM because the network becomes flooded with unwanted messages' traffic and it is difficult to avoid routing loops. Moreover, IP broadcast requires that all nodes examine a packet even if the information is not intended for that node, incurring a major performance penalty for that node because it must interrupt operations in order to perform this task at the operating system level.

DVM, therefore, uses multicast. Multicast reduces the load on nodes and is an alternative to broadcast. Multicast is simply a subset of broadcast. Multicast provides one-to-many delivery services for applications that communicate with several other nodes simultaneously. Multicast

protocols add a group ID so that several multicast streams can run concurrently [Gossweiler 94]. A node should be a member of one group at a time. The node will join and leave groups at any time. There is no restriction on the location or number of nodes in a group.

Multicast allows a group to communicate on a network via a single transmission by the source. The nodes in a group can communicate with each other, and only the nodes in the group receive the message, rather than every node on the network. Multicast protocols permit moderately large real-time bandwidths to be efficiently shared by an unlimited number of nodes. Multicast provides a way of overcoming these disadvantages whilst still retaining the low transmission overhead, but it is not widely available and is, like broadcast, unreliable. Using multicast for updates reduces aggregate message traffic. However, if the systems are to be geographically dispersed, then high-speed multicast is required.

If a client is connected to a server, it has to send all packets to the server. The server receives the packets, and decides which information should be forwarded to which of the other clients. This method greatly reduces the necessary bandwidth requirement for the client's connection to the network. However, the crucial thing about this approach is latency. If the latency is not extremely low the user will experience a perceptible delay between entity action on the source client and entity action on target clients. The user will perceive considerable lag if the latency is too high. In the DVM, therefore, communication to update the state of entities between nodes uses a peer-to-peer approach. Packets that transmit the current entity state to the nodes are sent in a multicast fashion. Theoretically, it would be ideal to use multicasting for this purpose.

4.4.2 Message Communication between nodes

As DVE systems are potentially distributed on different nodes, all communication between nodes is restricted to message communication. The DVM system uses two message

communication methods : *Loosely-coupled Message Communication* and *Tightly-coupled Message Communication*. This section describes the message communication between different nodes.

4.4.2.1 Loosely-coupled Message Communication

With loosely-coupled message communication, the source node sends a message to the destination node and does not wait for a reply. The two nodes proceed asynchronously and a message queue will build up between them. If the source node expects to receive a positive or negative acknowledgement, indicating whether the message has safely arrived at its destination or not, an additional effort is required. It is up to the source node to decide how to handle a failure in message transmission.

4.4.2.2 Tightly-coupled Message Communication

The source node sends a message to the destination node and then waits for a reply from the destination. Tightly-coupled message communication should only be used when a response is required. In the typical client/server situation, several clients request services from a server by sending messages to it. In this case, a message queue can build up at the server. The client can use tightly-coupled message communication and wait for a response from the server.

4.4.2.3 Summary of the Message Communication

Whether the client uses loosely-coupled or tightly-coupled message communication with the server is application dependent and does not affect the design of the server. It is possible for a server to have some of its clients communicating with it using tightly-coupled message communication while others use loosely-coupled message communication. When the client

waits for message from other client nodes in addition to the response from the server, it can use loosely-coupled message communication.

4.4.3 Data Portability

No data conversion need occur when a DVM application is based on a homogeneous system, where all operations run in the same environment. However, one of the goals of a DVM is to support distributed processing across heterogeneous environments. Communication in a heterogeneous environment will require data conversions. There are two kinds of data conversion : *Type conversion* and *Representation conversion*. Type conversion changes the data type of a value, for example, by rounding a REAL to an INTEGER. Representation conversion changes the binary representation of a value such as network byte ordering. DVM communications do not deal with type conversion. On the other hand, DVM requires that a representation conversion be performed when a typed value is transferred across environments that use different representations for such a value.

4.4.3.1 Data Representation

On networks, many data items transmitted between computers require more than a single byte of storage. However, unfortunately, some computers store the bytes that comprise a multibyte value in different orders. For example, 16-bit integers usually require two bytes. There are two ways to store this value. As can be seen in Figure 4.25, when the higher-order byte is stored at the lower address, it is referred to as big-endian (the most significant byte is stored first). Some machines store the lower-order byte first. This representation is referred to as little-endian (the least significant byte is stored first). To communicate between two machines which uses different byte-order, two communicating machines must agree on data representation.

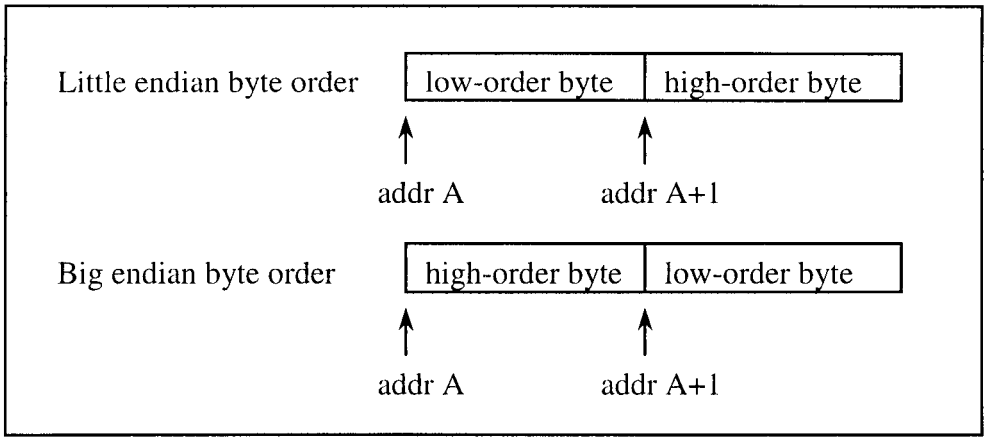


Figure 4.25 Byte Orders for a 16-bit quantity

4.4.3.2 Network Byte Ordering

There are potential byte order differences between different computer architectures and different network protocols. For a DVM system, therefore, a solution to this problem is for a network protocol to specify its network byte order. The TCP/IP protocol uses the big endian format for the 16-bit and 32-bit integers that it maintains in the protocol headers. The TCP/IP protocol maintains only integer fields, as the differences in the internal formats for floating point data are even worse. A DVM communication uses a big endian format because it is based on the TCP/IP protocol. The TCP/IP protocol only specifies the format for the fields that it maintains, it has no control over the format of the data that the DVM applications transfer across the network. To handle byte ordering for non-standard size integers, there are four functions that have been designed for the Internet protocols. The figure 4.26 shows the conversions done by these functions.

htonl	convert host-to-network, long integer
htons	convert host-to-network, short integer
ntohl	convert network-to-host, long integer
ntohs	convert network-to-host, short integer

Figure 4.26 Byte ordering functions

4.4.4 Data Compression

For a DVM system, compression of data can be used to save bandwidth. For example, the following cases 1 and 2 show some examples of JPEG data compression. The JPEG can typically achieve a high ratio of 10 : 1 to 20 : 1 without visible loss for compressing photographic images/textures. The first case shows an example where a typical image file is compressed by a ratio of 10 : 1, from a file of over 31 megabits down to a file of about 3 megabits. The second case shows that an image file is compressed by a ratio of 10 : 1 from, a file of slightly under 2 megabits down to a file of about 0.2 megabits. Table 4.3 shows the transmission time comparison at various bandwidths with and without compression. Large savings in network bandwidth can be gained by compressing the geometry data before sending it through the network.

Case 1 **:**

- 1280 × 1024 pixels at 24-bits/pixel = 31.45 Mbits
- Compression ratio 10:1 31.45 Mbits / 10 = 3.145 Mbits

Case 2 **:**

- 320 × 256 pixels at 24-bits/pixel = 1.97 Mbits
- Compression ratio 10:1 1.97 Mbits / 10 = 0.197 Mbits

Image File Size (JPEG)	Tel. Line 4.8 Kbps	Switched 56 Kbps	T – 1 [1] 1.544 Mbps	T – 3 [2] 45 Mbps
3.145 Mbits	655 sec	56 sec	2.04 sec	0.07 sec
0.197 Mbits	41.04 sec	3.52 sec	0.13 sec	0.004 sec

Table 4.3 JPEG Image File size and Transmission Time

Because data compression / decompression is actually independent and outside of this research, a DVM communication does not specify the detailed rules for data compression. However, a data compression techniques are a major factor that is needed to design a real-time DVM system for transmitting data among nodes. Without data compression and decompression, a DVM system will operate much more slowly and would require substantially higher bandwidth as well as more storage capacity. For a DVM system, therefore, the compression / decompression operation can be handled by software as well as hardware. Although the hardware method is more expensive, it is substantially faster than software-based solutions. However, more recently, the performance of software-based systems has been dramatically improved. Various commercial products (JPEG, ZIP, GZIP, etc..) have been announced. DVM can choose one of these commercial products.

4.4.5 Dead-Reckoning

As already described in Section 4.2.1, in general the biggest limitation of networks is the limited bandwidth. Bandwidth can be saved when transmitting state information by using dead-

[1] A network facility capable of transmitting information at the rate of 1.544 Megabits per second (nominally 1.5 Megabits per second). Also known as DS1.
[2] A network facility capable of transmitting information at the rate of 44.736 Megabits per second (nominally 45 Megabits per second). Also known as DS3.

reckoning. In general, dead-reckoning describes an approach to state transmission where prediction of an entity's actions is used to cut down on the amount of information that needs to be sent. The dead-reckoning algorithm greatly reduces network traffic, allowing more nodes within the limits set by network bandwidth. A DVM, therefore, uses dead-reckoning algorithms to reduce network traffic, although the method can cause problems if updates are sent too infrequently.

In a DVM, a user has to do something in order to create a change of state in an entity at a client. This could be something like pressing the button of a joystick or mouse to move the entity. If a user does not do anything, there will be no events. The state will normally change every single frame if an entity is moving or rotating at all. The state is only omitted from sending if an entity has not changed at all. However, if an entity moves along the same path at a previously established velocity, instead of just sending an entity's location, a source client sends a message that contains the entity's location, orientation, a timestamp, a velocity vector, and a dead-reckoning flag. The other target clients in the network can extrapolate the entity's location without additional updates. The other target clients will already know where the entity is because they have the entity's old position and current velocity. Updates are only sent when needed and the amount of traffic is greatly reduced.

A DVM uses a simple error-based approach to dead-reckoning thresholds. Figure 4.27 shows the error-based dead reckoning diagram. In error-based dead reckoning, if the difference between the entity's computed position and the actual position exceeds a predetermined error threshold range, then the source client informs the other target clients to update the entity with the new position and velocity. Normally, only the target clients that receive the entity state update perform the prediction algorithm, in order to determine whether the deviation of the predicted position from the actual position is high enough to justify sending another state update. If a target client's entity strays only slightly from the dead-reckoning path, then the target client will decide that the difference is negligible, and does not send messages to the other

target clients. The target client only needs to send a message out to the other clients when the change is significant. When a target client receives a message to update the dead reckoned entity, the target client uses a snap back method so that the location of the entity is corrected immediately to reduce computational loads, although it results in the entities apparently jumping.

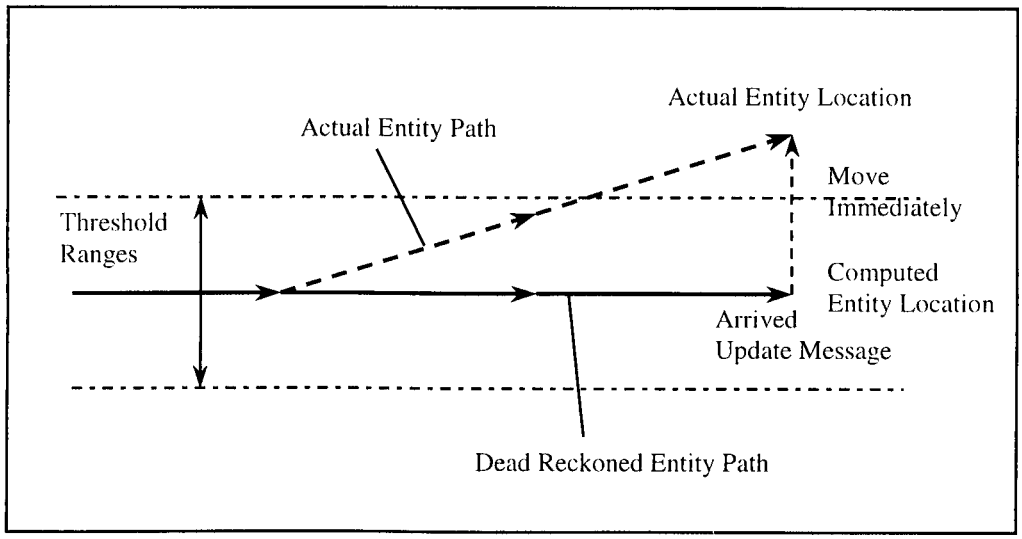


Figure 4.27 Error Based Dead Reckoning

4.4.6 TCP windows

As described in Section 4.3.3, in a DVM system, UDP is usually used, a TCP connection is only used to transmit large amounts of data like textures. When a TCP connection is used, sometimes the network connection between the nodes will have a bottleneck, limiting the rate of data transfer. Unless the bottleneck is the transmission rate of the sender, then if transmitting of data occurs too fast, it will result in congestion at the receiver. To throttle the transmission speed

down to a level where congestion does not occur, a TCP window is used. The primary reason for the TCP window is to control congestion.

A TCP window means that a sender can send a limited amount of data on a particular connection until it gets an acknowledgement back from the receiver that it is ready for more. For example, as can be seen in Figure 4.28, if a pair of hosts are talking over a TCP connection that has a TCP window size of 64 KB, the sender can only send 64 KB of data and then it must stop and wait for an acknowledgement from the receiver that some or all of the data has been received. If the receiver acknowledges that all the data has been received then the sender is free to send another 64 KB. If the sender gets back an acknowledgement from the receiver that it received the first 32 KB, although the second 32 KB is still in transit, then the sender could only send another 32 KB.

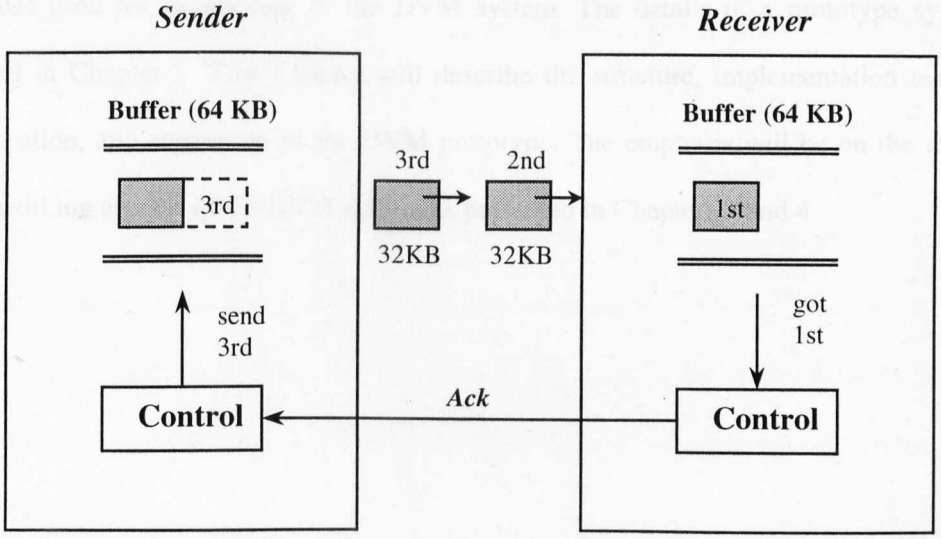


Figure 4.28 A TCP window for Congestion Control

For a DVM system the window size is determined by setting the socket send and receive FIFO buffer sizes. For communication between the nodes, the FIFO buffer sizes should be correctly

set before making the connection. In theory the TCP window size should be set to the product of the available bandwidth of the network and the round trip time of data going over the network [RFC, 98]. A way to determine the round trip time in the DVM is to use a ping from one node to the another and use the response times returned. For example, if a network had a bandwidth of 100 Mbits/s and the round trip time was 10 msec, the TCP window size should be 125 KB ($100 \times 10^6 \times 10 \times 10^{-3}$ bits).

4.5 Summary

This chapter has presented a networking architecture capable of implementing the DVM model described in Chapter 3. The details of the proposed networking architecture include the major goals to be achieved in DVM networking; the basic restrictions and their relationship to DVM communication, the networking subsystem for exchanging messages between nodes, and major techniques used for networking of the DVM system. The details of a prototype system are presented in Chapter 5. This Chapter will describe the structure, implementation techniques, demonstration, and evaluation of the DVM prototype. The emphasis will be on the modelling and networking aspects of the DVM system as presented in Chapters 3 and 4.

CHAPTER 5

A PROTOTYPE DVM SYSTEM

The overall modelling and networking architectures of the DVM system were described in Chapters 3 and 4. This chapter describes a DVM prototype which is used as a test-bed to show how the DVM system, as outlined in the previous chapters, might be implemented. The purpose of the DVM prototype is to test the correctness and validity of the modelling and networking architecture of a DVM system to show that it is efficient, and to describe how DVM networking can be implemented. A full implementation of the DVM design concepts would take a long time. Therefore, not all of the design's elements are fully implemented, though the prototype is sufficient to verify the viability of the ideas used in the proposed solution. Especially, the implementation of the prototype system concentrates on communication between nodes.

This chapter begins by describing the details of the prototype DVM system with a substantial case study. This is followed by an explanation of the fundamental techniques for implementation of the prototype system. Next, the demonstration results of the prototype

system are presented. Finally, the remainder of the chapter is concerned with performance testing.

5.1 Case Study : The Aircraft Manoeuvre Operations

The case study will incorporate many key points of the DVM techniques we have described in previous chapters. This section includes the scenario, implementation requirements, environment, structures, relationships for communication between components and the implementation sequences of the DVM prototype system.

5.1.1 Scenario

Small unit military operations have been chosen as a case study for the DVM prototype system. The case study has implemented the *aircraft manoeuvre operation*. The environment consists of a large 2D ground area. The system for the case study is based on a networked environment, and implements a distributed application where multiple users participate in the aircraft manoeuvre operation. Each user participating in the operation can sit at any of the client nodes, which are connected to each other through the network. Each user can enter and leave the operation at any time.

In this case study, a number of software units run on multiple nodes including the server and the client nodes. Each user's virtual world represents one part of the global VE. Client software is used to generate a virtual world for each user. The virtual world space size is the same across all the users. Each user's local VE is represented by the view scope.

The size of the view scope (FOV) is the same for all the users. If some users' FOV is sharing the same local virtual area, the users can see the virtual world with the same aircraft and terrain. Each user can have a view of all the aircraft that are currently in the same local VE. They are also able to see the same activity of all the aircraft in the FOV. If an aircraft is moved into the shared area of the local VE, its representation in the next local VE will appear. If an aircraft is moved out of the local VE, its representation in the local VE disappears, its representation in the next local VE is still displayed.

This system allows the networked users to control aircrafts. If a user has a permission to control an aircraft, the user can move the aircraft in any direction within the field of view (FOV), and stop at any time. The user can also change the speed and the direction of the aircraft. That is, each user can accelerate, decelerate or rotate the aircraft. When a user changes the state of an aircraft in the FOV, the system sends a message to inform the other users of the new position and velocity.

5.1.2 Implementation Requirements of the Prototype

This section describes the requirements for the implementation of the prototype system. The statement of requirements forms the basis for the design of the prototype. These requirements are as follows.

- It should support the synchronous update of the entity-state between nodes.
- It should support multiple servers and clients.
- It should support GUI-based user interactions.
- Data should be distributed on the multiple servers.
- Multiple users should be able to join or leave the system at any time.

- The system should allow clients access to any server.
- The system should allow users to control any entity.

5.1.3 Implementation Environments

The implementation of a DVM prototype system is undertaken at the network environment level. These environments consist of a number of hardware and software components. This section briefly describes these environments, including hardware, software, and networking components that are required to implement the DVM prototype system.

5.1.3.1 Hardware

A DVM prototype system should be executable on a number of different computers. However, the initial development of a prototype system was planned to implement in a homogeneous environment, namely a network of Silicon Graphics Indy workstations. Currently there are 8 workstations available. Although these systems would not be suitable for the fast implementation of the prototype system, if more specialised hardware is added, much of the graphics capabilities of this system can be improved.

