SUPPORTING QUALITY OF SERVICE IN DISTRIBUTED VIRTUAL ENVIRONMENTS

SHARATH RAMARAJ

A Thesis Submitted to the Faculty of

The Department of Computer Science

In Partial FulFillment of the Requirements for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

Denis Gracanin, Ph.D., Thesis Committee Chair

Ing Ray Chen, Ph.D.

Chang Tien Lu, Ph.D.

VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

Falls Church, Virginia

SEPTEMBER 19 2003

Keywords: Distributed Virtual Environments, Quality of Service, Real-time Optimizaton, Resource Allocation, Operations Research, Dynamic Programming, Neural Networks

Supporting Quality of Service in Distributed Virtual Environments

by

Sharath Ramaraj

(Abstract)

We present a resource allocation perspective to Quality of Service in Distributed Virtual Environments. The user of a DVE system will have improved Quality of Service if he/she is allocated the right amount of resources at the right time. Instead of allocating resources on a static basis, we adopt a dynamic need based resource allocation scheme that provides real-time resource allocation. Optimal resource assignments are calculated offline and a neural network is trained with the knowledge of optimal solutions from the offline Operations Research Techniques and it is then used to deliver near-optimal resource allocation decisions in real-time. We also present a case study of network bandwidth allocation and prove the usefulness of the technique.

To My Mother, Father, Advisor, the Lord Almighty

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Denis Gracanin for his enthusiasm in carrying out this research topic with me. His insights were invaluable in elucidating ideas that we explored for this project. This thesis would not have taken its shape if not for Dr. Gracanin's timely and thorough guidance. I feel fortunate to have a chance to work with such an excellent mentor and would like to express my gratitude to him.

My sincere thanks also go to Dr. Dusan Teodorovic , who went beyond the call of duty as a researcher and researched with me on real-time optimization topics. I cherished my intellectual relationship with him over the last two years.

The contributions of my colleagues and friends, Kalyan Pasupathy and Vijay Varadarajan were also considerable in the completion of this research. Thank you for your insights and discussions into the related topics in my research.

I would also like to thank my Parents and my Brother's family for all the timely support and drive they provided to help me complete my thesis. Special thanks to my Brother's kids for the free entertainment to help me relax.

And lastly, to Him who is my Lord and Savior, to Him be all the glory.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

The realization of virtual environments for collaborative work and navigation has been achieved through academic and commercial initiatives for the purposes of defense simulations and gaming. One important characteristic of all these virtual environments is the ability to bring together widely distributed and heterogeneous sets of users and computers together at the same time and perform tasks that would otherwise need the physical presence of each of these users. Widely distributed, in this context, means geographic separation of the networked terminals. The goal of a widely distributed architecture has been achieved in many systems. Shared local networks like LANs and point to point wide area networks like the Internet have been used as the medium of distribution in all of these distributed virtual environment systems.

To maintain realism, messages, event packets, state updates of all entities and regular heartbeat packets need to be communicated between users of the virtual environment. In a typical distributed virtual environment the network utilization of the application due to these messages is reasonable for a few users. But as the number of users increases, the network utilization increases and hence the network gets clogged with packets from the DVE application. In this situation, users of the DVE application do not get updates regularly and hence the scenes rendered on their screens becomes jittery. In this situation, the user is said to have a low Quality of Service (QoS). The goal of this research is to improve the QoS of users of DVEs.

Quality of Service of a DVE user can be measured in terms of the sense of presence and awareness of other users and entities that the user experiences in the DVE application. To support a realistic sense of presence

- The user's computer needs to receive updates regularly.
- It should also possess the capability to process these messages and visually render it to the user screen

Advances in graphics performance of hardware and computational power of processors have taken care of the second need pointed out above. The first need of receiving timely updates has not been realized for many end users with low-end network connections. This situation can be alleviated in many ways.

- The communication architecture can be carefully chosen to improve delivery times and latency.
- The system and software architectures can be designed to improve fault tolerance, consistency and latency
- Message filtering can be implemented so that each client gets only the messages of interest to it.
- Improved path prediction can be applied to reduce messages on the network
- Scalable event handling mechanisms can be incorporated to accommodate many users on the network to be able to use the application
- Improved rendering techniques can be used to reduce messages in transit
- All the above mentioned points can be seen as contenders for a resource pool and the resource allocation for that pool can be handled as part of a unified distributed architecture on a real-time and mathematically viable basis

The last point needs some explanation as that would form the basis of the research presented in this Thesis. Resource allocation, reservation and disbursement are some of the most important aspects of providing a high and fair Quality of Service in distributed Virtual Environments. The resources can be network bandwidth, general processor time, mathematical processor time, graphic processor time among various others. These resources have to be allocated in such a way that every user experiences a reasonable good Quality of Service so that they can perform their tasks effectively. In a Distributed Virtual Environment this means that the user must be able to navigate, interact and collaborate in the DVE without experiencing jittery scenes or slow responses to participant actions. It is in this context that we define Quality of Service in Distributed Virtual Environments.

1.1 Defining Quality of Service

High Quality of Service is defined as a state of the DVE system in which all users have the optimal amount of resources when they are needed to enable them to perform their tasks effectively. It is also important to mention that request for resources should be denied when it is known that the user does not need it.

This resource allocation process needs to be managed in such a way that it optimally allocates resources to all users. To determine the optimality of the resource allocation decisions, we have Operation Research Techniques. But these do not produce results in real-time. Nevertheless, we can still use the knowledge of optimal solutions gained through these methods to train a neural network. This trained Neural Network can produce near optimal decisions in real-time.

This research proceeds by reviewing previous work on each of these aspects of DVE applications. We go on to prove that resource allocation decisions can be made by Neural Networks to satisfy the real-time requirements of an interactive Virtual Environment. Based on this resource allocation model we propose a system architecture for DVEs that could improve the Quality of Service provided to each user without clogging the network.

CHAPTER 2
LITERATURE REVIEW

2.1 Awareness, Presence and Interaction

Awareness is a very important concept in distributed virtual environment systems. As stated in [1] awareness is an understanding of the activities of others, which provides a context for your own activity. Awareness management helps collaboration between users, by suppressing all the awareness information about users and objects that are not relevant for the current users' collaborative task. Awareness management also has an important role in the scalability of this kind of systems. By limiting the amount of information that must be processed by each user, awareness management can be a very effective mechanism for reducing resource usage, like network bandwidth and computer processing power. Given the importance of awareness management, different policies have been used depending of the systems requirements. and goals. RING [2] users are only aware of the objects they can see. This policy is adequate for environments that contain several visual barriers such as walls and doors, but it performs badly in densely populated open space environments.

Antunes et al. [3] propose an abstraction for awareness management in collaborative virtual environments as a solution to the problem of awareness information propagation in virtual environments. They proceed by defining awareness in an abstraction that has scope, scopeExpression, Interested party, InterestExpression, Interest space and awareness policy as the participants in the awareness management scheme. The Awareness Management abstraction defines three types of collaborations:

- register /un register scopes in a interest space
- register /un register interested parties in a interest spaces
- association/ dissociation of interested parties with scopes.

A scope is registered in an interest space through its addScope operation. An InterestSpace obtains the scope expression from the scope, operation getExp, and registers it in the awareness management policy. To unregister a scope from an interest space it is necessary to invoke its removeScope operation. This operation will remove the correspondent scope expression from the space's awareness management policy.

SPLINE [4] partitions the environment in spatial regions called locales. Each user is aware of all the objects in the current locale and in the immediate neighbors. The partitioning

of the environment is a very flexible mechanism for structuring the virtual environment. In SPLINE each locale defines its own independent coordinate system. The virtual environment results from the connection of several locales. Each connection between two locales defines a 3D transformation that describes the relations between the locale's coordinate systems, which allows the creation of non-Euclidean environments. This approach also eases the extension of the environments, since it is only a matter of defining new locales and connecting them to the existing ones. Finally, the locales approach also provides an effective mechanism for controlling awareness. Allowing the awareness of the adjacent locales gives users the notion of spatial continuity, increasing at the same time the system scalability. There are some situations where different policies could be more useful. For instance, in some situations one could be interested only in the current locale, and in other situations, in the current locale and perhaps a small subset of the adjacent locales.

NPSNET [5] divides the environment into fixed size regions called cells and each user defines an area of interest trough an aura. Users are only aware of those cells, which their aura intersects with. The size and shape of the cells were chosen taking into account the target application domain, military simulation. The goal of awareness management in this system is to improve system scalability and to support a large number of simultaneous users.

To really interact with virtual environment, it is necessary to use object picking and manipulation and as it is a collaborative environment the object on one Client must be seen moving on all the other connected Clients. In order to provide greater interaction within the virtual environment proximity detection should be implemented. Proximity, Collision Detection and Response, and Gravity are some important factors that will govern the interactivity level in situations. Although gravity is not directly combined into the same task, it does work hand in hand with Collision Detection and Response.

MASSIVE-1 [6] supports the spatial model of interaction [7], [8]. In this model users defined auras and interaction between two users can only happen when the user's auras intersect. The model also uses the concepts of focus and nimbus to compute the awareness level that a user can have of another user or object. The focus represents an observing object's interest in a particular medium. The nimbus represents an observed object's projection in a particular medium. The awareness level that an object A has of an object B in a particular medium M is a function of A's focus and B's nimbus in M.

In MASSIVE-2 [9] the spatial model of interaction is extended with the third-party objects concept that represents objects that can affect other objects and users awareness levels by changing their values of aura, focus and nimbus. The spatial model of interaction is perhaps one of the most complex and flexible awareness management model. It is suitable

for controlling user interaction in large-scale virtual environments. However, the model is too much focused on the spatial aspect of this system, making it difficult to manage awareness using semantic or organization considerations.

In MASSIVE-3 [10] an extension to the SPLINE [4] model is adopted. In this extension the environment is also divided into locales and the locales can be connected through boundaries. Each locale can have several aspects each representing a certain type of awareness information. The awareness management is performed, by selecting the locales and correspondent aspects that are relevant to a particular user. The system allows the programmer to select, or even adapt, the policy for selecting the relevant locales and aspects. Since locale's aspects can be arbitrarily defined it is possible to support awareness management taking into account organizational associations between the objects of an environment. Due to the existence of different application requirements it is necessary to support different awareness management policies. All the mentioned systems only offer support for a particular policy or family of policies. Only MASSIVE-3 allows some degree of adaptation, by letting the programmers to choose the policy for selecting the relevant locales for a particular user. However, the adaptation is confined to a particular kind of awareness policy, based on locales and aspects. It is not possible to use different policies. This problem is normally due to the lack of proper design abstractions that are, not only, able to solve the problem at hand, in this case awareness management, but also able to support the several variations that exist for the problem solution.

To cope with this problem, Antunes et al. [3] present an object-oriented awareness management abstraction that is flexible enough to support different awareness management policies. Instead of trying to provide a one-size-fits-all solution for awareness management, the defined abstraction allows for different policies to be defined, and for programmers to choose the most appropriate policy for the application being developed.

Musse et al. [11] propose a method for crowd modeling in virtual environments with autonomous agents. This crowd model aims at providing a way to populate virtual worlds giving a realistic sense of autonomous group presence. The crowd motion is based on goals. Each group has a list of goals to follow which is distributed in different ways to the individuals. This approach to crowd behavior represents an emergent behavior, i.e. the global effects which arise as a function of the local rules applied. As an example, consider the situation where a population of one hundred individuals in a virtual world is needed, then the system can define a crowd formed by one hundred agents and their crowd behavior. However, in the absence of a crowd model, a hundred real participants would be needed running at the same time and connected together with a network to share the same environment.

At a high information level, a crowd is treated as a single entity formed by agent groups who have specific behaviors. The simulation of these group's behaviors is possible if the crowd system knows the simulation environment viz. the goals, interest points and obstacle positions. In this case, the system is able to compute the regions over which the group positions must be distributed. For this scheme of crowd specification to work, some of the main requirements are Environment Information, the distribution of crowd information by groups and the distribution of group information by agents. Also crowd behaviors like seeking goals and relationships between individuals, flocking behavior and collision avoidance behavior need to be defined. This approach is claimed to reduce the amount of system resources needed when there is a need to model a larger environment with a large number of participants. This is a more of a crowd simulation approach.

Greenhalgh et al. [12] provide a way for crowd participation in virtual environments. Their approach is concerned with the mechanics of managing awareness and communication among human participants in crowded DVEs. This contrasts with previous research into crowd behavior modeling and simulation. This is concerned with participation in crowds as opposed to simulation of them. Specifically, they have introduced a framework for reasoning about and developing different kinds of crowds with different effects on spatial awareness and communication. This framework is based on a spatial model of interaction called third party objects. The key points of this framework are that crowds can play a significant role in configuring communication and awareness thereby introducing aggregate views of their members, the use of awareness to activate crowds supports crowds whose effects are based on membership and also on an observer's level of focus and that crowds may be mobile or fixed and may be created and destroyed at the system level, the application development level or by participants themselves. Crowd modeling and participation techniques are basically aggregation techniques that aim to improve scalability by and message transmission by treating the crowd as a single entity. For these techniques crowds of participants and autonomous agents act as participants and the crowd itself may be autonomous.

The VIVA system [13] proposes that awareness of users is needed not only in VR systems but also in other media. They argue that awareness of others and their activities is important for all media and applications, not just DVEs. The authors also point out that the web and most document handling applications are unaware of others in the workgroup or community and this limitation runs counter to major CSCW findings about how work is really achieved, and to positive experiences with DVEs. Such awareness, whether back-grounded or fore-grounded, needs to be constantly available. They briefly present a Web application (CRACK!) that 'provides people-awareness' in the Web, and between the Web and VR's,

and that should be extensible to other standard document handling programs, such as word processing and spreadsheet.

Interaction mechanisms are essential to support real-time natural interaction in the Collaborative Virtual Environment (DVE) with limited network bandwidth and computer processing power thus making the development of an efficient interaction management mechanism key to DVE development. In distance based interaction management, the distance between a user and the entities around him is used to decide the user's ability to know the entities. When the network bandwidth is limited, only the data from the entities nearest to the user is actually sent to that user's host. There are two kinds of applications using this distance based management mechanism. The first one is to use spatial distance to enable the interaction (eg Spatial Aura). Aura is an abstract sphere around the entity and move with the entity, like a magnetic field. When the aura collision is detected between two entities, the interaction between them is enabled. Another form is to enable the interaction through the horizon count method. The horizon count indicates the maximum number of entities that the user is prepared to receive updates from. Their distances from the avatar of the user will sort the entities. Another kind of application is to use spatial distance to calculate the LOD.

Acuity based interactions use the ratio between size and distance. Anything bigger or closer would be more relevant; anything smaller or farther away would be less relevant. In this mechanism, every user has an acuity setting, which indicates the minimum size-over-distance ratio required for an entity to be known to him.

Region based interaction division is widely used in the DVE systems, as the locale in Spline [4], the zone in VRClass [5], the hexagonal cell in NPSNET [5], and the closed community and building/room in NetEffect [4]. It also appears as third party objects in MASSIVE-2 [9]. An example of group based interaction management is the NPSNET. The virtual environment of the system is logically parted by associating spatial, temporal and functionally related entity classes to form network multicast groups. Each user has a local Area of Interest Manager (AOIM) that is used to identify the multicast groups that are potentially of interest to them, and to restrict network traffics to these groups.

The key concepts of the spatial management mechanism [7], [8] include medium, aura, awareness, focus, nimbus and adapters. Medium is a communication type such as audio, visual or text. Aura is an object-specific and medium-specific subspace in which interaction may occur. Awareness is used to quantify one object's significance to another object in a particular medium, which depends on focus and nimbus. Focus represents an observing object's interests in a particular medium. Nimbus represents an observing object's wish to

be seen in a given medium. Adapters are objects that can modify other object's auras, foci and nimbi in order to customize or modify its interaction with other objects. After the interaction is enabled by the aura collision, awareness is negotiated through combining the observer's focus and the observed entity's nimbus.

Wang et al. [14] propose a scheme of behavior based interaction management in the Smart3UCD virtual environment system. In SmartCU3D, an Internet DVE system, a behavior based interaction management with adaptive message routing mechanism is used. In this mechanism, the message routing in the system becomes adaptive to the application and the users' runtime interaction. It is achieved by giving the Object-Oriented style collaborative behavior description along with the 3D environment definition for every individual DVE application. The collaborative behavior description defines the functional roles selected by the users in the application, and the behaviors for every functional role. They also give the definitions for the roles and behaviors of the shared interactive objects within the 3D environment. When the interactive entities interact with each other in the DVE, the required message for coordinating the interaction and maintaining the world's consistence will be generated according to the specific behavior definition for the interactive entity's functional role. And the message will be routed among the affected interactive entities in the DVE. After such a message arrives, the interactive entities respond it with corresponding behavior based on their behavior definition. The authors argue that this approach enables the system to control the information flow from higher levels. Thus it gives a way to exploit the potential ability on managing the system resource and improves the real-time performance, scalability of the system.

The HIVE [16] architecture and data structures enable collaboration between 2D and 3D clients simultaneously. Geometric information is shared in the form of scene graphs using VRML 2.0 as the 3D modeling language for dynamic manipulation of the scene and extension with prototype nodes. In a 2D window-based conferencing application, the HIVE presents images of each participant in the conference, a shared whiteboard, and a shared browser that can be used to review viewgraphs or other data. The HIVE incorporates a variety of techniques such as speaker highlighting to provide additional non-verbal cues about who is speaking. The high fidelity audio module of the HIVE connects multiple clients together in a multicast group and enables sidebar conversations apart from the main group. Communication between HIVE clients is done using the Voyager mobile agent system. CAVERNSoft [17] is used for communication between the HIVE and VisualEyes [18]. Collaborative VisualEyes [18] is a retrofitted version of VisualEyes enabling global scale collaboration between VisualEyes applications. Collaborative VisualEyes clients share 3D scene graph information

by directly linking individual data nodes over a communication channel implemented with CAVERNSoft. CAVERNSoft creates a distributed shared memory for client applications that subscribe to data indexed by keys. It supports connecting an unrestricted number of participants at different host machines into a collaboration session. It supports client hardware with varying abilities to render graphical and audio data. It integrates shared applications into the collaboration session. It integrates a high-fidelity audio module in the environment and transmits speech between conference participants.

Lloyd et al. [19] introduce the mechanism of "formations" as an explicit way of defining and using groups of participants, defined through some expression of mutual interest, within DVEs. They aim to ease and enhance interaction for individuals, for example, in enhancing communication and navigation so that an individual can easily hear, speak to and keep up with a group of fellow participants as they move through a virtual world. They also aim to facilitate application and system management, for example, providing convenient ways of moving whole groups of participants at a time. The structure of a formation is comprised of an object of interest (the focal point of the formation. An object in the DVE that is the common interest for participants, and serves to bind them to the formation), Members (A participant that has indicated interest in the object of interest becomes a member of the formation and is subject to its influence and effects, and is also able to affect the formation.), Non-members (Participants that exist within the environment but do not express an interest in the object of interest. They are liable to a different awareness of the formation to members), Leaders (Formations may have a leader. The leader may have the potential to influence the effect of the formation in a different way or to a different extent than the normal members) and Projection (A way in which potential members can identify the presence and function of a formation within an environment.).

2.2 Communication Architecture

2.2.1 Protocol level Support

Protocol support at lower layers of the network protocol stack is not feasible because of the fact that those layers do not facilitate resource reservation. Many protocols have been proposed for Network and transport layers for negotiation. Dash is a Transport level protocol based on session reservation protocol. TENET is based on RCAP, RIP, RMTP and CMTP. Graceful adaptation is a key feature of the TENET approach. ST-II consists of two protocols ST(for data forwarding) and SCMP (for control messages). SCMP adds and removes nodes

from the broadcast tree and negotiates Quality of Service (QoS). The HeiTS protocol works on top of ST-II. It includes graceful degradation with 4 reliability classes. The Berkom approach is like HeiTS with more QoS parameters like Transport Service Data Unit, max size and MMTP. Application layer protocols like Resource reservation Protocol (RSVP) assume scalable media. These protocols are generally suited for any distributed multimedia application and can be extended to virtual environment applications as well.

The most commonly advocated and most powerful protocol for Distributed Virtual Environments is the IEEE standard DIS [20]. DIS is a network protocol based on IP multicasting and used for distributed simulation of military scenarios. It is so far the only existing standard for shared virtual environments. DIS is used by the NPSNET [14] system. It defines a number of units (PDU's) which are transferred to all other participants of the simulation to transfer the state of each object. While the protocol is very suitable for the specific application area, most PDU's are not suitable for general purpose virtual environments. The concept of transferring the state of each object frequently allows new participants or temporarily disconnected participants easily to catch up with the current state of the virtual world. However it puts a permanent network load on the network, even if object states are not changed.

This has been implemented in the VirtualArch [21], [22] platform. This protocol uses UDP for updating scenes in the VR environment. Since the Distributed Virtual Environment does not tolerate retransmissions or acknowledgements UDP is better suited. The Greenhalgh et al. [23] QoS architecture uses UDP over a reliable multicast network. It assumes the existence of multicast protocols like IGMP available in the network routers connected to clients. All clients are also expected to register themselves with multicast supporting routers to send and receive messages and updates.

The PARSEC [24], [25] gaming environment uses a hybrid combination of UDP and TCP based on DIS. It uses TCP for the connection establishment phases and then when the simulation is on, it uses UDP for updates and synchronizations. Houatra [26] implements QoS constrained event communications in the Continuum platform. Event communication protocols require scalability and reliability under real-time constraints (like Guaranteed service and/or best effort). Synchronization and message ordering should be enforced so that all users see the updates in the same order in which they were created. Group management and session control are also mandatory for event communication protocols.

ISTP (Interactive Sharing Transfer Protocol) [27] uses a heterogeneous communication infrastructure to support shared virtual environments. It is built on four existing underlying protocols: HTTP, RTP, TCP/IP and multicast UDP. While HTTP is used to transfer

files and other large amounts of data, RTP is used for the transmission of streams such as audio. Short messages, such as state synchronization are realized via unreliable multicast UDP. Additional TCP connections between servers and clients are used for reliable message transmission and recovery from UDP failures.

DWTP [28] the Distributed Worlds Transfer and Communication Protocol is an application layer protocol for connecting large scale virtual worlds and multiple users on the Internet. DWTP provides a set of daemons in order to realize the individual requirements for realizing distributed VR applications. Based on different network protocols it provides the basis for a universal protocol for shared virtual worlds. Events are used to keep distributed copies of shared virtual worlds consistent by transmitting appropriate synchronization data. Events may contain any kind of data and are usually rather small. The application can specify the required reliability for the transfer of events. Messages are actually a number of predefined events such as used for joining or leaving a shared virtual world. Some messages can contain additional data (e.g. for chatting, transmitting URL's, or sending requests). Files are heavy weight (large) objects, which require a reliable transfer. Files might be transferred peer-to-peer or to a group of recipients. Streams are used to transmit a continuous flow of data as used for audio or video. Streams do not require reliable transmission. DWTP provides a simple interface for these data types to the application or virtual environment, hiding the underlying network protocols. DWTP works over multicast and unicast setups. It uses a combination of TCP and UDP over multicast or unicast IP networks.

The Virtual Reality Transport Protocol [30] (vrtp) framework aims to provide the necessary infrastructure to support DVE with multi-user capabilities. The framework defined by VRTP focuses on the specification of the components along with their interfaces and interactions with one another. This approach gives the developers of choosing what they wish to adopt in their systems, without enforcing any constraints. The Virtual Reality Transfer Protocol (vrtp) [30] proposes a component framework for DVE systems, thereby providing a solution to the existing development problems. Each component has a well-defined role.

- Universal Platform is the backbone component of the vrtp framework. It consists of a flexible kernel with dynamic extensibility capabilities allowing the system to evolve in run-time. This component also provides low-level networking services in conjunction with a minimal directory service. The Universal Platform (UP) is designed according to the Application Layer Framing principles, thereby delegating to the application more network awareness and responsibility in the management of the network related resources.

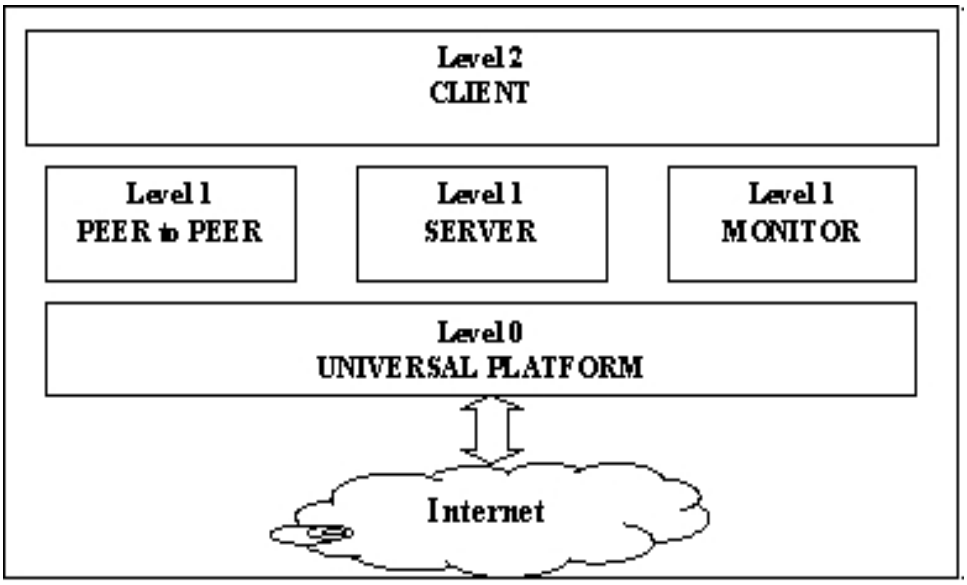

Figure 2.1: VRTP Architecture

- Monitor. The capability for monitoring has long been overlooked despite the demand of end-users to understand the reasons behind the degradation of their presence in a DVE, leaving them without the possibility of negotiating parameters to improve the properties they consider essential in detriment of others. This component will monitor the network status, providing the necessary feedback for the application to adapt as required taking into account the expectations of the user. In current network multimedia applications, such as video conference, the trend is to involve the user into Quality of Service (QoS) decisions regarding the network.

- Server. The server component is responsible for the management of heavyweight objects, retrieving them from the network when necessary. These objects correspond to significant amount of data, requiring reliable connection oriented communication for transmission. However this component may either be inactive or active, depending upon its role. In the case of it being inactive, the component is only responsible for retrieving data from the network; otherwise the component may assume an active role that enables end-users to host their own worlds within the domain of a DVE.

- Peer-to-Peer. This component handles lightweight objects, corresponding to state, control and event data being composed by small messages. The QoS associated to transmission varies according to the requirements of the semantics of the data. In the case of control data, there must exist reliability of delivery while in absolute state

dissemination non-reliability is permissible. However it is important to evaluate the impact of the reliability mechanisms on the latency of system. This component is also responsible for group management, which is necessary to alleviate the demand on network resources in exchange for computational resources without compromising the overall experience of the end-user. The effects of grouping entities according to their temporal, functional and spatial interests, have demonstrated positive results and it has become a common denominator in existing VE systems.

- Client. The former components represent the underlying system, while the client component encapsulates the application with all the particularities of a solution. The implementation may range from a simple plug-in for a browser to a full complex DVE for simulation purposes. Existing implementations of the Universal Platform layer include Bamboo [30] and JADE [31]. Both solutions are based on strong software engineering principles, which have previously existed in other fields of computer science, namely operating systems and middleware. Although Bamboo and JADE share common foundations in terms of functional objectives, they have divergent design architectures and consequently different implementations. These differences result from the adoption of different philosophies, which emerged from the fact that Bamboo adopted C++ as its implementation language, while JADE is based upon Java. The choice of implementation language should not exert any crucial impact to the final design of a system. However, Java consists of more than just a development language, offering a platform with inherent base functionality. So while Bamboo focuses on developing all the necessary mechanisms to support dynamic loading across different platforms and basic resource management, JADE moulds the functionality available in Java towards the same purpose. This results in JADE providing extensions not considered in Bamboo. Similar to JADE, Bamboo consists of a small flexible kernel, which is ever present in the background. The kernel allows dynamic code management of a system, in terms of loading and unloading segments of executable code. This management is only attainable by following the proposed design principles of partitioning the overall system functionality into components denominated Modules. How the code is organized into Modules depends entirely upon the design of a particular DVE system. The flexibility characteristic stems from the fact that provided that the Module interface is respected, then code of any nature may be integrated into the kernel. Both JADE and Bamboo provide minimal resource management, thereby avoiding problems normally associated with concurrency.