5.1.3.2 Software

Software development of prototype has been done on a UNIX-based system, namely the IRIX Operating System (version 6.2) that uses virtual memory. The prototype system was written using the IRIX Graphics Library (GL) on a Silicon Graphics workstation. This provides a set of subroutines created for the purpose of constructing both 2D and 3D computer graphics. The GL library is extensively described in the GL manuals [Silicon, 92]. The ANSI C language was

chosen as the implementation language to develop the prototype under IRIX OS. The standard C library of the UNIX version, which includes some functions not defined by the ANSI standard, was used.

5.1.3.3 Networking

The behaviour of the prototype system is that of a concurrent system. There are three environments, *multiprogramming*, *multiprocessing* and *distributed processing* for the development of concurrent systems. In the multiprogramming environment, there are multiple tasks sharing one processor. In the multiprocessing environment, there are two or more processors with a shared memory.

In the distributed processing environment, there are several computers connected to each other by a network. Each computer has its own local memory and there is no shared memory between the processors. Thus a distributed application consists of concurrent tasks distributed over the network and communicating via messages. The prototype system is mainly based on a *distributed processing environment*.

A Local Area Network (LAN) is usually used to link nodes within a single area and geographically dispersed sites are usually linked using the Internet. The prototype DVM system is based upon a LAN because the workstations are distributed over an Ethernet. The Ethernet link has a theoretical speed limit of 10 Mbps. Recently, networking mediums such as FDDI and Fast Ethernet, capable of operating at 100 Mbps, have become widely available.

The network configuration for the distributed processing environment is shown in Figure 5.1. As mentioned earlier, this configuration consists of a network of

workstations (Silicon Graphics Indy). This configuration represents the physical connections between the servers and the clients involved in the demonstration and the evaluation.

There are two servers, called S1 and S2, and six clients, called C1, C2, C3, C4, C5, and C6, on the Ethernet. The communication schemes of the prototype use *Unicast* and *Multicast*. As described in section 4.4.1, either *Unicast* or *Multicast* can be chosen to deliver messages from one client node to all other nodes in the same VE group. Unicast is used to establish communication between two server nodes. Multicast is mainly used to update the entity states between nodes.

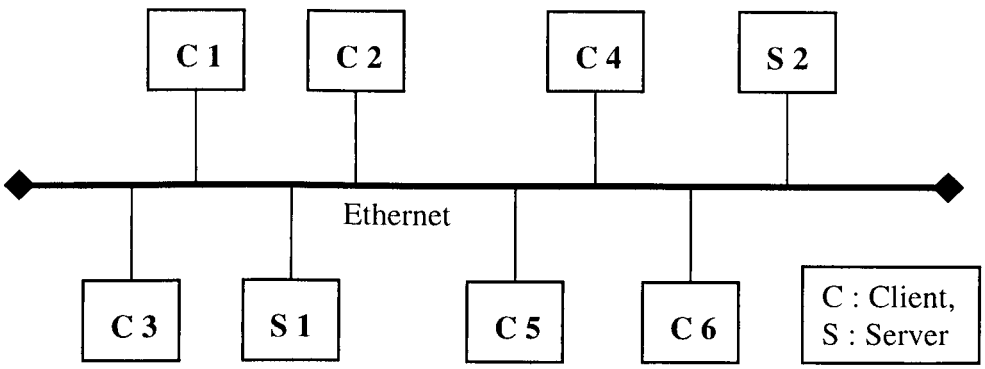


Figure 5.1 Network Configuration for Distributed Processing

5.1.4 Structure of the Prototype System

The prototype system contains two different types of software : the server and the client software. They exchange information with each other on the network. This section describes

the overall structure of the prototype system, and shows the interrelations among the modules within the system.

5.1.4.1 The Context Diagram

The context diagram is usually the first model to be constructed. The context diagram shows the total prototype system, its parts, and how they interrelate with the system. It specifies the boundaries of the prototype system, shows all the inputs to the system and outputs from the system, and clarifies the system requirements.

Figure 5.2 shows the system context diagram of the prototype system. The diagram shows the terminators that interface with the system. The terminators represent the people (e.g. client users, server managers) that use the prototype system. The terminators are parts of the world with which the prototype system must interact. The terminators can be used to represent the boundary. The terminators have their role and function. They receive information from the system, or send information to control it.

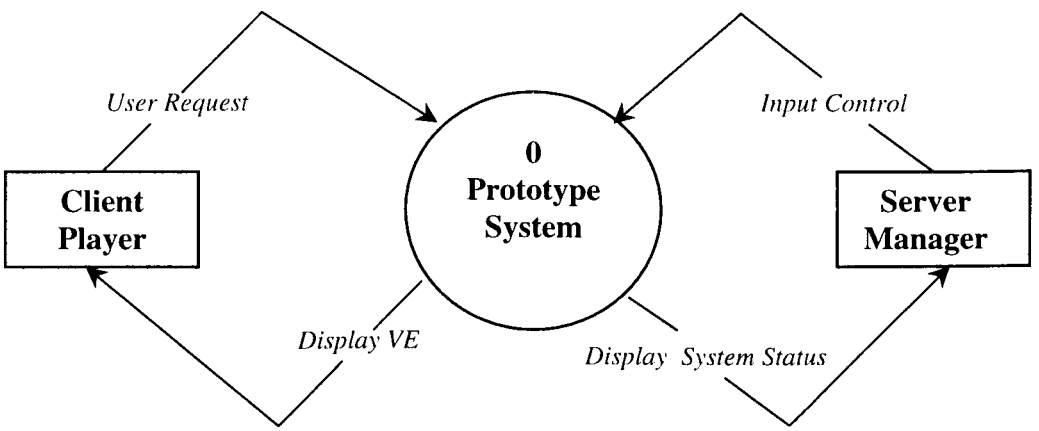


Figure 5.2 System Context Diagram

5.1.4.2 Dataflow Diagrams

Having completed the initial requirement analysis, the dataflow diagram (DfD) notation is used to complete the logical process model. Below the context diagram, the system has the level 1 DfD shown in Figure 5.3. The dataflow diagram is drawn from a restricted number of components, namely processes, terminators, dataflows and data stores. The DfD shows what data flow into, through, and out of the system, what data is stored, and how it is transformed by processes.

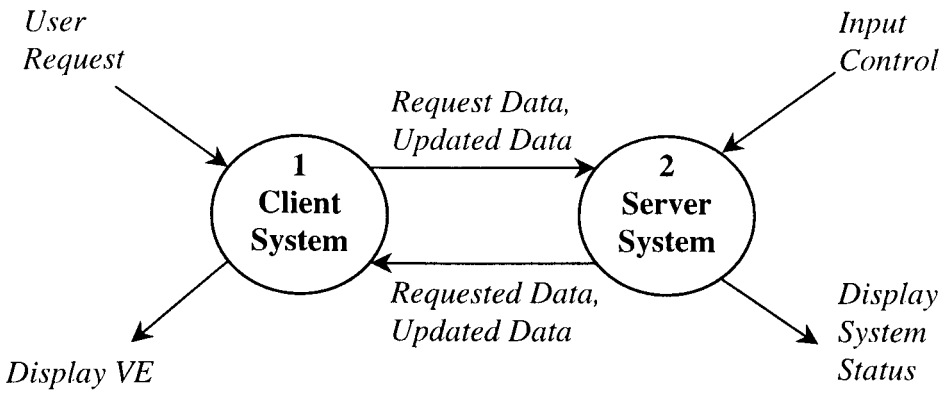
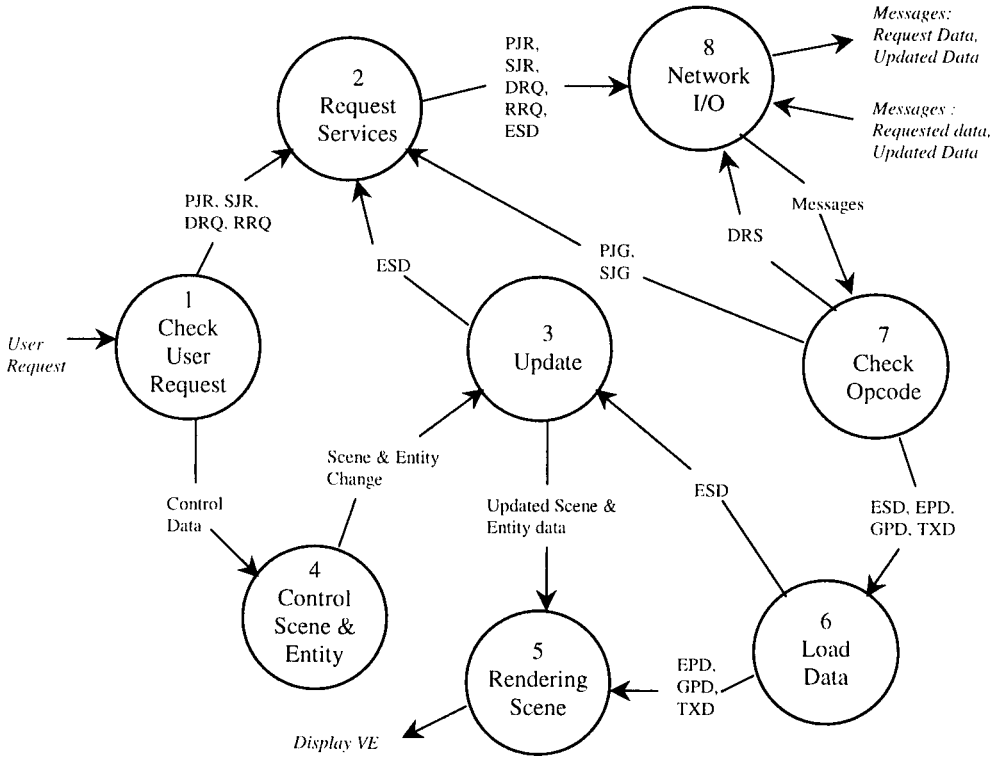
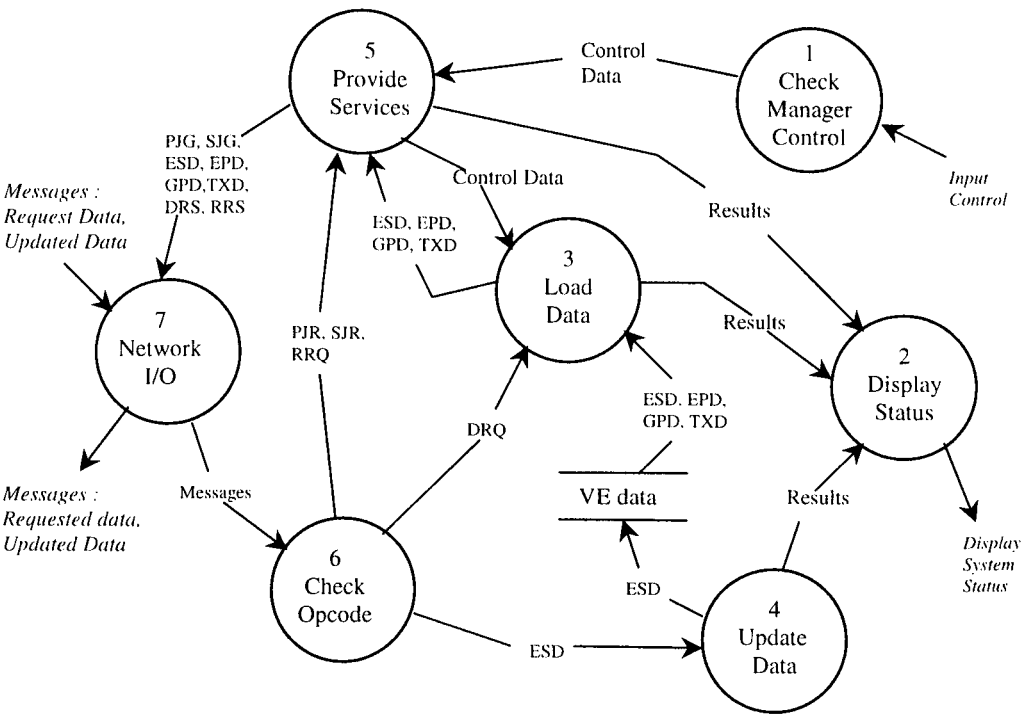


Figure 5.3 The Level 1 Data Flow Diagram

As shown in Figure 5.3, the data items flowing into and out of this DfD also appear in the context diagram. The system really performs two main processes : *Client system* and *Server system*. Rather than operating under the control of a central process, these are independently activated by control of the terminator. Figure 5.4 shows the Level 2 DfDs.



(a) DfD 1-1



(b) DfD 1-2

Figure 5.4 The level 2 DfD for the prototype system

5.1.5 Relationships for communication between DVM components

In the DVM prototype system, the server node includes the functions of four modules : Master Manager (MM), Network Input/Output Manager (NM), Computation Manager (CM), and Database Input/Output Manager (DM). The client node includes the functions of four modules : Master Manager (MM), Network Input/Output Manager (NM), Computation Manager (CM), and Rendering Manager (RM). Figure 5.5 shows the relationships for communication between DVM components defined in Section 3.2.1. The communication between NMs utilises the networking architecture described in the Chapter 4.

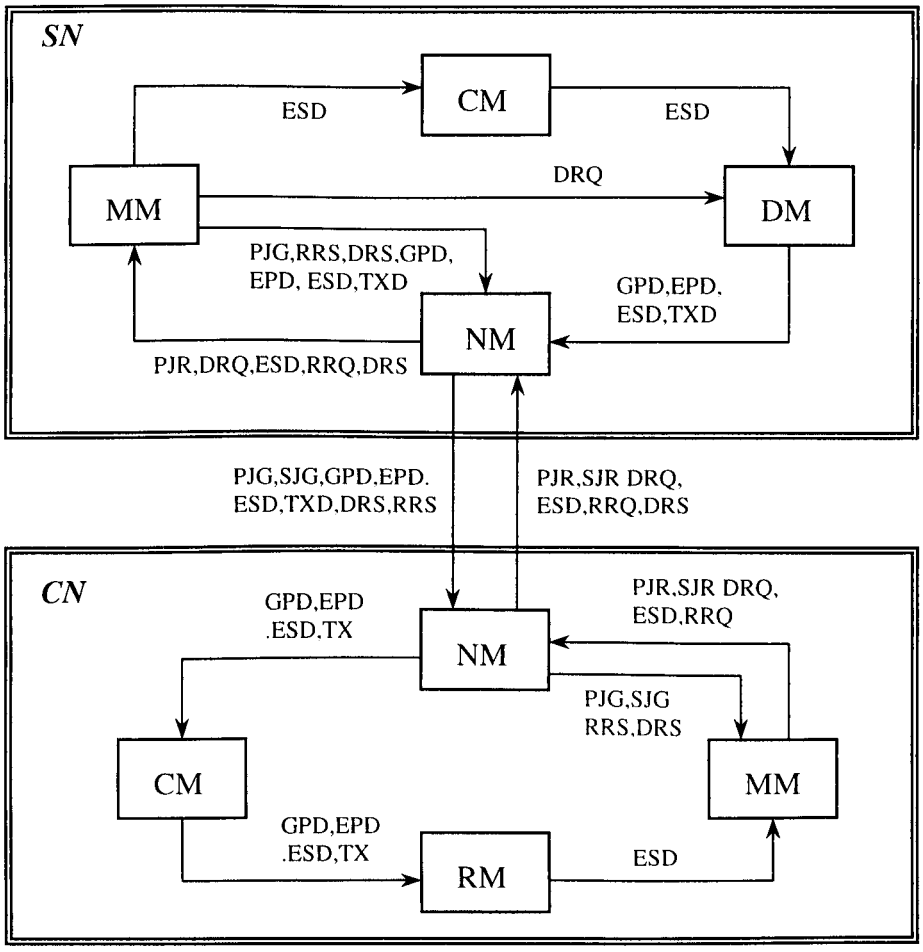


Figure 5.5 Relationships for communication between DVM Components

5.1.6 Implementation Sequences

This section describes the implementation of the prototype system. There are six phases in this implementation of the system. The phases include *System Initialisation*, *Initial Join*, *Initial Data Transmission*, *Screen Generation*, *Execution*, and *System Release*. Figure 5.6 shows the phases of the implementation.

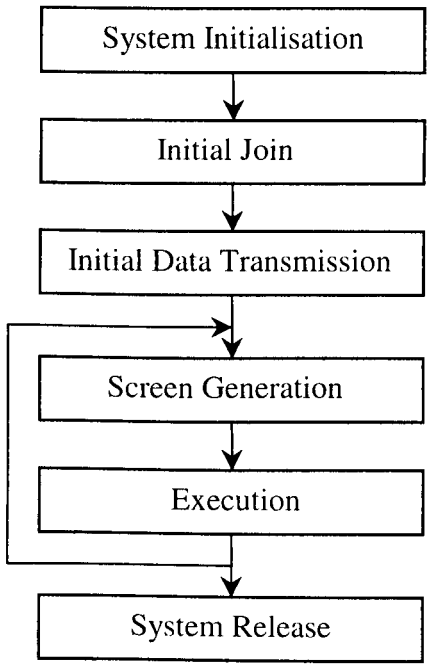


Figure 5.6 Phases of Implementation

5.1.6.1 System Initialisation

Figure 5.7 shows the procedure for the system initialisation. During the *System Initialisation* phase, each node starts its own program, initialises its system, starts up processes, and sets

some initial values of variables. The server program loads the initial data from database. After loading the initial data, the server program waits for the requests from its own clients. The client program sets up the screen to generate the local VE. After setting up the screen, the client program requests the initial join to the server.

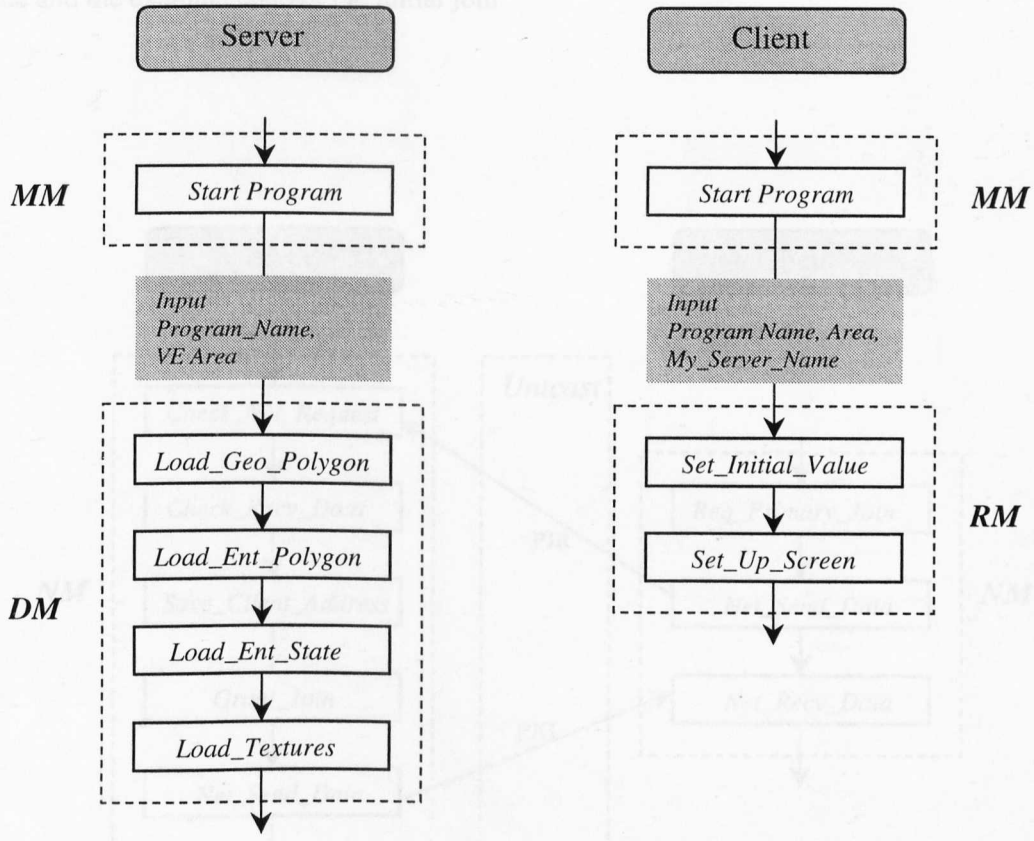


Figure 5.7 Procedure of the System Initialisation

5.1.6.2 Initial Join

Each client tries to join a server, the server maintains permissions to execute communication from all the nodes. When a client enters the system, the server saves the ID of the client. When the next client enters the system, it connects to the multicast channel, informing the other

clients that it has joined. As described in Section 4.3.5.1.1, to join the system a client first unicasts a PJR message to the server node and its request is accepted or rejected. The server node normally checks the requests from the client nodes, it responds by accepting or denying the requests. If the request is accepted, the server sends a PJG message to grant the initial join. Information about the client nodes joining the system is kept by the server node. The following Figures 5.8 shows the simple procedure to be followed when the client node joins the server node and the example code of the initial join.