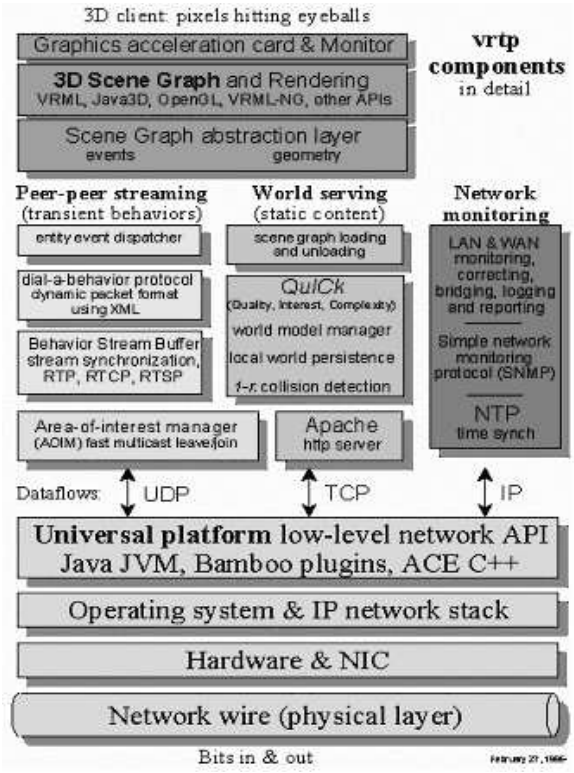

Figure 2.2: VRTP Components in Detail: Depicted from web3d vrtp working group

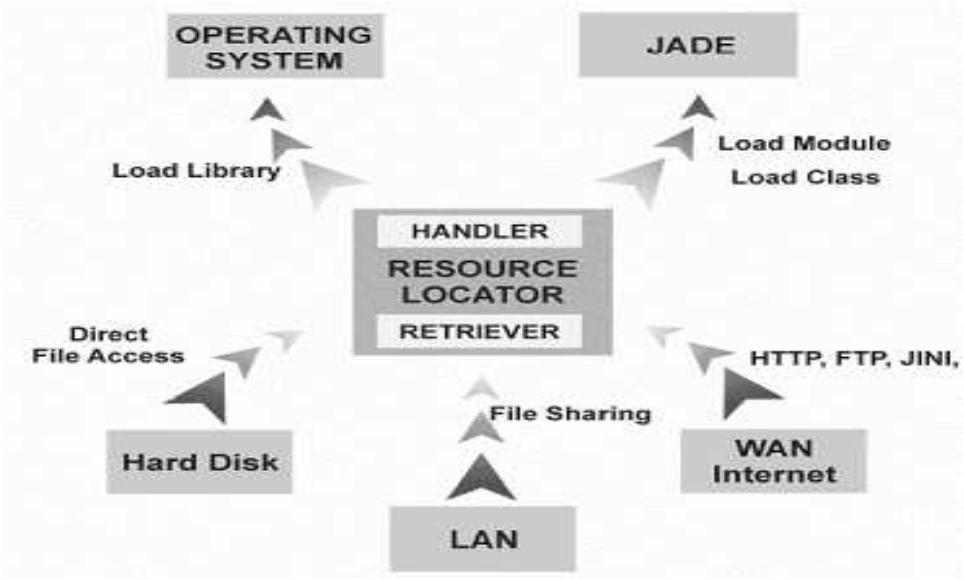

Figure 2.3: JADE Architecture: Depicted from [31]

2.2.2 Multicasting Systems

To better utilize network bandwidth, several DVEs use multicasting. Multicasting is a means of making the same information available to anyone on a network who subscribes to it, without replicating the data for each participant. Surprisingly, because of the nature of multicasting, this has at times appeared to be undesirable. Typically multicast traffic on one LAN is isolated or blocked by routers from traversing to other LANs within an intranet. In more elaborate network setups, LANs can be set up explicitly to vary the scope of multicast traffic to a local area, a region, or globally. Some examples of systems using multicasting are DIVE[32] developed by the Swedish Institute of Computer Science, HIVE [16] developed by HRL laboratories and NPSNETV [5] developed by the MOVES institute.

The DIVEBONE [33] developed by the Swedish Institute of Computer Science is a hybrid multicast/unicast communication infrastructure. For many users per site it retains the benefits of native multicasting. At the expense of some unicast traffic it allows a number of distributed sites to be connected relatively easily and reliably and without requiring access to the normal MBone. The proxy server allows multicast unaware processes to join regular multicast sessions by acting as a message multiplexor for all connected applications. The proxy server allows MBone unaware networks to support multicast sessions by acting as a message multiplexor for all connected proxy servers.

Multicast VRML Interchange Protocol MVIP [34] is another protocol introduced for shared virtual environments. Robinson et al. [34] introduce a method for encapsulating MVIP information over RTP. When one moves in a shared world, the others sharing the world must see this movement, and they must know how to map an Avatar, and other services onto this movement. The architecture is shown in 2.4. There are two classes of data that must be interchanged over a network to allow for this sharing: Time-sensitive data and informational data. In MVIP the time sensitive data flows over RTP while the informational data flows over RTCP. Time-sensitive data includes position/orientation changes that must be updated in a timely manner. These messages contain parameters (position and orientation with standard VRML meanings) that identify the current location (at time of packet generation) for that avatar.

HLA [35], [36] is an integrated architecture, which is developed to provide a common architecture for modeling and simulation. The HLA consists of three components, including federation rules, interface specification and the object model template (OMT). Federation rules describe the responsibilities of federates and their relationships with the run-time infrastructure (RTI). The interface specification identifies how federates will interact with the

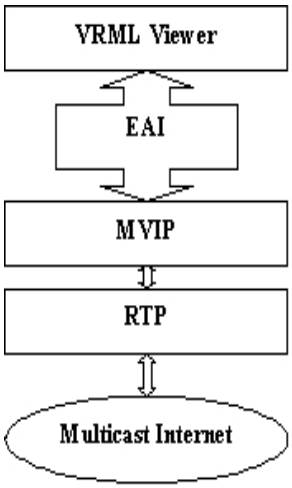

Figure 2.4: MVIP Architecture

federation via the RTI. The OMT provides a common framework for all objects and interactions managed by a federate. The RTI also provides sufficient services for simulation systems and specifies the federate interface using the following languages: CORBA IDL, C11, Ada 95, and Java. Moreover, the RTI services are classified into six categories that describe the interface between federates and the RTI. For example, federation management services allow the creation and destruction of a federation execution and data distribution management services provide information to the RTI for supporting efficient routing of data. DDM provides a flexible and extensive mechanism for isolating publication and subscription interests further. DDM services are used by federates to reduce the transmission and the reception of irrelevant data. A Federate is an application participating in a federation. A Federation: is a set of interacting federates that are used as a whole to achieve some specific objectives. A dimension is a named coordinate axis segment. All dimensions must be based on the same coordinate axis and have the same lower and upper bounds. A routing space is a multidimensional coordinate system through which federates can declare their intention by area of interest to send or receive objects' attribute values. A range is a continuous interval on a dimension defined by an ordered pair of values. An extent is a sequence of ranges, one for each dimension in a routing space. A region is a set of extents bounded to the same routing space, and is defined as a subspace within the routing space. Basically, the RTI provides a default region for every routing space and the default region covers the entire routing space. In general, a region used for the update of instant attributes or sending interactions is called an update region. The notation of an update region is expressed as UR. Similarly, a region

used for subscription of either attributes or interactions is called a subscription region.

2.2.3 Remote Procedure Calls

Remote Procedure Calls are a more dynamic way of communication between collaborating processes especially suited for virtual world models. Alice, a complex DVE system, has been developed on top of DIVER [37]. DIVER is a Virtual Environment (VE) development system, which transparently distributes rendering and input processes, implicitly decoupling the application computations from the rendering computations. DIVER's graphics-independent module spawns server processes on other machines and communicates function calls from the application using Remote Procedure Calls (RPCs).

2.2.4 Mobile Agents

Mobile agents are pieces of code, which can migrate from one machine to another under their own control. Simply put they can control their own time and place of execution. These code fragments travel over a network like any normal data packet. But they carry more information than static data packets. These are basically dynamic data. They carry more information with less data. There are many types of agents like collaborating agents, negotiating agents. These possibilities make them apt for DVE architectures. According to Peters et al. [57], software agents are most relevant and applicable for the use in real-world domains. Their intelligent behavior enables them to automate and delegate cognitive tasks that were not feasible for machines in the past. Agents support users individually during a session. They act as a representative of their 'employer' in the task they are assigned to. Three major attributes may describe the behavior of software agents in general Autonomy, Cooperation and Learning.

The DVECOM system proposed by Choukair et al. [22] is a classic example of a mobile agent system working in DVE systems. This system uses mobile agents to collect orthogonal update metadata. This metadata contains criticality, pertinence, degradation degree tolerance of a particular update message in the DVECOM system. The VirtualArch platform basically supports Agent Migration within the client server architecture.

Jung et al. [38] describe a virtual environment for multiple autonomous and instructable agents. The distributed simulation framework allows for easy integration of novel agents into the environment. Through the distributed implementation of simulation, visualization, and agent control the workload of the simulation is shared among a number of computer systems. The architecture is client server architecture with the agents acting as autonomous clients.

Agent control clients communicate with the server via a fixed protocol. Messages from the server define the agents' perception. By sending messages to the server, agents can act in the world. The three components of the architecture are the server (for holding the world model, transporting messages to/from the clients, generating sensor data), Applet client (for accepting messages from the server, interpreting these and controlling the virtual world) and Control client (for defining the behavior of an agent and interpreting perceptions)

QoS negotiation architectures have a potential use for mobile agents in negotiating QoS parameters between user preferences and basic availability of system resources. This potential has been exploited in the QoS management architecture proposed by Hafid. The CINEMA [39] system implements XNRP (Negotiation and Resource Reservation) protocol based on the mobile agent paradigm. XNRP proceeds in a phased manner with user agents negotiating between user preferences and system resource availability. This architecture is especially useful because of the fact that mobile agents are not bulky but carry a lot of information, which can be useful in processing DVE messages, state information, synchronization data, and causal ordering of scene rendering updates.

2.3 System and Software Architecture

Multi User virtual environments have gained significance due to their usefulness in military simulations and entertainment value. Their value has also improved along with the improvements to computer graphics, audio and real-time processing, multi-user games have also improved in terms of visual interactivity. To accommodate the great demand for reality and interactivity, information critical for rendering of remote entities must be issued as frequently as possible. This simple scheme is however inadmissible because of the following two factors [40]:

- The ever increasing requirement for state updates of remote entities will overload the simulation engine; and
- Network latency and limited bandwidth will put an upper bound to the rate at which entities can exchange information with each other.

These two factors lead to the issue of software architectural scalability. Scalable software architecture can be defined as a general framework that supports a virtual environment with an increasingly larger number of concurrent dynamic entities and/or players without fundamental modifications to that architecture. The design of a software architecture must take

the above two factors into account because faster computers and networks alone will not satisfy the requirements for increasing the number of participants in a virtual environment over time. To improve QoS for networked virtual environments it is important to have a scalable network and software architecture which can handle the required load and also eliminate the unwanted traffic and message packets in the network. There are many approaches proposed to solve this problem.

Cai et al. [41] propose a scalable architecture for improving QoS and scalability in general in networked gaming environments. A client (i.e., a player or a participant) will join a server according to the position of the avatar it controls. To support secured communication for interactions and accounting information as well as to speedup periodic update messages (e.g., position updates), a hybrid communication scheme using both TCP and IP multicast is used between clients and the associated server. The RunTime Infrastructure (RTI) services enable the communication among servers. The High Level Architecture (HLA) [35], [36] Data Distribution Management (DDM) is employed to limit the amount of communication between the servers. In addition, the Ownership Management (OM) is also employed to implement the need for transferring the avatars between servers.

Fully distributed and centralized architectures can be combined to form a distributed client-server architecture where there are multiple servers providing services to the clients. The distributed client-server architecture retains the advantage of the simple centralized (client-server) architecture. In addition, by sharing the load of computation and communication amongst multiple servers, more players would be able to participate in the same virtual environment. There are two principal approaches to divide the work amongst servers in a distributed client-server architecture [42]

- Virtual World Subdivision: The virtual world is partitioned into logical groups and each group is assigned to a server. Clients connect to the server according to the group to which its avatar belongs.

- Participant Subdivision: Clients are grouped and assigned to a server according to the physical distance between the client and the server. Clients connect to the server according to the geographical area in which they are located.

In the virtual world subdivision approach, a server only maintains a part of the entire virtual world. But, a client may need to migrate to a different server if it changes its logical group. In the participant subdivision approach, a client will connect to the same server throughout the game. But, each server may need to maintain a copy of the entire virtual world. In [5], three possibilities for partitioning a virtual are suggested world:

- Spatial Partitioning: This is based on partitioning the virtual world into areas, which can be processed in parallel and independently. Therefore, the players in the same part of the virtual world can interact with each other through the same server.

- Temporal Partitioning: Entities that require the same rate of update are grouped together. Groups requiring a higher update rate can then have a larger share of the total network bandwidth.

- Functional Partitioning: Entities are grouped according to functional classes (e.g., a battalion in war-gaming). Those in the same functional class can then communicate with each other frequently through a multicast group.

Foster et al. [43] propose a high performance software architecture for meta-computing applications, which integrates diverse communication substrates, transport mechanisms, and protocols, chosen according to where communication is directed, what is communicated, or when communication is performed. This architecture allows multiple communication methods to be supported transparently in a single application, with either automatic or user-specified selection criteria guiding the methods used for each communication. The realization of this approach requires solutions to challenging problems: separate specification of communication operation and communication method; identification of applicable communication methods; selection from among alternative methods; and the incorporation of multiple communication methods into an implementation. This architecture allows programmers to specify communications in terms of high-level abstractions such as message passing or remote procedure call, while supporting diverse low-level methods for actual communications. Communication methods can be associated with individual communication operations, and the selection of an appropriate method can be guided by both automatic and user-specified criteria. The architecture also incorporates solutions to various problems that arise when multiple communication methods are incorporated into a single implementation. Central to this multi-method communication architecture is an abstraction called a communication link, which provides a concise, mobile representation of both the target of a communication operation and the methods used to perform that operation. Multi-method communication addresses the need for integrating the usefulness of various options in Transport Mechanisms, Network protocols, QoS, Interoperability of tools and security.

W-VLNET [44] uses a client-server architecture with a server database of the virtual world. The world is fully replicated over all the clients. Any changes to the world are sent to all the clients through the server. All clients have a copy of the static objects.

Any dynamic objects are distributed through the server. There is a growing interest in flexible system infrastructures, which can be modified (often dynamically) to support new and various mechanisms and techniques. High availability long-lived shared virtual worlds will demand mechanisms that allow the whole system to evolve while it is still running, for example adding new network protocols and other system behaviors. The NPSNET-V system [5] provides a particular pattern of extensibility (leveraging Java's class loading capabilities) that allows network protocols, objects classes and graphical models to be dynamically added to the running system. At a lower level, the Bamboo system [30], [45] provides a dynamic module loading system that spans multiple languages (including C++ and Java), and that is intended to support extensible virtual environment systems of this kind.

The DEVA [46] system adopts a model of long-lived server environments, which dynamically load and compose objects from behavior fragments, to create flexible and composable object and world behaviors. The MASSIVE [6], [9], [10] series of architectures by Chris Greenhalgh et al. support different architectures. Some of them are the aura-based replication over peer-to-peer unicast communication in MASSIVE-1 [6]; region (third-party) based replication and abstraction over hybrid peer-to-peer and server-based multicast in MASSIVE-2 [9]; and policy-driven locale based replication over adaptive client-server communication in MASSIVE-3 [10]. Greenhalgh et al. propose an approach which uses distributed event filters and a notion of deep behaviors to reduce the amount of messages in the network. In this architecture all messages go to a commonly known API event pipe over the network.

The distribution architecture in the DEVA [46] system is logically a client /server architecture, which provides a single definitive locus of control for the Virtual Environment using its server component, with 'mirrors' of the entities being maintained in each client process. Behind the scenes, however, DEVA manages the delegation of control dynamically to the most appropriate parts of the system thereby achieving the highest fidelity of perceptual and causal coherency attainable for the application at hand. The 'server' is in fact a cluster of processors running identical processes called 'server nodes' that together form a single multi-threaded parallel virtual machine capable of processing large numbers of entities. The server provides a computing resource for multiple virtual environments, and maintains a far heavier processing load than any one user's client could manage at any one time. A networking layer provides lightweight position independent messaging between entities. Entities are created in and managed by the server node processes, and client processes such as a visualizer or user application - connect to the server to interact with and obtain state information about the entities. The server is persistent. Each server node is designed to manage multiple entities. Each entity is assigned a unique 'pool ID' - an offset into the list

of entities managed by a given server node. The virtual server and pool IDs uniquely define the location of an entity in the server. When an entity is created a hash function is applied to the entity's name to obtain a second virtual server. This virtual server manages the name of that entity; separate virtual servers manage the name of the entity and the entity itself. The addressing mechanism allows entities to dynamically migrate across server nodes to help balance-processing load. When an entity moves it only needs to inform its name manager of its new location. The migration of both entities and name management allows server nodes to be dynamically added ,and removed from a running server.

The infrastructure has a ConstraintsFilter to enforce an explicit ordering on all events (part of MASSIVE-3's exploration of consistency mechanisms), "LocalNowRouting" filter which sends a copy of the event to the local pending pipe for immediate enactment, and another copy of the event to the sending pipe for distribution. and a Unicast that sends the event to the server over a TCP connection, an UpdateSceneGraph which enacts the event on the local database replica, an "EventPipeRouting" filter that passes the event to another specified event pipe and Multicast sends the event to all connected clients (with the optional exception of the originating client). The API Pipe is always present in the system; the other pipes are added dynamically as the event filters request them. Deep behaviors are used to specify the low level system behavioral aspects of each shared world data item. This is done explicitly using annotations in the code. By making (the declarations of) deep behaviors part of the shared state of the virtual world this system exploits the normal (default) data distribution mechanisms to distribute deep behaviors around the system as required. This allows them to affect event filters and event pipes on multiple machines in a coordinated fashion. Deep behaviors are provided as a layer of abstraction over the event filter infrastructure, hence making extensibility and configurability more meaningful to users and programmers. The variant Deep Behavior demonstrates the flexibility of the framework by providing facilities not usually provided by virtual environment systems. Rather than allowing arbitrary updates to items, updates to items tagged with the variant behavior create proxy items related to the original item by a syntactic consistency mechanism. Other clients viewing the item see its original state and can themselves create related proxy items representing their desired changes to the state of the item. The actual mechanism for creating these subjective views and relating the proxy to the original item will depend on the awareness management facilities of the virtual environment system. The prototype implementation uses aspects to create overlay environments for each variant. The system uses delayed persistence and caching to improve performance and extensibility of the platform.

Yu et al. [47] present a continuous consistency model in the TACT system where application designers can bound the maximum distance between the local data image and some final consistent state. This space is parameterized by three metrics, numerical error, order error, and staleness. TACT, a middleware layer that enforces consistency bounds among replicas, allows applications to dynamically trade consistency for performance based on current service, network, and request characteristics. They argue that their approach is general by describing how a variety of services can express their consistency requirements using TACT and by showing how a number of existing consistency models can be expressed within the TACT framework.

The VIRTUS DVE system [48] is a Client/Server system easier to maintain. Collision handling, user and ownership management, and the guarantee of consistency generate less problems. The server may act a message filter and keep unnecessary communication to a minimum. As a client /server system VIRTUS can be extended to a hybrid or multicast system with less effort than pure peer-to-peer systems. Since there is direct communication only between clients and the server, no modifications can be missed by the server. VIRTUS employs centralized data management with data replication on demand. The server knows all worlds in the environment and all objects in all these worlds, handles ownership and guarantees consistency. It is notified of changes in any of the worlds and supports persistency of object states after client termination. Late-coming users immediately have an up-to-date view at the environment. The server manages a database of worlds comprising the universe (VirtusUniverse) and organizes the communication to the clients. Each client knows at most one VirtusWorld at any moment. When a client switches from one world to another, it receives all necessary information about the new world from the server. Centralized data management does not limit the creation of objects to the server side. Clients may also create new objects and insert them to the scene. They are registered by the server and presented to other clients. A special case of necessary client side object creation is the insertion of avatars.

Mauve et al. [49] discuss the importance of and investigate the existing Application Programming Interfaces (APIs) for reliable multicast communications in distributed interactive media. They discuss methods of structured receiver groups, unstructured receiver groups, forward error correction, congestion control, and efficient state and event transmissions for effective distributed media communication.

The ATLAS [50] DVE system by Lee et al. addresses the concern of scalability in virtual environments by first defining the components of scalability and then addressing each component of the problem. According to them the main concerns in scalability include interest

management, communication architecture, data replication, and concurrency control. It uses a Peer/Server model to take advantage of both the peer to peer model and also the multiple server model. Here consistency management is done through a server and inter-client communication is done through multicast. ATLAS assumes that a whole virtual world is divided into several logical regions and that each participant can communicate with other participants in a region to which he belongs and those in neighboring regions. It supports the interest management scheme based on user interests and spatial distance. It also leverages human heuristics. Inter-region communication is also supported in a scalable manner. Concurrency control can be done in three ways. They are pessimistic (block all users and grant lock on request), optimistic (allow all accesses and resolve conflicts when they occur) and predictive. The prediction based concurrency control is to allow users to lie in the optimistic scheme as well as to eliminate the need for repairs like the pessimistic scheme to support real time interaction. It is suitable to allow real time interactions for users.

The ATLAS system uses the entity-centric prediction based concurrency control scheme that satisfies the needs for scalability in terms of interactive performance as the number of users increases. Only the users surrounding a target entity multicast the ownership requests by using the multicast group address assigned to the entity. The number of messages per owner decreases drastically since the owner receives only from users joining the entity multicast group instead of from all users in the same region. This reduces network bandwidth consumption. Therefore, an owner makes prediction with the reduced number of messages, which, in turn, results in a short request processing time. It allows the owner to determine the next owner and pass the ownership in time. For efficient data management of a virtual world, the authors have proposed a scheme using user-based caching and pre-fetching exploiting the object's access priority generated from spatial distance and individual user's interest in objects in DVEs [51]. The scheme leverages the locality obtained from interactions between a user and objects during user's navigation in a virtual world, interest in objects in the world. To incorporate the user's interest into the access priority of objects, the authors leverage the fact that a user tends to repeatedly visit objects interesting to it or highly popular objects likely attracting it. To enumerate the level of interest and popularity of an object, the authors introduce two values, interest score and popularity score of an object, respectively. The interest score of an object is set per user and represents how much the user expresses its interest to the object. The popularity score of an object is set per world and represents how many people in the world express their interest to the object. By combining these two values with the spatial relationship, ATLAS improves the performance of caching and pre-fetching since the interaction locality between the user and objects are reflected in

addition to spatial locality. The number of access times for a given object determines interest and popularity scores.

For further improvement of cache hit rate, the ATLAS system incorporate user's navigation behavior into the spatial relationship between a user and the objects in the cache. It is also observed that a user usually alternates a navigation mode between wandering and moving. This fact is used to provide more accurate information for caching and pre-fetching.

The CAVERNsoft system [17] uses many parallel sockets over a high bandwidth connection to provide a virtual environment over an Internet scale network. The HIVE [16] architecture and data structures enable collaboration between 2D and 3D clients simultaneously. Geometric information is shared in the form of scene graphs using VRML 2.0 as the 3D modeling language for dynamic manipulation of the scene and extension with prototype nodes. In a 2D window-based conferencing application, the HIVE presents images of each participant in the conference, a shared whiteboard, and a shared browser that can be used to review viewgraphs or other data. The HIVE incorporates a variety of techniques such as speaker highlighting to provide additional non-verbal cues about who is speaking. The high fidelity audio module of the HIVE connects multiple clients together in a multicast group and enables sidebar conversations apart from the main group. Communication between HIVE clients is done using the Voyager mobile agent system. CAVERNSoft is used for communication between the HIVE and VisualEyes. CAVERNSoft creates a distributed shared memory for client applications that subscribe to data indexed by keys. It supports connecting an unrestricted number of participants at different host machines into a collaboration session. It supports client hardware with varying abilities to render graphical and audio data. It integrates shared applications into the collaboration session. It integrates a high-fidelity audio module in the environment and transmits speech between conference participants.

The Community Place system [51] (CP) is a client-server system. It consists of a central server, running at a well know Internet node, which is responsible for enabling browsers to view a single shared VRML scene. Browsers connect to this server when they load the VRML file associated with the shared scene. The Community Place architecture consists of three components namely the Browser, Server, and the Application Programming Models. The browser works in conjunction with a HTML browser. The CP browser loads the 3D data file (in VRML2.0 format), in the course of which it finds an entry describing the location of the server to be used for this shared 3D scene. The CP browser then contacts the server via the Virtual Society Client Protocol (VSCP) that runs above 1P. The server informs the CP browser of any other users in the scene, including their location, and any other 3D

objects not contained in the original scene description downloaded from the web server. This architecture uses VRML 2.0 standards and uses Java as its scripting language. . CP uses the associated HTML browser to subsequently download any scripts referred to in the VRML file. Scripts are able to manipulate scene graph nodes by generating events that are delivered to the node and change one or more of its properties. The scripts can also dynamically generate other VRML nodes. This provides a framework for event communication, notification, and behavioral specification inside the CP system. The server (also known as the CP Bureau) acts as a position tracker and message forwarder.

Each user's browser, as it navigates through the shared scene, sends position information to the server. The server then uses AOI (area of interest) algorithms to decide which other browsers need to be aware of these position changes. The server sends out the position to the chosen browsers, which in turn use the information to update the position of the local representative, the avatar, of the remote user. The role of the server is limited to managing state on behalf of connected users. It is generally unaware of the original scene loaded by the browser. The second role of the server is to carry out a similar function for any script level messages that are generated by a browser as a result of user interaction. The third component is the application-programming model. The CP system provides two models for application building, the first is known as the Simple Shared Script (SSS) model, and the second as the Application object (AO) model. The two share some elements but are targeted at different applications and different authors. Both models use the message sending API that CP supports. Messages can be sent to all browsers to all except the sender and to the owner or master of an object.

According to Pekkola et al. [13] DVE scalability is usually thought of in terms of the intriguing technical problem of appropriate and smooth representation of large crowds. They propose that this issue needs to be resolved along with the issue of scalability in more profound situations where scalability is also connected with re-configurability of worlds. Current DVE's usually offer the ability to move between worlds. Scalability is thus directly implicated in user's ability to reconfigure and create new worlds. There is a further issue of 'downward' scalability, like the ones anticipated in the MASSIVE-1 [6] 2D. Variants of the same DVE need to be available to devices of differing power, e.g. mobiles. Similarly, DVE affordances like background awareness of others need to be available to devices running different media, whether text, audio, or video. This could be called 'sideways' scalability. The VIVA system [13] proposed by these authors, addresses scalability in a multiple server environment with the following parameters. They argue that any algorithm that assigns users dynamically to different servers, and also switches users between the servers, has a

number of distinct and partially conflicting goals. It is difficult to set a single value that measures its' efficiency. However, performance can be measured in achieving each particular goal, and then solving a multi-criteria optimization problem. The parameters proposed are

- Number of server switchovers per time unit
- Total network traffic
- Number of hops when transmitting broadcast traffic to recipients
- Average latency in client-to-client connections
- Average delay caused by a server switchover
- Robustness and reliability: number of server splits (and possible unrecoverable client errors).