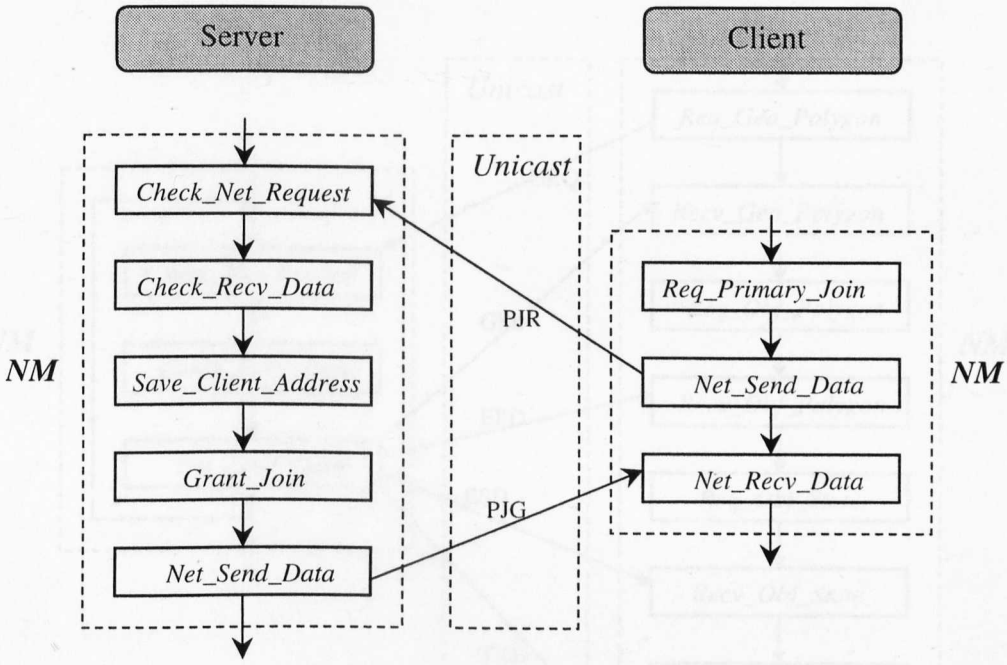


Figure 5.8 Procedure of the Initial Joi

5.1.6.3 Initial Data Transmission

The following Figures 5.9 and 5.10 show the procedure and the example code for the initial data transmission. After joining the server, the client node requests the initial data from the

server node. In the case study (Section 5.1), the client loads initial data from the server node to create terrain and aircraft on the screen. The server provides its own initial data to the clients by First-Come-First-Served (FCFS). This is a very simple operation and consists merely of keeping in strict order all the requests coming into the server. Later requests are not allowed to complete their operation before earlier ones.

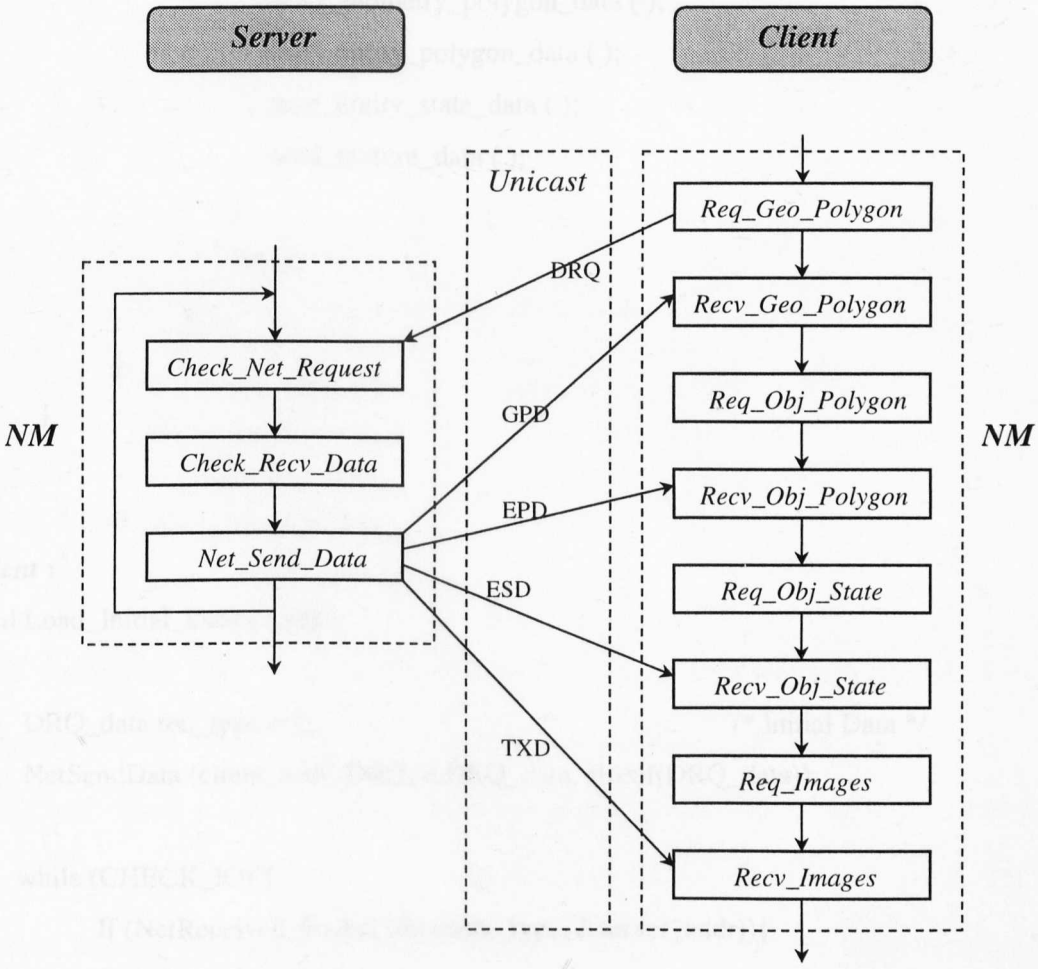


Figure 5.9 Procedure of the Initial Data Transmission

Server :

```
void Check_Network_Request (void) {
        :
      If  (NetReceived_Packet (&packet_type, &client_addr)) {
          Switch (packet_type)  {
                :
              case  DRQ :
                   NetRecvData(&DRQ_data, sizeof (DRQ_data));
                   If (DRQ_data.req_type == 1) {                    /* Initial Data */
                      send_geometry_polygon_data ( );
                      send_entity_polygon_data ( );
                      send_entity_state_data ( );
                      send_texture_data ( );
                   }
                   break;
                :
           }
     }
}
```

Client :

```
void Load_Initial_Data (void) {
       :
    DRQ_data.req_type = 1;                                  /* Initial Data */
    NetSendData (client_addr, DRQ, &DRQ_data, sizeof(DRQ_data));
      :
   while (CHECK_IO) {
         If (NetReceived_Packet (&packet_type, &server_addr)){
         Switch (packet_type) {
                case  GPD :
                     NetRecvData(&GPD_data, sizeof (GPD_data));
                     Receive_geometry_polygon_data ( );
                     break;
                case  EPD :
```

```
                    NetRecvData(&EPD_data, sizeof (EPD_data));
                    Receive_entity_polygon_data ( );
                    break;
                case ESD :
                    NetRecvData(&EPD_data, sizeof (EPD_data));
                    Receive_entity_state_data ( );
                    break;
                case TXD :
                    NetRecvData(&EPD_data, sizeof (EPD_data));
                    Receive_texture_data ( );
                    break;
            }
        }
    }
}
```

Figure 5.10 Example codes of the Initial Data Transmission

5.1.6.4 Screen Generation

After loading the initial data from the server, the client node will generate the screen for rendering the entities and terrain of the local VE on the monitor. Figure 5.11 shows the screen of the user. There are three windows displayed on the screen. The screen is a combination of the 2D field of view (FOV), the local VE view, and the information view panel. The FOV window displays the current section of the local VE. The size and the initial location of the FOV have default settings. The local VE view window displays the 2D terrain of the whole local VE area. It shows the position of the FOV. The window of information view panel contains textual information such as the speed, orientation and position of the aircraft, and the FOV position.

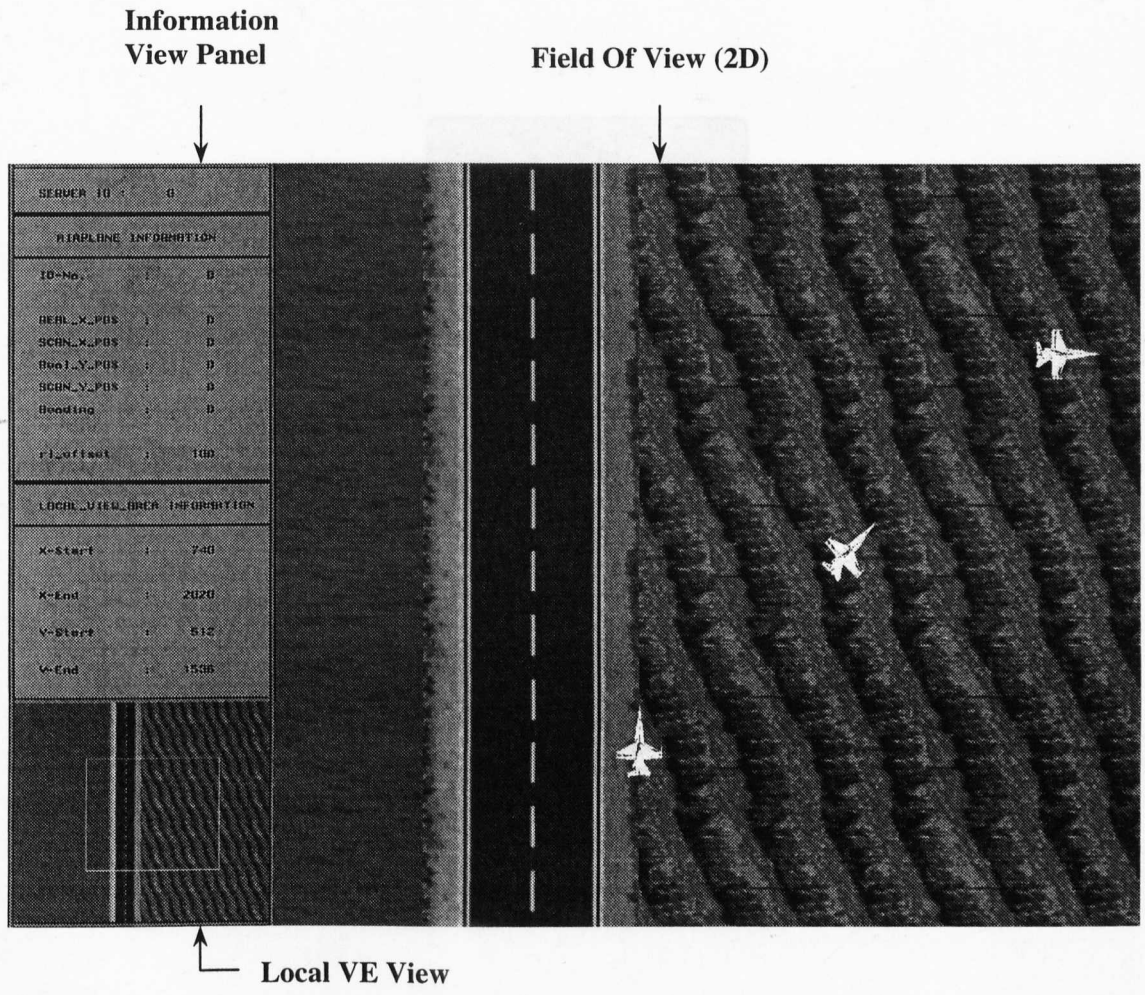


Figure 5.11 The User's Screen

Figure 5.12 shows the procedure for the screen generation. In the case study (Section 5.3.1), the FOV generates the aircraft and the terrain on the screen. Then, the Local VE View generates the terrain of the whole local VE and draws the boxes to show the position of the FOV. The user can control the FOV, which can be moved in any direction, and regenerated. Finally, the Information View Panel shows information about the aircraft that the user is controlling, and the position of the FOV. The prototype uses a simple aircraft as an example entity model. Because the terrain in the prototype is flat, the prototype uses simple aircraft motion based on a 2D scene.

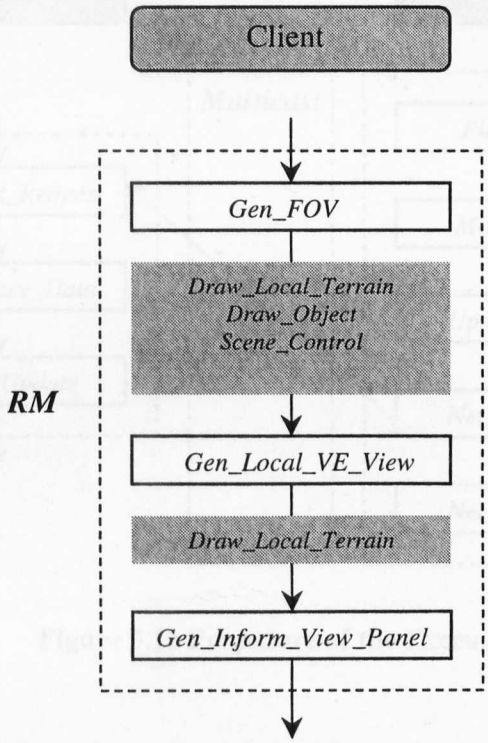


Figure 5.12 Procedure of Screen Generation

5.1.6.5 Execution

In the execution phase, the entities are controlled and updated. Figures 5.13 and 5.14 show the procedure and example code for entity control and update. The client program looks for the control of the entities from the user (move, rotate, quit) or the other users, and multicasts the updates of entities to the other nodes. To control the entities directly, combinations of mouse and key inputs are used. The speed and direction of the entities are modified directly according to user input. The state vector of the entity is made up of its speed and direction. If the state of the entities is modified, then the client transmits the updates to the server and the clients in the same local VE.

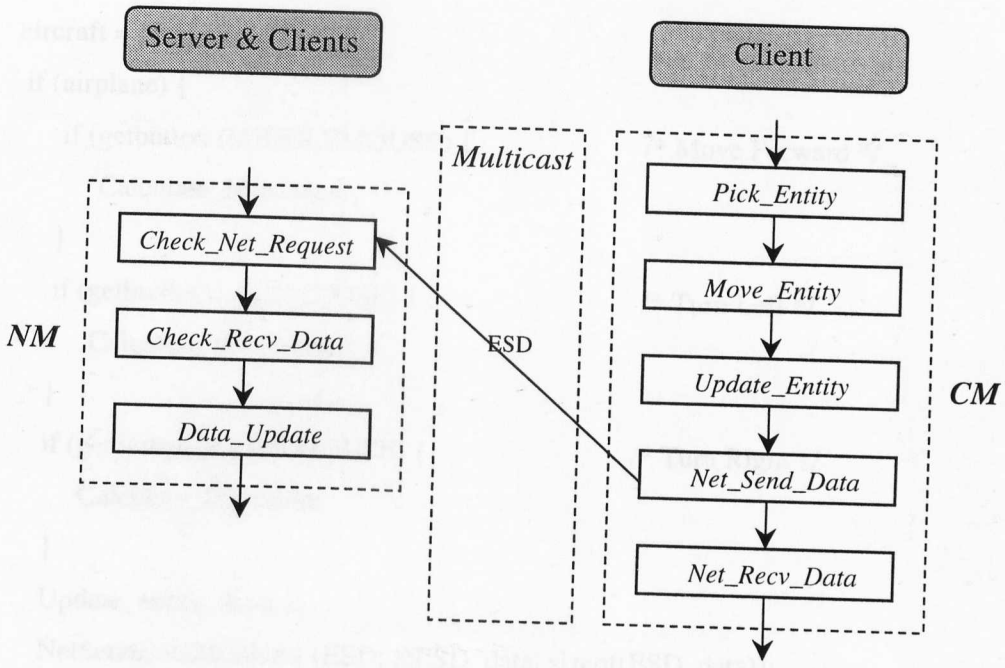


Figure 5.13 Procedure of the Execution

Server :

```
void Check_Network_Request (void) {
        :
    If (NetReceived_Packet (&packet_type, &client_addr)) {
        Switch (packet_type) {
            :
            case  ESD :
                NetRecvData(&ESD_data, sizeof (ESD_data));
                Receive_entity_state_data ( );
                break;
            :
        }
    }
}
```

```
Client :
void Control_Entity (void) {
    aircraft = pick_aircraft ( );
    if (airplane) {
        if (getbutton (MIDDLEMOUSE) {          /* Move Forward */
            Calculate_Movement;
        }
        if (getbutton (LEFTMOUSE) {            /* Turn Left */
            Calculate_Direction;
        }
        if (getbutton (RIGHTMOUSE) {           /* Turn Right */
            Calculate_Direction;
        }
        Update_entity_data( );
        NetSendDataMulticast (ESD, &ESD_data, sizeof(ESD_data));
    } else
        :
}
```

Figure 5.14 Example code for Entity Control

5.1.6.6 System Release

The final phase, the release phase, is initiated by the client. A client node can release the Join by sending a release message (RRQ) to the server node. A client simply sends a release message (RRQ) to inform the server that this client is leaving the system. This action allows the server to remove the client from the system. Figure 5.15 shows the procedure for the system release.

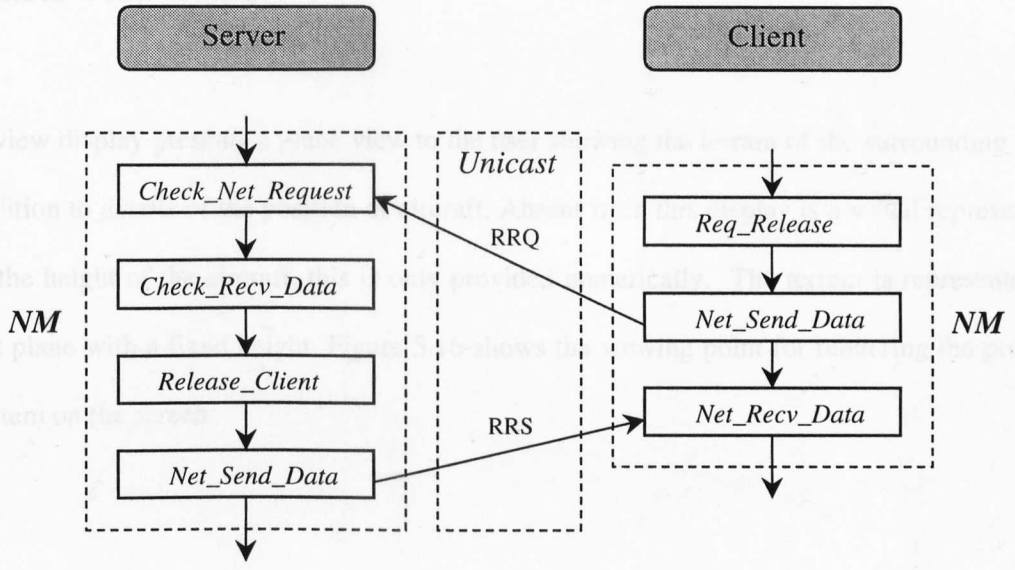


Figure 5.15 Procedure of Release

5.2 Fundamental Techniques for Developing the Prototype

This section describes the fundamental techniques used for implementing the prototype system. These techniques mainly concentrate on graphical and communication aspects.