The algorithm works by dynamically dividing a space into convex polygons, each representing an area served by a particular server. Clients (users) are located inside a convex, and connected to an appropriate server. Since each borderline is shared only by a limited number of adjacent servers, any modification to a partition is possible to accomplish without centralized control, by negotiations between relevant servers. The algorithm assigns a weight for each server. The number of 'excess' users located in each server affects the weight. 'Excess' is the estimated number of users who, if moved to another server, create the best possible quality of service for the remaining users. If each server already has an optimum number (or less) of users, the borders are not moved and the algorithm is terminated at this point.

Ishibashi et al. [52] provide an adaptive QoS paradigm for multimedia streams in 3D virtual spaces, with an enhancement to the virtual time rendering media synchronization algorithm to determine exertion of media synchronization control and accuracy of media synchronization according to the importance of each object. The scheme carries out CPU load control, traffic control and media synchronization control in accordance with CPU load, network load and importance of each object.

Aahrus et al. [53] provide a two tiered server architecture for QoS improvement in gaming environments running on a generic Internet. This architecture is based on a server tier and a concentrator tier. The server tier uses Distributed server architectures which spread the server load on several machines working in parallel, usually with separated computing tasks. The concentrator tier does the functions of connection and bandwidth management, event routing between clients and servers, duplication of events to clients, filtering of events to

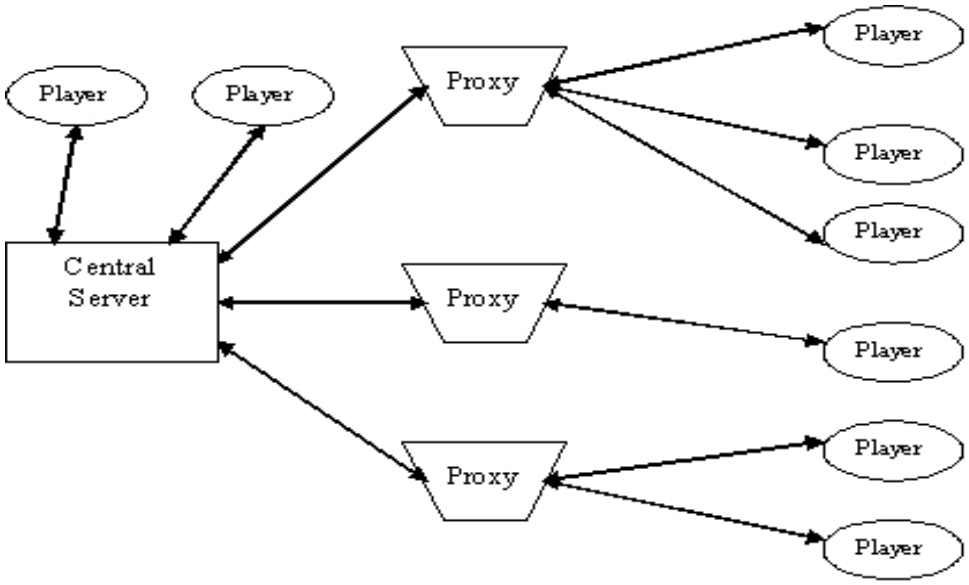

Figure 2.5: Proxy Architecture for Networked Computer Games

clients, Bandwidth adaptation, Interest management, server-side relevance filtering based on game state and concentrator-side relevance filtering based on networking conditions. The network between servers and concentrators is characterized by high bandwidth, low latency, and low packet loss. The network between concentrators and clients is in the worst case characterized by low bandwidth, high latency, and significant packet loss.

Mauve et al. [54] propose a proxy based architecture for networked gaming environments, as opposed to a strictly client server or peer to peer architecture. A central server is in charge of maintaining the game state, being the final authority on consistency and prevention of illegal state manipulations. A client may connect directly to the server and communicate in the same way as in a centralized client-server architecture. However, a player may also connect to a proxy instead of the main server. It is expected that such proxies are located close the players (e.g., at their respective service-providers). The proxies can be thought of as extensions of the server. The server can trust them to a certain degree, since the proxies are not under the control of the players. Consequently, some server functionality can be delegated to the proxies and is therefore located closer to the players. Furthermore, the proxies themselves are interconnected. Therefore they form an overlay network which may be used to alleviate some of the inherent problems of a centralized architecture. Congestion prevention in the gaming network can be made possible to route traffic around congested areas. In case all possible paths to a server are congested, or the server cannot handle the

current load even with the help of the proxies, individual proxies can perform access control. This allows the server to continue operation without being flooded by connection requests. The proxies only admit additional players as long as the generated traffic is fair to competing flows and can be handled by the server.

In addition to using the overlay of proxies to avoid congested areas of the network, it can improve the overall robustness of the architecture. Proxies can also be used to check if the behavior of a player conforms to certain rules. The server no longer has to perform this very time consuming tasks and can in turn support a larger number of simultaneous clients. Proxies can be used to detect whether the players use so called "bots" or try to manipulate the game's protocol; it acts as a special type of fire wall for the game server. A gaming proxy can even act as a form of packet normalizer [9], preprocessing client information and ensuring that only valid client data is distributed to other clients and the server. Latency optimization for discrete and random events in the gaming network can also be achieved with this proxy architecture.

DTWP [28] provides protocol level support for two basic mechanisms to realize large-scale virtual environments. They include adding additional daemons (on additional hosts) to reduce the load on the existing ones l splitting worlds into smaller parts (regions), each part using its own network connections Since there is no direct connection between the individual daemons (all daemons send and receive messages via the multicast connection) unless transmission failures occur, adding daemons usually reduces the load of the existing ones. Additional world daemons might be used to provide additional dial in points (initial downloads). Increasing the number of world daemons is useful, if a large number of users, which cannot be handled in time by the available world daemons, frequently join a shared virtual world. Instead of publishing additional dial in points as URL's for those world daemons, the original world daemon(s) can simply be configured to redirect requests to the new world daemons.

Additional recovery daemons can be used to reduce the number of recovery requests on a single daemon. This might be necessary, if a large number of transmission failures occur. Adding unicast daemons provides the possibility to support a large number of participants even if they do not have access to multicasting. A single unicast daemon should not be connected to a large number of participants. Tests showed that the limit for the number of participants to be supported by a single unicast daemon is between five and twenty depending on the activity of the participants and the requested services (e.g. with/without audio). The network load of unicast daemons can also be reduced by configured individual daemons for the transfer of events and stream data.

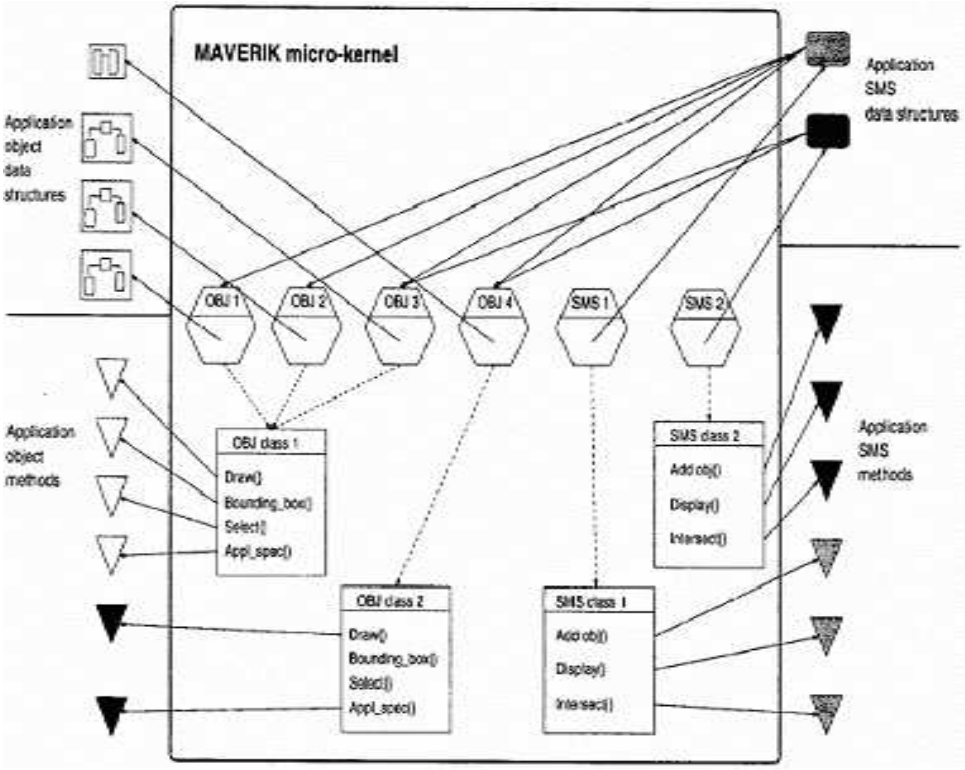

Figure 2.6: Micro-Kernel Architecture in the MAVERIK system: Depicted from [55]

In addition to increasing the number of daemons for a particular world, DWTP [28] also supports splitting the whole virtual world into smaller parts or regions. Each of these parts uses its own network connections. Thus the load of the daemons responsible for a certain part decreases. The VR application has to make sure, that the participant of such a shared virtual world is only connected to those parts, which are currently visible to him or her.

The MAVERIK [55] System has its software architecture based on micro-kernel functionalities. In this architecture each object can be potentially made of a set of polygon meshes or of predefined functions for basic shapes like circles, cylinders etc. Each of these definitions is either stored as objects in the environment which have individual definitions from provided scratch or as a collection of predefined objects (the definition for these predefined objects is stored in the micro-kernel). The MAVERIK system allows both data representations and algorithms to be customized for individual applications. As a consequence, the data describing an environment is assumed to exist outside the MAVERIK micro-kernel. For the kernel to provide useful functions, it therefore defines a framework for accessing this data, displaying it, and enabling the user to interact with the environment. MAVERIK sup-

ports object definition and management, large model display and management, interaction, navigation, collision detection and avoidance, plus all of the usual low-level features such as managing different display devices (e.g. stereoscopic projection displays, head-mounted displays), and handling input devices such as 3D trackers and voice recognition hardware. The MAVERIK micro-kernel provides a framework, which permits customization of spatial management methods. In a manner analogous to object definition, the kernel uses classes and methods to store and access spatial data. An application defines a class for each object storage technique, registers the call-back functions corresponding to the different methods for each class, and defines generic object management structures - called spatial management structures (SMSs) - to store and manage MAVERIK objects. Objects can be inserted in any number of SMSs and processing can be performed on the SMS most suited to a particular task.

MASSIVE-3 [10] adopts the concept of "Locales", introduced in the SPLINE system [4], to provide a powerful and expressive means of structuring and composing a virtual world. In a locale-based system the locale is the fundamental unit of world composition. Each locale typically corresponds to a distinct region of the virtual world, such as a room, corridor, open space or vehicle, and can contain virtual objects as well as users' embodiments (their representations with the virtual world). A key feature of the locale approach is that there is no single global coordinate system for the whole virtual world. Instead, each locale defines its own independent coordinate system. Linking together a number of locales composes a complete virtual world. Each link from one locale to another includes a 3D transformation that defines the relationship between the locales' coordinate systems. Locales are further subdivided into one or more "Aspects". These are the fundamental unit of interest management in MASSIVE-3, and each aspect is realized as one environment database. All distributed data in MASSIVE-3 must be contained in a particular aspect of a particular locale. Functional classes, organizational scopes, fidelity and cost define an aspect. Typically, different functional classes of data will only be of interest to a subset of observers, and this can be exploited if those functional classes are separated within the interest management framework. A single aspect can contain one or more functional classes. Organizational scopes will normally be defined by the application to reflect organizational associations between objects. MASSIVE-3 allows a locale to contain aspects with the same functional and organizational scopes, but different fidelities. An observer can then choose the aspect with the fidelity that suits their requirements and available resources.

Carson et al. [56] propose a hybrid system of multicast and client-server approaches. Predominantly the system is distributed: hosts communicate using multicasting rather than

relaying data through a server. Communication occurs using a mixture of best effort and ACK-based reliable multicasting. Use of reliable multicasting is reserved for messages, which must be delivered to avoid deadlock, such as messages to grant and release, shared objects. The majority of traffic, such as avatar position data, is sent via best effort to avoid the higher cost of using the reliable multicasting protocol. In the case of avatar position, data packet loss is not so serious as a machine can catch up when the participant moves again. The system also employs a lightweight server called the Gatekeeper to perform some services such as providing access to the virtual world, recording the state of the world and controlling access to shared objects. The principal function of the Gatekeeper is to act as a gateway for participants entering the virtual environment. It listens for connections via a TCP/IP connection. When a client connects, information about the multicast address and packet TTL to use are transmitted along with information about the state of the world. The state contains information about shared events that have been triggered which have a lasting effect on the state of the world, for instance someone turning on a light in the world. The state also contains information about shared objects, such as the positions of the object and whether they are still available for collection. Once a participant has entered the world, the Gatekeeper ensures that participants are still connected, by monitoring heartbeat messages sent out periodically by clients. If Heartbeat messages are not sent out in a certain time frame then the Gatekeeper attempts to query the particular client. If the client does not reply to the query then the Gatekeeper takes steps to remove the participant from the virtual world so preventing it from being cluttered by "dead" avatars.

Peters at al. [57] support agent based communication in the VETAF architecture (a collaborative environment for a virtual task force). In this system the agent communication is carried out using an "Agent Communication Language". They build their agents on "A Simple Agent Platform" (ASAP). All components within an ASAP Core Module communicate among each other through the use of events. This process is based on a general broadcast of all messages and events to ensure that all other agents are notified. The ASAP Core Module represents one agent society. The Agent Controller is the runtime environment providing the general user interface to monitor the local agents. In addition, the controller offers the possibilities to access system resources. Conditioners alert an agent when a specified event occurs and External Commands provide the interface to non-agent programs.

Lu et al. [58] propose a mechanism required to support a large-scale distributed simulation on the Web, and adopted high level architecture (HLA) [35], [36] developed by the Defense Modeling and Simulation Office (DMSO), to provide a more flexible distributed simulation environment. They also use the 3-level control mechanism (3LCM) with HLA-

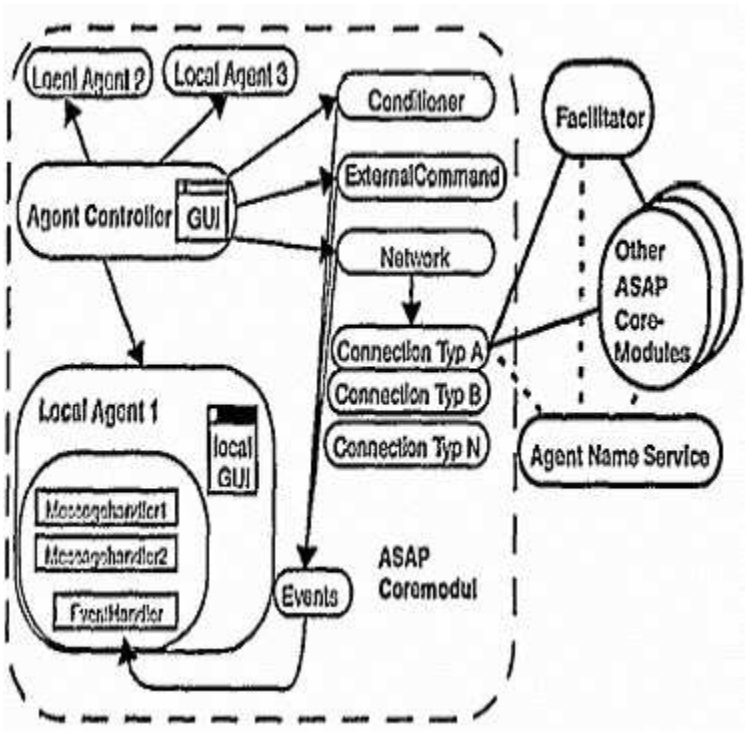

Figure 2.7: ASAP Architecture: Depicted from [57]

based middleware in the federate Server to adaptively maintain information consistency and minimize message traffic for distributing information among client hosts and the federate servers.

The NetEffect [59] architecture identifies six key techniques to improve overall improved scalability of the system. Those techniques are Partitioning and client migration, need-to-know-updating, group dead reckoning, point-to-point-connection, extensibility and dynamic load balancing. In NetEffect the virtual world is partitioned into many communities, which are distributed over multiple servers. A server can handle one or more communities. A user can establish a connection to any server depending on the user's choice on community. When a user wants' to change community, he is connected to the new server handling that community and disconnected from the old server. At any point of time, all users of a community are connected to the same server. Therefore user interaction with the virtual world does not generate any inter-server network traffic. This architecture is extensible due to the fact that servers are very loosely coupled. Adding more and more communities with new servers without affecting existing worlds can extend the virtual world. Moreover servers can be programmed to provide application specific support. This extensible architecture provides flexibility in increasing a system's computational resources. NetEffect provides a load balancing technique, which creates a uniform distribution of user density over the servers. If there are too many communities for available servers, it is possible for one server to handle multiple communities. In that case, the system periodically checks density of users at all communities and at all servers. To balance load among all the servers, the system dynamically transfers some communities from heavily loaded servers to ones with less load. Dynamic load balancing optimizes the performance of a network of computational resources.

The entire virtual world is divided into several communities, which are managed by peer servers. A peer server maintains the database for its communities and handles its clients' requests. A peer server also communicates with other servers, if necessary, but the inter-server communication has been minimized in this architecture in order to reduce network traffic. When a peer server starts up, it first connects to the master server, which assigns an identity to the peer server. The master server also takes care of initial connection and distribution of clients, and maintains each user's personal database. When users join the world, their clients are connected to the master server first. The master server decides the starting community for the client and instructs the client to connect to that community-server. This decision about which community the client is connected to may vary from application to application depending on game logic.

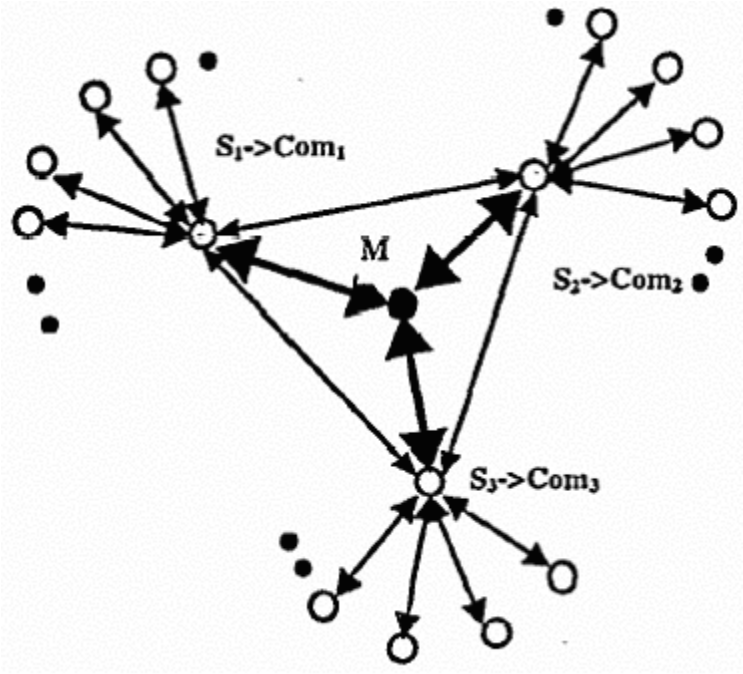

Figure 2.8: NetEffect Architecture: Depicted from [59]

Generic Internet Scalable Architecture (GISA) [53] is a scalable client-server architecture for gaming environments in which the server side is organized in two tiers: the server tier, which contains the game state logic, and the concentrator tier, whose task it is to perform connection and bandwidth management. The server tier is a distributed system that supports distribution of the game state over a number of physical machines in order to handle the load. Each server in the server tier communicates with all the other servers and all the concentrators. A server is an event based system; it receives events from various sources, and according to its internal state it generates new events that are routed to the appropriate recipients. Since the server tier distributes the game state over several servers, events gener-

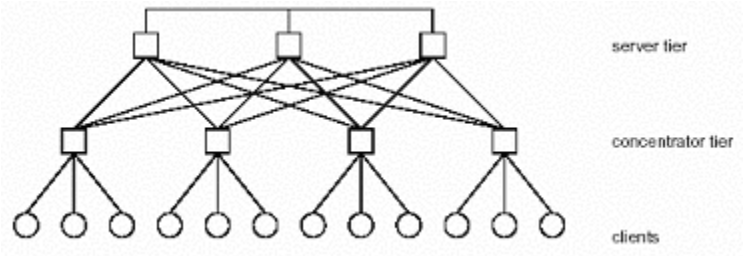

Figure 2.9: Generic Internet Scalable Architecture: Depicted from [53]

ated by the world model of one server may need to be passed to the world model of another server. The concentrator tier consists of a set of concentrators, each of which is connected to all the servers and a large number of clients. Events received from clients are routed to the servers in the server tier. Concentrator functions include connection and bandwidth management, event routing between clients and servers, duplication of events to clients and filtering of events to clients. The client communicates with the user through a graphical user interface and maintains a partial model of the game state. This model is not necessarily accurate; the game state maintained by the server tier is by definition correct, and the client state may deviate slightly from this state. The client game state is updated by events sent from the server tier via the concentrator tier, and player action events are sent from the client to the server tier. The client is robust towards varying network conditions, but the gaming experience may suffer.

Bauer et al. [60] propose a multi-server architecture with differences from usual multi-server architectures. The authors argue that If the server farm itself is distributed, such that only some subset of clients is handled by each part of the server farm, and then attain the desired level of performance. However, servers still need to coordinate states among themselves. If they were to do this naively, e.g. by sending every event they received to all other servers, then each server would receive all events and the situation would not be any better than in the completely centralized solution. Servers need to reduce the amount of information they forward to other servers by only sending relevant information. For example, a given event should only be forwarded to a server if there is at least one interested client registered with that server. If there is no correlation between interest in an event and physical location, then, as the number of clients rises, the probability that there is at least one client registered with a server interested in a given event approaches unity for a fixed number of servers. The correlation overhead of the distributed-server approach limits its scalability. This overhead is significantly reduced by enhancing the network with coordination, computation, and storage services that are provided by network-aware booster boxes in the proposed architecture. Booster boxes acquire network awareness by monitoring traffic, measuring network parameters such as delay, by participating in routing exchanges, or by acting as an application layer proxy server. Each booster box serves a number of clients in its network vicinity and performs caching, filtering, forwarding and redirecting of game events in a game-specific way. Compared to distributing the servers across the network, the main advantage of this approach is that booster boxes combine network and application awareness in a single entity. For example, forwarding decisions can be based on high-level knowledge, e.g. the sender's location in the virtual space, as well as on network-level knowledge, e.g.

current network delays. A piece of application-specific code, called booster, is executed on the booster box. The booster box provides interfaces that allow the booster to observe and manipulate data streams, and to participate in control and management protocol operations. Boosters use the following fundamental operations to reduce the load on the servers

- Caching: Boosters cache non-real-time information and answer on behalf of the server.
- Aggregation: Events from two or more clients might be aggregated. In the simplest case, several redundant events arrive within a given time frame, and only one is forwarded to the server. In other cases, the booster computes an aggregated event by performing a function that otherwise would have been performed on the server.
- Intelligent Filtering: Depending on the state of the game, events may no longer be relevant. These events can simply be dropped by the booster.
- Application-level routing: A game supporting millions of participants will be implemented on distributed servers. The booster acts as an application-level router, and sends an event only to those servers that are responsible for handling it.

Reynard et al. [61] extend previous work on texturing video streams into DVEs through de introduction of awareness driven video QoS. Movements in a shared virtual environment are mapped onto different levels of mutual awareness via the spatial model of interaction. In mm, awareness levels map onto different video services corresponding to different QoS parameters. This uses awareness driven QoS to seamlessly move between three different video services according to a user's movements. They argue that the general approach of situation video displays within a 3-D virtual world offers three main advantages: it may help establish a degree of spatial consistency between different video views; it allows a direct representation of the viewpoints of those who are accessing video images; and it uses natural movements to switch between different video views without disrupting the activity at hand. This technique of awareness driven video QoS then introduces greater flexibility into the way that such video displays are managed within a virtual world. Specifically, it allows users to establish an optimal allocation of limited resources (processing and network) across multiple video streams and contains explicit support for privacy.

The SpaceFusion [62] architecture is a client /server model, with a number of information services provided by different organizations at their own servers, with Selective integration, or Fusion, of these information sources at the user side based on his/her interests. The architecture assumes that many different servers provide different services. Therefore a

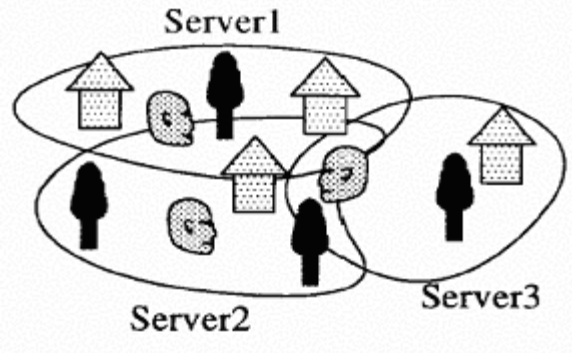

Figure 2.10: Fusion of various information from different servers in the Hyperfusion Architecture: Depicted from [62]

client can simultaneously connect to many different servers at the same time. SpaceFusion clients can connect to several servers simultaneously, and information from different servers are amalgamated and presented on the browser. This is the notion of Fusion. The concept of fusion in SpaceFusion architecture has three aspects.

- Horizontal fusion: This is the fusion of small spaces provided by possibly different servers, resulting in the seamless construction of a large space.
- Vertical fusion: This is the fusion of spaces provided by possibly different servers, resulting in the overlaying of a lot of information on the same space.
- Hyper fusion: This is the fusion of virtual space and information from the 3-d world.

Horizontal fusion allows spatial partitioning into smaller chunks, each of which is managed by possibly different servers. Thus, it enables local organizations to hold information about the local areas by themselves. By vertical fusion, different kind of information can be distributed into the appropriate servers owned by different organizations. Hyper fusion is another kind of fusion, and this fuses the virtual world and the real world. This must he a key concept of the cyberspace in the future.

In the PARSEC [24], [25] network gaming architecture shown in Figure 2.11, the game logic uses the game-code interface to pass state information to the protocol layer, which defines the way clients communicate with each other (peer-to-peer or client/server-based). Parsec uses three distinct protocols viz. Peer-to-peer, Slotserver and Gameserver The latter two protocols are client/server-based. The slotserver protocol is a hybrid between peer-to-peer mode and the gameserver mode. If this protocol is selected, Parsec clients will have

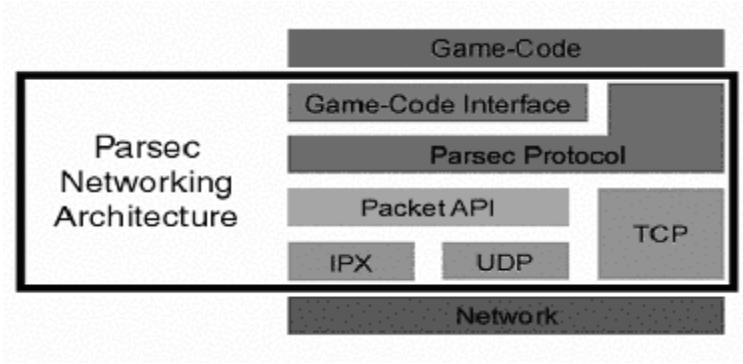

Figure 2.11: Parsec Gaming Architecture: Depicted from [24]

to connect to a server, but the actual game state transmission is done like in peer-to-peer mode, i.e. the server is not involved in the game-play at all. This mode is useful if one wants to play peer-to-peer games with people that are not connected to the same Ethernet segment. The protocol layer itself uses any of the available transport protocols provided by the operating system via the packet API, to actually transport information over the network. Both the protocol and the packet API can be switched on the fly, to allow the user to choose his preferred mode for network play, e.g. peer-to-peer/IPX or gameserver/UDP. Additionally TCP is used for initial connection establishment with the server, and some other communication tasks that are not time-critical.