5.2.1 Graphical Techniques

The implementation of the prototype concentrates on the communication aspects between the nodes, so the details of the graphical techniques used for rendering the VE are not presented. The prototype developed just provides details about the generation of the terrain and the aircraft.

5.2.1.1 View Display

A view display presents a plane view to the user showing the terrain of the surrounding area in addition to details of the position of aircraft. Absent from this display is a visual representation of the height of the aircraft, this is only provided numerically. The terrain is represented as a flat plane with a fixed height. Figure 5.16 shows the viewing point for rendering the prototype system on the screen.

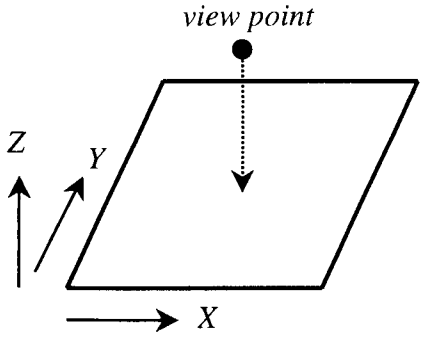


Figure 5.16 Viewing Point for Rendering

5.2.1.2 Screen Co-ordinates

Figure 5.17 shows the co-ordinates on the screen of the workstation. The *X* co-ordinate means how far to the East or to the West of the origin the point is on the *X* axis. The positive *X* values are to the East of the origin, and the negative *X* values are to the West of it. The *Y* co-ordinate means how far to the North or to the South from the origin, the point is on the *Y* axis. Positive *Y* values are to the North of the origin and negative *Y* values are to the South of it. The *Z* co-ordinate describes how far in front of or behind the origin the point is on the *Z* axis. Positive *Z* values extend towards the user and negative *Z* values extend away from the user, away from the origin.

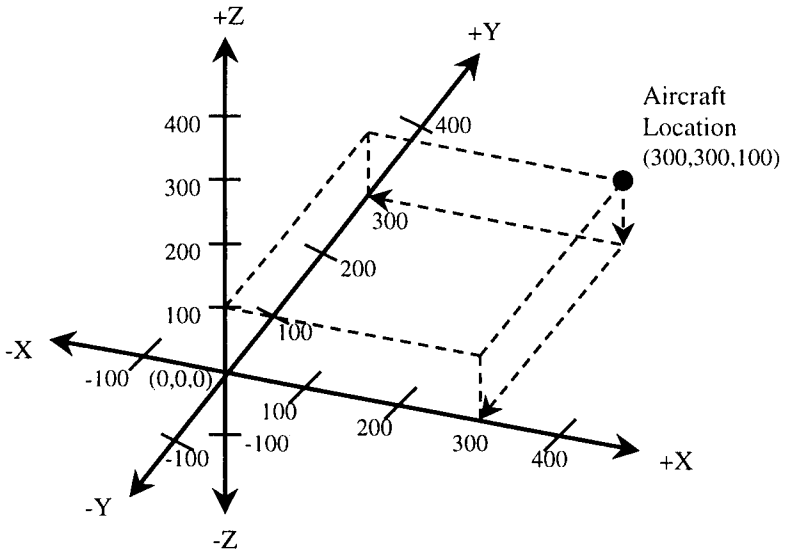


Figure 5.17 Screen Co-ordinates

5.2.1.3 Scale Factor

To scale the display the simulation on the screen, the scale factor is maintained at a fixed value showing a virtual area representing roughly 200 km by 200 km. This value represents an intermediate value between the 80 and 320 km of as being of primary importance for global situation awareness.

5.2.1.4 Terrain Representation

The polygonal technique is used for representing and processing terrain data in the geographic database. Therefore, the terrain of the DVM system is represented as a set of polygons. A polygon is specified by a connected sequence of vertices that all lie in a plane. The boundary of the polygon is defined by connecting the vertices in order : *v1* to *v2*, *v2* to *v3*, and so on, finally connecting *vn* back to *v1*. Even though the Graphics Library (GL) can support polygons with up

to 255 vertices, the performance is usually optimised only for polygons of 3 or 4 vertices. So the 4-sided (quadrilateral) polygons are used to draw the terrain, the shape of polygon is a square containing a grid of equidistant cells. A mesh made of quadrilaterals is called a quadrilateral strip (q-strip). The GL quadrilateral strips are used to render the terrain.

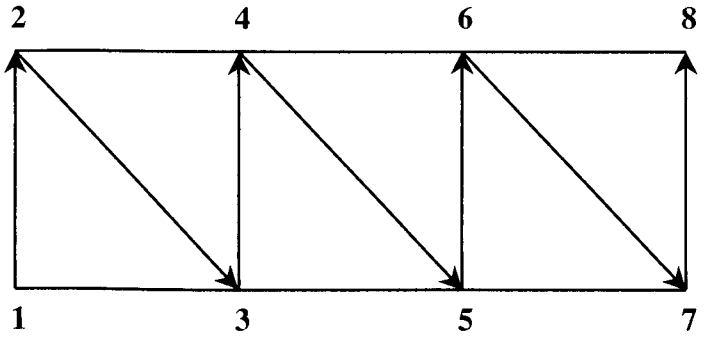


Figure 5.18 Sequence followed in representing the terrain

As shown in Figure 5.18, the grid was split into a series of rows and the rows were rendered as a group. The quadrilateral strips method can improve the performance. For example, to produce the three squares, the quadrilateral strips use this sequence : (1,2,3,4), (5,6), (7,8). However, if the squares are individually rendered, this sequence draws the three quadrilaterals : (1,2,3,4), (3,4,5,6), and (5,6,7,8). It would require nearly twice as many calls.

5.2.1.5 Aircraft Representation

Aircraft are represented using a model supplied with the ModelGen software package developed by Software System Inc. Each aircraft requires a position and an orientation allowing it to be placed accurately above the terrain. Storing values in a file makes it possible to fix the positions of all aircrafts at the start of each simulation.

5.2.1.6 Basic Rendering Techniques

This section describes the techniques needed to render the prototype system on the screen. These techniques include *Z-buffering*, *Colour Display*, and *Texturing*.

5.2.1.6.1 Z-buffering

Whilst drawing a scene it is imperative that entities within the field of view are ordered correctly so that closer entities exclude those behind them. To achieve this a technique known as Z-buffering is used [Silicon, 92]. A Z-buffer is an area of memory that stores a value representing each pixel's depth. Thus when drawing a pixel it is first tested against the corresponding pixel in the Z-buffer. If it is closer, i.e. visible, then the screen pixel is updated along with the depth in the Z-buffer.

5.2.1.6.2 Colouring

As colour is an integral part of the display, the image generation system must be able to provide an adequate number of different colours. Systems are generally based around an 8-bit, 16-bit or 24-bit colour table. With only 8 bits per pixel, allowing 256 colours, artefacts such as banding can be seen and so a 24-bit system, allowing over 16 million colours is generally preferable.

5.2.1.6.3 Texturing

Texturing is one of the most effective methods employed for adding realism to a scene [Silicon, 92]. Rather than simply assigning a colour to the surface of an entity, texturing applies an image to the surface. This image can be used to add realism as in adding an image of wood to a

table, or else to simplifying an entity by enabling the underlying entity to be simplified with the detail being made up by the texture.

5.2.2 Communication Techniques

Sockets are the basis for network programming in the prototype. So this section describes details of the basic techniques used in network programming with sockets. As described in Section 4.3.4, there are various API types, including Unix, Windows and Java, etc. For the DVM prototype, the scope will be limited to the Unix environment. The DVM prototype will concentrate on the API for Interprocess Communication (IPC) in the UNIX environment. In the future, a DVM prototype could provide environments for WinSock, Java, and maybe others. The implementation of a DVM prototype also concentrates on the API for C programmers under Unix.

5.2.2.1 Domains

Sockets created by different programs use host names and port numbers to refer to one another. These names must be translated into network addresses for use by IP communications. Each network address format is called a domain. We restrict the implementation of the prototype to the *Internet domain* (AF_INET). The following structure shows *Socket System Call* for the prototype system.

int *socket* (int *family*, int *type*, int *protocol*);

The *family* is AF_INET (Internet protocols) for IP communications, the *type* depends on whether TCP or UDP is used, and the protocol is typically set to zero for most user applications.

5.2.2.2 Socket Addresses

The combination of the network addresses and the port numbers is referred to as a socket address. Networking system calls require two socket address structure representations. One is the protocol-specific socket address structure that the user employs, and the other is common to all protocols that the kernel uses. Figure 5.19 shows the socket address structure in <sys/socket.h> .

```
struct sockaddr {
u_short     sa_family;          /* Internet address family, AF_INET */
char        sa_data [14];       /* up to14 bytes of protocol-specific address */
};
```

Figure 5.19 Socket Address Structure

The *sa_data* contains a destination address and port number for the socket. The *sa_family* for the prototype system will be AF_INET. Figure 5.20 shows structures for the Internet address family in <netinet/in.h>.

```
#include <netinet/in.h>
struct in_addr {
u_long      s_addr;         /* 32 bit netid / hostid, network byte ordered */
};
```

```
struct  sockaddr_in {
short int           sin_family;     /* Address family  */
u_short             sin_port;       /* 16 bit Port number  */
struct  in_ addr    sin_addr;       /* 32 bit netid / hostid  */
char                sin_zero [8]    /*  not used  */
};
```

Figure 5.20 Address Structures for Internet family

The *sin_family* corresponds to *sa_family* in a *struct sockaddr* and should be set to "AF_INET". The *sin_port* and *sin_addr* in a *struct sochaddr_in* must be in *Network Byte Order*. Network byte order uses big-endian format where the most significant byte is stored first. The *sin_zero* should be set to all zeros with the function bzero ().

5.2.2.3 Internet Addresses

Most communication protocol suites define some type of addressing that identifies networks and hosts. Each host on the Internet is assigned a unique 32-bit Internet address, each Internet address encodes both a network ID and a host ID. The 32-bit Internet address is represented by the dotted decimal notation (eg. 150.237.28.214).

The network ID falls into various classes according to the size of the network address. Class A uses 8 bits for the network address with 24 bits left over for other addressing. Class B uses 16 bit network addressing. Class C uses 24 bit network addressing and class D uses all 32 bits. The University of Hull is registered as a Class B network, so the prototype DVM system uses a 16 bit network address with 16 bits left to identify each machine.

Any organisation with an Internet address of any class can subdivide the available host address space in any way it desires, to provide subnetworks. Internally, the University network is divided into subnetworks. 8 bits are used for this. 8 bits are finally used for host addresses within a department subnet. This places a limit of 254 machines that can be on each subnet.

5.2.2.4 Port Addresses

As described in Section 4.3.4, a service exists on a host, and is identified by its port. Ports in the region 1-255 are reserved by TCP/IP. The system may reserve more. So DVM processes must use ports numbered above 1023.

5.2.2.5 Socket System Calls

In a DVM application, a server NM communicates with a client NM. Socket-Related system calls are required to perform the communication between these processes. The socket system calls can be used in both the connectionless protocol (UDP) and the connection-oriented protocol (TCP).

In the connectionless protocol, the client does not establish a connection with the server. The client sends a datagram to the server, and the server waits until data arrives from some client. By using the network address of the client process, along with the datagram, the server can send its response to the correct process.

In the connection-oriented protocol. first the server is started, then later a client is started. Before the communication starts, each process opens a socket of its own. The server then binds its socket descriptor to its local Internet address and port number, and waits for the connection

from the client processes. After the connection is established, the client process initiates the requests, and the server process responds.

5.3 Demonstration of the Prototype System

This section illustrates details of the demonstration. In order to show whether the system is properly working or not, a graphical display is provided. Especially, the focus in the implementation of the demonstration is the communication between the nodes. The demonstration is divided into the system integration and the system execution phase.

5.3.1 System Integration

The system integration includes the system initialisation, the initial join, the initial data transmission, and the initial display generation. First, to start the system integration, the users start their own program. The user at the client inputs the program name, the area name of the local VE to load, and the name of the server to access. The system manager at the server node inputs the program name and the area name of the local VE to load. The server loads the initial data, including geometry polygon data, entity polygon data, entity state data, and texture data from the database.

After loading the initial data, the server waits for requests from its own clients. The client sets the initial value of variables, and sets up the screens. Then the client tries to create the initial join with the server, when the server saves the ID of the user. After joining the server, the client node requests the initial data from the server node, and loads the initial data from the server

node. Finally, after loading the initial data, each client node displays the aircraft and the terrain of the local VE on the screen.

5.3.2 Execution

The client program runs on a schedule, and each execution consists of reading inputs, processing them, producing updates to the state of the entities, generating the VE screens, and transmitting updates to the others. The inputs can come from the networks, or from the controls operated by the users. The inputs can be changed by the users. To maintain all of the aircraft positions and velocities, a complete database is kept on the server node. In the aircraft information display, "Heading" represents the direction of the aircraft, the values of the X and the Y axis components (Latitude and Longitude) represent the aircraft's position, and the value of the Z-axis component (Altitude) is a fixed value.

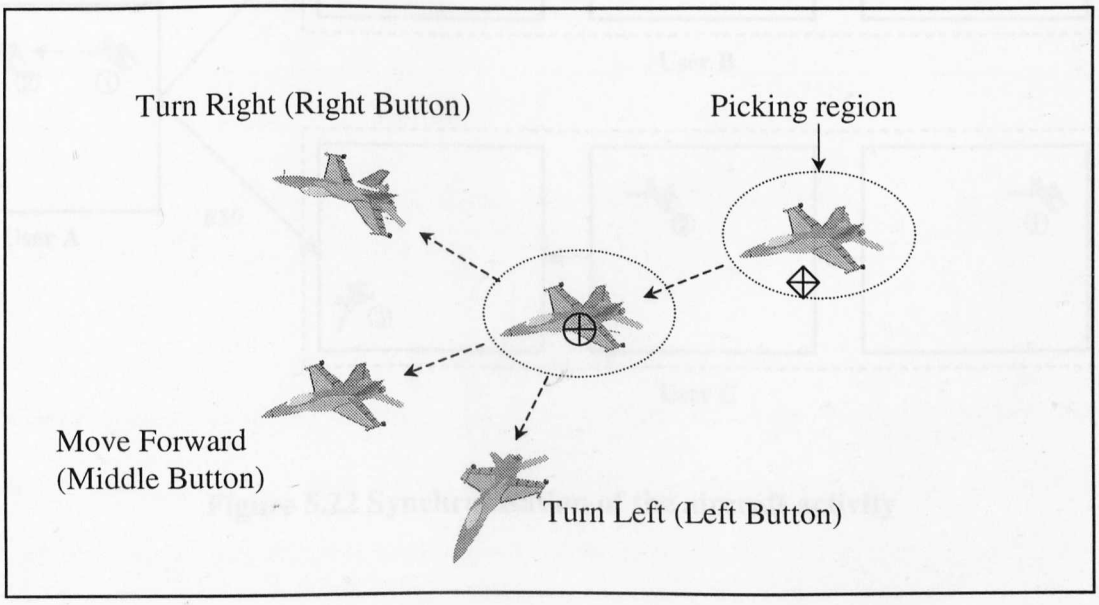


Figure 5.21 Control of the aircraft

Figure 5.21 shows the method for the control of the aircraft. The user can control the movements and the activities of the aircraft using mouse and keyboard. A user can control only one aircraft at a time. To control the aircraft, the user moves the cursor on the aircraft, and the shape of the cursor changes. The cursor as a diamond indicates which user has the right to control the aircraft. The buttons of the mouse are used to change the direction and the position of the aircraft. The user presses one of the buttons of the mouse. The right button is used to turn right, the left is used to turn left, and the middle is used to move forward. To change the speed of the aircraft, the user positions the cursor on an aircraft and presses the function keys of the keyboard. The F5 key is used to increase the speed, and the F6 key is used to decrease the speed. If the user does not press any of the keys or buttons, the aircraft continues at the same speed and direction until it reaches the end of the VE area.

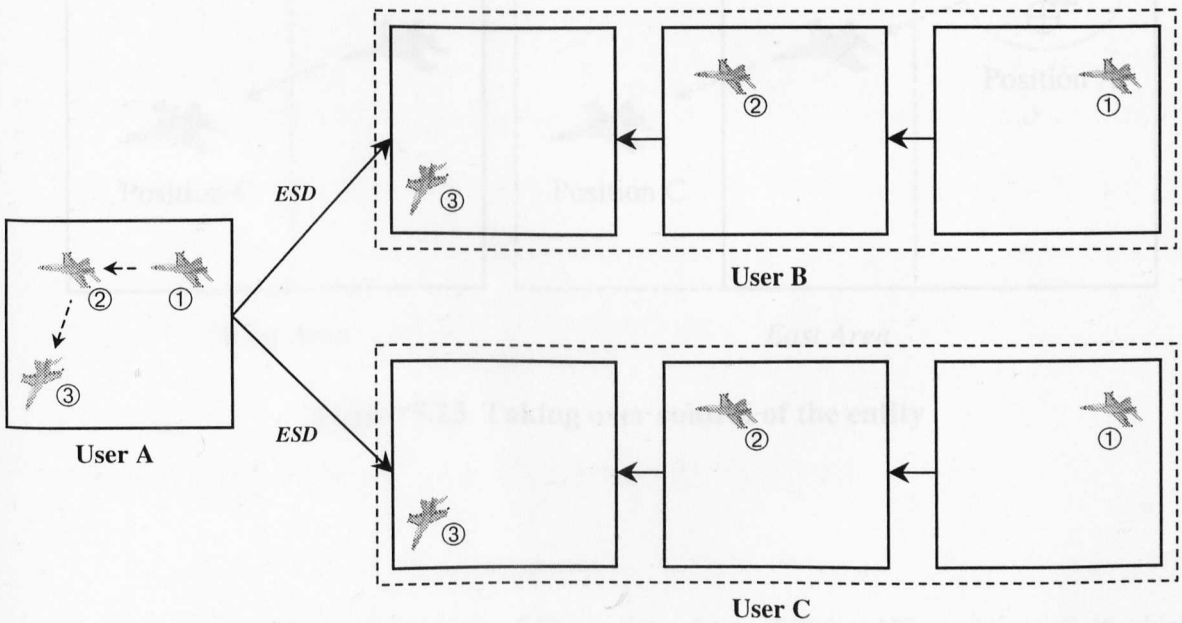


Figure 5.22 Synchronisation of the aircraft activity

The VE world can respond to actions by altering the status and multicasting the updates to the other VE worlds. Figure 5.22 shows the synchronisation of the activity of an aircraft. While the users are controlling the aircraft, it updates the other users' view by sending out information about the activities of the aircraft. The DCP packet units are used to transmit this information. When a user moves an aircraft, messages are sent by multicasting. At the same time, the aircraft in the other users' view changes its position or direction. Not only can the user see the movement, but also all the other users; the aircraft moves on all screens as a result of the update.

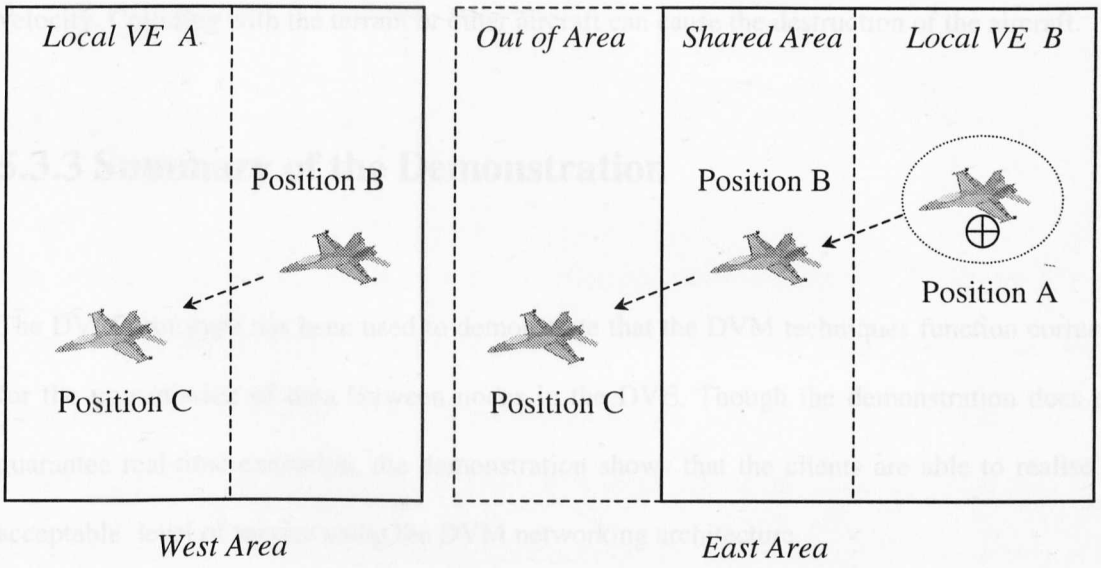


Figure 5.23 Taking over control of the entity

The environment illustrated in figure 5.23 consists of two different VE worlds, each of which has different aircraft and area. Two different VE worlds representing the East Area and the West Area are built at different workstations individually. Figure 5.23 shows the views (FOV) of the users in the two areas. As shown this figure, if a user in the east area moves an aircraft into the shared area, the user in the west area can also see the aircraft. The users in both the east

and the west can control the aircraft in the shared area. If the aircraft moves out of the east area and moves into the west area, the user in the east area cannot control the aircraft. Only the user in the west area can control the aircraft, the user can take over control.

If an aircraft moves at the same speed and direction, the users don't need to transmit the state of the aircraft at every time step. Whenever an aircraft is moving, each client node computes the aircraft's state. If the movement does not deviate much from the dead-reckoning, then no messages are sent. The aircraft still moves on all screens as a result of the dead-reckoning algorithm. If the aircraft does deviate significantly from the dead-reckoning path, then the user informs all the other users to update their aircraft with the new correct aircraft position and velocity. Colliding with the terrain or other aircraft can cause the destruction of the aircraft.

5.3.3 Summary of the Demonstration

The DVM prototype has been used to demonstrate that the DVM techniques function correctly for the transmission of data between nodes in the DVE. Though the demonstration does not guarantee real-time execution, the demonstration shows that the clients are able to realise an acceptable level of service using the DVM networking architecture.

5.4 Evaluation of the Prototype System

The key aspects of the theory presented in this thesis were implemented on the prototype system. A performance test of the prototype system was carried out to evaluate the overall effectiveness. There are many methods that can be used to test the performance. The prototype system has been subject to two performance tests : the *Initial Data Transmission Time Check* and the *Synchronisation Capability Check*. This section will describe the results of the

prototype system tests as a whole. All of the benchmarks used in this chapter were run under similar conditions. The tests were conducted on multiple node systems. No users were permitted access to the networks during testing.

5.4.1 Test 1 : Initial Data Transmission Time Check

In this test, a server transmits the initial data to the clients, and its *Initial Data Transmission Time* is checked to evaluate the performance of the prototype system. The Initial Data Transmission is taken for two network configurations : a case that uses *multiple servers* and another case that uses a *single server*.

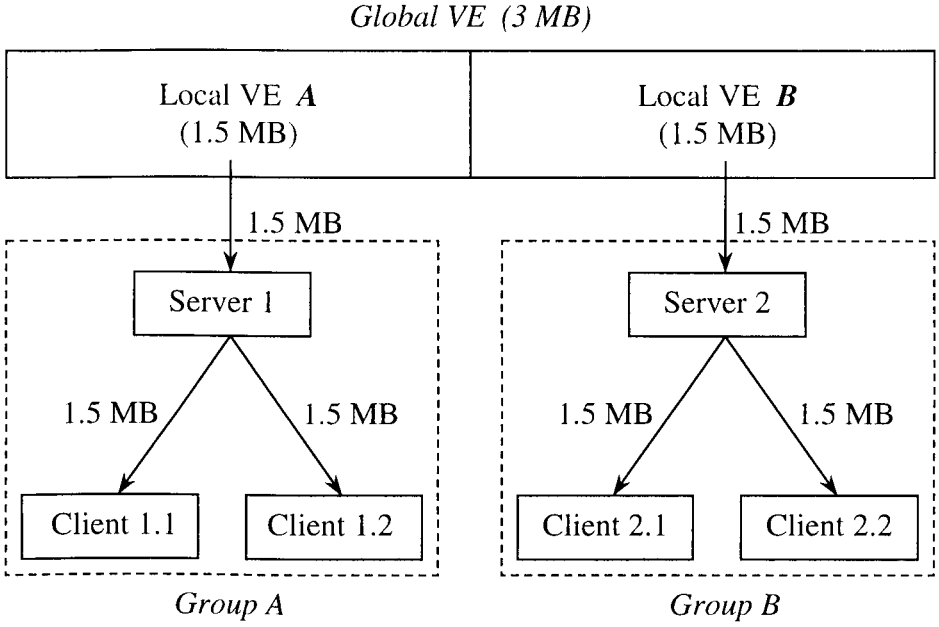
5.4.1.1 Configurations

The two network configurations for the evaluation are shown in Figure 5.24. Figure 5.24 (a) shows a configuration that includes two servers with their own client nodes. Each server transmits the initial data to its own client nodes. Figure 5.24 (b) shows a configuration that includes a server with its own client nodes. The single server transmits the initial data to its own client nodes. In both configurations, disk accesses to load the initial data occur only at the beginning of the system execution.

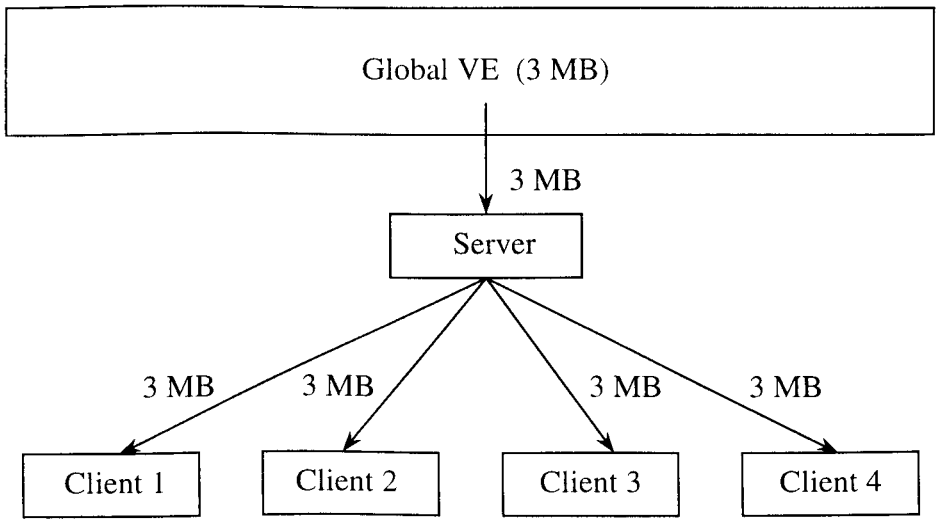
5.4.1.2 Checking the Initial Data Transmission Time

The value of the initial data transmission time does not include the processing delay in the workstations. The value means just the propagation time from the server to the client and the response time. As shown in Figure 5.25, to check the initial data transmission time, each server

checks the arrival time of the DRQ packet from the clients, and the last response time from the clients. These two values are used to calculate the elapsed time.

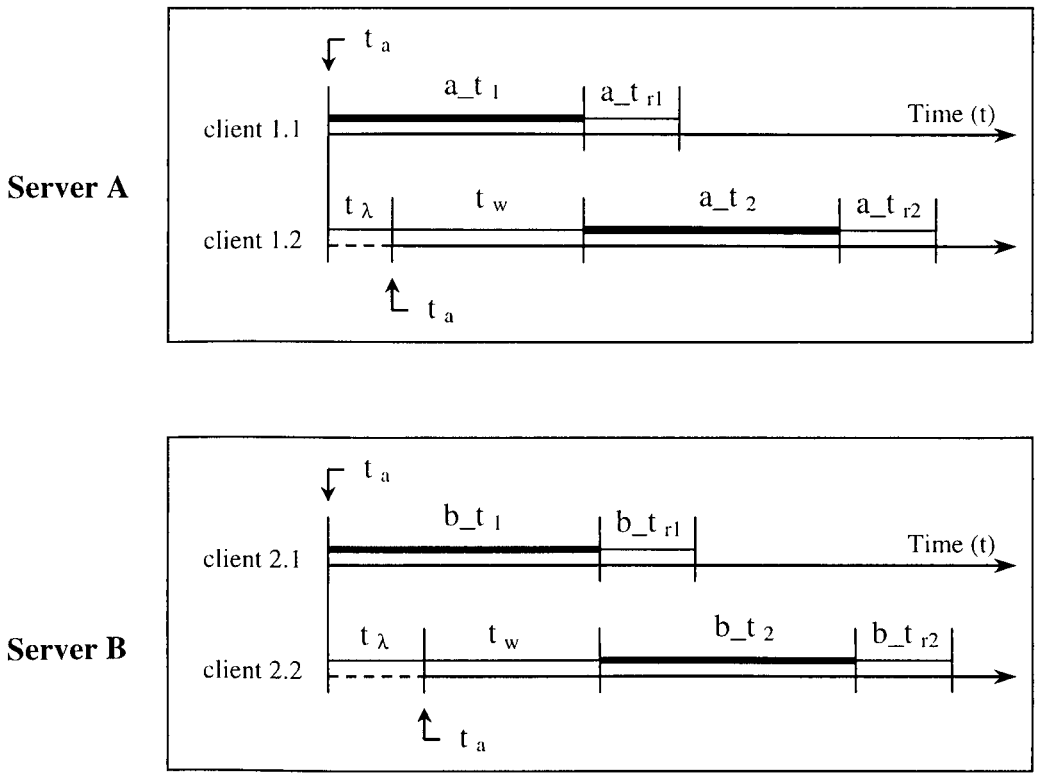


(a) Case 1 : Configuration using Multiple Servers

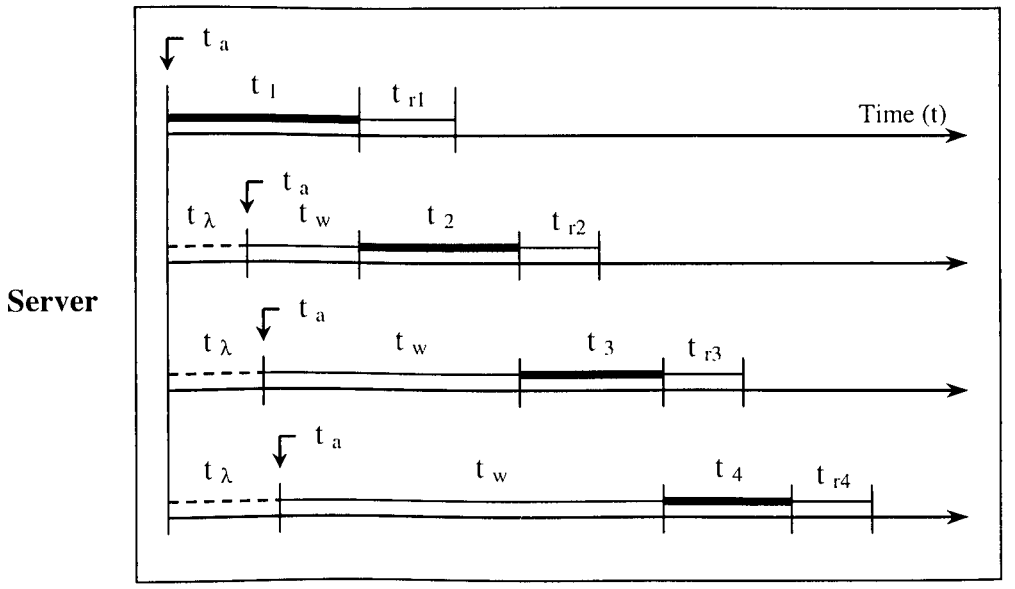


(a) Case 2 : Configuration using a Single Server

Figure 5.24 Network Configurations for the Initial Data Transmission



(a) Case 1 : Using Multiple Servers



(b) Case 2 : Using Single Server

Figure 5.25 Time Progression Diagrams for Transmission of Initial Data

The examples of the time progression diagrams for transmission of initial data are shown in Figures 5.25 (a) and (b). Figure 5.25 (a) shows a case where multiple servers transmit the initial data to the client nodes, Figure 5.25 (b) shows a case where a single server transmits the initial data to the clients.

In these figures, the value of t_a indicates the *arrival time* of the DRQ packets from the clients to the server, the value of t_λ indicates the *time gap* between DRQ packets arriving from the clients to the server. The value of t_w indicates the *waiting time* while each client waits for the service from the server. The value of t_x (x = 1,2,3,...) indicates the *data transmission time* while the server transmits the initial data to the client. And the value of t_{rx} (x = 1,2,3,...) indicates the response time before the clients confirm the completion of data transfer to the server.

The following cases 1 and 2 show some examples for calculation of Total Initial Data Transmission Time from the servers. Case 1 shows an example where two servers transmit the initial data of about 10 MB to their own client nodes. If T_B is bigger than T_A, T_B is the Maximum Total Initial Data Transmission Time (T_{max}). Case 2 shows an example where a single server transmits the initial data of about 20 MB to its own client nodes. Maximum Total Initial Data Transmission Time (T_{max}) is calculated by adding each transmission time and the response time from the last client in sequence.

Case 1 : **Using Multiple Servers**

Total Initial Data Transmission Time from the server A : $T_A = a\_t_1 + a\_t_2 + a\_t_{r2}$

Total Initial Data Transmission Time from the server B : $T_B = b\_t_1 + b\_t_2 + b\_t_{r2}$

Maximum Total Data Transmission Time : $T_{max} = \max (T_A , T_B)$

Maximum Data Transmission Time :

$$T_{max} = t_1 + t_2 + t_3 + t_4 + t_{r4}$$

5.4.1.3 Test Results

The time taken to transmit the initial data is to some extent dependent on network conditions. The final evaluation is produced using these average values. *Average Total Initial Data Transmission Time* is used to compare the difference of the performance in two cases. *Average Maximum Total Transmission Time* is the results of the comparison of *Average Total Initial Data Transmission Time* taken to transfer the initial data with several different sizes.

Although all of the results produced during Initial Data Transmission Time Check Tests cannot be presented here, Tables 5.1 and 5.2 show representative samples of the results. Table 5.3 shows the final test results. These results are the average values of a hundred test results.

Figure 5.26 is a graphical representation of the results whilst Table 5.3 details the average maximum total initial data transmission time. The solid line shows the Average Maximum Total Transmission Time using single server. The dashed line shows the Average Maximum Total Transmission Time using multiple servers. As can be seen, with an increase in the number of clients, the Average Maximum Total Transmission time for the case of multiple servers increases less sharply than the case of using a single server.

Number of Test	Group A (T_A)	Group B (T_B)	T_{max} (sec)
	1	1	
1	4.72	5.86	5.86
2	4.71	5.50	5.50
3	4.25	4.99	4.99
4	4.69	5.30	5.30
:	:	:	:
100	4.41	5.62	5.62
Average (sec)	4.57	5.44	5.44

(a) Numbers of Clients = 2

Number of Test	Group A (T_A)	Group B (T_B)	T_{max} (sec)
	2	1	
1	7.35	5.21	7.35
2	7.64	5.45	7.64
3	7.20	5.20	7.20
4	8.05	5.26	8.05
:	:	:	:
100	8.20	5.72	8.20
Average (sec)	7.71	5.39	7.71

(b) Numbers of Clients = 3

Number of Test	Group A (T_A)	Group B (T_B)	T_{max} (sec)
	2	2	
1	8.10	8.74	8.74
2	7.65	7.78	7.78
3	7.99	8.52	8.52
4	7.48	7.96	7.96
:	:	:	:

100	7.76	8.10	8.10
Average (sec)	7.81	8.24	8.24

(c) Numbers of Clients = 4

Number of Test	Group A (T_A)	Group B (T_B)	T_{max} (sec)
	2	3	
1	7.32	11.60	11.60
2	7.25	11.47	11.47
3	7.45	11.79	11.79
4	7.95	12.32	12.32
:	:	:	:
100	8.13	12.54	12.54
Average (sec)	7.64	11.96	11.96

(d) Numbers of Clients = 5

Number of Test	Group A (T_A)	Group B (T_B)	T_{max} (sec)
	3	3	
1	11.70	12.10	12.10
2	11.43	11.41	11.43
3	11.87	12.43	12.43
4	11.98	12.51	12.51
:	:	:	:
100	12.17	12.74	12.74
Average (sec)	11.83	12.23	12.23

(e) Numbers of Clients = 6

Table 5.1 Maximum Total Transmission Time based on Multiple Servers

Number of Test	Numbers of Clients				
	2	3	4	5	6
1	12.46	19.94	25.88	32.82	37.79
2	13.15	18.76	24.76	31.95	38.45
3	14.64	19.10	23.93	33.35	38.92
4	12.70	17.90	26.22	32.67	39.23
:	:	:	:	:	:
100	14.20	20.12	24.79	33.46	39.14
Average (sec)	13.44	19.15	25.17	32.83	38.76

Table 5.2 Maximum Total Transmission Time based on Single Server

Number of Servers	Number of Total Target Clients				
	2	3	4	5	6
1	13.44	19.15	25.17	32.83	38.76
2	5.44	7.71	8.24	11.96	12.23

Table 5.3 Average Maximum Total Transmission Time (s)

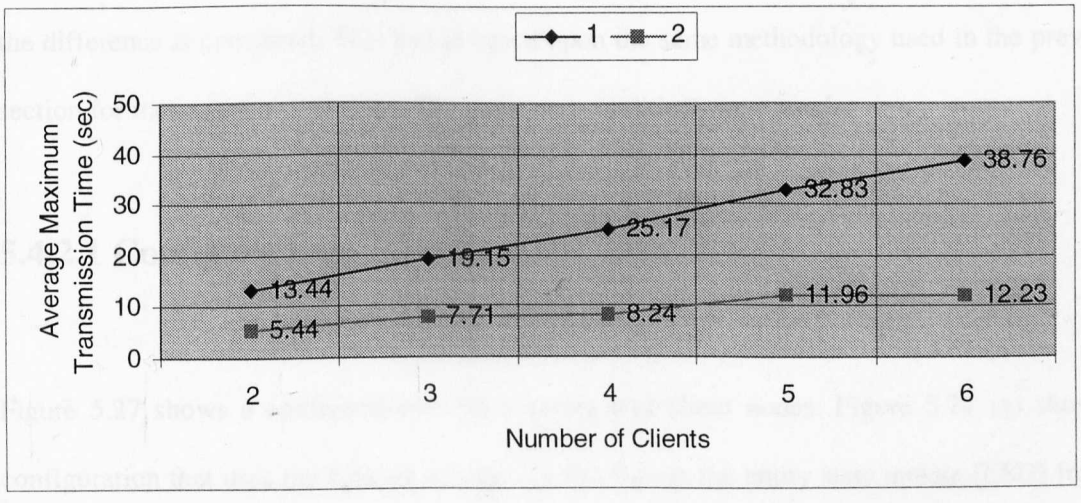


Figure 5.26 Comparison of Average Maximum Total Transmission Time

5.4.2 Synchronisation Capability Check

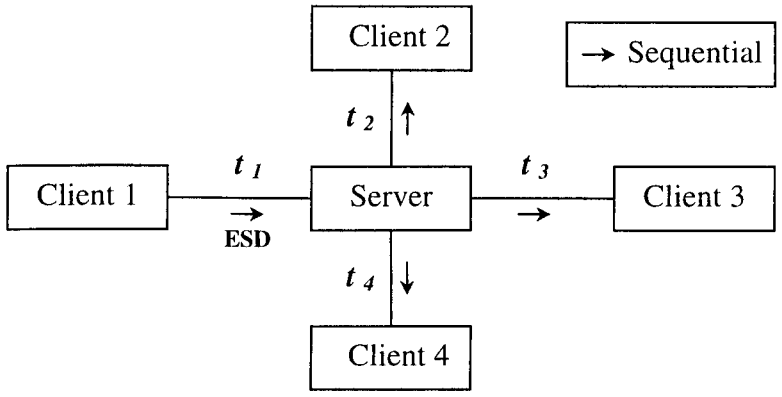
The synchronisation capability is directly related to the system performance. To achieve good synchronisation of entity movements among multiple users across a network, a good system performance is required. The biggest influence on the system performance is the number of entity state updates that the nodes must handle. Data such as entity state updates will be sent frequently whereas initial data will be sent less frequently. The number of entity state updates sent is dependent upon the frequency of changes and events at each client. The number of nodes in a system plays an important part in the system performance. The higher the number of clients, the sharper the increase in entity state updates. The number of entity state updates has an effect on the system performance. To increase system performance, it is important to reduce the number of entity state updates that the nodes must handle.