2.4 Message Culling

The W-VLNET [44] system uses message caching in two ways to reduce the amount of messages going over the network. The Avatar files themselves are compressed into zip files, making the transfer to the Server lighter in comparison with uncompressed files (normally 7-8 times larger), but even these files are between 600K and 1M and hence a caching mechanism is also employed to reduce wait times and bandwidth utilization. The caching mechanism works in two ways; firstly it acts in the normal way, which is to check if a copy of the file exists locally (the files name, date stamp and size are compared with the local copy) and then just transfers the basic information (like posture/position), which is extremely small in comparison. The second caching mechanism is used if the user has a low bandwidth connection to the server; it basically uses a default avatar representation (also stored locally) for all avatars, hence reducing the requirement to download other client's representative avatar.

Greenhalgh et al. [63] propose an approach which uses distributed event filters and a notion of deep behaviors to reduce the amount of messages in the network. Message culling in the ATLAS [50] system is achieved using caching of local objects. Locality is determined by temporal locality, spatial locality and interest based locality.

Chim et al. [64] discuss the need for caching and pre-fetching in virtual environments as a method of improving performance in virtual environments. They also propose a multi resolution caching mechanism that is complemented object pre-fetched mechanisms for pre-fetching future accessed objects. Multi resolution modeling is a method of level of detail rendering in which various levels of details of the object are cached at a peer to improve network performance so that various levels of detail of the same object are not requested repeatedly over time. Also, the various levels of detail of the object are pre-fetched to reduce bottlenecks at the time when the object is needed. Replacement is based on object access patterns rather than common LRU mechanisms. MRM (Most Required Movement) method is used to accomplish replacement in the cache. Pre-fetching is done with the help of data structures called Historical Movement Vectors. An Exponential Weighted Movement Average (EWMA) is calculated from this vector to predict access probability.

In the Community Place system [52], the server which is also known as the CP Bureau acts as a position tracker and message forwarder. Each user's browser, as it navigates through the shared scene, sends position information to the server. The server then uses AOI (area of interest) algorithms to decide which other browsers need to be aware of these position changes. The server sends out the position to the chosen browsers, which in turn use the information to update the position of the local representative, the avatar, of the remote user.

In the DIVE system [32], it has been observed that participants form sub-groups where activities occur in clusters or peer-to-peer within the global session. This mimics the way the spatial model is used in the real world. The observation can be exploited to decrease overall message passing if one can deliver packets only to the recipients they are intended for, i.e. those within the sub-group. In this way, the amount of global traffic is limited, and the number of incoming messages to each user is reduced.

Using the three dimensions of space is a well-known approach to partition VEs into several disjoint Areas of Interest. Static geographical regions are used in applications based on natural terrains, such as in DIS based systems [20].

A different approach uses intersecting volumes to model interaction between participants. This notion of a spatial area of interest associated with a user has evolved out of work in the COMIC project [65]. The spatial area, known as an aura determines a boundary; objects or users outside the boundary can not be influenced or interacted with. In contrast, all

objects within the boundary are candidates for influence or interaction. The COMIC model goes further by defining two notions, focus and nimbus, to represent the degree of interest users have in each other. The focus represents the degree of interest one user brings to bear on another. The nimbus represents the degree of attention one user pays to another. The combination of the focus and nimbus of two interacting users defines their level or degree of interaction.

To achieve this, the server in the CP system is structured with an aura manager that is responsible for tracking the spatial location of any user (or AO object) and for determining if two user's auras have collided. If they have, the aura manager causes those two objects to join a consistency group that is defined as a set of objects who have shared data that must be maintained consistent. Proxies, i.e. local representatives of the remote object, denote the actual replicas. In the case where the objects are all local to one server, these proxies are generally pointers to the master object. In essence, the aura manager is responsible for defining groups of spatially co-located objects that need to maintain a degree of consistency. As it decreases the degree of sharing, this mechanism is used to reduce the amount of information that has to be sent out from the server as a result of any state changes.

In SmartCU3D [14], [15] an Internet DVE system, a behavior based interaction management with adaptive message routing mechanism is used. In this mechanism, the message routing in the system becomes adaptive to the application and the users' runtime interaction. It is achieved by giving the Object-Oriented style collaborative behavior description along with the 3D environment definition for every individual DVE application. The collaborative behavior description defines the functional roles selected by the users in the application, and the behaviors for every functional role. They also give the definitions for the roles and behaviors of the shared interactive objects within the 3D environment. When the interactive entities interact with each other in the DVE, the required message for coordinating the interaction and maintaining the world's consistence will be generated according to the specific behavior definition for the interactive entity's functional role. And the message will be routed among the affected interactive entities in the DVE. After such a message arrives, the interactive entities respond it with corresponding behavior based on their behavior definition. The authors argue that this approach enables the system to control the information flow from higher levels. Thus it gives a way to exploit the potential ability on managing the system resource and improves the real-time performance, scalability of the system.

Deep compression is recommended as a method of message size reduction in real-time streaming media by Cohen-Or et al [66]. This method involves the use of view dependent texture streaming which is useful for an environment consisting of a polygonal geometric

model, textures, and a number of light sources and furthermore texture-intensive, that is, the size of the environment database is dominated by the textures, while the geometry-space is significantly smaller than the texture-space. Nevertheless, the size of the environment is too large to be downloaded from the server to the client in an acceptable time. Streaming the environment requires the server to transmit the animation script and, according to the camera viewing parameters, to transmit the visible parts of the model. However, the size of the textures necessary for a single view is too large to be streamed in real-time. Instead of using these original textures, the authors argue it is better to use nearby views as textures. These views can be regarded as view-dependent textures that are effectively valid only for texturing the geometry viewed from nearby viewing directions. This method of message size reduction is also a viable option for virtual environments where state update communications are sent regularly. These updates can be compressed based on similarity between the previous state updates. Identical updates can even be discarded.

Many other systems use interest management as a means of reducing the number of messages relayed through the network. Some examples include Masa et al.'s e-Agora system [125] which implements interest management through division of the DVE system into domains and sub-domains. SPLINE [4] has a built-in interest management policy that replicates and renders the user's current locale and its immediate neighbors. The main function of interest management in MASSIVE-3 [10] is to determine what should be replicated, received, rendered, etc. In the approach adopted here interest management is performed at the level of locales and aspects, i.e. the interest management system must choose which locales should be replicated, and which of these should be rendered. MASSIVE-3 unlike SPLINE does not have a single policy for interest management. MASSIVE-3 includes standard policies to select locales or aspects based on topological distance, Euclidean distance, awareness, and benefit/cost. Different policies can be specified for different functional and/or organizational types, and for different fidelities. Multiple policies can then be combined using logical conjunctions. MASSIVE-3 uses independent policies for replication, graphical rendering and audio (although the rendering and audio policies can only choose among currently replicated aspects, whereas the replication policy chooses among all known aspects).

SPLINE [4] partitions the environment in spatial regions called locales. Each user is aware of all the objects in the current locale and in the immediate neighbors. The partitioning of the environment is a very flexible mechanism for structuring the virtual environment. In SPLINE each locale defines its own independent coordinate system. The virtual environment results from the connection of several locales. Each connection between two locales defines a 3D transformation that describes the relations between the locale's coordinate systems, which

allows the creation of non-Euclidean environments. This approach also eases the extension of the environments, since it is only a matter of defining new locales and connecting them to the existing ones. Finally, the locales approach also provides an effective mechanism for controlling awareness. Allowing the awareness of the adjacent locales gives users the notion of spatial continuity, increasing at the same time the system scalability. There are some situations where different policies could be more useful. For instance, in some situations one could be interested only in the current locale, and in other situations, in the current locale and perhaps a small subset of the adjacent locales.

Han et al [67], [68] propose a method of interest group based message filtering scheme that filters data transmissions based on user interests and spatial distance. It leverages human heuristic such that, for instance, in a virtual shopping mall, users often tend to move to and crowd specific places with their own interests and to interact with those whom have similar interests. In this proposed scheme, users of the same interests dynamically form a group when they get close. Each user in the group multicasts update messages to the rest of the group whenever it moves or interact with the world. On the other hand, when the group is included or collided with the interest area of a user who is not a member of the group, that is, not sharing the same interests, the representative of the group sends the aggregated update information of the group with low frequency to the user. The member with high priority among the members of a group is elected as a representative. A user with multiple interests should deal with his most interesting objects with highest priority and hence the lower priority messages can be filtered off because multiple interests cause an inconsistency in representation. Each interest group has its own multicast address.

In class-based filtering, classes of all objects that will participate in a virtual world are predefined. Objects in the virtual world are classified based on these classes. Users register their interests in classes before participating in the world and thus receive messages only from the objects of the registered classes. If a class is a subclass of another, users who register their interest in the latter also receive all the messages from objects of the former. High Level Architecture (HLA) [35], [36] adopts this filtering mechanism. The class-based filtering does not work well with DVE systems such as virtual shopping malls or virtual communities where interest of users change dynamically. Some systems attempt to combine filtering mechanisms for fine-grained data filtering. The hybrid approach focuses on balance between fine-grained data partitioning and computational overheads. The three-tiered architecture [69], [70] is an example of the hybrid approach that provides a fine-grained relevance filtering mechanism. It provides three tiers of data filtering hierarchically. The first and second tiers use spatial distance based filtering and in the third tier, protocol dependant filtering is used. If each user

has its own multicast address, the user can select the necessary data from subscribers (other objects). The third tier allows users to receive only necessary data from others with any kind of a protocol that is designed to support a specific filtering adapting to characteristics of a target DVE system.

The HLA Run Time Infrastructure (RTI) [35], [36] provides a set of services used to interconnect simulation during a federation execution. These RTI services are grouped into the six categories: federation management, declaration management, object management, ownership management, time management, and data distribution management. This article focuses on the functionality of the HLA data distribution management (DDM) services. The data distribution management in the HLA is used to reduce message traffic over the network and data sets required to be processed by a receiving host. The fundamental concept to support value-based filtering of DDM services is the routing space. Based on the routing space in the HLA DDM, federates can express an interest in receiving or sending data via subscription and update regions. The subscription and update regions can dynamically change in both size and location over time as an object's state in the routing space changes. So far the RTI does not automatically provide information filtering based on known functions, it only determines which federates should discover which objects by matching subscription and update regions. When a subscribing federate extends its subscription region in a routing space only, false positive updates can occur explicitly. As a result, the filtering efficiency of a subscription region deteriorates if the attribute updates fall into the extended section.

Lu et al. [58] propose a dynamic filtering strategy (DFS) to solve the false positive updates and add more accurate filtering ability to the RTI applications. This scheme depends upon filtering efficiency to dynamically change the position of the subscription regions or reduce the regions in order to avoid unreasonable extension of the subscription regions. A reduction in the extension of the regions can reduce false positive updates. They also present a Web-based distributed architectural design in which a 3-Level Control Mechanism (3LCM) is integrated into an HLA-based simulation modeling and HLA based middleware. The 3LCM is a technique to support a large number of clients and maintain as much information consistency as possible. It comprises three different kinds of methods, including the shared repository technique, information interest management, and predictive simulation scheme, to dynamically maintain a reasonable turnaround time and the information consistency of shared objects within the Web-based environment.

Trefftz et al. [71] report a method of message caching which aggregates incoming as well as outgoing messages and transmits the latest part of the grouped messages which indicates the most current value of the entity state. Caching the local updates before sending them

to the network has a global impact on the use of the network. The updates will be cached in a local events cache and, depending on the value of an outgoing-time-out, the number of updates sent to the network will be reduced considerably. A larger value of outgoing-time-out will avoid having a fast machine flooding the network and the slower machines with too frequent updates. Caching remote updates before updating the local scene has a local impact. The local data structure describing the Virtual World becomes a critical shared resource that can be updated by either local or remote events in order to avoid inconsistencies. Caching remote updates favors local updates, in which the user is probably more interested. Slower nodes use caching of incoming messages to reduce further the flooding effect of frequent updates generated by faster machines. Messages are received and stored in the local cache, but the scene is not necessarily updated immediately. Participating computers with a faster frame rate than the reciprocal of outgoing-time-out do not need to slow down the rate of remote updates, but slower computers do, in order to favor local updates. Caching outgoing events reduces the bandwidth usage, since the number of messages sent per unit time is reduced. The impact is especially significant when the participating nodes share a segment or a local area network or when the bandwidth is a scarce resource. Caching outgoing events also reduces the time other nodes spend processing remote events. Caching incoming messages, on the other hand, has only a local effect. If computing resources are scarce, in most applications it is best to give priority to updates generated from local events, since they will most likely be of most importance for the local user.

Park et al. [72] propose a method of relevance filtering and interest management based on multi-resolution spatial models to eliminate unnecessary state updates. Relevance realization is addressed on a real-time basis and relevance is checked against the avatar's current position. Relevance realization in interest management makes use of an entity based model. Level of Detail is introduced into interest management to reduce and optimize relevance realization. To make the granularity of the movement detection and the estimation of relevance different, they propose multi-resolution spatial modeling and update filtering. Both are controlled by resolution distance. If the resolution distance of a player increases, the load of relevance realization reduces as the low frequency of filter update relates with the movement detection and the request of relevance estimation.

NetEffect [59] allows the user to create a proper place-hierarchy for a community. When users are outside a building, they do not need to know about any update of objects inside the building. When users enter the building, they get object updates from the server. This technique is called need-to-know updating, since it defines the scope of a user within a smaller area (building or room) within a community. A server sends updates to a user only for those

objects in the place where the user is located. Thus need to know updating can reduce network traffic significantly.

2.5 Path Prediction Methods

One kind of path prediction method is based on the statistical method using random process analysis and past navigation patterns. It considers the fact that most people follow some regular paths for their regular activities, such as going from home to the office. However, a person may occasionally choose to have a path different from his/her regular movement pattern. Two models are proposed in [73] to model these two situations. One is responsible for the recognition of regular paths. The other is responsible for predicting non-regular movement using the Markov model, which is based on Markov process analysis. Unfortunately, the analysis requires collecting a large amount of data and may also involve expensive computations.

Dead reckoning [40] is a popular technique used in DVE systems to reduce bandwidth consumption in transmitting the positional information of moving objects. Each participating machine runs a simulator to extrapolate or predict future states of a moving object from its current state. There are many possible ways to extrapolate the movement of an object. The most popular method is the polynomial predictor, which is included in the DIS protocol of IEEE standard 1278. Although the 2nd order polynomial predictor is commonly used, polynomials of other orders may also be used in the predictor.

- zero order: $\mathrm{p_{new}} = p$
- first order: $\mathrm{p_{new}} = p + t \times v$
- second order: $\mathrm{p_{new}} = p + (t \times v) + (1/2 \times a \times t^2)$

where p is the initial position, pnew is predicted position, v is the velocity, a is the acceleration, and t is the time at which pnew is to be predicted. This method assumes that the motion model is deterministic and follows a simple formula. In [74], a hybrid approach was suggested. The first order polynomial is used if the acceleration is either minimal or substantial; otherwise, the second order polynomial is chosen. This method provides a generic model for many kinds of motion.

Another prediction model is the EWMA [75] scheme. This method predicts future movements based on weighted past movement vectors, with the current vector given a weight of 1 and each preceding vector decreased by a factor of

a, such that

$0 <= a <= 1$ and

$1 + n \times m = a \times n \times m + (1 - a) \times n \times m \times r$ where

$1 + n \times m$ and $n \times m$

are the predicted movement vectors for step $n + 1$ and n, respectively.

$n \times m \times r$ is the actual movement vector at step n.

The EWMA scheme can quickly adapt to the change in the user's movement pattern. The predicted movement vectors always depend heavily on the recent movement vectors.

The Kalman-based predictor [76] is more sophisticated than the polynomial predictor. It was originally used to filter measurement noise from linear systems by minimizing the mean square estimation error in a recursive way. In other words, it finds a solution of minimal error, which is the difference between the actual entity state and the observed or measured entity state. It may work with the polynomial predictor or other prediction models to further reduce the prediction errors. Similar to the polynomial predictor or dead reckoning, this method assumes the predicted motion to follow a fixed motion model.

Apart from polynomial-based prediction, there are customized prediction methods for special objects. For example, in [77], the movement of an aircraft is modeled by a circular path because an aircraft typically moves spirally in their designed virtual environment. Another method is the Gauss-Markov process modeling to predict head motion. It is because head motion is usually consisted of some burst changes in viewing angle accompanying with long static viewing direction afterwards. This kind of specialized prediction usually produces more accurate results because it takes into account the motion behavior of the target objects and extracts more information than the polynomial predictor does. The tradeoff is the loss of generality. A motion prediction method specifically designed for 3D navigation will certainly help reduce the amount of data to be pre-fetched.

Chan et al. [78] propose a hybrid motion prediction method based on an elliptic model for high velocity mouse motion and a linear model for low velocity motion. They use the kalman [76] filter to determine the pulse constants for the mouse motion and also the EWMA values of absolute velocity and length of the pulse instead of their mean values. This scheme also uses caching and pre-fetching based on these predictions to improve the performance of the system.. The objects that are possibly in view in the next step are pre-fetched from the server. The next step is predicted based on the kalman and the EWMA algorithms.

NetEffect [59] uses the group dead-reckoning technique in updating the movement and position information of all available users in the same community. It doesn't involve any inter-server communication, which reduces network traffic tremendously.

2.6 Event Handling

Poellabauer et al. [79] propose a scheme for coordinated CPU and event scheduling at the operating system level so that multimedia applications can benefit from such schemes and hence the user level QoS is improved. They argue that real-time and multimedia applications would be affected by scheduling techniques that do not take advantage of interactions between the CPU process schedules and the event-scheduling framework. Rather techniques that do take advantage of these interactions would benefit from such adaptive schedulers. They describe an event delivery mechanism, termed ECalls, that supports such coordination. They show ECalls' ability to reduce variations in inter-frame times for media streams. This paper proposes to extend multimedia scheduling by also explicitly taking certain events of importance to applications into account. Examples of events are notifications of changes in kernel state like the receipt of a network packet, a response by a kernel service to a request issued earlier by the application, or an exception experienced within the kernel. They show that the timely delivery and processing of such events is particularly important for time-constrained multimedia applications like virtual environments, interactive distributed simulations [40], or distributed games that integrate streaming audio and video [24], [25] of media streams should be minimized. Game events like position updates and certain actions of avatars must be delivered in a timely fashion. Techniques like proportional share scheduling of tasks and communications can reduce jitter for continuous media streams. However, the coordination of task scheduling with important game events can further reduce variations in inter-frame times and increase responsiveness to player actions. The authors aim to improve the performance of multimedia and real-time applications by making these applications and the system services they use event-aware. Toward this end, they provide an efficient operating system mechanism, termed ECalls (Event Calls) [79], for the exchange of events between applications and the OS services they utilize, and (2) develop policies for coordinating the scheduling of event delivery with the scheduling of operating system services. The particular service addressed in this paper is task scheduling. ECalls is an event-based mechanism that links user space applications with certain kernel-level services, allowing them to share information and events in an inexpensive and flexible way. ECalls is implemented as an extension to the Linux 2.2.13 kernel, and it supports real-time and multimedia applications by

- delivering events between applications and kernel services in a timely fashion,
- enabling both to efficiently share relevant information, and
- influencing process scheduling in response to the receipt of new events and/or pending

events.

The SIENA [80], [81] Internet based event notification service is an example of scalable event messaging for real-time networks. SIENA is implemented as a distributed network of servers that provide clients with access points offering an extended publish/subscribe interface. The clients are of two kinds: objects of interest, which are the generators of notifications, and interested parties, which are the consumers of notifications. This would be useful for middleware based event messaging for distributed virtual environments. The W-VLNET system follows a multithreaded approach with no need for shared memory communication between processes rather elegant inter thread communication.

Another network-level technology that is at least somewhat related to SIENA [80] is active networks. An active network is a network with programmable switches. In a sense, the programmability feature of active networks is a form of content-based routing, since the content of the packets can govern packet routing in a very expressive manner and can be used to achieve routing optimizations. But while active networks themselves are not a general-purpose event notification service like SIENA, they could nevertheless possibly be used as an implementation platform. Two prominent efforts are intended to lead specifically to event notification services. These are the CORBA Notification Service and the Java Message Service. It is important to note, however, that both these efforts do not themselves realize an event notification service. Rather, they simply specify interfaces to be implemented by a service, with the critical issue of how an implementation is to provide a scalable service left unaddressed. Commercial products such as SoftWired's iBus, TIBCO's TIB/Rendezvous, Talarian's SmartSockets, Hewlett-Packard's E-speak, and the messaging system in Vitria's BusinessWare provide implementations of an event notification service. While these products support distribution of the service, they are not specifically designed to support wide-area scale to the degree discussed in this paper. In particular, they typically achieve simple distribution through a federation of centralized servers with statically configured interserver routing. There are several research efforts concerned with the development of an event notification service, including IBM's Gryphon, Elvin, JEDI, Keryx, and the recent work of Yu et al. [82].

Yu et al. [82] propose an event notification service implemented using a peer-to-peer architecture of proxy servers, with one server being the distinguished root server. They believe a hierarchical arrangement of servers to have superior scalability to a non hierarchical one. Elvin is a centralized event dispatcher that has a rich event-filtering language that allows complex expressions of interest. The centralized architecture facilitates efficient event filtering, although it poses severe limitations to its scalability. Keryx also provides structured

event notifications and filtering capabilities extended to the whole structure of events. Keryx has a distributed architecture similar to that of USENET News servers. The architectures of TIB/Rendezvous and JEDI are also hierarchical, although their subscriptions are based on a simplified regular expression applied to a single string the subject in TIB/Rendezvous or the entire notification in JEDI. Even simpler is the selection mechanism offered by iBus that adopts channel-based addressing.

In the DWTP [28] protocol events are used to keep distributed copies of shared virtual worlds consistent by transmitting appropriate synchronization data. Events may contain any kind of data and are usually rather small. The application can specify the required reliability for the transfer of events. Messages are actually a number of predefined events such as used for joining or leaving a shared virtual world. Some messages can contain additional data (e.g. for chatting, transmitting URL's, or sending requests).

2.7 Level of Detail Rendering

In virtual reality simulations the speed of rendering is vitally important especially in applications where real-time interaction with the simulation is required. One of the techniques that is used for controlling the frame rate is that of having several different geometrical representations for each object and then assigning a level of detail to each object in such a way as to keep the frame rate high whilst maximizing the visual quality of the images that are rendered. The most well-known level of detail assignment algorithms are the Funkhouser [83], [84] algorithm and the algorithm where the level of detail is assigned with respect to the distance of the object from the viewer which is called the distance method.

Surface algorithms are the methods that produce the highest quality work on the surface of polygonal objects, With a simplified assumption of dealing only with triangles, information on which triangles are neighbors is needed. Local operations can be applied to remove triangles and fill the holes created by that process. Such algorithms can take into account local curvature and can generate simplifications with guaranteed error bounds. However, they are constrained to objects with well-connected surfaces. Unfortunately, this constraint is often not fulfilled by CAD models. Many of these algorithms are also constrained to preserve the genus of the object, and can therefore not simplify the objects beyond a model-dependent level.

Clustering algorithms consider the fact that Real-world applications almost always involve ill-behaved data, and for very large scenes and slow connections, it should be possible to produce very coarse approximations as well as moderately coarse ones. Clustering al-

gorithms are level of detail (LOD) generation methods that ignore the topology of objects and force a reduction of the data set. The key idea here is to cluster multiple vertices of the polygonal object that are close in object space into one, and remove all triangles that degenerate or collapse in the process, The problem here is that exact control over local detail is not so easily possible, but such an algorithm can robustly deal with any type of input data, and produce arbitrarily high compression. Vertex clustering can either be done with a simple uniform quantization or a binary tree. Schmalstieg [85] proposes a level of detail algorithm based on octree quantization for vertex clustering because of its superiority to uniform or binary tree quantization techniques.

The representation of objects using several different LODs was first proposed by Clark in 1976. Since then a huge amount of research has gone into techniques for creating LODs and using them to reduce the complexity of the rendered geometry. However, most of the common LOD methods are static. They use certain unchanging thresholds, such as the distance of object from the viewpoint or the projected area on the screen, applied to each object individually, for deciding what LOD to use. They merely reduce the geometry rather than regulate it since they make the complexity of rendering dependent on the complexity of the visible part of the scene at each frame. It wasn't until much more recently that dynamic methods (predictive or adaptive appeared. Here the thresholds for selecting the LODs change at each frame depending on the amount of visible geometry so that the frame rate remains within the desired limits and ideally is constant.

There may be two approaches for encoding the object models in order to reduce the amount of information to be sent through the network. The first approach is by applying either a geometry compression method or a level of details (LOD) method to reduce the storage size of the models. Most geometry compression methods consider the geometry information shared by neighboring polygons and reduce the amount of data needed to represent the polygon mesh. LOD methods consider the fact that the details of an object become less visible as the object moves away from the viewer and thus fewer polygons may be used to represent a distant object. These methods can potentially reduce the amount of data to be transmitted by sending models of appropriate resolutions to the client.

The second approach is to encode the object models for progressive transmission. This means that the models are encoded in such a way that partially transmitted models can be rendered and progressively refined as more information is received. Hence, the client no longer needs to wait for the whole model to be transmitted before rendering and can thus provide a more immediate visual feedback to the user. This approach has only recently been attracting a lot ,of attention and the progressive mesh is among the first in this area. In

this method, an object model is decomposed into a base mesh and a sequence of progressive records. The base mesh represents the minimum resolution model of the object. A progressive record stores information of a vertex split that may slightly increase the resolution of the base mesh by introducing two triangles into it. Hence, by applying the sequence of progressive records to the base mesh, the model will gradually increase in resolution until it reaches the highest resolution when all the records have been applied. The resolution of the model can be decreased by reversing the above operation.

There are several algorithms for level of detail rendering Funkhouser's algorithm works as follows. For each frame the initial level of detail for each object is the level of detail that the object had for the previous frame (or the lowest level of detail if the object was not rendered in the last frame). Then in order to try to reach an optimal solution the object that has the largest reward will have its LOD increased. The total time taken for the frame will now have increased. In order to get the time below the required frame rate limit the object with the lowest reward for its next LOD downwards will have its detail reduced by one. This increase/decrease stepping is performed until the object that has its level of detail increased is the same object who then has its LOD reduced again. The algorithm then terminates for that frame. It is in essence an iterative "greedy" algorithm. The benefit function is defined as

Benefit(O, L, R) = $Size(O) \times Accuracy(O, L, R) \times Importance(O) \times Focus(O) \times Motion(O) \times Hysteresis(O, L, R)$ where

O is the object being rendered,

L is the level of detail and

R is the rendering algorithm being used.

The central idea in the Market model of LOD rendering [120] is to take the conventional way of having a central algorithm that performs an optimization algorithm to decide how much time to give to each object and turn it around. Now, just like in a trading market, each player (object) is given an initial allotment of resources (time) to trade with. For each frame the assignment of the levels of detail becomes a session of trading where each object will try to buy or sell the time it has depending on the need it has for the time it currently owns. If more than one processor was available then parallel trades could be allowed as long as some locking mechanism was added to prevent the same time being traded independently to two different objects at the same moment.