To achieve this effectively, it will be necessary to choose a communication scheme that can reduce the number of communications between nodes. The communication scheme affects the total transmission time. In this test, therefore, a client transmits a state update to the others by using the *Unicast* or the *Multicast* communication scheme, its *total update time* is tested and the difference is compared. This test is based upon the same methodology used in the previous section for message transmissions. The same test conditions were used.

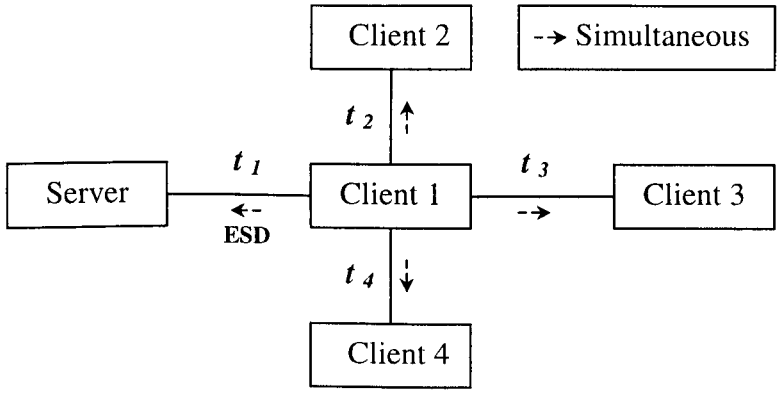
5.4.2.1 Configurations

Figure 5.27 shows a configuration with a server and client nodes. Figure 5.27 (a) shows a configuration that uses the Unicast scheme. In this figure, the entity state update (ESD) from a client is sent to the server first, and the server forwards it to the other target clients. Figure 5.27 (b) shows a configuration that uses Multicast. Multicast allows a single transmission to reach

each node running the VE system. With a Multicast only one message would be required which would reach a server and three client nodes almost simultaneously. A client node could multicast update messages to the other target nodes.



(a) Case 1 : Unicast



(b) Case 2 : Multicast

Figure 5.27 Configurations for the entity state updates

5.4.2.2 Checking of the Total Entity State Update Time

To check the total entity state update time from a source node to the target nodes, the experiments had been taken at two methods: a method that checks *Round Trip Time* and another method that checks *One-way Trip Time*.

5.4.2.2.1 Round Trip Time Check

The time progression diagrams for updating of the entity states by the Round Trip Time Check are shown in Figures 5.28 (a) and (b). In these figures, the value of t_x ($t_{1, 2, \ldots}$) indicates the value of $t_{\lambda i}$ ($t_{\lambda 1, } t_{\lambda 2, \ldots}$) indicates the *time delay* (eg. software overhead) at each node when transmitting an ESD message to one or more nodes, the *data transmission time* required to transmit an ESD packet from a client node to the other nodes, and the value of t_{ri} ($t_{r1, } t_{r2, \ldots}$) indicates the *response time* from the each target node that received the ESD.

Figure 5.28 (a) is a case where a source client sends the ESD to a server first, and forwards it to the target nodes by Unicast in sequence. Figure 5.28 (b) is a case where a client sends the ESD to the target nodes simultaneously by Multicast. A source node checks the return times of the response messages from the target nodes, in order to calculate elapsed time.

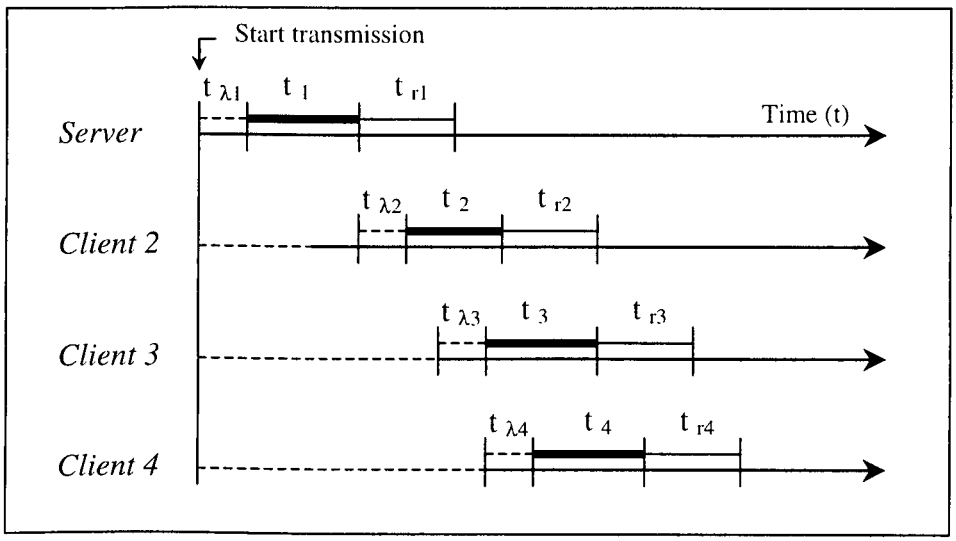
Total Entity State Update Time (T_{tot}) is checked by calculating the difference between the time of transmitting the ESD from a source node to the first target node and the time at which the last response arrives from the target nodes.

Case 1 : Unicast

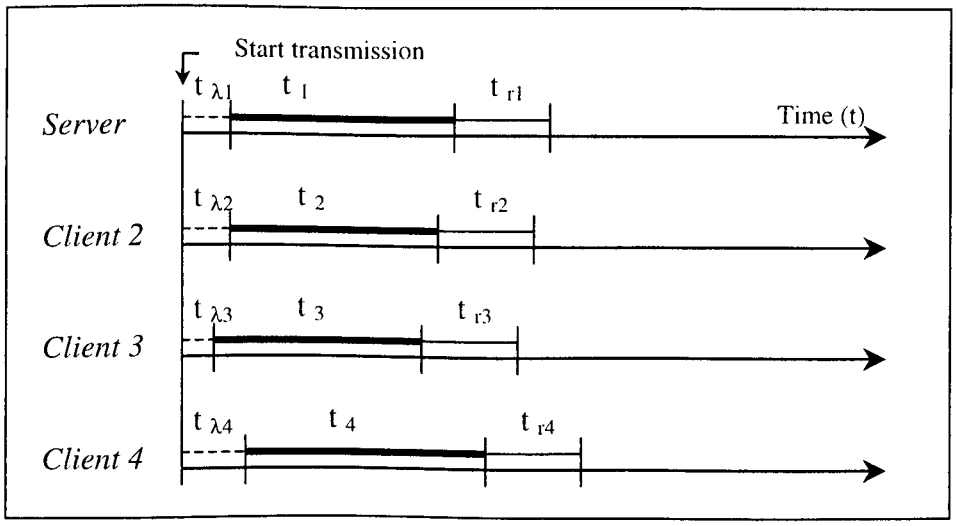
Total Entity State Update Time : $T_{tot} = t_{\lambda 1} + t_{1} + t_{\lambda 2} + t_{\lambda 3} + t_{\lambda 4} + t_{4} + t_{r4}$

Case 2 : Multicast

Total Entity State Update Time : $T_{tot} = \max (t_{\lambda i} + t_i + t_{ri})$ $i = \{ 1, 2, 3, 4 \}$



(a) Case 1 : Unicast



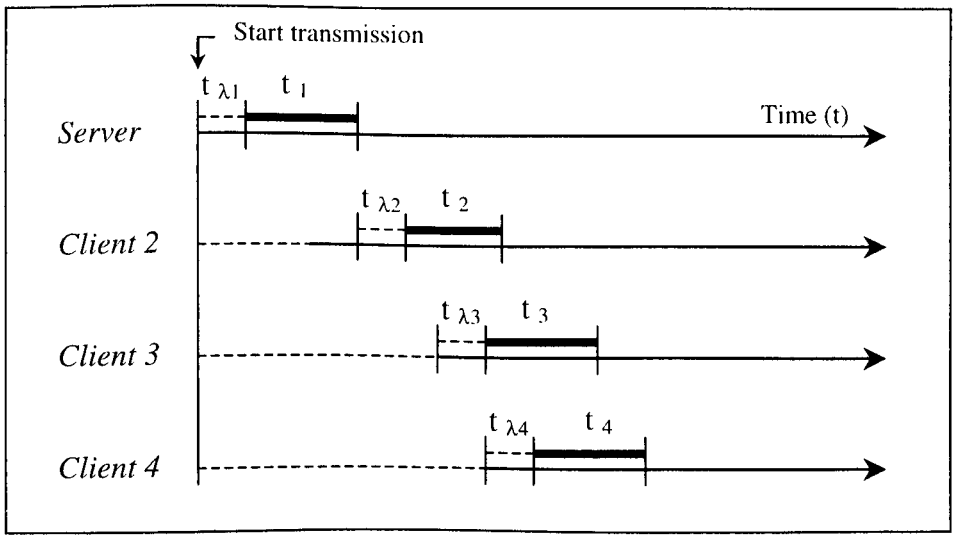
(b) Case 2 : Multicast

Figure 5.28 Total Entity State Update Time Check by Round Trip

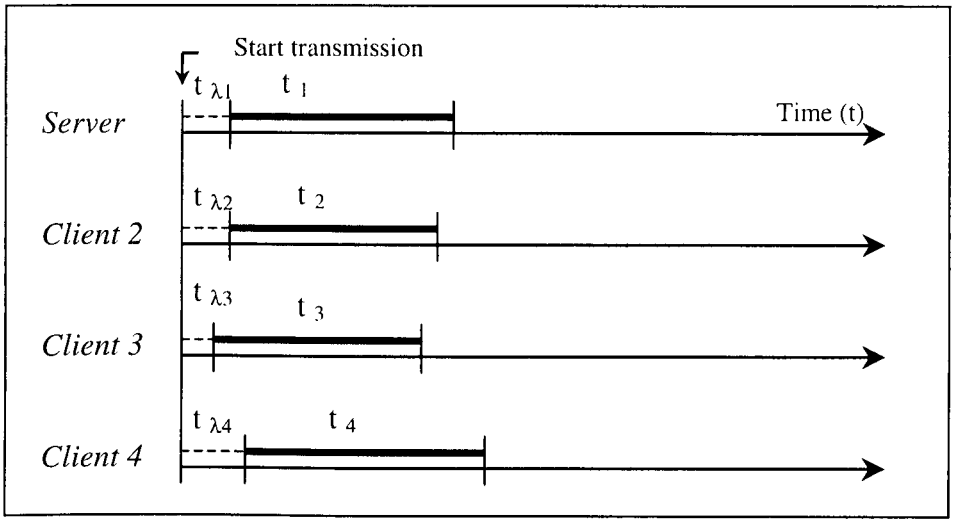
5.4.2.2.2 One-way Trip Time Check

The time progression diagrams for updating of the entity states by the Round Trip Time Check are shown in Figures 5.29 (a) and (b). In these figures, the value of t_x ($t_{1, 2, \ldots}$) indicates the *data transmission time* required to transmit an ESD packet from a client node to the other nodes, and the value of $t_{\lambda i}$ ($t_{\lambda 1}, t_{\lambda 2, \ldots}$) indicates the *time delay* (eg. software overhead) at each node when transmitting an ESD message to one or more nodes.

Figure 5.29 (a) is a case where a client sends the ESD to a server first, and forwards it to the target nodes by Unicast in sequence. Figure 5.29 (b) is a case where a client sends the ESD to the target nodes simultaneously by Multicast. Each target node checks the arrival time of the ESD packet from the source client node, in order to calculate elapsed time.



(c) Case 1 : Unicast



(d) Case 2 : Multicast

Figure 5.29 Total Entity State Update Time Check by One-way Trip

Total Entity State Update Time (T_{tot}) is checked by calculating the difference between the time of transmitting the ESD from a source node to the first target nodes and the time that the ESD arrives at the last target node.

Case 1 : Unicast

Total Entity State Update Time : $T_{tot} = t_{\lambda 1} + t_1 + t_{\lambda 2} + t_{\lambda 3} + t_{\lambda 4} + t_4$

Case 2 : *Multicast*

Total Entity State Update Time : $T_{tot} = \max (t_{\lambda i} + t_i) \quad i = \{ 1, 2, 3, 4\}$

5.4.2.3 Test Results

This section compares the results of two experiment methods that described in the previous section. Average Total Entity State Update Time is used to compare the difference between two

cases : *Unicast* and *Multicast*. The time taken to update the ESD packet is also dependent on network conditions. The final test results shown here are *the average values of a hundred test result*

5.4.2.3.1 Round Trip Time Check

Tables 5.4 and 5.5 show a number of sample results. Table 5.6 shows the final test results of the comparison of *Total Entity State Update Time* taken to transfer the ESD. Figure 5.30 is a graphical representation of the results whilst Table 5.6 details the update time. The solid line shows the Average Total Entity State Update Time of using Unicast. The dashed line shows the Average Total Entity State Update Time of using Multicast.

As can be seen in this figure, there is a linear relationship between the time taken to update the ESD and the number of nodes. Although the average total entity state update time of the case of Unicast sharply increases much more than that of the case of using Multicast, the greater the number of clients, the more updating time increased by using Muticast. In the case using Multicast, the average value of total entity state update time is about 0.168 sec. In the case using Unicast, the average value of total entity state update time is about 0.194 sec.

This result means that the transmission speed is very slow, although a relatively small amount of data is sent. The actual transmission time is not the issue, rather it is the software overhead. Therefore, this method is inadequate for the experiment that checks the transmission times of the ESD.

	Number of Test	Numbers of Clients				
		1	2	3	4	5
Total Update Time (sec)	1	0.17	0.19	0.19	0.20	0.21
	2	0.16	0.18	0.19	0.21	0.22
	3	0.17	0.17	0.20	0.22	0.23
	4	0.18	0.18	0.19	0.20	0.22
	:	:	:	:	:	:
	5	0.16	0.18	0.20	0.21	0.23
	Avg. Time	0.168	0.18	0.194	0.208	0.222

Table 5.4 A Case of Multicasting by Round Trip

	Number of Test	Numbers of Target Clients				
		1	2	3	4	5
Total Update Time (sec)	1	0.16	0.17	0.17	0.19	0.19
	2	0.17	0.16	0.18	0.18	0.19
	3	0.13	0.15	0.17	0.17	0.18
	4	0.17	0.17	0.16	0.17	0.18
	:	:	:	:	:	:
	100	0.15	0.15	0.16	0.17	0.17
	Avg. Time	0.156	0.16	0.168	0.176	0.182

Table 5.5 A Case of Unicasting by Round Trip

Communication Schemes	Number of Target Clients				
	2	3	4	5	6
Multicast	0.156	0.16	0.168	0.176	0.182
Unicast	0.168	0.18	0.194	0.208	0.222

Table 5.6 Average Total Entity State Update Time (sec) by Round Trip

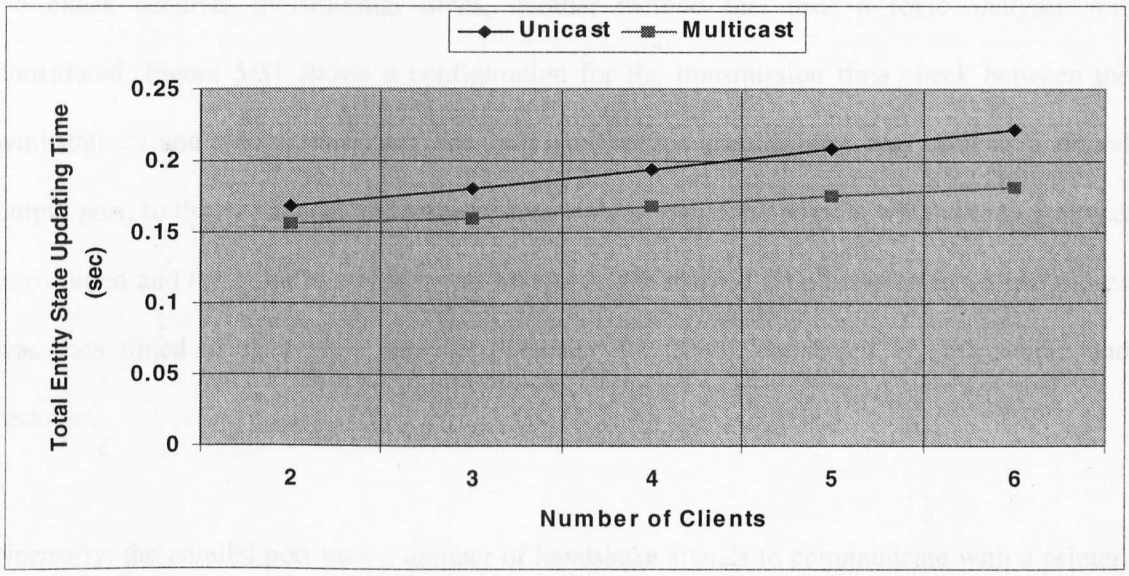


Figure 5.30 Comparison of Average Total Entity State Update Time by Round Trip

5.4.2.3.2 One-way Trip Time Check

To check the transmission time of the ESD from a source node to the target nodes by one-way trip, The experiment was considered at two methods : a method that uses the *time synchronisation* between the workstations and another method that uses a *logic analyser*.

First, to synchronise time between the workstations, the internal clocks on all the Indys were synchronised with the time server of MCC (Manchester Computing Centre) several times in succession. The NTP (Network Time Protocol) was used to synchronise the time. The architecture and design of NTP is described in [Mills, 92]. After several attempts, the largest offset was 0.044 seconds and the smallest was 0.003 seconds. It was not possible to synchronise any better than this, therefore NTP cannot be expected to give accurate results. This method is also inadequate for the experiment that checks the transmission times of the ESD.

To check accurate transmission times, another method that uses a logic analyser was considered. Figure 5.31 shows a configuration for the transmission time check between the workstations and a logic analyser. The Indy workstation parallel port was used as a digital output port, so that the sender and receiver were able to signal the point at which the packet was transmitted and the point at which it was received. The elapsed time between these two pulses was then timed using a logic analyser (Thandar TA 2000), connected to both sender and receiver.

Normally, the parallel port uses a number of handshake signals to communicate with a printer, and eight digital outputs to transfer data. For the purposes of this experiment, only one of these outputs was used: D1. The signal pulses were produced by momentarily taking this output high (logic 1) then low (logic 0).

In order to achieve this, it was necessary to override the existing handshake protocol used by the port. This was achieved by wiring the BUSY signal to Ground, to indicate that the device is always ready to accept data, and wiring /STB to /ACK. This is necessary because the Indy would normally pulse /STB low and then wait for a corresponding /ACK pulse from the printer. In wiring these two pins together, the Indy effectively acknowledges itself.

This makes it possible to use the port as a simple digital output, by opening the port '/dev/plp' as a file stream and writing appropriate values to it. For example, writing the value 255 takes all outputs high, while writing the value 0 takes all outputs low. The time taken to write to the port is assumed to be constant on both Indys, so that timing the elapsed period between the leading edge of the two pulses (sender and receiver) gives an accurate measure of the period between packet transmission and receipt. It is not significantly affected by processing overhead, for example.

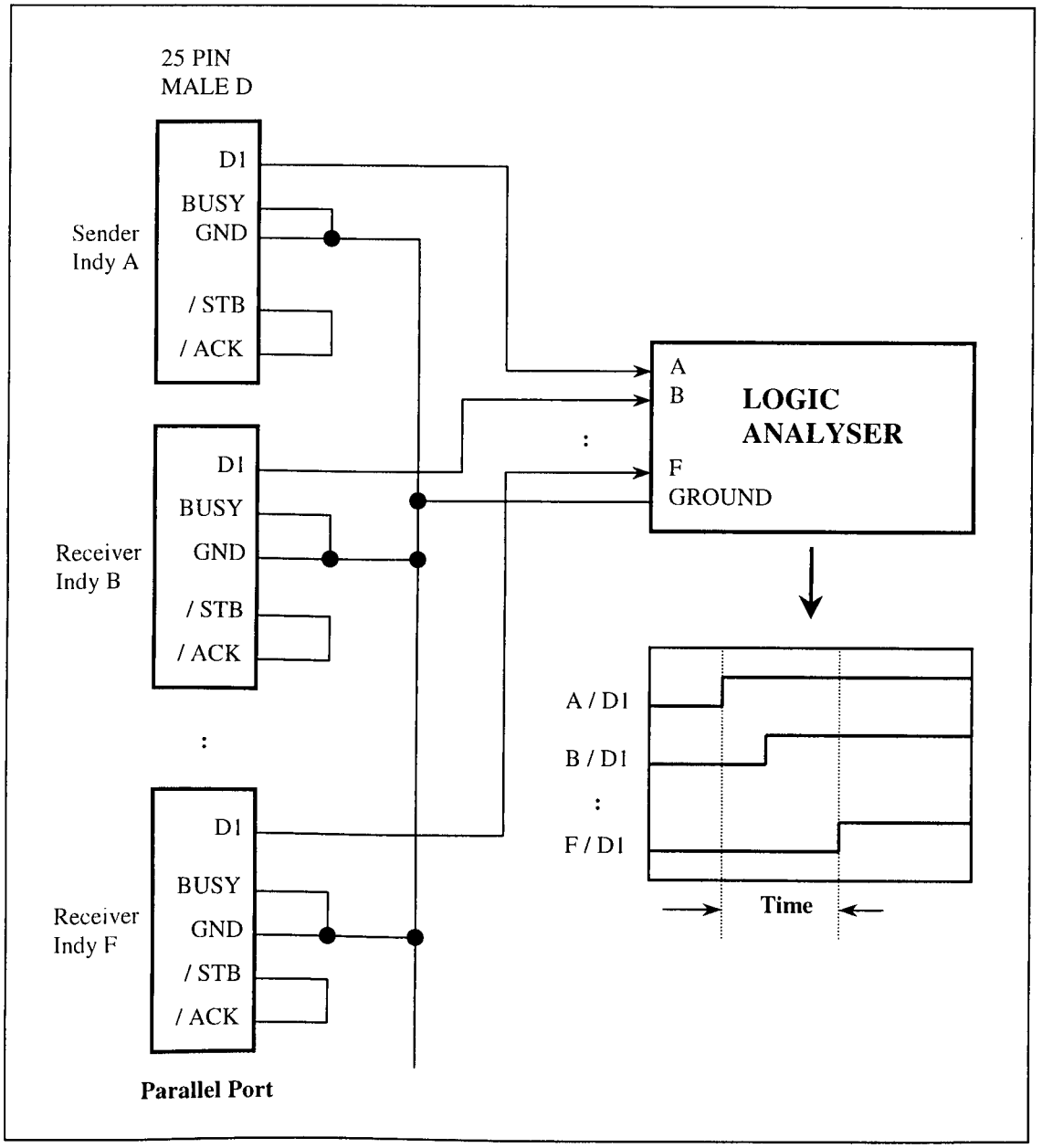


Figure 5.31 A Configuration for Transmission Time Check using a Logic Analyser

The logic analyser produces a graph of signal levels (high/low) against time. It functions by sampling a number of digital inputs at regular intervals. The sampling rate is programmable over a large range from kHz to MHz, so that high timing resolution is possible. There is a trade-off, since the analyser has a fixed buffer size for storing samples, so that using a faster clock

will use the available buffer more quickly while producing more accurate timing results. For the purposes of this experiment, the clock was set to the maximum possible rate that would allow the entire transmit-receive cycle to fit within the buffer. The sample frequency used was 25 MHz, which results in a timing resolution of ± 40 us.

The time interval between pulses was measured by manually marking the start of each pulse with a cursor, and then reading off the elapsed time from the analyser display. The cursor can be positioned exactly at particular samples, so that any error depends purely upon the timing accuracy of the analyser, and is not affected by the manual placement of the cursor.

Although all of the results produced during Total Entity State Update Time Check Tests cannot be presented here, Tables 5.7 and 5.8 show a number of sample results. Table 5.9 shows the final test results. Figure 5.30 is a graphical representation of the results whilst Table 5.9 details the update time. The solid line shows the Average Total Entity State Update Time using Unicast. The dashed line shows the Average Total Entity State Update Time using Multicast.

As can be seen in this figure, there is a linear relationship between the time taken to update the ESD and the number of nodes. Using the Unicast scheme, the update time increases as the number of clients increase. The average total entity state update time of the case of Unicast increases more sharply than the case where Multicast is used.

In the case using Multicast, there is very little difference in the average total entity state update time between the nodes, which means that the users across the network would perceive entity movement to be well synchronised. However, in the case using Unicast, the average difference of total entity state update time between the first target node and fifth target node is about 11.62 ms, which means that the more the number of target clients increases, the bigger the

difference of total entity state update time between the nodes, the users across the network would not see well synchronised movement of entities.

	Number of Test	Numbers of Target Clients				
		1	2	3	4	5
Total Update Time (ms)	1	8.98	10.04	10.92	9.84	10.92
	2	11.28	8.72	8.56	10.72	8.16
	3	10.56	10.68	10.4	11.36	10.76
	4	9.56	10.84	11.24	10.96	10.96
	:	:	:	:	:	:
	100	10.82	9.76	11.44	9.68	11.92
	Avg. Time	10.34	10.37	10.39	10.42	10.48

Table 5.7 A Case of Multicasting by One-way Trip

	Number of Test	Numbers of Target Clients				
		1	2	3	4	5
Total Update Time (ms)	1	16.24	19.52	21.68	25.04	28.2
	2	16.16	19.15	23.12	24.32	28.56
	3	14.88	17.52	21.88	24.28	28.32
	4	16.92	17.96	20.88	23.96	25.88
	:	:	:	:	:	:
	100	16.84	19.12	22.04	24.96	26.52
	Avg. Time	16.09	18.67	21.82	24.43	27.71

Table 5.8 A Case of Unicasting by One-way Trip

Figure 5.32 shows that a Multicast method would produce greater performance benefits as more client nodes are added to the system. One of the most limiting aspects of the real-time implementation is the use of Unicast communications between nodes. Figure 5.32 indicates the performance increase that would be experienced if the updates could be sent to all nodes simultaneously rather than sequentially, using Multicasting, that is.

Communication Schemes	Number of Total Target Clients				
	2	3	4	5	6
Unicast	16.09	18.67	21.82	24.43	27.71
Multicast	10.34	10.37	10.39	10.42	10.48

Table 5.9 Average Total Entity State Update Time (ms) by One-way Trip

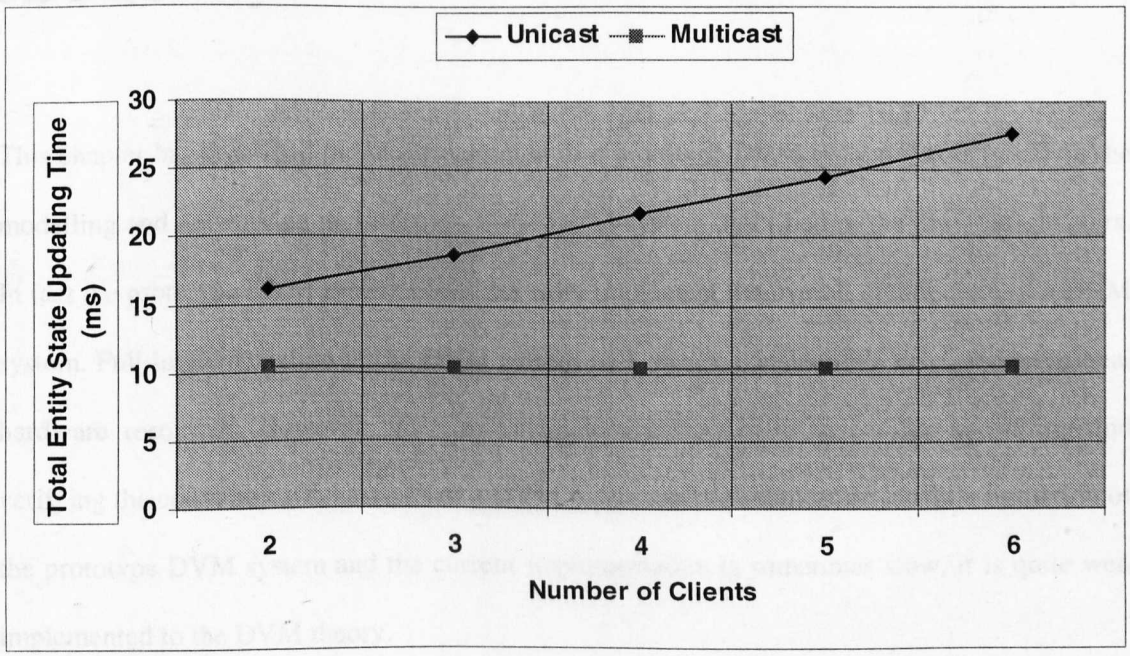


Figure 5.32 Comparison of Average Total Entity State Update Time by One-way Trip

5.4.3 Summary of Evaluation

This section has presented an evaluation of the DVM prototype concentrating on the system's performance. As can be seen from the above results, the system performance is dependent upon the configurations. A DVM model can produce different performance ratings on different configurations and communication schemes. The test results show that a configuration with two servers produces better performance than a configuration with a single server, and a configuration that uses a Multicast scheme produces greater performance than a configuration that uses a Unicast scheme. Although the most limiting factor in a distributed configuration is the network latency that has a substantial impact on performance, by allocating data to multiple server nodes, the system performance can be increased. The distribution of a VE becomes a necessity rather than a luxury.

5.5 Summary

This chapter has described the implementation of a prototype DVM system that is based on the modelling and networking architecture of the DVM system described in the previous chapters. In fact the prototype DVM system could not fully implement the overall architecture of a DVM system. Full implementation of the DVM system will require considerable time, and additional hardware resources. However, the current implementation is sufficient for specifying and verifying the communication aspects of a DVM. Although communication is still a limitation of the prototype DVM system and the current implementation is sometimes slow, it is quite well implemented to the DVM theory.

CHAPTER 6

CONCLUSION

This thesis has described a concept called DVM for designing and constructing distributed virtual worlds, a networking architecture for a DVM system, and an experimental prototype system for the development of DVM applications. This final chapter presents the experiences of the research, the contributions made by this thesis, and outlines the scope for future work. Finally, the chapter concludes with some final comments on the thesis.

6.1 Research Experience

The DVM research left us with three main experiences that show the difficulties of the full implementation of the DVM design concepts, the limitations of our environments including hardware and software, and technical limits for communication.

First, it was almost impossible to implement the full design of the DVM concepts within the limited research time. The full design and implementation of a complex DVM requires huge effort and takes a long time. The DVM design concepts were a broad multidisciplinary research

area that includes many different aspects of computer science. As described in Chapter 3 and 4, a large and complicated DVM includes large numbers of computers and a greater quantity and variety of information should be displayed on the screen in a manner that is effective for participants.

The design of a DVM requires many of the features that are necessary in large, scalable, complex virtual environments. The features of a DVM must adapt to increasing fidelity requirements, changing database needs, higher resolution terrain models, higher communication speed and improving input and output devices. These features require changes in the hosts, protocols, bandwidth, and software architectures. Therefore, the prototype DVM system could not exhaust all the considerations for implementing a DVM but has concentrated the aspects critical to scaling environments, as described in Chapter 5.

Secondly, the important consideration when creating a distributed virtual environment is to know the limitations of our environments including hardware and software. While good architectural design of a DVE system often helps, not every problem can be solved by good design. Better hardware and software environments can afford us greater capability to build a large and complex DVE system. However, for some applications, such environments are not available at prices that participants will support.

In the current environment for the implementation of the prototype DVM system, the hardware limitations are typically more restrictive than the software limitations. So the work concentrated less on the graphical aspects of distributed virtual environments and instead focused on multi-user and communication aspects.

Third, technical limits for communication in a DVM still inhibit general use of a real-time DVM. The importance of real-time performance and the difficulty of its implementation make

this an important issue in DVM research. Communication latency is currently the largest factor responsible for inhibiting progress of real-time DVMs. Perfect communication reliability and synchronisation between nodes has been shown to negatively affect the real-time response of the entire system and is also prohibitively expensive in terms of bandwidth. Therefore, a scalable DVE protocol, which can be used over wide area networks must sacrifice some reliability for real-time performance.

Experience with the demonstration of a DVM prototype system has shown that socket communications using the multicast mechanism of TCP/IP were not guaranteed: sometimes messages sent did not reach their destination, and communication failed by an unexpected signal. However, by experiments, the performance advantages of multicast messages outweigh the risk of occasional lost message, although the protocol does not guarantee delivery. The communication latency and reliability were reasonably satisfactory. The users were able to realise fairly reliable and synchronous service. So DVM experience shows that high-speed, multicast communication should be required for a DVM system that is geographically dispersed.

6.2 Research Contribution of this Thesis

This thesis has presented only limited aspects of DVM system development. However, the DVM system does not support a complete architecture for specifying and verifying properties of a complex DVE. Based on the experience of the prototype development, we are satisfied that the DVM architecture can enable specifying and verifying, in many small steps, the properties of many DVE systems. The DVM system provides two primary contributions, including modelling and networking architecture.

The first contribution comes from the modelling of the DVM system. As described in Chapter 3, a DVM model based on the Client-Server architecture, can be effectively used as a framework or a platform for the further development of large virtual worlds. Compared with a DVE based on a single server and multiple clients, a DVM based on multiple servers and multiple clients has obvious advantages.

A DVE can bring together a large number of participants, and the participants can explore the DVE. In the client-server system based on a single server, a server may have a bottle neck that significantly slows down the DVE when many participants are connected. However, by distributing the connections from participants to multiple servers, it can reduce the possibility of the bottle neck problem.

In the DVE based on a single server, it may be less fail-safe and could even crash the whole system if the server fails. However, in the DVM, although a server fails, the whole system does not crash because the DVE is managed by multiple servers and it is also possible to recover by receiving current entity state data from its own client nodes.

DVE applications have been less successful because of the large amount of data in such applications. A large virtual world requires large quantities of VE data, leading to extensive download times. Therefore, the DVM distributes the large VE over a network of servers to avoid long download times. By experiments, this indicates that the DVM technology can significantly contribute to reducing the amount of data maintained by a single server and the data transmission time.

The second contribution of this thesis comes from the networking architecture of the DVM system as described in Chapter 4. A networking architecture for the DVM system has been designed and implemented. The DVM networking system provides a new paradigm and an

application layer protocol (DCP) for data communication in DVEs. However, communication is still a limitation of the DVM system. Nonetheless, through experiments as described in Chapter 5, the DVM system has shown significant contributions to the area of communication for the DVE system. The DVM networking scheme has been shown to work correctly, and therefore provides the possibility of a DVE system that uses multiple servers. The DVM also offers more efficient protocols, by sending messages only to relevant nodes. These methods proved are able to efficiently manage distributed data with very little latency over short periods of time.

In the DVE there is a lot to be gained by reducing the bandwidth throughout a network of computers because much time will be spent communicating. The DVM uses a method that optimally utilises network bandwidth by downloading only the exact portion of the VE that is necessary for rendering, instead of downloading entire VE data sets. This approach can reduce long download times and improve network utilisation by adaptive use of both rendering and network capacity. The *Communication Protocol for a DVM system* (DCP) provides message formats and an implementation structure for reducing the network bandwidth between nodes in the DVM system. By implementing the DCP protocols, the DVM system run effectively over networks. Further enhancement of DVM communication can be expected accordingly with the future development of technology. The DVM concepts concerning distribution strategy and inter-process communication may serve to aid the development of larger, advanced DVEs in the future.

6.3 Future Work

The DVM system provides many exciting challenges in different areas of computer science, including distributed databases, user interfaces, information retrieval, parallel processing, communication protocols, and so on. Because the DVM concept is fairly recent, its component

technologies still have many issues to be resolved. A complete DVM system requires a continuing developmental effort. Future work lies in expanding the facilities of DVMs to provide more realistic virtual worlds. The following sections consider directions for future work to develop and improve DVMs.

6.3.1 Three Dimensional Implementation

The methods presented in the DVM prototype have been all been two dimensional. As described in Chapter 5, the reason for choosing a two dimensional method for the development of DVM prototype was that all the packet types used in two dimensions are easily extended to three dimensions. However, to be able to improve realism it is likely that full three dimensional models will be required.

6.3.2 Multiple Processes

The current implementation of the DVM prototype generates one process per processing node for the communication and the generation of virtual environments. On a network of computers it is possible to have more than one process per processing node. When communication and processing (eg. displaying graphics) occur concurrently, the DVM can improve performance by running multiple processes on a single node.

6.3.3 Portability

The prototype DVM implementation is currently limited to homogeneous systems, such as IRIX OS. Although the portability is a separate research area, portability is an important area to run the DVM system on various heterogeneous systems. The DVM should be ported to an

architecture independent system. To transfer existing code to the new architecture, higher level programming tools should be developed to extend the lifetime of a program beyond that of the computer for which it was originally developed.

6.3.4 Extending Functionality

The current implementation of DVM is limited in the functionality provided to the user. To improve the reality of the DVM, the extension of new functionality requires more developments that include the user interface, communication, VE generation, entity behaviour, and so on.

6.3.5 Enriching the behaviour of entities

The prototype DVM included relatively simple behaviour of entities. To control the movement of entities through a more realistic three dimensional space, a completely dynamic DVM model is required. This requires further work to enrich the behaviour of entity operations, allowing for constrained manipulation along arbitrary axes.

6.3.6 Audio Integration

As described in Chapter 5, the prototype DVM concentrated on the visual aspects, audio integration was not considered. However, integration of audio communications with the DVM is required to get a realistic VE. Sounds associated with entities in the VE are essential. Therefore, a more advanced DVM model should include techniques which accurately generate realistic sounds.

6.4 Final Comments

As distributed virtual environments become richer and more complex, the need for technology for developing these DVEs will grow dramatically. This growing need will result in a fundamental change to the future of distributed virtual environments. The DVM system has presented a new paradigm for designing and implementing the DVEs. The DVM system has illustrated how communications, views, data, and processes can be distributed and managed within a DVE. Therefore, it is hoped that the DVM architecture will constitute a good start towards addressing the demands of DVE applications.

APPENDIX

DCP Packet Formats

This appendix shows twelve type formats of packets that present in chapter 4. These were not included in the main body of thesis because these packet formats are changeable.

C.1 Primary Join Request (PJR)

PJR messages (opcode 1) contains Client_Identification (*Client_ID*) field. Every client has a unique *Client_ID*. The *Client ID* is a sequence of bytes in *netascii* terminated by a zero byte. A node that receives netascii mode data must translate the data to its own format.

2 bytes	string	1 byte
Opcode	Client_ID	0

C.2 Primary Join Grant (PJG)

PJG messages (opcode 2) contains Operation_Time (*Optime*) field. The *Optime* means the global time. The *Optime* is assigned a positive integer that represents the seconds, for example, in the range 1 through 100.

2 bytes	6 bytes
Opcode	Optime

C.3 Secondary Join Request (SJR)

SJR messages (opcode 3) contains the Current_Area_Number (*Curr_Area_#*) and Direction (*DIR*) fields. The *Curr_Area_#* presents the local VE area number. The *Curr_Area_#* is assigned a positive integer, for example, in the range 1 through 9999. The *DIR* shows a positive integer which presents the location number (east : 1, west : 2, south : 3, north : 4) of next local VE area.

2 bytes	8 bytes	3 bytes	2 bytes
Opcode	Optime	Curr_Area _#	DIR

C.4 Secondary Join Grant (SJG)

SJG messages (opcode 4) contains Next_Server_Name (*Nxt_Sv_Name*) field. The *Nxt_Sv_Name* field allows the server to name up to 10 characters.

2 bytes	8 bytes	string
Opcode	Optime	Nxt_Sv_Name

C.5 Release Request (RRQ)

RRQ messages (opcode 5) contains just *client_ID* field.

2 bytes	8 bytes	string
Opcode	Optime	Client_ID

C.6 Release Response (RRS)

RRS messages (opcode 6) contains just *server_ID* field.

2 bytes	8 bytes	string
Opcode	Optime	Server_ID

C.7 Data Request (DRQ)

DRQ messages (opcode 7) contains just *Data_Type* (*Dtype*) field. The *Data_Type* (*Dtype*) field contains the integer numbers (GPD : 1, EPD : 2, ESD : 3, TXD : 4, Initial Data : 5, Next Area Data : 6) associated with the types which are requested.