Multi resolution modeling is a method of level of detail rendering in which various levels of details of the object are cached at a peer to improve network performance so that various levels of detail of the same object are not requested repeatedly over time. Also, the various

levels of detail of the object are prefetched to reduce bottlenecks at the time when the object is needed. Scope overlapping of world objects is determined continuously and the objects are rendered at the optimal resolution (Rendering at more than the optimal resolution will waste system resources). This also uses the area of interest management. To et al. [121] propose a scheme of multi-resolution modeling. This adaptive multi-resolution method is based on edge decimation. During the preprocessing stage, the model is simplified by collapsing edges and at the same time constructing a set of hierarchies to represent the parent-child relationship of the vertices. These hierarchies are referred to as the vertex trees and the root nodes of these trees form the base mesh of the model. To reduce the cost of pointer usage, the vertex trees are linearized to a set of one-dimensional arrays and stored at the server. At run-time, when the client requests for an object model, the server will transmit the base mesh nodes of the vertex trees first. Other nodes of the vertex trees will then be transmitted from top to bottom progressively; however, we may selectively transmit nodes from different vertex trees according to the view and animation parameters of the client at the time of the transmission.

Central to the framework proposed by Pasman et al [122] for level of detail generation is the extension of every node in the scene graph with an accuracy curve. This curve describes, for that node, the resource load as a function of the accuracy in the final image. It represents not a single point, but the full range of possibilities, such that in the end a suitable point can be found quickly. This accuracy curve can be calculated for any object and group of objects in the scene graph. For the leaf nodes, the curve is specified, estimated or derived mathematically, and for the intermediate nodes in the scene graph the curves of the children are combined into a new curve. The application can pick the required accuracy and/or resource load at the top node, and trigger the use of that configuration or decide to take other actions. LODs are generated or updated only after the application has chosen the proper operation point. This framework is guaranteed to schedule level of detail with quality and cost and works on the fly. The application directly controls the trade-off between the amount of resources to be used and the accuracy of the final images.

2.8 The Resource Allocation Perspective

2.8.1 Various Approaches To Resource Allocation

Research cited in the previous sections aim to optimze resource usage by techniques like behavior based interaction management, visibility based interest management and interest group based interest management. This also includes the use of Aura, Nimbus models by

Greenhalgh et al. [12]. These are indirect methods of conserving systemwide resources. The resource reservation protocol RSVP is more direct in the sense that it directly deals with the resource pool itself. The problem with RSVP is that it is too static. It reserves resources for several sessions at length even though the resource may not be used during several long time intervals. This is undesirable and hence gave rise to some dynamic variants of RSVP. Even these variants do not make use of mathematical models like Operations Research models rather, the current usage and request scenario. Using the current scenario for resource allocation decisions is good but should not be ad-hoc. The problem with OR based techniques do not produce results in real-time. Teodorovic et al. [146] thus came up with a hybrid resource allocation model which uses the current resource request and usage scenario. This data is fed to a trained neural network which gives resource allocation decisions in real-time.

This neural network is trained using optimal data obtained from running Operations Research models. This approach continues from our previous research where we optimized the resource allocation decisions in a Dynamic Stochastic Knapsack [138]. We found the decision times to be very much compliant with the real-time requirements of Distributed Virtual Environments. Based on this, we present a resource centric perspective of Quality of Service and a system architecture using this design.

2.8.2 Generalized resource Modeling

The stochastic knapsack problem was first developed in the late 1980's as a model to study admission control in telecommunication networks, so-called loss networks in particular[139], [140]. A certain communication link has a given capacity c (for instance, c is the bandwidth or the number of circuits), so that the number of jobs or calls with different capacity requirements that can be accommodated and processed at any time by the link is limited. When the capacity is saturated, incoming calls will be blocked and lost. Therefore, the link can be viewed as a knapsack, with a given capacity or size c. Jobs arrive at random time epochs, with random sizes and random occupancy times (if admitted). Ross and Yao[141] discuss An O(F) algorithm to determine the revenue of threshold policies. They consider the problem of the optimal static control where for each class a portion of the knapsack is dedicated for the general case of K classes. They also present a finite-stage dynamic programming algorithm for locating the optimal static control and variants of the optimal static control which allow some sharing among classes. Lin et al. [142] study problems that arise in revenue management and dynamic/flexible pricing Based on a dynamic programming for-

mulation, they identify certain properties of the value function, which lead to a lower and upper-orthant structure of the optimal policy. This motivates a class of control that they call switch-over policies, in which the switch-over times are optimally derived via convex programming. Based on these policies, we develop pricing models to optimize the price reductions over the decision horizon. In a related study Simon et al. [143] discuss admission control for deadline-driven packet scheduling policies provide real-time performance guarantees by associating a deadline with each packet and then transmitting packets according to increasing orders of deadlines. The paper develops a probabilistic model to analyze admission control methods for the general class of non-preemptive deadline-oriented packet scheduling policies. The authors present a general-purpose schedulability theorem for non-preemptive earliest deadline first packet scheduling. They then show how to use a stochastic knapsack to compute acceptance probabilities. Kleywegt et al. [144], [145] analyze an optimal strategy for accept/reject decisions to the knapsack and optimal stopping times for stopping the problem before the deadline. Given the waiting cost and the time horizon of the problem, the objective is to determine the optimal policy that maximizes the expected value (rewards minus costs) accumulated. Assuming that all items have equal sizes but random rewards, optimal solutions are derived for a variety of cost structures and time horizons, and recursive algorithms for computing them are developed. Optimal closed-form solutions are obtained for special cases. The DSKP has applications in freight transportation, in scheduling of batch processors, in selling of assets, and in selection of investment projects.

2.8.3 Real-Time optimization using Neural Networks

Real-Time optimization using machine learning techniques is adopted by us in Teodorovic et al. [146] in designing traffic adaptive control strategies. The authors use knowledge gained from predictive optimization techniques to train fuzzy sets and neural networks. These trained components are then used to build real-time controllers and develop planning strategies. In our previous research [9] we have used a similar technique to simulate a Real-Time traffic adaptive control system. These techniques were previously used for solving specific problems in traffic control systems.

CHAPTER 3
SYSTEM SETUP AND METHODOLOGY

3.1 The DIVE system

Distributed Interactive Virtual Environments (DIVE) [5][8] developed at the Swedish Institute of Computer Science is an experimental platform for the development of virtual environments, user interfaces and applications. DIVE is especially tuned to distributed, multiuser VEs. The DIVE architecture focuses on software and networking solutions that enable high interaction at each participating site. DIVE incorporates many basic features that applications can build upon such as a shared distributed hierarchical world database of entities through which user and application interactions take place. This database is replicated actively so that a copy resides at each site. Updates to entities takes place first at the local copy and are then propagated to all participants. DIVE uses peer-to-peer multicast communication. It identifies two types of traffic, 1) entity state updates, that are crucial and are sent over reliable multicast, and 2) continuous data stream like audio and video that are sent over unreliable multicast. It provides support for live audio, web integration and scripting using Tcl language. Applications can be created with ease in the DIVE environment simply by defining the worlds, the entities in the world and their characteristics in a DIVE virtual world file. Actions and interaction capabilities can be built into the entities by attaching tcl scripts to them that define the interaction behavior. These definitions of the worlds can be downloaded from anywhere in the Internet using URLs, which makes composition very simple. The applications can make use of the underlying features provided by DIVE to support multi-user interaction and communications as well as state consistency. This makes building and deploying applications in DIVE very simple indeed. Recent version of DIVE also provides many scalability extensions such as subjective views, high-level application events, lightweight groups, rendering extensions and database extensions. Some of the important features of DIVE that contribute to network behavior are outlined below [5].

3.2 Experimental Setup

To characterize the network behavior of DVE applications and model this using a modeling tool, it is important to first study a live application and analyze its behavior under varying parameters such as number of participants, levels of interaction and system characteristics. Sample applications provided with the DIVE binaries offer a clean and simple experimental environment to conduct this study.

3.2.1 DIVE Software

The main components of DIVE are:

- diva - A DIVE application that functions as a visualizer, implementing a 3D graphics and sound interface. diva is basically an advanced browser for DIVE worlds.
- diveserver - A server that contains a list of DIVE applications and the VR worlds they are viewing.
- proxyserver - DIVE applications use a peer-to-peer multicast communication model. If a network doesn't support multicast, the proxyserver can be used to relay messages to one that does.
- vishnu - A symbolic link to diva.

3.2.2 The Experimental DIVE Application

The experimental virtual world was adapted from the examples provided with the DIVE installation. The example world, tutorial.vr was modified to include specific objects, which have been programmed to respond to various types of interaction commands such as mouse clicks, moving (translation and rotation), collision detection etc. The resulting world is called demo.vr. The T-shaped pink figure is the default avatar representing the participant. The world consists of a flat landscape in which are placed 5 objects - a yellow ball, a red box, a blue box, a robotic figure, a blockie and a white podium named To Dive Town. The behavior of each object is as below: Figure 3.1 shows the demo world.

3.3 The Distributed Multi-user Testbed

A DIVE based distributed VE was setup connecting a number of computers on a LAN. The test machines were of varying hardware and OS platforms like Sun Sparcs running Solaris and Intel platforms running Windows NT/XP. One of the machines functioned as a dedicated nameserver, running the diveserver application. The multicast address on which this nameserver listens and the range of multicast addresses that can be assigned to various worlds by the nameserver are specified in the default configuration file.

The steps required to set up a multi-user DIVE session are:

- Start diveserver, the nameserver on the designated machine.

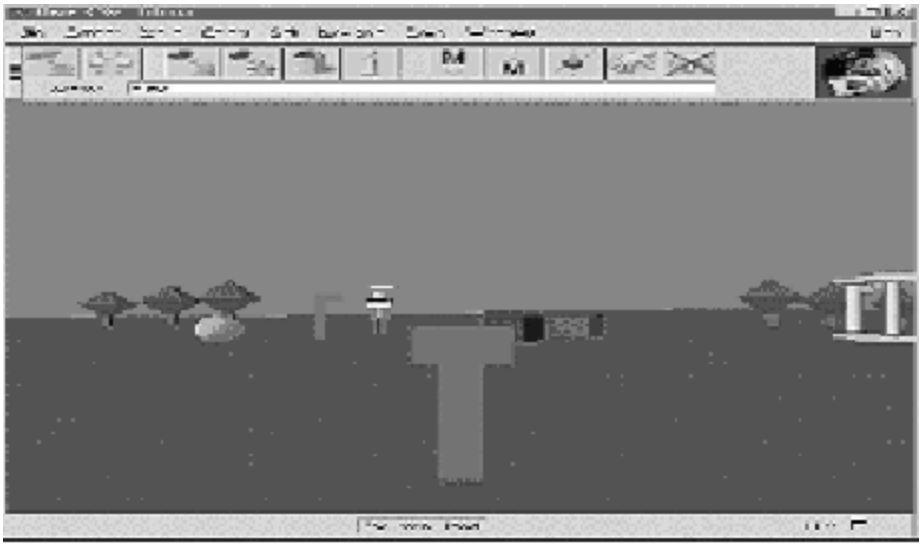

Figure 3.1: DIVE sample screen

- Participating peer machines join the world. A file named demo.vr describing the demo world must exist in the default search path. The command is as below:

vishnu -world demo Each participant has a representative avatar, the default being the figure called blockie. All the participants start at the same location in the world and are able to see each other. They can move around in the world using the arrow buttons on a standard keyboard. The Figure 3.2 shows the demo world. Various experiments were conducted in this experimental world to study the network behavior of the application. Participants executed actions in the world and their effect on the network was monitored using packet capture tools.

3.4 Data Capture

Along side DIVE, a network packet sniffer application, Ethereal, is also started on a machine that is not executing the application or running the diveserver. The packet sniffer is configured to monitor the multicast address and port on which the world is communicating and it records every packet sent and received by every host involved in the distributed virtual world. Ethereal records packet information in numerous standard formats such as tcpdump, Sniffer, windump etc, which makes these trace files portable. The Figure 3.2 below shows a screen shot of the ethereal packet filter. The Figure 3.2 shows the filter string used to capture DIVE packets belonging to the demo world. In this experiment, the world is listening to the

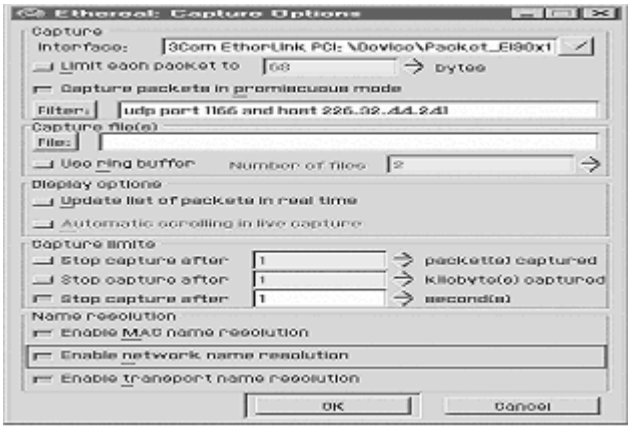

Figure 3.2: Ethereral Packet Filter

UDP port 1166 (default) and mulitcast address 226.32.41.241

The Figure 3.3 shows the result of executing the packet capture for 5 seconds.

3.5 Sample Experiments

The Figure 3.3 shows the captured DIVE packets belonging to the demo world

The method in which traffic is generated is important. While we want a controlled environment to study the application, we also want to understand application behavior under real life situations. For the former, it is important to time and plan each interaction and perform all possible scenarios of applications behavior. For the latter, the application should be executed, as it would be used in practice with real participants, distributed geographically across multiple locations accessing the VE over heterogeneous networks using heterogeneous systems. As we are interested in understanding the network characteristics of DIVE based DVE applications at this point, we conducted only controlled experiments. Once we have achieved a good understanding of the application, it can be modeled using a network-modeling tool like OPNET. Various real-life scenarios can then be simulated to study the performance of the DVE in large-scale realistic distributions. A technique to model such an application is described later. All the experiments consist of carefully choreographed actions executed at specific points on a predefined timescale. Thus each action can be recognized and reproduced unambiguously if needed. In addition every packet put into the network can be accounted for. A point to note is that all the experiments were conducted using ten different types of avatars as well as with no avatar at all so as to filter out avatar specific packets. Various scenarios were studied. Few of the important ones are outlined

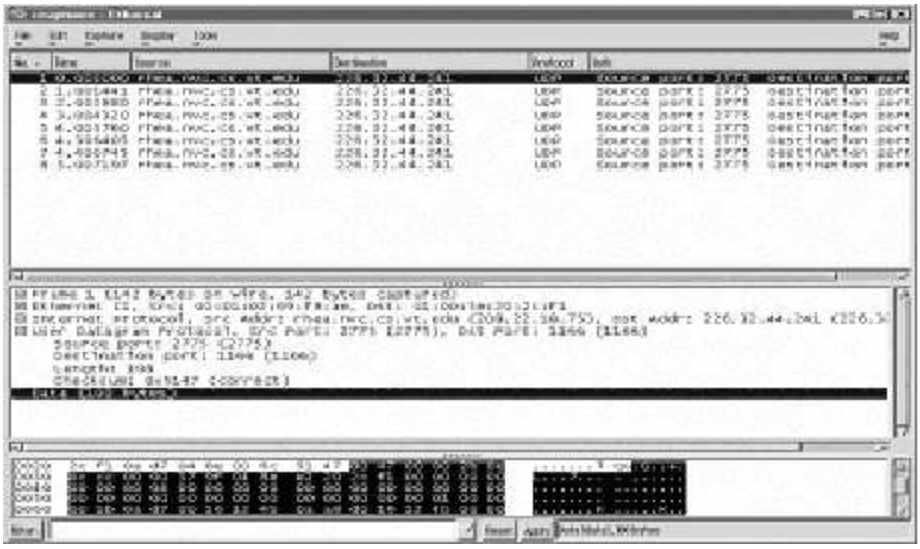

Figure 3.3: Ethereral packet capture output

below:

- Steady-state behavior - A participant joined the world, the application was allowed to settle down to steady state and packets were captured for 20 seconds. The aim of this experiment was to study the network traffic at steady state.

- First participant joins a world - The ethereal packet capture software was started, the first participant joined the world, and the application was allowed to settle down to steady-state by not performing any action within the world. This experiment distinctly identifies the packets generated when the first user joins the world.

- New user joins an existing world - One participant (the first) joined the world, the application was allowed to settle to steady-state by performing no actions in the world, the packet capture software was started, and 20 seconds later a second participant joined the world. The application was again brought to steady-state. The aim of this experiment was to identify any differences in the join operation between the first and subsequent participants.

- Participant exits the world - The world with two participants was brought to steady-state, ethereal was started, 30 seconds later, one of the participants left the world. This captured the network traffic generated when a user leaves a world.

- Last participant exits the world - The experiment 4 above was repeated for the last participant to study any difference in traffic in the case of the last user.

- Avatar move operations - The DIVE browser, diva, supports avatar navigation using the arrow keys on the keyboard. This default behavior is in addition to any object specific movement behavior that may have been programmed into the avatars object definition. Forward/backward translation is executed using the up/down arrow buttons. The avatars viewpoint can be rotated left or right using the left/right arrow buttons. To capture these behaviors separate experiments were conducted for translation and rotation. The procedure, however, is the same and will not be repeated twice here. The participant joined the world, after steady state was reached, ethereal was started, 20 seconds later the avatar was moved forward/backward or rotated clockwise/anti-clockwise for 10 seconds, and 12 seconds later the packet capture was stopped.

- Interactions -Clicking on an object - The DIVE browser, diva, also enables the user to interact with objects using the mouse by clicking on them. If the object is programmed to respond to these user-initiated events, the response will be triggered. By default, clicking an object produces not response. The effect of clicking an object is, therefore, highly customized behavior and no general conclusions can be drawn about this operation. But, given the fact that most objects, upon clicking, responds either by moving or rotating or perform some complex operations that combine translation and rotation in some form, we decided to test this as well. In the demo world used in our experiments, three objects - yellow ball, red box and blue box - have been programmed for click events. The yellow ball bounces once, red box rotates, and the blue box performs a translation, as shown in Figure 3.4. To study this operation, a participant joined the world; at steady state ethereal was started, 20 seconds later the red box was clicked, another 20 seconds later the blue box was clicked.

- Interactions - Moving and Rotating an object using the mouse - In addition to click operations, the diva client, by default, supports moving and rotating objects using the mouse. The middle button of the mouse can be used to move and the right button to rotate an object. These are default operations and require no programming support within the object. These operations were captured as part of the above experiment. Twenty seconds after clicking the blue box, the yellow ball was moved using the middle mouse button for 5 seconds, 20 seconds later the blockie object in the world was rotated

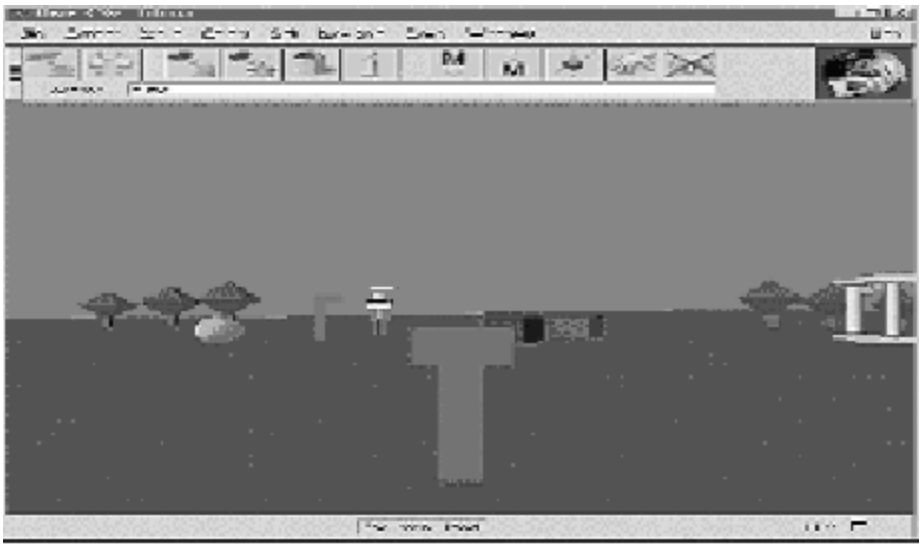

Figure 3.4: DIVE screen before Interaction

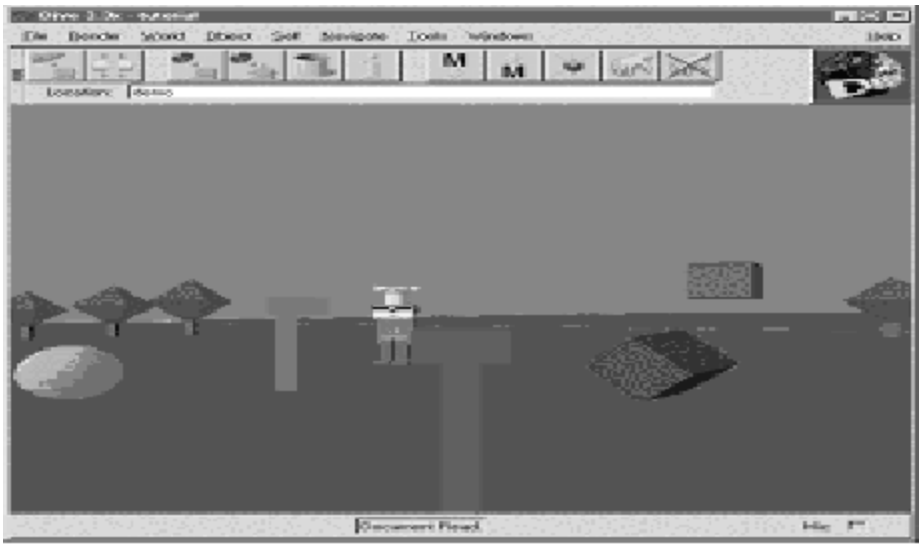

Figure 3.5: DIVE screen after Interaction. Object responds to mouse clicks wit movement.

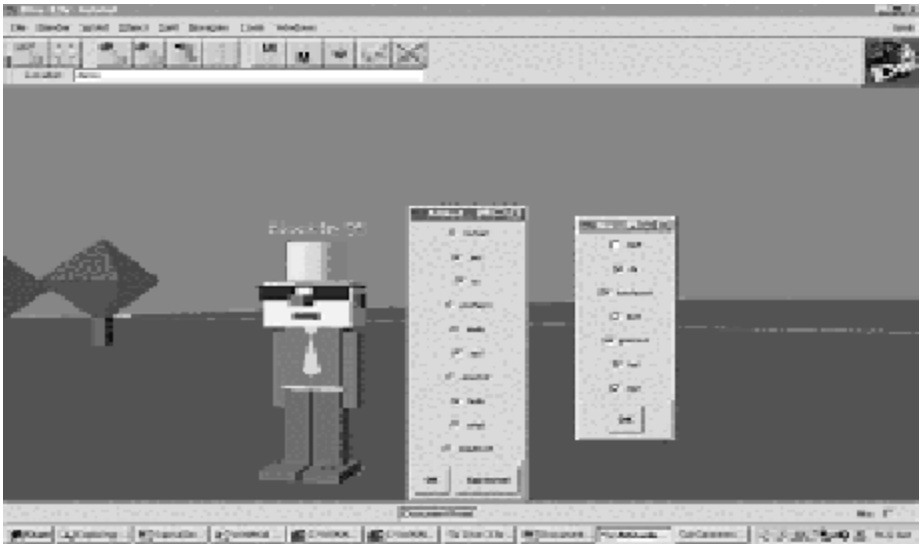

Figure 3.6: DIVE Interaction control panel

using the right mouse button for 5 seconds and 20 seconds later the packet capture was stopped. Taken together, the experiments 7 and 8 study customized object behavior for click interactions and standard object behavior for translation and rotation.

- Interactions - More customized behavior - The robot object has been programmed to respond in various ways to mouse clicks through a customized control panel specific to this object. The control panel, shown in Figure 3.6, has commands to change the expression on the robots face, make the robot nod its head, bend at the waist, tap its foot etc. Also its garment can be modified using the garment control panel, for e.g., remove the glasses, the belt, the tie, the hat; replace the slacks with a skirt, etc. So, it is obvious that it is highly object-specific behavior that involves a few rotations (nod of the head, bending at the waist), but largely aesthetic in nature. This study is interesting because, according to DIVEs philosophy each local host performs as many operations locally as possible, and tries to minimize network traffic. This experiment is an ideal candidate as the operations mostly affect the appearance of the object and does not cause any effect on other objects. The sequence of operations were are follows: At steady-state the controls panels are opened and ready, ethereal was started and in 20 second intervals the following operations were performed - nod (button "yes"), bend (button "hello"), tapping the foot (button "impatience"), changed expression to "sad", removed hat, glasses and belt.

Figure 3.7: Output trace file generated by Ethereal. Detailed information about each frame and all the higher layer PDUs that it encapsulates are generated. Of importance to our study is the time of creation of each frame and the size of the encapsulated UDP packet.

- Dynamic world Composition - DIVE permits new objects to be added to a world on the fly by downloading the object definition file using a URL. The new object can be loaded in front of the requesting participants avatar. This operation involves loading the object and updating the world database followed by informing all the other participants of this new object. It was tested by the following sequence of operations - at steady-state, ethereal is started, 20 seconds later the new object is loaded from a file URL (object definition is available locally) and packet capture was stopped 20 seconds later.

The packet traces captured by ethereal are written to a text file for further analysis. The information we were interested in is of two types - the number of packets generated at the specific points in time when actions were performed and the size of each packet. Packet contents are not important either to the study of network behavior or to model applications later in OPNET. Hence we did not attempt to analyze the data in each packet. The packet data generated by ethereal gives detailed information about each data link layer frame placed into the network, as shown in Figure 3.7. It then breaks it down to the various higher level protocol PDUs embedded in the frame up to the UDP layer. It separates UDP header and data information. In each frame, we were interested in the time instant at which the frame was generated and the size of the UDP packet in the frame. We were also interested in

analyzing this information over a number of experiments to identify any existing pattern of traffic generation and to filter out avatar and object specific packets, if any. A number of programs were written in java to automate this data-mining task. Each program read in a set of ethereal output trace files listed in a configuration file, extracted the requisite information and then wrote the result to either a text file or a comma-separated-values (.csv) file. The latter was then input to Microsoft Excel to produce charts, which are good visual aids and easier to interpret. A program was also written that could analyze the large amounts of data mined from each trace file for commonalities and patterns. For example, all the output trace files for a join operation were input to a java program, which would then extract all the different packet sizes in each trace file and produce a .csv output file each. The .csv files were then input to another java program that would look for patterns and commonalities in the packet sizes, which helped filter out avatar specific and object specific packets and identify the packets generated exclusively by the join operation.

3.6 Off-line Dynamic Programming Optimization

Dynamic programming approach is used to find the combination of item counts for all the flows that maximizes the objective function. The item count for each flow is considered to be a single stage. For this problem, item count for any flow is assumed to be an integer value that can vary between the minimum and maximum for that particular flow. In the rest of the paper an example with four different flows will be used.

Let us consider the following definitions: q_1 is the quantity of items in flow 1; q_2 is the quantity of items in flows 1 and 2; q_3 is the quantity of items in flows 1, 2 and 3; and q_4 is the quantity of items in flows 1, 2, 3 and 4. Each possible value of q_1, q_2, q_3, and q_4 is represented by a node associated with the respective stage i ($i = 1, 2, 3, 4$). The length of arc (q_{i-1}, q_i) that represents reward value of the flow i is denoted as $R_i(q_{i-1}, q_i)$. The length of the shortest path to node q_i at stage i ($i = 1, 2, 3, 4$) is denoted as $f_i(q_i)$ (by definition $f_0(q_0) = 0$). The reward function for each flow i is: $\text{reward}_\text{i}(q_i) = w_i \times q_i \times r_i$, where w_i is the weight per unit for items in flow i and r_i is the reward per unit for items in flow i.