2 bytes	8 bytes	2 bytes
Opcode	Optime	Dtype

C.8 Data Response (DRS)

The DRS message (opcode = 8) has a data_number (*Data_#*) field. The *Data_#* field contains the positive integer number (GPD : 1, EPD : 2, ESD : 3, TXD : 4, All Done : 5) which presents the type of acknowledgement. The Message_Operation_Time (*MsgOptime*) contains the Optime of the received message.

2 bytes	8 bytes	2 bytes	8 bytes
Opcode	Optime	Data_#	MsgOptime

C.9 Geometry Polygon Data (GPD)

Each GPD message (opcode = 9) has an *Area_Number* (*Area_#*), *Polygon_Number* (*Poly_#*), *Offset*, and *Vertex* fields. The *Area_# field* on the GPD message contains the integer numbers (East : 1, West : 2, South : 3, North : 4) for directions. The *Offset* is assigned a positive integer that means the distances from the origin of global VE. The *Vertex* fields are assigned positive integer numbers that presents vertices of each polygon.

2 bytes	8 bytes	2 bytes	2 bytes	2 bytes	2 bytes		2 bytes
Opcode	Optime	Area_#	Offset_X	Offset_Y	Poly_#[*0*]	..	Poly_ #[*n-1*]

2 bytes		2 bytes	2 bytes		2 bytes		2 bytes	
Txt_#[*0*]	..	Txt _#[*n-1*]	Vt_X[*0*][*0*]	..	Vt_X[*n-1*][*0*]	..	Vt_X[*0*][*3*]	..

2 bytes	2 bytes		2 bytes		2 bytes		2 bytes
Vt_X[*n-1*][*3*]	Vt_Y[*0*][*0*]	..	Vt_Y[*n-1*][*0*]	..	Vt_Y[*0*][*3*]	..	Vt_Y[*n-1*][*3*]

2 bytes	2 bytes		2 bytes		2 bytes
Vt_Z[*0*][*0*]	..	Vt_Z[*n-1*][*0*]	..	Vt_Z[*0*][*3*]	..	Vt_Z[*n-1*][*3*]

C.10 Entity Polygon Data (EPD)

The EPD message (opcode = 10) has *Polygon_Number* (*Poly_#*), *Patch_Number* (*Patch_#*), *Vertex* (*Vt*), *Normal* (*Nor*), and *Colour* (*Col*) fields. These fields are assigned positive integer numbers.

2 bytes	8 bytes	2 bytes		2 bytes	2 bytes		2 bytes
Opcode	Optime	Poly_#[*0*]	..	Poly_#[*n-1*]	Patch_#[*0*]	..	Patch_#[*n-1*]

2 bytes		2 bytes	2 bytes		2 bytes	2 bytes		2 bytes
Vt[*0*][*0*]	..	Vt[*0*][*2*]	Vt[*n-1*][*0*]	..	Vt[*n-1*][*2*]	Nor[*0*][*0*]	..	Nor[*0*][*2*]

	2 bytes		2 bytes	2 bytes		2 bytes
..	Nor[*n-1*][*0*]	..	Nor[*n-1*][*2*]	Col [0]	..	Col [2]

C.11 Entity State Data (ESD)

The ESD message (opcode = 11) has Data_Type (*Dtype*), Area_Number (*Area_#*), Entity_Type (*Otype*), Entity_ID (*Obj_ID*), Offset, Position (*POS*), Direction (*DIR*), Speed (*SP*) and Dead-Reckoning_flag (*Dflag*) fields. The *Etype* field contains a positive integer number (Plane : 1, Car : 2, Human : 3) which presents the type of entity. The *Obj_ID* and *SP* fields are assigned a positive integer, typically in the range 1 through 9999. The *POS* fields are assigned positive real numbers. The *DIR* field is assigned a positive integer, typically in the range 1 through 360. The *Dflag* field sets flag (On : 1, Off : 0).

2 bytes	8 bytes	2 bytes	4 bytes	4 bytes	8 bytes	8 bytes
Opcode	Optime	Dtype	Area_ #	Etype	Ent_ID	Offset_X

8 bytes	8 bytes	8 bytes	8 bytes	6 bytes	8 bytes	2bytes
Offset_Y	POS_X	POS_Y	POS_Z	DIR	SP	Dflag

C.12 Texture Data (TXD)

The TXD message (opcode = 12) has Texture Number (*Txt_#*) and Imagefile (*ImgF*) fields. The *Txt_#* field is assigned a positive integer, typically in the range 1 through 9999. The *ImgF* field contains the image data of texture.

2 bytes	8 bytes	2 bytes	n bytes
Opcode	Optime	Txt_#	ImgF

GLOSSARY

AM	Audio Manager. Audio Manager is a process that provides services related to the audio display. Defined in section 3.2.1.9.
ASCII	American Standard Code for Information Interchange.
Broadcast	A name for a Data Link Layer frame which is addressed to all stations attached to that Data Link.
CN	Client Node. Client Node manages the local VE, interact with I/O devices and perform graphics rendering. The client node establishes communications with the server node and through it or directly, transmits the data to the server or the other client nodes. Described in section 3.2.1.2.
CM	Computation Manager. Computation Manger is a process that runs on the server and client nodes. It is in charge of the computation of all events. Defined in section 3.2.1.6.
DCP	Inter-process Communication Protocol for a DVM system. The DCP is an application-layer protocol that permits the integration between nodes in the DVM system. The DCP describes the format to send and receive

messages between nodes.

DD — Decimal Degrees.

DM — Database Input / Output Manager. Database Input / Output Manager handles a collection of data forming the virtual environment. Its primary function is to load the initial data. Described in section 3.2.1.5.

DMS — Degrees, Minutes, and Seconds

DRQ — Data Request. The *Data_Type* (*Dtype*) field contains the integer numbers associated with the types which are requested.

DVE — Distributed Virtual Environments. When VEs are distributed on multiple computers across a network, it is called as a Distributed Virtual Environment (DVE).

DVM — DVE systems with Multiple-Servers. DVM is a model of DVE based on the client-server model with multiple servers. DVM is an experimental platform for the development of distributed virtual environments, user interfaces and applications based on shared 3D synthetic environments.

ENTITY — One of the essential system components. The VE state is represented by the sum of each entity's state. Some VEs contain a large number and variety of active entities such as vehicles, humans, and planes, which

are under active control. There are also passive entities, such as buildings, trees and hills.

EPD — Entity Polygon Data. The EPD message (opcode = 8) has *Polygon_Number* (*Poly_#*), *Patch_Number* (*Patch_#*), *Vertex* (*Vt*), *Normal* (*Nor*), and *Colour* (*Col*) fields.

ESD — Entity State Data. The ESD message (opcode = 9) has Data_Type (*Dtype*), Area_Number (*Area_#*), Entity_Type (*Otype*), Entity_ID (*Obj_ID*), Offset, Position (*POS*), Direction (*DIR*) and Speed (*SP*) fields.

GPD — Geometry Polygon Data. The GPD message (opcode = 7) has an *Area_Number* (*Area_#*), *Polygon_Number* (*Poly_#*), *Offset*, and *Vertex* fields.

IM — Interaction Manager. Interaction Manager is a process that provides services related to interaction devices. IM is usually in charge of the input and output of all the devices. Described in section 3.2.1.8.

IP — Internet Protocol. The standard used for sending the basic unit of data, an IP datagram, through the Internet.

LAN	Local Area Network. A network connecting various electronic devices in a localised geographical area such as a single office building or campus.
MM	Master Manager. Master Manager maintains all the state variables of entities and the geometric model of the virtual environment. It is also responsible for routing messages appropriately. Described in section 3.2.1.3.
NM	Network Input / Output Manager. Network Input / Output Manager is responsible for dealing with the network. It is a process that makes it possible for users using the same VE to communicate with each other. The NM keeps a list of other nodes that it is connected to. Described in section 3.2.1.4.
Multicast	Multicast provides one-to-many and many-to-many delivery services for applications that communicate with several other nodes simultaneously.
PJG	Primary Join Grant. The Primary Join Grant message contains Operation_Time (*Optime*) field.
PJR	Primary Join Request. The Primary Join Request message contains Client_Identification (*Client_ID*) field. Every client has a unique *Client_ID*.

Port	The identifier used to select a particular process within a TCP/IP host.
RM	Rendering Manager. Rendering is a process that runs on the client node. It provides services for rendering and managing the visual representation, including shaded, texture-mapped surfaces, and coloured lighting. Described in section 3.2.1.7.
RRQ	Release Request. The Release Request message contains just *client_ID* field.
SJG	Secondary Join Grant. The Secondary Join Grant message contains Next_Server_Name (*Nxt_Sv_Name*) field.
SJR	Secondary Join Request. The Secondary Join Request message contains the Current_Area_Number (*Curr_Area_#*) and Direction (*DIR*) fields.
SN	Server Node. Server Node provides a bridge between its client nodes and databases. The server node is also responsible for maintaining the data consistency between its client nodes. Described in section 3.2.1.1.
Socket	The concatenation of IP address and TCP port which together specify a particular process or service within the Internet.
TCP	Transmission Control Protocol. A transport protocol providing connection-oriented, end-to-end reliable byte data transmission in

	packet-switched computer subnetworks and internetworks.
Topology	A description of how stations on a network connect to a cable.
TXD	Texture Data. The Texture Data message (opcode = 10) has Texture Number (*Txt_#*) and Imagefile (*ImgF*) fields.
UDP	User Datagram Protocol. This protocol within the TCP/IP Protocol Suite is transaction oriented, and delivery and duplicate protection are not guaranted.
Unicast	Unicast establishes communication between two nodes.
VE	Virtual Environment. Virtual Environment is the thing being partly or wholly simulated by the VR system.
VR	Virtual Reality. Virtual Reality is used to refer to the whole subject area, its hardware, software, applications, etc.

REFERENCES

[Barrus, 96] Barrus J. W., Waters R.C. and Anderson D.B., " Locales and Beacons: Efficient and Precise Support for Large Multi-User Virtual Environments," MERL Technical Report TR95-16a, 1996.

[Birman, 87] Birman K., Josep T., Schmuck F., "ISIS - Distribute Programming Environment, Version2.1 -User's Guide and Reference, 1987.

[Bishop, 92] Gary Bishop, "Research Directions in Virtual Environments, Report of an NSF Invitational Workshop," Computer Graphics, Volume 26, Number 3, August 1992, pp. 166-170.

[Blau, 92] Brian Blau, Charles Huges, Michael Moshell and Curtis Lisle, "Netwrked Virtual Environments," Computer Graphics, Special Issue on !992 Symposium on Interactive Computer Graphics, March 1992, PP 157-160.

[Boursier, 82] Patrice Boursier, Michel Scholl, "Performance Analysis of Compaction Techniques for Map Representation in Geographic Databases," Computer & Graphics Vol. 6, No 2, 1982, pp. 73-79.

[Broll, 95] Broll W. "Interacting in Distributed Collaborative Virtual Environments," Proceedings of the IEEE VRAIS'95-Virtual Reality Annual International Symposium, 1995, pp. 184-185.

[Bryson, 94] S. Bryson and S. K. Feiner, "Research frontiers in virtual reality," SIGGRAPH '94 Proceedings, 1994, pp. 473-474.

[Burdea, 94] G. Burdea and P.Coiffet, "Virtual Reality Technology," John Wiley & Sons, Inc, 1994.

[Calvin, 93] Calvin J., Dickens A., Gaines B., Metzger P., Miller M., Owen D, "The SIMNET Virtual World Architecture," Proceedings of the IEEE VRAIS'93 Conference, 1993, pp. 450-455.

[Carlsson, 93] Carlsson and Hagsand O., "DIVE-a Multi-User Virtual Reality System," Proceedings of the IEEE VRAIS'93 Conference, 1993, pp. 390-400.

[DIS, 94] DIS(1994) Standard for Distributed Interactive Simulation- Application Protocols. Version2.0, Fourth Draft. Institute for Simulation and Training, University of Florida, U.S.A.

[DMSO, 97] DoD High Level Architecture. On-line paper. http://hla.dmso.mil/

[Durlach, 95] Durlach, Nathanliel I. and Mavor, Anne S., "Virtual Reality: Scientific and Technological Challenges, National Academy Press, Washington, D.C. 1995.

[dVS, 95] dVS for UNIX Workstations User Guide Rev. 3.0, pp.13-15.

[Flipcode, 99] Online Paper, Daily Game Development News & Resources, (Web address =http://www.flipcode.com/network/)

[Gigante, 93] Michael A. Gigante, R A Earnshaw, H Jones "Virtual Reality Systems," Academic Press Ltd, 1993, pp. 8-13,16.

[Gossweiler, 94] Gossweiler, R., and others, "An Introductory Tutorial for Developing Multiuser Virtual Environments," Presence, 3(4), Fall 1994, pp. 255-264.

[Green, 99] M. Green, MR Toolkit. On-line paper. Computer Graphics Research Laboratory, Department of Computing Science, The University of Alberta, Edmonton, Alberta, Canada T6G 2H1 (Web address = http://web.cs.ualberta.ca/~graphics / MRToolkit.html)

[Grimsdale, 93] Grimsdale C., "Virtual Reality Evolution or Revolution," proceedings of the third annual conference on Virtual Reality, London, April 1993, pp. 15-18.

[Hagsand, 96] Contact address:Olof Hagsand The distributed Collaborative Environment Group, Swedish Institute of Computer Science, Box 1263, s-164 28 kista, Sweden.
(Web address = http://www.sics.se/dce/dive.html)

[Halsall, 92] Fred Halsall, "Data Communications, Computer Networks and Open Systems," Addison-wesley, 1992, pp. 143-161.

[Hawkes, 96] Rycharde Hawkes, "A Software Architecture for Modelling and Distributing Virtual Environments," Phd Thesis, The University of Edinburgh ,1996.

[Hensgergen, 96] Eric Van Hensgergen. On-line paper.
(Web address = http://www.csh.rit.edu/~airwick/dist.html)

[IEEE, 87] IEEE Standard for Radix-Independent Floating-Point Arithmetic.

[Kalawsky, 91] Kalawsky, R.S. "Visually coupled systems to virtual reality: an aerospace perspective," In computer Graphics '91, Blenheim Online, 1991.

[Kanarick, 91] Kanarick, C.M. "A technical overview and history of the SIMNET project," Proceedings of the SCS Multi-conference on Distributed Simulation, 1991, pp. 104-111.

[Katz, 94] Warren Katz, "Military Networking Technology Applied to Location-Based, Theme Park and Home Entertainment Systems," Computer Graphics, Volume 28, Number 2, May 1994, pp.110-114.

[Liskov, 93] Liskov B. "Practical Uses of Synchronised Clocks in Distributed Systems," Distributed Computing, 6, 1993, pp. 211-219.

[Loral, 92] Loral Systems Company, "Strawman Distributed Interactive Simulation Architecture Description Document," Vol. 1, Advanced Distributed Simulating Technology Program Office, Orlando, Florida, March 1992.

[Macedonia, 95] Macedonia, Michael R., Zyda, Michael J., Pratt, David R., Brutzman, Donald P. and Barham, Paul T., "Exploiting Reality with Multicast Groups," IEEE Computer Graphics & Applications (revised from appearance in the VRAIS '95 Proceedings), September 1995, pp.38-45.

[Mastaglio, 95] Thomas W. Mastaglio and Robert Callahan, "A Large-Scale Complex Virtual Environment for Team Training," IEEE , July 1995, pp. 49-55.

[Mills, 92] Mills, D., "Network Time Protocol (Version 3) specification, implementation and analysis," RFC 1305, University of Delaware, March 1992, 113 pp.

[Mullender, 93] Sape Mullender, "Distributed Systems," Addison-Wessley, 1993, p.2.

[Nees, 94] Richard J. Nees, "Electronic Image Communications," Learned Information, Inc., Medford, NJ, 1994, PP. 35-62.

[NPSNET, 95] NPSNET(1995) NPSNET IV.7J System Overview. Manual.

[Poils, 95] Michael F. Poils, Stephen J. Gifford, and David M. Mckeown Jr., "Automating the Construction of Large Scale Virtual Worlds," IEEE , July 1995, pp. 57-58.

[Pratt, 93] David R. Pratt, "A software Architecture for the Construction and Management of Real-Time Virtual Worlds," PhD Thesis, Naval Postgraduate School, Monterey, California, June 1993.

[Pryce, 96] Nat Pryce, "The Use of Behaviour Specifications in Distributed Virtual Environments," Imperial College London, April 1996.

[Rheingold, 91] Rheingold, "Virtual Reality," New York: Summit Books, 1991.

[RFC, 87] RFC 1001 "Protocol Standard for a NetBios Service on a TCP/UDP Transport : Concepts and Methods," March 1987.

[RFC, 89] RFC 2415 "Simulation Studies of Increased Initial TCP window size," September 1998.

[Sawler, 91] Sawler, Robert, Matusof, "Issues Concerning Cue Correlation and Synchronous Networked Simulators," AIAA, 1991.

[Schmalstieg, 96] Dieter Schmalstieg and Michael Gervautz, "Demand-Driven Geometry Transmission for Distributed Virtual Environments," Computer Graphics, Volume 15, Number 3, 1996, pp.421-431.

[Serrano, 94] Juan Flaquer,Alejando M. Garcia-Alonso, Nicolas Serrano, "Solving the Collision Detection Problem," IEEE Computer Graphics & Applications, May 1994, pp.36-42.

[Shaw, 93] Shaw, C. and Green, "The MR Toolkit Peers Package and Experiment," IEEE Symposium on Research Frontiers in Virtual Reality, October 25-36, 1993, San Jose, CA: pp. 463-469.

[Sheridan, 93] Sheridan, Zeltzer, "Virtual RealityCheck," Technology Review, Oct. 1993, pp. 19-28.

[Silicon, 92] Silicon Graphics, Inc.(1992), "Graphics Library Programming Guide Volume I & II".

[Snowdon, 95] Snowdon D.N., "AVIARY: A Model for a General Purpose Virtual Environment," PhD Thesis, Department of Computer Science, University of Manchester, 1995.

[Stallings, 93] William Stallings, "Computer Organisation and Architecture," 4th ed., Macmillan Publishing Co., 1993, pp. 251-291.

[Steven, 94] Macedonia, Michael R., Zyda, Michael J., Pratt, David R., Barham, Paul T. and Zeswitz, Steven, "NPSNET : A Network Software

Architecture for Large Scale Virtual Environments," Presence, Vol. 3, No 4, Fall 1994, pp. 265-287.

[Sturman, 94] David J. Sturman, David Zeltzer, "A Survey of Glove-based Input," IEEE Computer Graphics & Applications, January 1994, pp. 30-31.

[Stytz, 96] Martin R. Stytz, "Distributed Virtual Environments," IEEE Computer Graphics & Applications, May 1996, pp. 19-20.

[Talpede, 95] Talpede R. and Ammar M.H.(1995) Single Connection Emulation(SCE): An Architecture for Providing a Reliable Multicast Transport Service. Proceedings of the International Conference on Distributed Computing Systems, Ch. 62: pp.144-151.

[VRML, 97] Virtual Reality Modelling Language, ISO/IEC DIS 14772. On-line paper. (Web address = http://vag.vrml.org/VRML97/DIS/)

[Wang, 94] Q. Wang, "Networked Virtual Reality," Master Thesis, University of Alberta, 1994.

[Wang, 95] Q. Wang, Mark Green, and Chris Shaw, "EM-An Environment Manager For Building Networked Virtual Environments," Proceedings of the IEEE VRAIS'95.

[West, 93] A.J. West, T.L.J. Howard, R.J. Hubbold, A.D. Murta, D.N. Snowdon, D.A. Butler, "AVIARY - A Generic Virtual Reality Interface for Real

Applications ,"Virtual Reality Systems, Academic Press, 1993, pp. 213-236.

[Wilson, 92] Michael J. Zyda, David R.Pratt, James G. Monahan, Kalin P. Wilson, "NPSNET: Constructing A 3D Virtual World," ACM, Proceedings of the 1992 Symposium on Interactive 3D Graphics, pp.147-155.

[Zyda, 95] Donald P. Brutzman, Michael R. Macedonia and Michael J. Zyda, "Internetwork Infrastructure Requirements for Virtual Environments," Computer Science Department, Naval Postgraduate School, Monterey California, USA. Available at (Web address = http://www.stl.nps.navy.mil/~brutzman/vrml/vrml_95.htm/)