The Dynamic Programming recursive equations are:

$$\begin{aligned} f_0(q_0) &= 0 \\ f_j(q_i) &= \max_{\text{feasible } (q_{i-1}, q_i)} \{f_{i-1}(q_{i-1}) + R_i(q_{i-1}, q_i) \qquad i = 1, 2, 3, 4 \end{aligned}$$

The basic constraint on the knapsack is that the capacity C (total weight) must be greater

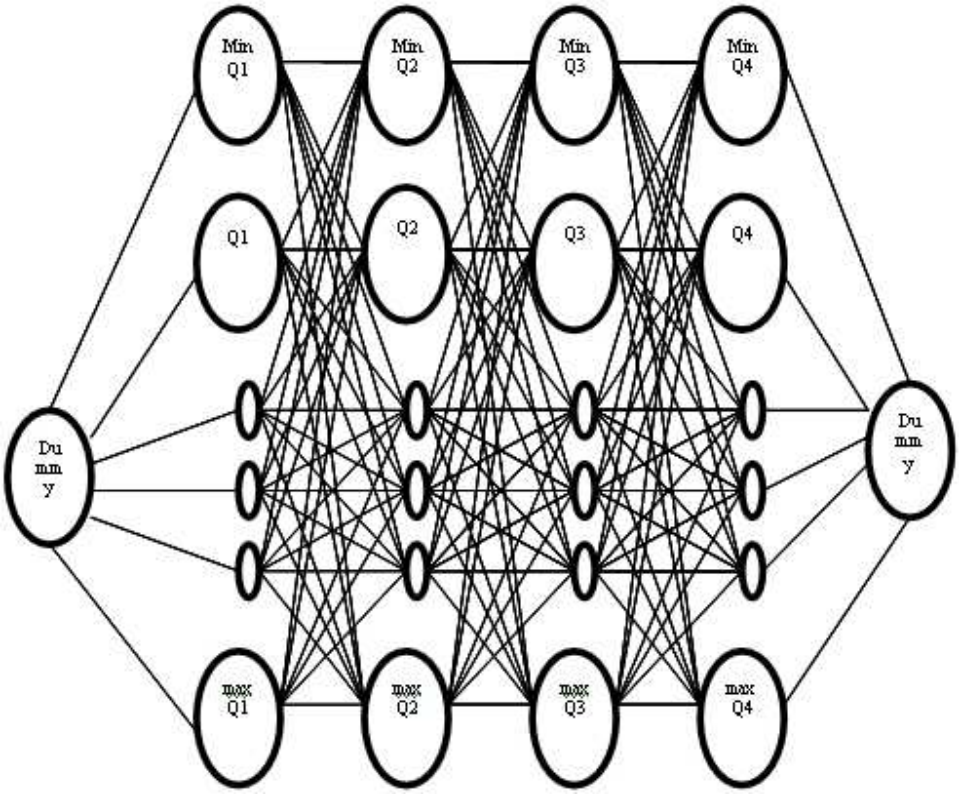

Figure 3.8: The Dynamic programming network model

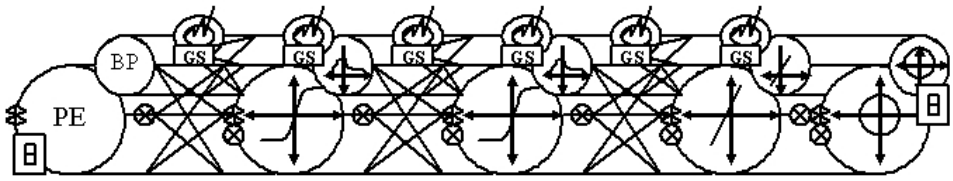

Figure 3.9: The Multi Layer Perceptron Network with feed forward elements

or equal to $\sum_{i=1}^{4} w_i \times q_i \times r_i$. The other constraint on the Dynamic Programming involves the minimum and maximum quantities admissible for each flow. The optimized combination of item quantities for the four stages is the combination of the quantities where the total cumulative reward at the final stage is maximal. Given the maximum and minimum quantities for each flow and also the knapsack configuration variables (like the capacity of the knapsack, the weight per unit for each flow of items and the reward per unit for each flow of items), the Dynamic Programming determines the optimal value of quantities for each flow.

3.7 Neural Network Based Real-Time Optimization

The neural network used in this study is a Multi-Layer Perceptron (MLP). It is the most common supervised neural network. It consists of multiple layers of axons (processing entities) connected in a feed forward fashion. A software tool called NeuroSolutions was used to build and train the neural network. NeuroSolutions uses the back-propagation of errors to train the MLP. The output generated by the Dynamic Programming algorithm is reformatted in such a way that it takes into account the arrival pattern generated by the Petri-net model. This arrival pattern and optimal solution are written as input-output mappings. The mapping is constructed in intervals of δt. The value of δt is chosen in such a way that, there could possibly be only one arrival in any of the flows. An example mapping structure is shown in Table 3.1. The column t is the cumulative time. It increases by increments of δt. The columns Rq_1 to Rq_4 indicate the cumulative rewards of arrived requests in each flow. The columns A_1 to A_4 indicate the cumulative reward of accepted items in that particular flow after the particular request. The columns R_1 to R_4 indicate the difference between the reward if the request was accepted and the optimal reward. As stated above, if this value is negative or zero, the request is accepted, else the request is rejected. All columns except the column t are fed to the MLP since the input-output mapping is time independent. The columns Rq_1 to Rq_4 and A_1 to A_4 are used as inputs. The columns R_1 to R_4 are used as

t	Rq_1	Rq_2	Rq_3	Rq_4	A_1	A_2	A_3	A_4	R_1	R_2	R_3	R_4
δt	**36**	0	0	0	36	0	0	0	-828	-96	-105	-90
$2\delta t$	36	0	0	**30**	36	0	0	30	-828	-96	-105	-60
$3\delta t$	36	0	**35**	**60**	36	0	35	60	-828	-96	-70	-30
$4\delta t$	36	**24**	35	60	36	0	35	60	-828	-96	-70	-30
$5\delta t$	36	24	35	60	36	0	35	60	-828	-96	-70	-30
$6\delta t$	36	**48**	35	**90**	36	24	35	90	-828	-72	-70	0
$7\delta t$	36	48	35	90	36	24	35	90	-828	-72	-70	0
$8\delta t$	**72**	**72**	35	**120**	72	48	35	90	-792	-48	-70	30

Table 3.1: Input and output mapping

outputs. A neural network is constructed for each output. Hence, in this scenario we had 4 neural networks.

CHAPTER 4
MEASUREMENTS AND RESULTS

Shown below are the graphs depicting the packet count at 1 second intervals produced for some of the experiments.

4.1 Participant Join and Steady State

The figures 4.1 and 4.2 below show two join experiments using two separate avatars.

The spike of packets in the 0-1 second interval signifies the packets generated for a new participant. After the initial spike, the traffic settles down to a steady rate. These packets are significant in that they are avatar specific. In Figure 4.1 7 packets of size 1380 bytes are generated in 40 seconds and in Figure 4.2, a total of 7 packets were generated in 40 seconds, their sizes ranging from 1080 bytes to 1472 bytes. These packets are always preceded by one 120 byte packet. Similar results with varying avatar specific packet sizes were observed in the other experiments as well. Hence it can be concluded that during the join operation, there is a spike of packets including avatar specific packets and join operation specific packets. At steady-state there exist avatar specific entity state packets at a rate of 1 every 5 seconds.

4.2 New participant in an existing world

As evident from Figure 4.3, when a second user joins the world, a spike of packets is generated. Before the instant of the join, only the first users avatar-specific entity state packets existed in the network. After the second user joined the world, its entity packets also start arriving at the rate of 1 in every 5 seconds. Hence it is obvious that the number of these kinds of packets will grow linearly with the number of participants in a simplistic application such as this one. A point to note here is that irrespective of the number of users, a 100 byte packet is generated every second as evident from Figures 4.1, 4.2 and 4.3. It is not clear what these packets contain, but the results are very consistent.

4.3 Participant Exit

Figure 4.4 shows that at the 30th second when one of the two participants exited the world, two packets were generated of sizes 92 and 112 bytes each. The exit can be confirmed by the fact that before the 30th second there were twice the number of avatar-specific entity packets,

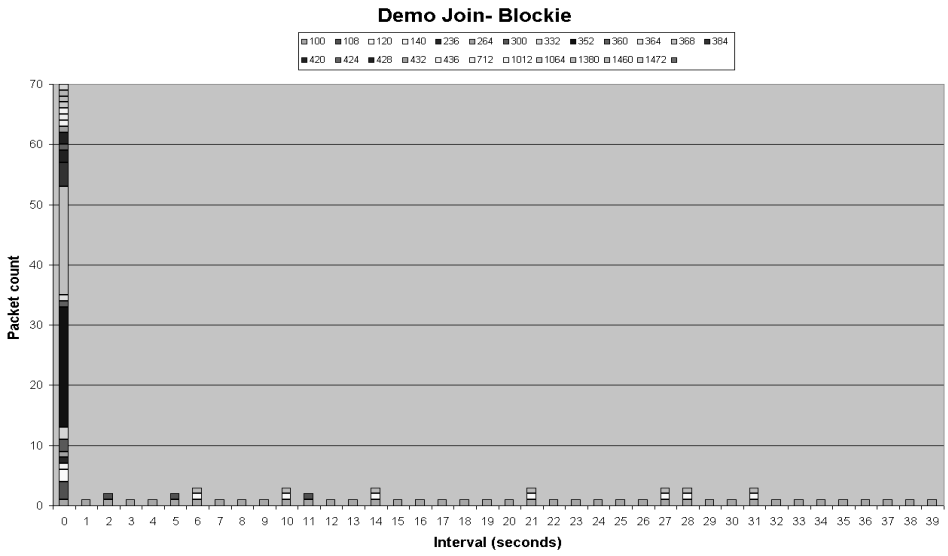

Figure 4.1: Join Operation. First participant joins the world.

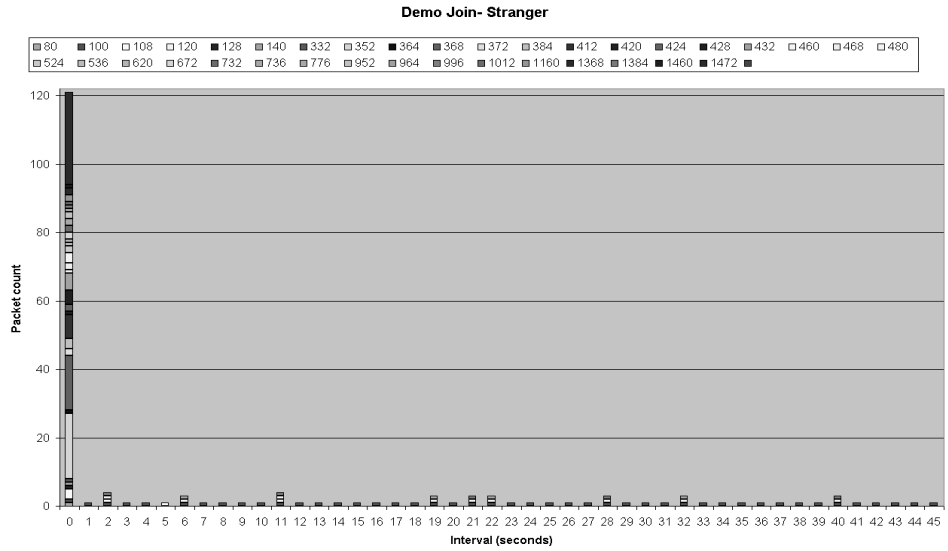

Figure 4.2: Join Operation. First participant joins the world with human avatar.

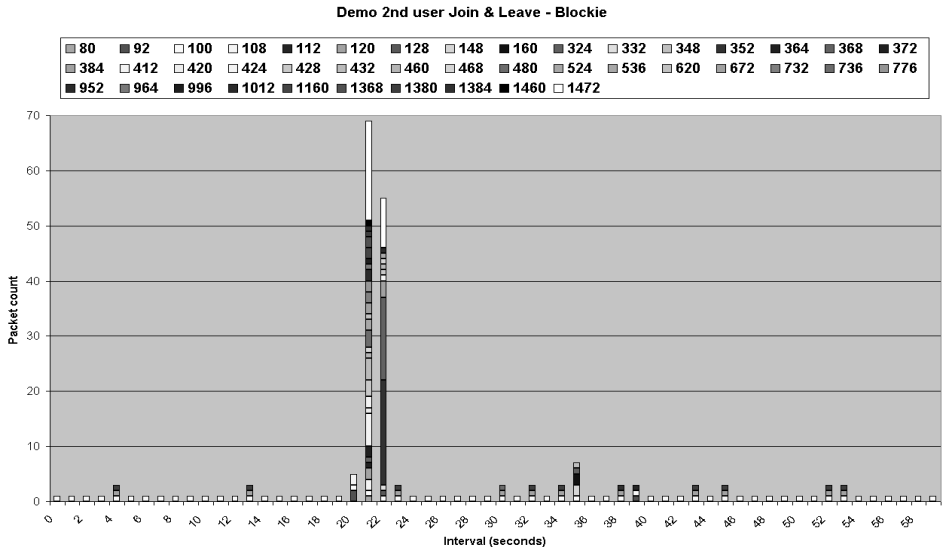

Figure 4.3: Second participant joins the world in the 20th second.

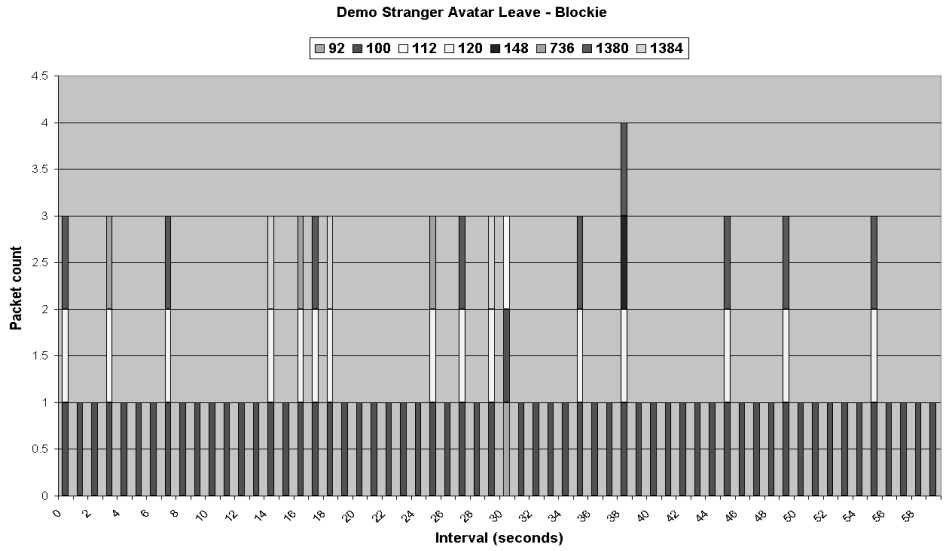

Figure 4.4: Participant Exit Operation. One of the two participants leaves the world.

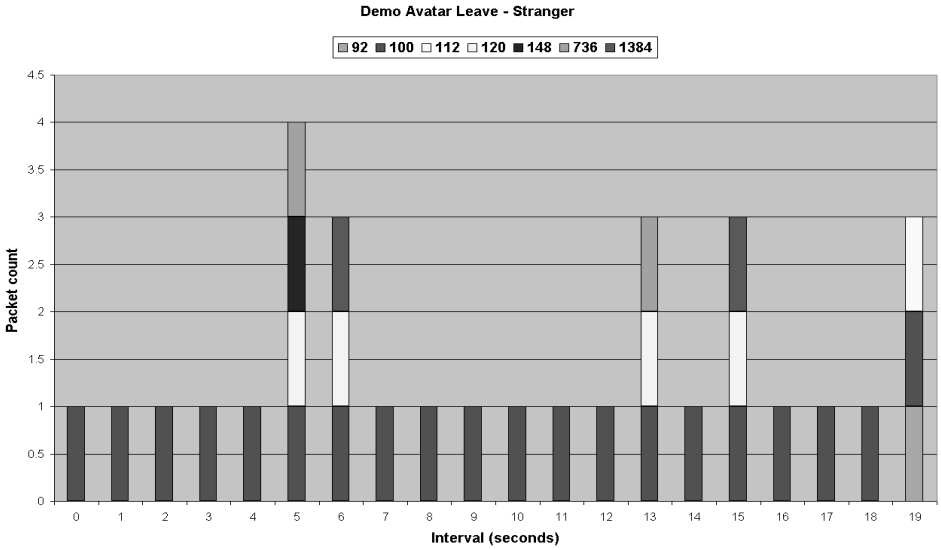

Figure 4.5: Participant Exit Operation. Last participant leaves the world

which were verified for correctness, and afterwards the number of packets are reduced to half and corresponds to the avatar of the remaining participant. Figure 4.5 shows that the same 2 packets of sizes 92 and 112 bytes are generated when the last participant leaves the world Another point to note is that there is no network traffic at all following the exit of the last user.

4.4 Keyboard Move

The Figures 4.6 and 4.7 below show the traffic generated by an avatar move and rotate operations using the arrow keys on the keyboard. In the Figure 4.7, the participant starts moving forward at the 20th second and continues to move for 10 seconds. As evident from the graphs, this action generates only 2 packets each of size 200 bytes, one at the start of the move operation and one in the end. Hence it can be concluded that all the changes to the scene as seen by the user due to the move operation are processed and rendered locally. Other users see this participant move within the world and it can be safely concluded that some form of dead reckoning together with convergence is employed to display the action on other hosts.

The same conclusions can be drawn from Figure 4.8 about the rotation operation. A 200B packet is generated at the start of the rotation to the left at the 20th second and one in the end at the 25th second. Rotation to the right started at the 25th second as indicated

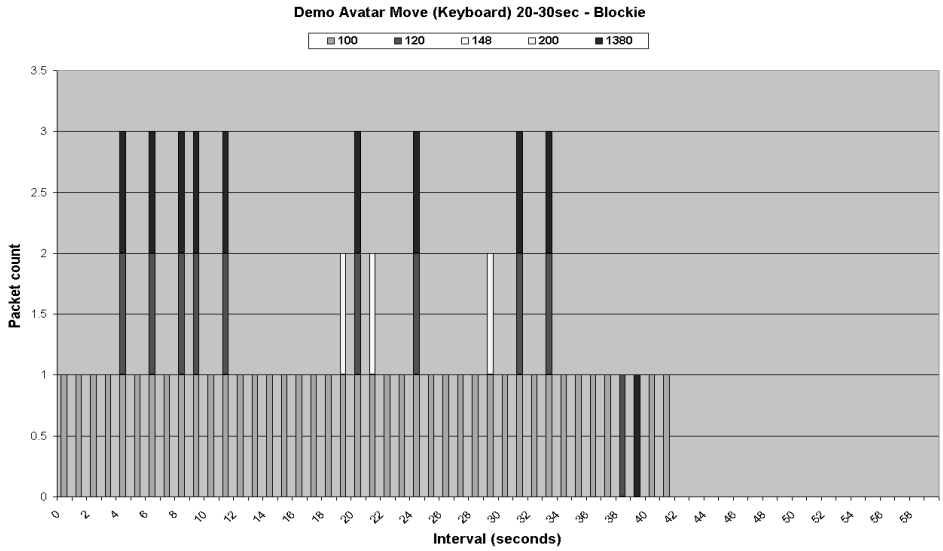

Figure 4.6: Single step move using the keyboard, Blockie Avatar

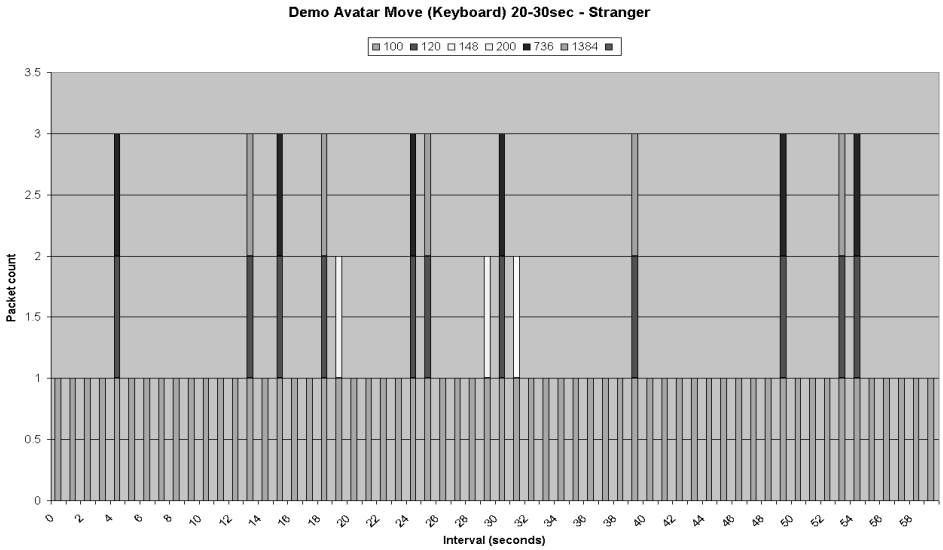

Figure 4.7: Single step move using the keyboard, Stranger Avatar

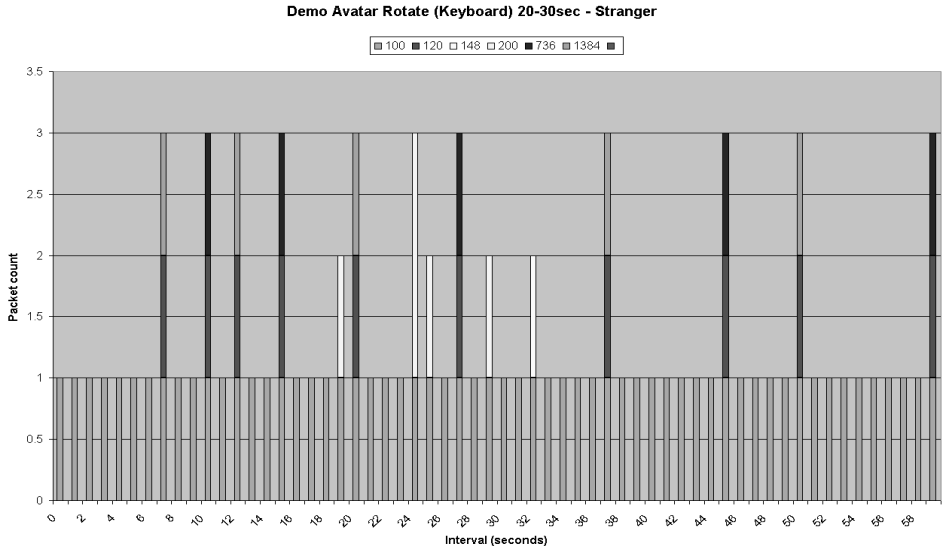

Figure 4.8: Single step Rotate using the keyboard, Stranger Avatar

by the third 200B packet and ended at the 30th second with another 200 byte sized packet. All operations affecting the presentation of the scene and entity state are performed locally at all hosts. It should be mentioned here that while the arrival of 200 byte packets at the start and end of the operation was consistent across all the experiments, the number of 200 byte packets varied from 1 to 2. Hence, it would be safer to assume a uniform distribution between 1 and 2 for the number of packets generated by a move operation.

4.5 Interactions

- Clicking, moving and rotating objects using the mouse As explained earlier, the first two operations are object specific behavior in response to mouse click events, while the latter two are standard operations supported by the DIVE client. From the graph in Figure 4.9 it can be seen that a mouse event generates a 144byte packet at the time the event was triggered. Subsequent traffic depends on the type of mouse event and the response. Standard move and rotate operations using the mouse generates a 132B packet at the end of the action, here after 5 seconds at the end of the translation/rotation. The red box rotated upon clicking, which resulted in another 144B packet along with 172B packets. Packets of size 172B have consistently appeared whenever any objects position was changed. Hence we can assume that if an action results in a change in an object location/orientation 172B packets are generated. The number of such packets

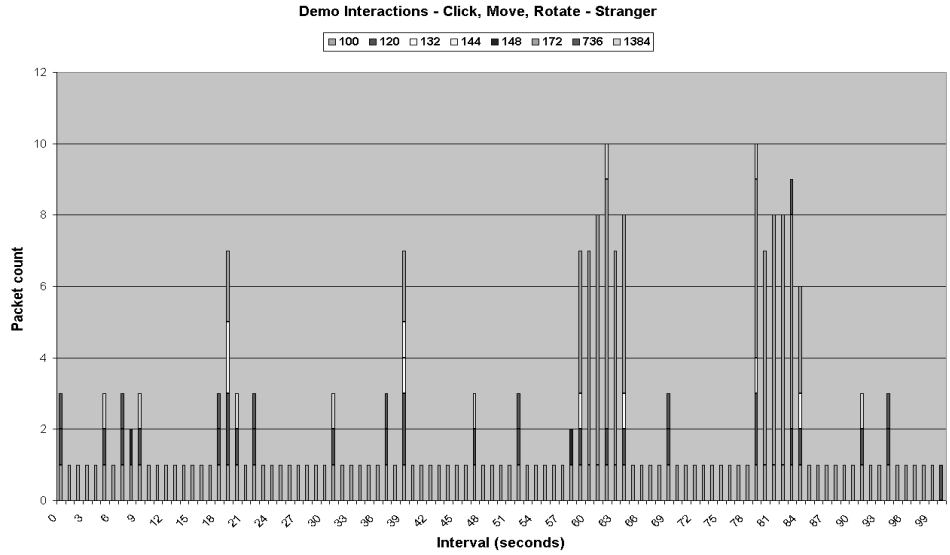

Figure 4.9: Mouse Interactions in general

ranged anywhere from 2 to 20.

- Object-Specific responses to interactions. Figures 4.10 and 4.11 show some of the traffic generated by customized behavior of objects.Stranger object has human-like limbs and body that can move like a normal person. The resulting actions are complex and involve translations and rotations along numerous axes. Sufficient experiments were not conducted to identify any pattern and commonalities in the packets generated by specialized object behavior to interactions. Hence more research is needed in this area.

4.6 Dynamic World Composition

The Figure 4.12 show the packets generated when a new object was added to the existing demo world. In this case, the object added was the blockie object itself. The world had one participant, whose avatar was a blockie too, as verified by the avatar-specific entity packets generated during steady-state. Figure 4.1 showed the packet generated when a participant joined with the blockie avatar. Visually comparing Figures 4.1 and 4.12 shows that a number of common packets are generated in both cases; these can be concluded to be the result of the blockie definition. Every new object produces traffic specific to it. Hence, the only generalization that can be drawn about traffic generated by the addition of new objects is in

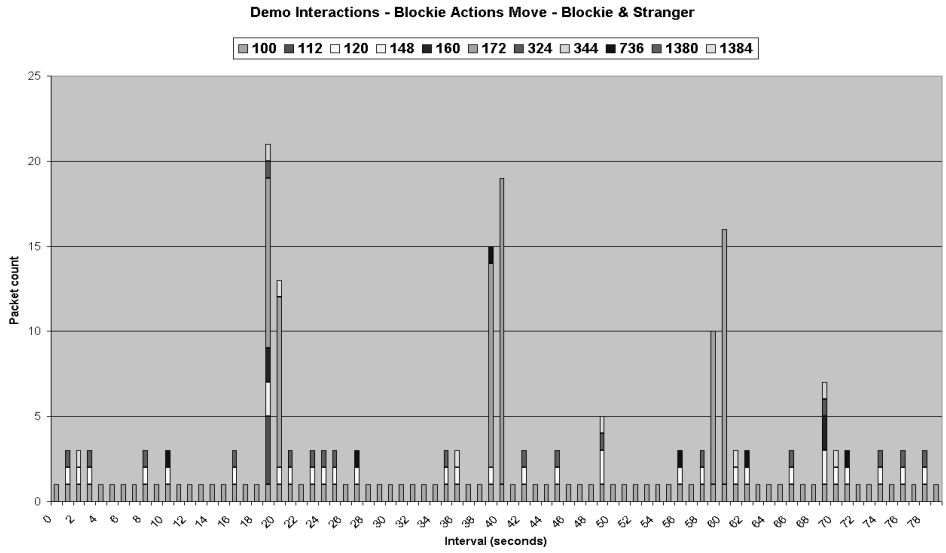

Figure 4.10: Object specific behavior to interactions. The robot object nods its head at the 20th second, bends at the waist in the 40th second and taps its foot at the 60th second.

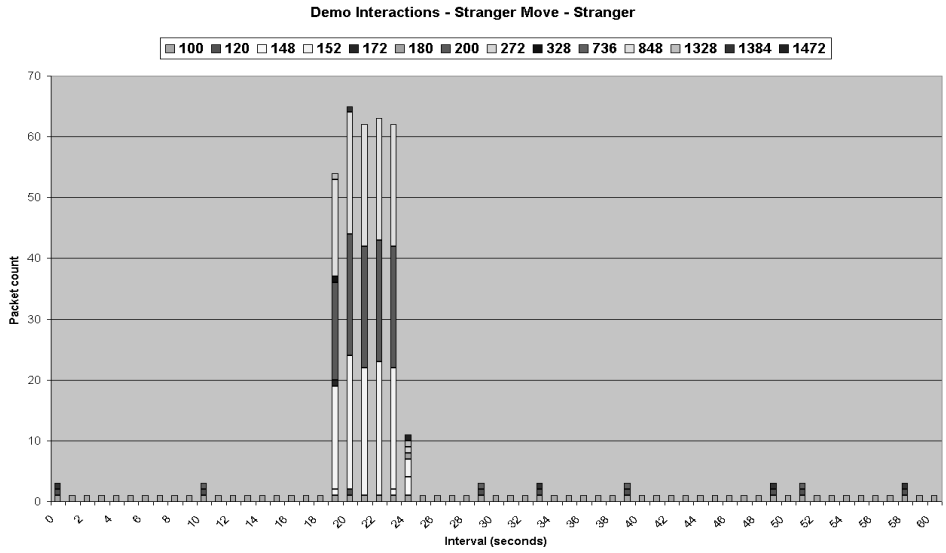

Figure 4.11: Object specific behavior; shows move operation of a sophisticated avatar, stranger.

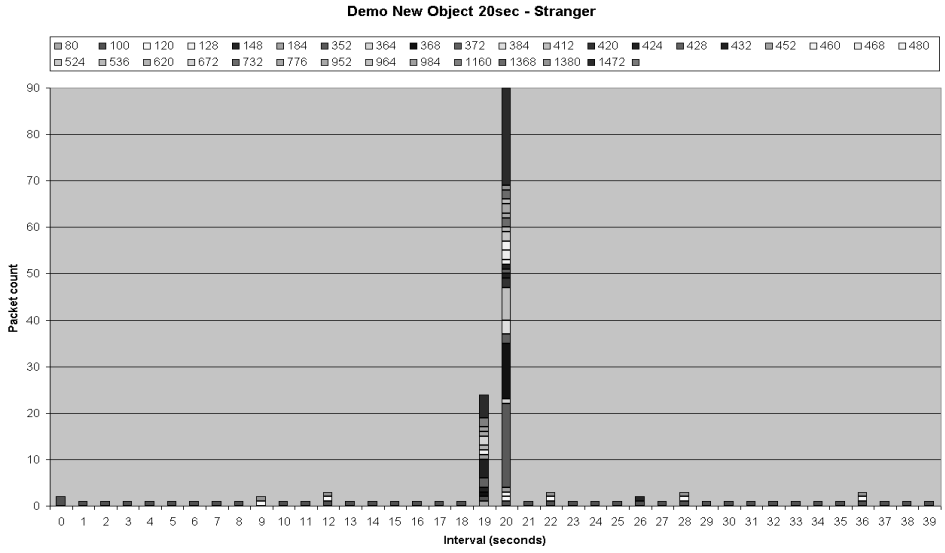

Figure 4.12: Dynamic World Composition. A new blockie object was added to the world at the 20th second.

terms of range of packet sizes and numbers. It was found that objects of various complexities generated packets ranging in size from 80 to 1500 bytes for low to high complexity and their numbers ranged from 1 to 100.

The conclusions drawn from the experiments are summarized in the Table 4.1 below.

4.7 Results

Data from the Dynamic Programming optimized knapsack was collected for up to 100 different arrival patterns. The reformatted data used as input to the neural network amounted to about 45000 rows of data. Of these, 20000 rows were used to train the neural network. The remaining rows were used for testing the trained MLP network. A sample of the testing output obtained from the neural network is shown here. The sample output shown is for one output variable R_2 obtained from its corresponding MLP network. Similar results were obtained for other output variables. Table 4.2 shows optimal (expected) values and values obtained (generated) by NeuroSolutions. The significant point to be noted is that the expected and obtained values ranged from -1300 to 10000. The difference between the expected and obtained values of the output resulted in a Mean Squared Error (MSE) value of 0.000001. The learning curve of the MLP and a plot obtained between the desired output and obtained output is shown in Figure 4.13. The simulations were executed on a Pentium 4

Action	Size(B)	No.	Time(secs)	Distr.	Rate/sec
All	100	1		Uniorm	1
Automatic Ping	148	1		Uniorm	0.05
User Join	108	3	1		
			12	Uniform	0.25
	140	1	1		
	1472	1	1,prob-0.7		
		10-60	1,prob -0.3		
	332	1	1		
	352	2-20	1		
	368	2-25	1		
	428	1-20	1		
	1012	1-2	1		
	120	1	1		
Avatar specific	100-1400	1	1		
Object specific	50-1500	30-140	1		
User Exit	92	1	1		
User Exit	112	1	1		
Avatar move	200	1	1 at start		
Avatar rotate	200	1	1 at move end		
(keyboard)	200	1	1 at start		
Steady State	200	1	1 at end		
Entity pkts	100-1400	1		Uniform	0.2
Mouse Event	144	1	1 at start of action		
Mouse Move Rotate	132	1	End of move rotate		
Object	172	2-20	Every sec of move		

Table 4.1: Summary of request packets generated with each user action

based computer running the Windows operating system. The dynamic programming transformations completed execution in forty minutes for a hundred arrival patterns. The neural network training sessions lasted for thirty minutes.

4.8 Case Study: Network Bandwidth Allocation in a DVE

We conduct experiments based on the generalized Dynamic Stochastic Knapsack model with a specific Resource case in DVEs namely Network Bandwidth Allocation. It was for a similar case that the first Knapsack problem was originally formulated. The original knapsack problem was formulated to handle call admission policies in the telecommunication industry. Optimal call admission policies were initially arrived at using Offline methods like Operations

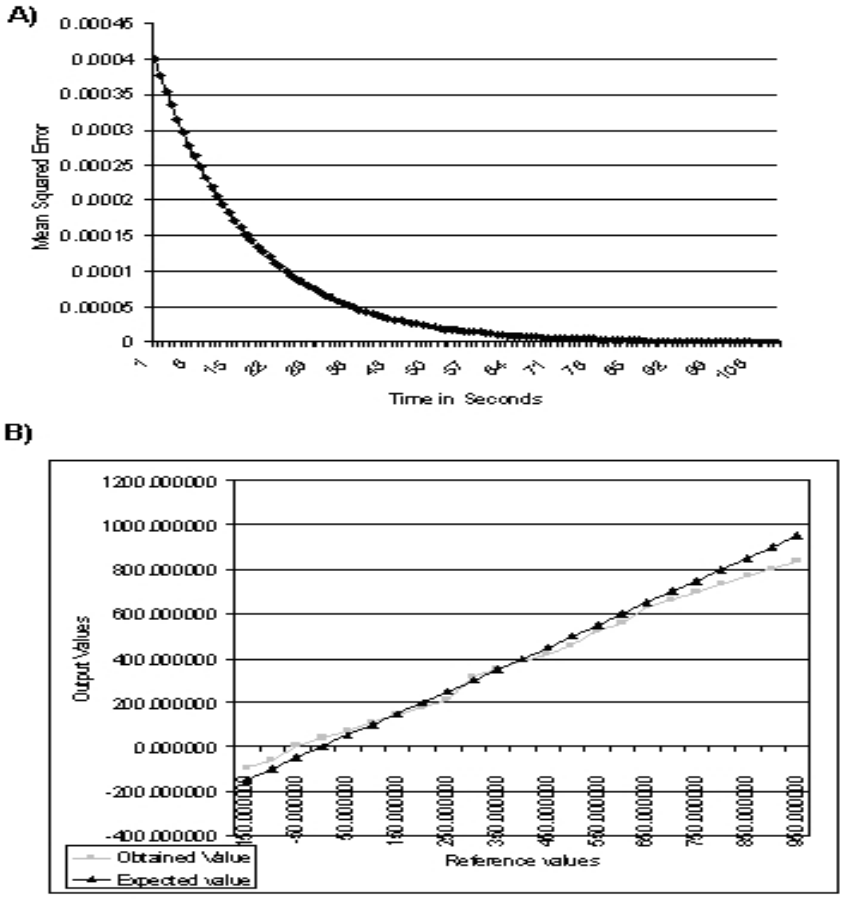

Figure 4.13: A) Learning curve (MSE versus time) and B) The results

Optimal	Obtained	Optimal	Obtained
-104.999893	-97.305122	-104.999893	-97.305122
-70.000038	-62.999664	-70.000038	-62.999664
0.000000	5.957107	0.000000	5.957107
34.999851	40.232655	34.999851	40.232655
70.000038	74.710869	70.000038	74.710869
104.999893	108.949783	104.999893	108.949783
140.000076	143.563110	140.000076	143.563110
174.999924	178.090408	174.999924	178.090408
210.000122	212.387802	210.000122	212.387802
315.000000	316.538696	315.000000	316.538696
349.999847	351.236389	349.999847	351.236389

Table 4.2: Desired versus obtained output

Research techniques.

We took the case of the DIVE distributed virtual environment system. The traffic generated for each user action, interaction scenario, steady state scenario, movement and navigation etc was measured as explained in section 3. Each scenario has an associated number of packets generated. This constitutes the request for the bandwidth resource (Knapsack space). The actual bandwidth allocated to this application at each resource controller will be te equivalent of the knapsack capacity. The DVE application's network controller's primary "resource allocation decision" component is the neural network based real-time we have designed. The reward associated with each request is the measure of QoS needed for each class of request.

In this case study we treat network bandwidth as the resource for which there are various contenders. As in the Generalized Dynamic Stochastic Problem we have various classes of requests. The requests have been classified for this study as follows

- Steady State Packets
- Interaction (Keyboard and Mouse Based interactions)
- Object State, User State
- Movement and navigation (Translation, Rotation, Specialized movements like walking running etc.)
- Load Objects, New User Join Operation
- Remove Objects, User Leave operations

Each class of requests has a reward associated with it. Also each class of requests will have a specific size associated with each request. Each request has a constant part and an optional variable part. The command part of each request forms the constant part and the data associated may vary with the specific request. Also each class of requests arrives at a specific rate. The rate we have applied to this case study is based on models of interaction and navigation suggested by Greenhalgh et al.[] from their research.

We associate the rate of requests an active user would make for each class of requests and extend it to "multiple-user" scenarios. The requests are assumed to arrive following a poisson distribtion hence we generate an arrival pattern following this distribution at a mean rate that we specified above for each class and subclass of request.

For example a user join operation has commands that join the user and load objects into his virtual environment. The data part will consist of the objects themselves, the environment and world description, the user avatar definition etc. Whereas a User Leave operation has only the leave command. No variable part is associated since a leave operation does not include transporting object data. A simlar case is applied for the Object load and Unload operations.

Class ID	Reward
1	45
2	25
3	15
4	10
5	4
6	1

Table 4.3: Reward configuration for each class of requests in this study

Constant	Value
Capacity	6000 to 10000 Bps
Decision Interval δt	0.01 seconds
Time for each sample	90 seconds
Number of samples	6
User count	45,50,55,60, 65, 70

Table 4.4: Experiment constants in this study

The higher the reward for a class of requests, the higher the QoS and priority associated while allocating bandwidth for that request. Shown above is a table containing the configuration variables for the network resource allocation problem. The typical arrival rates,

packet classification, and packet sizes for each user is shown in the table below. We ran the

Class Description	Packet Size	Class ID	Arrival Rate
Steady State	100	6	60.0
Steady State	120	6	12.0
Steady State	148	6	12.0
Steady State	736	5	3.0
Steady State	1384	5	6.0
Interaction	316	2	20.0
Movement	448	2	15.0
Object State	-2474	1	5.0
User State	-1384	1	5.0
Join Operation	-22404	4	1.0
Load Object	-5424	4	2.0
Leave Operation	240	3	1.0
Remove Object	436	3	2.0

Table 4.5: Typical arrival rates, packet classification, and packet sizes for each user

knapsack model finetuned for optimal network bandwidth allocation and request generation. Dynamic Programming was applied to the Network resource to calculate the optimal allocation decision at each stage. The optimal resource allocation table is shown below. This is derived from the offline technique. A sample is shown below. The first 6 columns depict the request in each flow. The next six columns show the optimal output generated by the offline technique. To reduce the error that the perceptron has while learning, we assign one neural netowrk for each output. In this case we have six MLPs. The trained neural network was able to achive a Mean Squared Error of 0.000001. The expected outputs and obtained output samples for the Rs1 output is tabulated here.

The learning curve and Expected Vs. Obtained output plots for the Mult-Layer Perceptron are shown below.

We summarze the results of this study and the resarch in the next chapter abd describe a possible architecture that would utilize this trained neural network to implement optimal network resource allocation in DVEs.

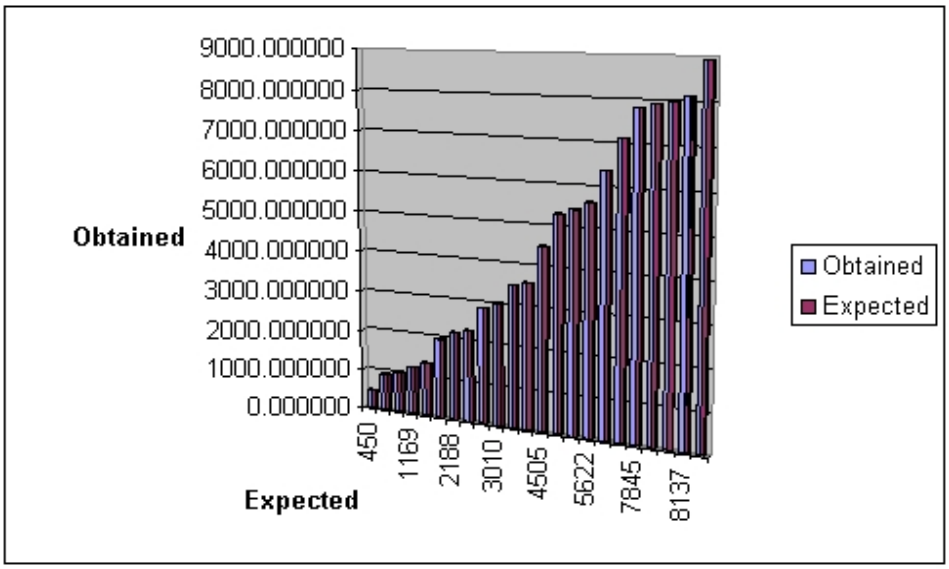

Figure 4.14: Comparison between expected and obtained outputs from various parts of the result

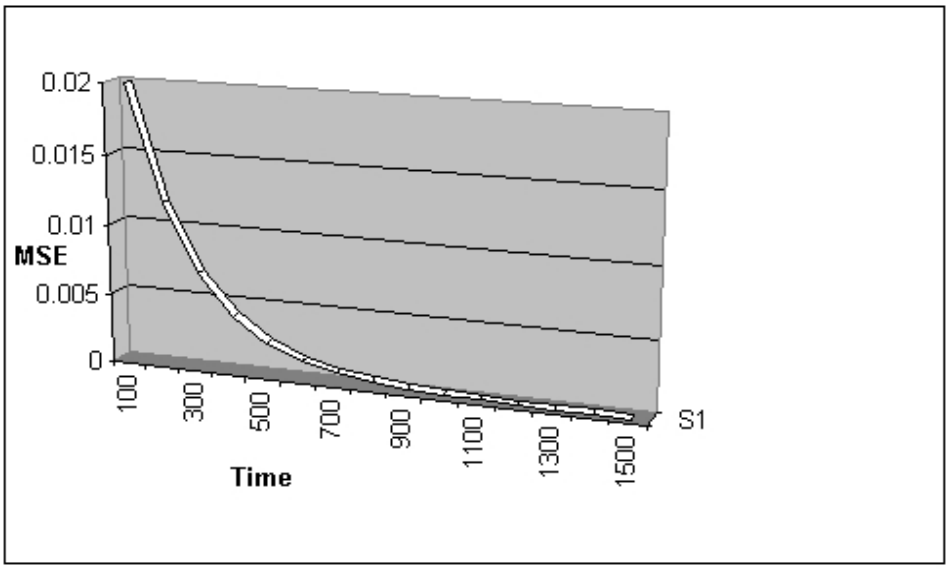

Figure 4.15: Learning Curve as a plot of Mean Square Error Vs. Time

Interval	Rq1	Rq2	Rq3	Rq4	Rq5	Rq6	Rs1	Rs2	Rs3	Rs4	Rs5	Rs6
$1\delta t$	3577	448	0	0	0	300	3577	448	0	0	0	300
$2\delta t$	0	1080	0	0	0	348	0	1080	0	0	0	348
$3\delta t$	16170	316	0	0	0	440	10000	0	0	0	0	0
$4\delta t$	6170	632	0	0	0	940	6170	632	0	0	0	940
$5\delta t$	0	316	0	6529	0	348	0	316	0	6529	0	348
$6\delta t$	2689	316	240	0	0	248	2689	316	240	0	0	248
$7\delta t$	0	1528	436	0	2856	320	0	1528	436	0	2856	320
$8\delta t$	0	764	0	10820	1384	628	0	764	0	9236	0	0
$9\delta t$	0	1528	240	1584	2768	1228	0	1528	240	1584	2768	1228
$10\delta t$	0	1080	0	0	0	220	0	1080	0	0	0	220
$11\delta t$	0	1344	0	0	0	248	0	1344	0	0	0	248
$12\delta t$	0	1080	0	10099	0	400	0	1080	0	8920	0	0
$13\delta t$	3956	1528	0	13817	0	748	3956	1528	0	4516	0	0
$14\delta t$	0	896	0	9301	0	948	0	896	0	9104	0	0
..	..	..	..	..	..	..	..	..	..	..	..	..
$9000\delta t$	..	..	..	..	..	..	..	..	..	..	..	..

Table 4.6: Typical arrival rates, packet classification, and packet sizes for each user

Expected value	Obtained Value
3577	3576.989001
0	0.001101
10000	9999.987771
6170	6169.964565
0	0.100237
2689	2689.019862
0	0.001802
0	0.017201
0	0.153702
0	0.062331
3956	3957.083961
0	0.249206
6838	6837.670023
3189	3188.734507

Table 4.7: Expected and Obtained outputs for Request category 1: Highest QoS

CHAPTER 5
CONCLUSIONS AND PROPOSALS

5.1 Conclusions

Not many DVE architectures address QoS as a direct issue. In this research we adopted a resource allocation perspective to Quality of Service after analyzing various existing architectures. We emphasize the need for a dynamic resource management subsystem that takes the current resource request and availability scenario into consideration and provides results based on learning from mathematical optimization models. It is also important that the resource manager itself does not become a hurdle to QoS. It must hence produce allocation decisions in real-time.

We thus introduced a novel use of machine learning that produces results in real-time based the knowledge it gains from the offline optimization techniques. We modeled the resources in a DVE as a generalized Dynamic Stochastic Knapsack, simulated the arrival of requests to the knapsack, applied the dynamic programming optimizations for these request arrival patterns, compiled this data and fed it to a Multi-Layer Perceptron Network and used this trained neural network as resource allocation decision support tool. We demonstrated from our experiments that the expected and obtained decision outputs from this trained neural network were very close with a Mean Square error of around 0.00001. Also this output was received in several microseconds. This is very desirable in a distributed virtual environment.

5.2 Proposed Architecture

In this section we propose a QoS architecture based on the above resource allocation tool.

AOIMP refers to Area of Interest Management Proxy Server Process KINP refers to Knowledge implementing Neural-Network based Proxy server process KANS refers to Knowledge acquiring Neural Server (Learning and Offline Optimization Server)

All KINPs and KANSs collectively form the decision service subsystem that provide the resource allocation decision support. The AOIM processes reduce the amount of messages released in the network. Each DVE client registers with one Proxy server. The KINP, AOIMP and session management proxies can be implemented as light weight kernel processes on a single proxy server machine, since all these tasks have to be carried out for each client.

Each KINP proxy server must be totally synchronized with all other KINP processes. These KINP processes would contain the trained neural network that would provide resource

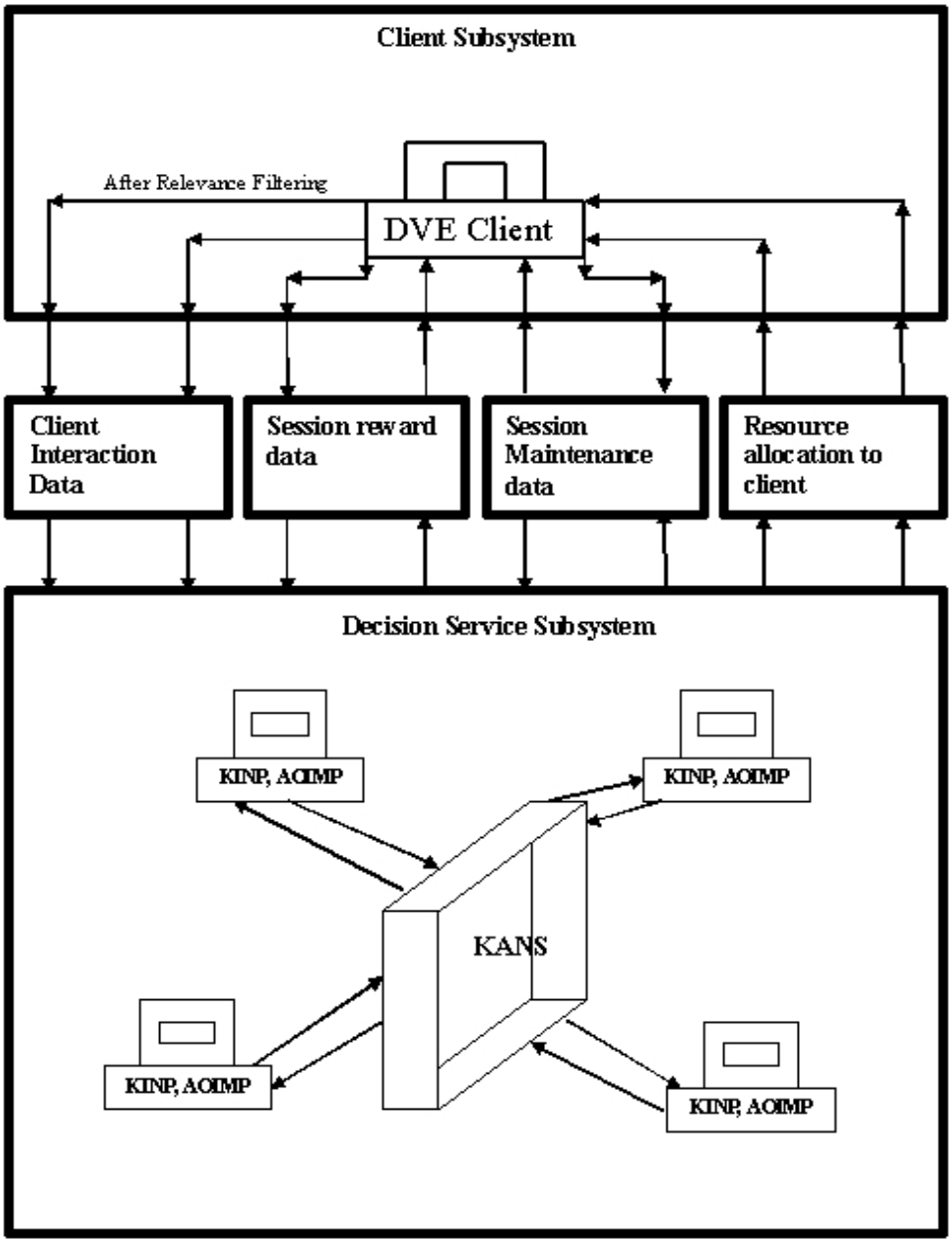

Figure 5.1: Proposed QoS Architecture based on Resource allocation technique developed in this study

allocation decisions based on the current resource request and usage scenario at its clients. This scenario is calculated based on the interaction data and session reward data passed on to KINP by the client. KINPs also pass on current client interaction data and resource usage data to the KANS server.

The KANS server is in a state of constant learning with inputs provided by the KINP proxies. This data is fed to Offline Optimizes and the knowledge of optimal solutions gained through this process is used to educate the neural network. The trained neural network's weights are then updated at all the KINP proxies. It should be mentioned again that the Multi Layer Perceptron learns by updating the weights on the connections between its layers.

The connection between the proxy servers and the KANS servers is assumed to be stable and of very high speed. Each client can have broadband connections to the network.

REFERENCES

[1] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In In Proc. Computer Supported Cooperative Work (CSCW 92), 1992.

[2] T. A. Funkhouser. RING: A Client-Server System for Multi-User Virtual Environments. Computer Graphics (1995 SIGGRAPH Symposium on Interactive 3D Graphics), Monterey, CA, April, 1995, p. 85-92.

[3] M. Antunes , A. R. Silva and J. Martins, An abstraction for awareness management in collaborative virtual environments, Proceedings of the ACM symposium on Virtual reality software and technology, November 15-17, 2001

[4] J. Barrus, R. Waters, and D. Anderson. Locales: Supporting Large Multiuser Virtual Environments. In IEEE Computer Graphics and Applications, pages 16(6):50100, November 1996.

[5] M. Macedonia, M. Zyda, D. Pratt, P. Barham, and S. Zestwitz, "NPSNET: A Network Software Architecture for Large- Scale Virtual Environments", Presence: Teleoperators and Virtual Environments, Vol. 3, N. 4, 1994

[6] C. Greenhalgh and S. Benford. "MASSIVE: a Distributed Virtual Reality System Incorporating Spatial Trading," in Proc. IEEE 15th International Conference on Distributed Computing Systems (DCS'95), Vancouver, Canada, May 30 - June 2, 1995, IEEE Computer Society

[7] C. Benford and L. Fahlen, "A Spatial Model of Interaction in Virtual Environments", in Proc. Third European Conference on Computer Supported Cooperative Work (ECSCW'93), Milano, Italy, September 1993.

[8] C. Greenhalgh and S. Benford, MASSIVE: A Distributed Virtual Reality System Incorporating Spatial Trading, in Proceeding of the 15th International Conference on Distributed Computing Systems(DCS95), 1995, pp.27-34.

[9] C. Greenhalgh (1996): "Dynamic, embodied multicast groups in MASSIVE-2", Technical Report NOTTCS-TR-96-8, Department of Computer Science, The University of Nottingham, UK.

[10] C. Greenhalgh, J. Purbrick and D. Snowdon. Inside MASSIVE-3: Flexible Support for Data Consistency and World Structuring, in Proceedings of the Third ACM Conference on Collaborative Virtual Environments (CVE 2000), September 2000

[11] S. R. Musse , C. Babski , T. Capin and D. Thalmann. Crowd modeling in collaborative virtual environments, Proceedings of the ACM Symposium on Virtual reality software and technology 1998, p.115-123, November 02-05, 1998, Taipei, Taiwan

[12] S. Benford , C. Greenhalgh , D. Lloyd, Crowded collaborative virtual environments, conference proceedings on Human factors in computing systems, p.59-66, March 22-27, 1997, Atlanta, Georgia, United States

[13] S. Pekkola, M. Robinson, M. O. Saarinen , J. Korhonen, S. Hujala, T. Toivonen, Collaborative virtual environments in the year of the dragon, Proceedings of the third international conference on Collaborative virtual environments, p.11-18, September 2000

[14] Wang W., Lin Q., Ng J., Low C.P., SmartCU3D: a Collaborative Virtual Environment System with Behavior Based Interaction Management, in proceedings of ACM, VRST 2001, Canada, Nov, 2001.

[15] Wang W., Lin Q., " SmartCU3D: a Behavior Based Interaction Management Mechanism for Internet CVE, in Proceeding of IEEE ICME'2001, Tokyo, Japan, Aug 2001.

[16] M. Daily, M. Howard, J. Jerald, C. Lee, K. Martin, D. McInnes, P. Tinker, R. Smith. DDRIVE: Distributed Design Review In Virtual Environments. Proceedings of Collaborative Virtual Environments, Septemeber, 2000.

[17] K. S. Park, Y. J. Cho, N. K. Krishnaprasad, C. Scharver, M. J. Lewis, J. Leigh, A. E. Johnson (2000,) CAVERNsoft G2: A Toolkit for High Performance Tele-lmmersive Collaboration. Proceedings of the ACM symposium on Virtual reality software and technology

[18] R.C. Smith, R. R. Pawlicki, J. Leigh, D. Brown, Collaborative VisualEyes, in proceedings of the Fourth International Immersive Projection Technology Workshop,

[19] D. Lloyd, S. Benford, C. Greenhalgh: Formations: explicit group support in collaborative virtual environments. Proceedings of the ACM symposium on Virtual reality software and technology December 1999 162-163

[20] J. Locke, An Introduction to the Internet Networking Environment and SIMNET/DIS,http://wwwnpsnetcs.nps.navy.mil/npsnet/publications/DI.psZ2000

[21] Z. Choukair, D. Retailleau and M. Hellstrom. Environment for Performing Collaborative Distributed Virtual Environments with QoS. Poccedings of the Seventh International Conference on Parallel and Distributed Systems (ICPADS'00) July 04 - 07, 2000

[22] Z. Choukair and D. Retailleau. Integrating QoS to Collaborative Distributed Virtual Reality Applications" Proceedings. Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2000. (ISORC 2000).

[23] C. Greenhalgh, S. Benford and G. Reynard: A QoS architecture for collaborative virtual environments. Proceedings of the seventh ACM international conference on Multimedia (1) 1999: 121-130

[24] A. Varga, M. Hadwiger. The Parsec Networking Architecture. Available from http://www.parsec.org/netdocs/

[25] A. Varga, "PARSEC: Building the networking architecture for a distributed virtual universe", CESCG'99, Budmerice, Slovakia, April 26-27, 1999.

[26] D. Houatra. QoS-Constrained Event Communications in Distributed Virtual Environments. Proccedings of the International Symposium on Distributed Objects and Applications. September 21 - 23, 2000

[27] R. C. Waters, D. B. Anderson and D. L. Schwenke. 'Design of the Interactive Sharing Transfer Protocol', WET ICE '97 – IEEE Sixth Workshops on Enabling Technologies for Collaborative Enterprises: Infrastructure for Collaborative Enterprises, IEEE Computer Society Press, June 1997, pp. 140-147.

[28] W. Broll "DWTP-An Internet Protocol For Shared Virtual Environments," VRML 98 Third Symposium on the Virtual Reality Modeling Language

[29] D. Brutzman, M. Zyda , K. Watsen, and M. Macedonia. Virtual Reality Transfer Protocol (vrtp) Design Rationale, Workshops on Enabling Technology: Infrastructure for Collaborative Enterprises (WET ICE): Sharing a Distributed Virtual Reality, MIT, Cambridge Massachusetts, June, 1997

[30] K. Watsen, and M. Zyda. Bamboo-A Portable System for Dynamically Extensible, Real-time, Networked, Virtual Environments. Virtual Reality Annual International Symposium, Atlanta Georgia, 1998, 252–259.

[31] M. Oliveira, J. Crowcroft and M. Slater. Component framework infrastructure for virtual environments, Proceedings of the third international conference on Collaborative virtual environments, p.139-146, September 2000

[32] O. Hagsand. Interactive Multiuser VEs in the DIVE system. IEEE Multimedia, Spring 1996, 30–39.

[33] E. Frcon, C. Greenhalgh and M. Stenius. The DiveBone-An Application-Level Network Architecture for Internet-Based CVEs. ACM Symposium on Virtual Reality Software and Technology, University College London UK, December 1999.

[34] John L. Robinson, John A. Stewart, Isabelle Labbe: MVIP - audio enabled multicast Proceedings of the fifth symposium on Virtual reality modeling language (Web3D-VRML) February 2000 p103-109

[35] K.L. MORSE. 1997. HLA data distribution management: Design document, version 0.7. Available through the Internet.

[36] K. L. Morse and J.S. Steinman. 1997. Data distribution management in the HLA: Multidimensional regions and physically correct filtering. In Proceedings of the 1997 Spring Workshop on Simulation Interoperability (Mar. 3-7).

[37] R. Gossweiler, C. Long, S. Koga and R. Pausch. DIVER: A Distributed Virtual Environment Research platform Virtual Reality, 1993. Proceedings., IEEE 1993 Symposium on Research Frontiers in , 1993

[38] B. Jung and J. Milde, An open virtual environment for autonomous agents using VRML and Java, Proceedings of the fourth symposium on The virtual reality modeling language, p.7-11, February 23-26, 1999

[39] K. Rothermel, I. Barth and T. Helbig: Cinema - An architecture for distributed multimedia applications. Architecture and Protocols for High-Speed Networks 1993: 253-271

[40] Singhal, S. and Zyda, M. Networked Virtual Environments Design and Implementation. Addison Wesley, July 1999.

[41] W. T. Cai, P. Xavier, S. J. Turner and B. Lee. A Scalable Architecture for Supporting Interactive Games on the Internet. 16th Workshop on Parallel and Distributed Simulation May 12 - 15, 2002

[42] T. K. Capin, I.S. Pandzic, D. Thalmann and M.N. Thalmann Avatars in Networked Virtual Environments, John Wiley, June 1999.

[43] I. Foster, J. Geisler, C. Kesselman and S. Tuecke. Multimethod Communication for High-performance Metacomputing Applications. Proceedings of Super Computing 96

[44] C. Joslin, T. Molet and N. Magnenat-Thalmann: Advanced real-time collaboration over the internet. Proceedings of the ACM Virtual Reality Software and Technology, VRST 2000, October 22-25, 2000 25-32

[45] Watsen, K. and Zyda, M., "Bamboo - Supporting Dynamic Protocols for Virtual Environments", Proc. IMAGE Conference, Arizona, August 1998

[46] S. Pettifer, J. Cook, J. Marsh, and A. West. Deva3: Architecture for a large scale virtual reality system. In Proceedings of ACM Symposium in Virtual Reality Software and Technology 2000, pages 33-39. ACM Press, October 2000. I

[47] H. Yu and A. Vahdat, "Combining Generality and Practicality in a Conit-Based Continuous Consistency Model for Wide-Area Replication." Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS), April 2001

[48] Saar, K. VIRTUS: A Collaborative Multi-User Platform. In Proceedings of the 4 th Symposium on the Virtual Reality Modeling Language (VRML'99), pp.141-152, 1999

[49] M. Mauve and V. Hilt. An application developer's perspective on reliable multicast for distributed interactive media, ACM SIGCOMM Computer Communication Review, v.30 n.3, p.28-38, July 2000

[50] Lim, M., Han, S., and Lee, D., ATLAS Internal Specification. Project Report, 2001. http://cds.icu.ac.kr/research/area/dve.

[51] Park, S., Lee, D., Lim, M. and Yu, C. Scalable Data Management Using User-Based Caching and Prefetching in Distributed Virtual Environments. ACM Symposium on Virtual Reality Software and Technology, Canada, November 2001, 221–226.

[52] Rodger Lea , Yasuaki Honda , Kouichi Matsuda , Satoru Matsuda, Community Place: architecture and performance, Proceedings of the second symposium on Virtual reality modeling language, p.41-50, February 24-26, 1997, Monterey, California, United States

[53] L. Aarhus, K. Holmqvist, M. Kirkengen. Generalized two-tier relevance filtering of computer game update events. Proceedings of the first workshop on Network and system support for games 2002. pp10 - 13

[54] M. Mauve, S. Fischer and J. Widmer. A Generic Proxy System for Networked Computer Games, Proceedings of the first workshop on Network and system support for games 2002

[55] R. Hubbold, J. Cook, M. Keates, S. Gibson, T. Howard, A. Murta, A. West, and S. Pettifer. GNU/MAVERIK : A micro-kernel for large-scale virtual environments. Presence: Teloperators and Virtual Environments, 10:22-34, February 2001.

[56] J. A. Carson and A. F. Clark. Multicast Shared Virtual Worlds Using VRML97. Proceedings of the fourth symposium on Virtual reality modeling language February 1999

[57] Ralph Peters, Andreas Graeff, Christian Paul. Integrating Agents into Virtual Worlds. Proceedings of the 1997 workshop on New paradigms in information visualization and manipulation, 1997

[58] T. C. Lu, C. N. Lee and W. Y. Hsia. Supporting large-scale distributed simulation using HLA. ACM Transactions on Modelling and Computer Simulation, Vol.10, No.3, pp.268-294, July 2000.

[59] Tapas K. Das , Gurminder Singh , Alex Mitchell , P. Senthil Kumar , Kevin McGee, NetEffect: a network architecture for large-scale multi-user virtual worlds, Proceedings of the ACM symposium on Virtual reality software and technology, p.157-163, September 1997

[60] D. Bauer, S. Rooney and P. Scotton. Network infrastructure for massively distributed games. Proceedings of the first workshop on Network and system support for games 2002 Pages: 36 - 43

[61] G. Reynard, S. Benford, C. Greenhalgh, and C. Heath. Awareness Driven Video Quality of Service in Collaborative Virtual Environments, Proc. CHI '98, 18-23 April, 1998, pp. 464-471, ACM Press.

[62] H. Sugano, K. Otani, H. Ueda, S. Hiraiwa, S. Endo and Y. Kohda . SpaceFusion: a multi-server architecture for shared virtual environments. Proceedings of the second symposium on Virtual reality modeling language 1997 Pages: 51 - ff

[63] Purbrick, J. and Greenhalgh, C., An Extensible Event-based Infrastructure for Networked Virtual Worlds, to appear in Proc. IEEE VR 2002, Orlando, Florida, 2002,

[64] Chim, J., Lau, R., Si, A., Leong, H., To, D., Green, M. and Lam, M. Multi-Resolution Model Transmission in Distributed Virtual Environment. ACM Symposium on Virtual Reality Software and Technology, November 1998, 25-34.

[65] S. Benford, A. Bullock, L. Fuchs and J. Mariani. Computable Models and Prototypes of Interaction", (Eds.), COMIC project deliverable D4.2, ISBN 0-901800-55-4, October 1994.

[66] D. Cohen-Or, Y. Mann and S. Fleishman, Deep compression for streaming texture intensive animations, Proceedings of the 26th annual conference on Computer graphics and interactive techniques, p.261-267, July 1999

[67] Han, S., Lim, M. and Lee, D. Scalable Interest Management Using Interest Group based Filtering for Large Networked Virtual Environments. ACM Symposium on Virtual Reality Software and Technology, Korea, October 2000, 103108

[68] Han, S., Lim, M., Lee, E. and Lee, D. A Scalable Network Support for Internet-based 3D Virtual Shopping Mall. HCI'02, Phoenix Park, Korea, February 2002.

[69] H.A. Abrams, K. Watson, and M. J. Zyda, "Three-tiered interest management for large-scale virtual environments,', Proc. ACM symposium on Virtual reality software and technology, 125-129 (1998).

[70] H.A. Abrams. "Extensible Interest Management For Scalable Persistent Distributed Virtual Environments," PhD Thesis, Naval Postgraduate School, Monterey, California. (1999).

[71] Helmuth Trefftz, Ivan Marsic: Message caching for local and global resource optimization in shared virtual environments. Proceedings of the ACM Virtual Reality Software and Technology, VRST 2000, October 22-25, pp97-102

[72] C. Park , H. Ko and T. Kim, Multi-resolution spatial model for large-scale virtual environment, Proceedings of the ACM symposium on Virtual reality software and technology, October 22-25, 2000

[73] G. Liu and G. Maguire Jr. A Class of Mobile Motion Prediction Algorithms for Wireless Mobile computing and Communications. Mobile Networks and Applications, 1(2): 113-121, 1996.

[74] S. Singal and D. Cheriton. Exploiting Position History for Efficient Remote Rendering in Networked Virtual Reality. Presence, 4(2):169-193, 1995.

[75] J. Chim, M. Green, R.W.H. Lau, H.V. Leong, and A. Si. On Caching and Prefetching of Virtual Objects in Distributed Virtual Environments. In Proc. of ACM Multimedia, pages 171-180, 1998.

[76] R. Azuma and G. Bishop. A Frequency-Domain Analysis of Head-Motion Prediction. In Proc. of ACM SIGGRAGH'95, pages 401-408, 1995.

[77] A. Katz and K. Graham. Dead Reckoning for Airplanes in Coordinated Flight. In Proc. of Workshop on Standards for Interoperability of Defense Simulations, pages 5-13, Mar. 1994.

[78] A. Chan , R.W.H. Lau , B. Ng, A hybrid motion prediction method for caching and prefetching in distributed virtual environments, Proceedings of the ACM symposium on Virtual reality software and technology, November 15-17, 2001

[79] C. Poellabauer, K. Schwan and R. West, "Coordinated CPU and Event Scheduling for Distributed Multimedia Applications" , in Proceedings of the 9th ACM Multimedia Conference (ACM SIGMM), September 2001

[80] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an internet -scale event notification service. In Proceedings of the Nineteenth ACM Symposium on Principles of Distributed Computing (PODC 2000), July 2000.

[81] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. ACM Transactions on Computer Systems, Vol. 19, No. 3, August 2001.

[82] H. Yu, D. Estrin, and R. Govindan. A hierarchical proxy architecture for Internet-scale event services. In Proceedings of WETICE '99, Stanford, CA, June 1999.

[83] Thomas A. Funkhouser. Network Topologies for Scalable Multi-User Virtual Environments. IEEE VRAIS '96, San Jose, CA, April, 1996.

[84] Thomas A. Funkhouser. Network Services for Multi-User Virtual Environments. IEEE Network Realities '95, Boston, MA, October, 1995.

[85] D. Schmalstieg, Lodestar: an octree-based level of detail generator for VRML, Proceedings of the second symposium on Virtual reality modeling language, p.125-132, February 24-26, 1997

[86] D.W.Fellner and A.Hopp. VR-Lab - A Distributed Multi-User Environment for Educational Purposes and Presentations. In Proceedings of VRML'99 , paes 121- 131. 1999

[87] M.Hosseini, S.Pettifer and N.D.Georganas, Visibility Based Interest Management in Collaborative Virtual Environments, Proc. ACM Collaborative Virtual Environments Conf. (CVE 2002), Bonn, Germany, Sept. 2002

[88] A. Wilson, M. Lin, D. Manocha, B. Yeo, and M. Yeung. Videobased rendering acceleration algorithms for interactive walkthroughs. Proc. of ACM Multimedia, pages 75-84, 2000.

[89] Thomas C. Hudson, Ming C. Lin, Jonathan Cohen, Stefan Gottschalk, and Dinesh Manocha. V-COLLIDE: Accelerated collision detection for VRML. In Rikk Carey and Paul Strauss, editors, VRML 97: Second Symposium on the Virtual Reality Modeling Language, New York City, NY, February 1997. ACM SIGGRAPH / ACM SIGCOMM, ACM Press.

[90] Olof Hagsand , Rodger Lea , Marten Stenius "Using Spatial Techniques to decrease Message Passing in a Distributed VE Systems" a publication of ACM SIGGRAPH in VRML 97 -Second symposium on the Virtual Reality Modeling Language. pp. 7-15

[91] Martin K., Creation and performance analysis of user representations in collaborative virtual environments. ACM Computing Surveys (CSUR), v.31 n.2es, June 1999

[92] Broll W., Meier, E., Schardt, T. "The Virtual Round Table - A Collaborative Augmented Multi-User Environment" CVE2000, The Third International Conference on Collaborative Virtual Environments, San Francisco, USA, Sept. 10-12 2000

[93] Swindells C., Dill J C., and Booth, K. S., System lag tests for augmented and virtual environments, Proceedings of the 13th annual ACM symposium on User interface software and technology, p.161-170, November 06-08, 2000

[94] L. Pantel, L. Wolf: "On the Suitability of Dead Reckoning Schemes for Games", First Workshop on Network and System Support for Games (NetGames2002), April 16-17, 2002, Braunschweig, Germany.

[95] C. Winkelholz and T. Alexander (2002) Approach for Software Development of Parallel Real-Time VE Systems on Heterogenous Clusters Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization, Pages: 23 32

[96] Thomas A. Funkhouser, Patrick Min, and Ingrid Carlbom, Real-Time Acoustic Modeling for Distributed Virtual Environments, ACM Computer Graphics Annual Conference Series, Proceedings of SIGGRAPH 99,August 1999, pp. 365-374

[97] Thomas A. Funkhouser , Ingrid Carlbom, Gary Elko, Gopal Pingali, Mohan Sondhi, and Jim West, A Beam Tracing Approach to Acoustic Modeling for Interactive Virtual Environments, ACM Computer Graphics Annual Conference Series, Proceedings of SIGGRAPH 98, July 1998, pp. 21-32 (Adobe Acrobat .pdf file, 570 Kb).

[98] Patrick Min and Thomas Funkhouser, Priority-Driven Acoustic Modeling for Virtual Environments, Proceedings of EUROGRAPHICS 2000 , Interlaken, Switzerland, August 2000 (Adobe Acrobat .pdf file, 350 Kb).

[99] Roberts, D. and Sharkey, P. Maximising Concurrency and Scalability in a Consistent, Causal, Distributed Virtual Reality System, Whilst Minimising the Effect of Network Delays. IEEE Workshops on Enabling Technology: Infrastructure for Collaborative Enterprise, 1997, 161–166.

[100] Yang, J. and Lee, D. Scalable Prediction Based Concurrency Control for Distributed Virtual Environments. IEEE Virtual Reality 2000, New Brunswick NJ USA, March 2000, 151–158.

[101] Schwartz, P., Bricker, L., Campbell, B., Furness, T., Inkpen, K., Matheson, L., Nakamura, N., Shen, L., Tanney, S. and Yeh, S. Virtual Playground: Architectures for a Shared Virtual World. ACM Symposium on Virtual Reality Software and Technology, Taipei Taiwan, November 1998, 43–50.

[102] Macedonia. M. and Zyda, M. A Taxonomy for Networked Virtual Environments. IEEE Multimedia, January-March 1997, 4(1):48–56.

[103] Macedonia, M., Zyda, M., Pratt, D., Brutzman, D. and Barham, P. Exploiting Reality with Multicast Groups. IEEE Computer Graphics and Applications, September 1995, 38–45.

[104] Lim, M. and Lee, D. Improving Scalability Using Sub-Regions in Distributed Virtual Environments. International Conference on Artificial Reality and Telexistence, Tokyo Japan, December 1999, 179–184.

[105] Lee, E., Lee. D., Han, S. and Hyun, S.J. Prediction-based Concurrency Control for A Large Scale Networked Virtual Environment Supporting Various Navigation Speeds. ACM Symposium on Virtual Reality Software and Technology, Canada, November 2001, 227–232.

[106] Lee, D., Yang, J. and Hyun, S.J. Scalable Predictive Concurrency Control for Large Distributed Virtual Environments with Densely Populated Objects. ACM Symposium on Virtual Reality Software and Technology, Korea, October 2000, 109–114.

[107] Greenhalgh, C. and Benford, S. Boundaries, Awareness and Interaction in Collaborative Virtual Environments. IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Cambridge Massachusetts USA, June 1997, 193–198.

[108] Benford, S. and Greenhalgh, C. Introducing Third Party Objects into the Spatial Model of Interaction. European Conference on Computer Supported Cooperative Work, Lancaster, September 1997.

[109] Carey, R., Bell, G. and Marrin, C. ISO/IEC 14772-2:2001, The Virtual Reality Modeling Language (VRML) - Part 2: External authoring interface (EAI)

[110] Mike Fraser , Tony Glover , Ivan Vaghi , Steve Benford , Chris Greenhalgh , Jon Hindmarsh , Christian Heath, Revealing the realities of collaborative virtual reality, Proceedings of the third international conference on Collaborative virtual environments, p.29-37, September 2000

[111] K. Rothermel, G. Dermler, W. Fiederer. (June 03 - 06, 1997) QoS Negotiation and Resource Reservation for Distributed Multimedia Applications , 1997 International Conference on Multimedia Computing and Systems (ICMCS '97)

[112] Gabriel Dermler, Walter Fiederer, Ingo Barth, Kurt Rothermel . (June 17 - 23, 1996)A Negotiation and Resource Reservation Protocol (NRP) for Configurable Multimedia Applications. 1996 International Conference on Multimedia Computing and Systems (ICMCS '96)

[113] W. B. Mitchell, D. Economou, S. Pettifer, and A. West. Choosing and using a driving problem for cve technology. In Proceedings of ACM Symposium in Virtual Reality Software and Technology 2000, pages 16-23, October 2000.

[114] D. Economou, B. Mitchell, S. Pettifer, and A. West. Cve technology development based on real world application and user needs. In Proceedings of IEEE WETICE 2000. IEEE Press, June 2000.

[115] S. Daubrenet, S. Pettifer, and A. West. Relationships: providing structure and behavior for shared virtual environments. In Proceedings of 7th UKVRSIG Conference, pages 117-126. University of Strathclyde, September 2000.

[116] S. Pettifer, J. Cook, and J. Mariani. Towards real-time interactive visualisation virtual environments: A case study of q-space. In Proceedings of the International Conference on Virtual Reality 2001, pages 121-129. ISTIA Innovations, May 2001.

[117] Masaya Okada, Hiroyuki Tarumi, Tetsuhiko Yoshimura, and Kazuyuki Moriya: "Collaborative Environmental Education Using Distributed Virtual Environment Accessible from Real and Virtual Worlds", Applied Computing Review, Vol.9, No.1, pp. 15-21, ACM, Spring 2001.

[118] Masaya Okada, Hiroyuki Tarumi, Tetsuhiko Yoshimura, Kazuyuki Moriya, and Tetsuro Sakai: "DigitalEE: A Support System for Collaborative Environmental Education Using Distributed Virtual Space", Systems and Computers in Japan, Vol. 33, No. 8, pp. 51-63, John Wiley and Sons, July 2002

[119] J.H.P. Chim, M. Green, R.W.H. Lau, H.V. Leong, and A. Si. On caching and prefetching of virtual objects in distributed virtual environments. In ACM Multimedia, 1998.

[120] J.Howell, Y.Chrysanthou, A.Steed, M.Slater, A Market Model for Level of Detail Control. Proceedings of ACM VRST 99

[121] Greenhalgh, C. M. and Benford, S. D., Introducing Regions into Collaborative Virtual Environments, Internal report available from the authors of this paper (submitted to IEEE VRAIS'97).

[122] Macedonia, M., Zyda, M., Pratt, D., Brutzman, D. and Barham, P., "Exploiting Reality with Multicast Groups: A Network Architecture for Large-Scale Virtual Environments", Proc. IEEE Virtual Reality Annual International Symposium (VRAIS'95), North Carolina, March, 1995

[123] Leigh, J., Johnson, A. E., DeFanti, T.A., Issues in the Design of a Flexible Distributed Architecture for Supporting Persistence and Interoperability in Collaborative Virtual Environments, proceedings of Supercomputing '97 Nov 15-21, 1997

[124] Richard M. Fujimoto, Exploiting temporal uncertainty in parallel and distributed simulations, Proceedings of the thirteenth workshop on Parallel and distributed simulation, p.46-53, May 01-04, 1999

[125] Michal Masa , Jiri Zara. Generalized interest management in virtual environments. Proceedings of the 4th international conference on Collaborative virtual environments September 2002

[126] Seunghyun Han , Mingyu Lim , Dongman Lee, Scalable interest management using interest group based filtering for large networked virtual environments, Proceedings of the ACM symposium on Virtual reality software and technology, October 22-25, 2000

[127] Purbrick, J. and Greenhalgh C., Extending Locales: Awareness Management in MASSIVE-3 in Proceedings of the IEEE Virtual Reality 2000 Conference (VR 2000), New Brunswick, NJ, USA, March 2000, IEEE Press

[128] Greenhalgh, C., Purbrick, J., Benford, S., Craven, M., Drozd, A. and Taylor, I., Temporal links: recording and replaying virtual environments, in Proceedings of the 8th ACM international conference on Multimedia (MM 2000), ACM Press

[129] H.Nakanishi, C.Yoshida, T.Nishimura and T.Ishida, "FreeWalk: Supporting Casual Meetings in a Network, " Computer Supported Cooperative Work, p 308-314, 1996.

[130] Mark, J Pullen. "The Internet-Based Lecture: Converging Teaching and Technology", ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE), July 2000

[131] Danny S. P. To , Rynson W. H. Lau , Mark Green, A method for progressive and selective transmission of multi-resolution models, Proceedings of the ACM symposium on Virtual reality software and technology, p.88-95, December 20-22, 1999

[132] Hindmarsh, J., Fraser, M., Heath, C., Benford, S. and C. Greenhalgh (1998) Fragmented Interaction: Establishing mutual orientation in virtual environments. Proceedings of CSCW'98. ACM Press. p217-226

[133] Gianpaolo U. Carraro , John T. Edmark , J. Robert Ensor, Techniques for handling video in virtual environments, Proceedings of the 25th annual conference on Computer graphics and interactive techniques, p.353-360, July 1998

[134] L. Pantel and L. C. Wolf. On the impact of delay on real-time multiplayer games. In Proceedings of the 12th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), Miami Beach, FL, May 2002.

[135] Mike Wray, Vincent Belrose. Avatars in LivingSpace. Proceedings of the Fourth Symposium on Virtual Reality Modeling Language, 1999 pp13-19

[136] Goddard T., Sunderam V. S., 1999, "ToolSpace: Web Based 3D Collaboration", Proceedings of the International Conference on Intelligent User Interfaces, January 5-8, 1999, pp. 161-165

[137] Barrus, J., Waters, R. and Anderson, D. Locales: Supporting Large Multiuser Virtual Environments. IEEE Computer Graphics and Applications, November 1996, 16(6):50–57.

[138] Ramaraj, S., Teodorovic. D., Gracanin, D.: Real-Time Optimization of Dynamic Stochastic Knapsack Problem Using a Multi-Layer Perceptron Network, MLMTA 2003, Las Vegas, NV

[139] K.W. Ross and D.D. Yao, "Monotonicity properties for the stochastic knapsack," *IEEE Transactions on Information Theory*, vol.36, no.5, pp. 1173–1179, Sept. 1990.

[140] G.Y. Lin, Y.Lu, and D.Yao, "The stochastic knapsack revisited: Structure, switch-over policies, and dynamic pricing," 2002, available at http://www.ieor.columbia.edu/yao/knaps7.pdf

[141] R.Simon and T.Znati, "A probabilistic analysis of admission control policies for deadline-driven service disciplines," in *Proc. of the 31st Annual IEEE Simulation Symposium*, 5–9 Apr. 1998.

[142] A.J. Kleywegt and J.D. Papastavrou, "The dynamic and stochastic knapsack problem," *Operations Research*, vol.46, pp. 17–35, 1998.

[143] A.J. Kleywegt and J.D. Papastavrou, "The dynamic and stochastic knapsack problem with random sized items," *Operations Research*, vol.49, pp. 26–41, 2001.

[144] D.Teodorovic, P.Lucic, J.Popovic, S.Kikuchi, and B.Stanic, "Intelligent isolated intersection," in *Proc. of the 10th International IEEE Conference on Fuzzy Systems*, Melbourne, Australia, Dec. 2001.

[145] D.Teodorovic, V.Varadarajan, M.R. Chinnaswamy, and S.Ramaraj, "Evolution of real-time traffic adaptive signal control algorithms — a review," in *Proc. of the ICORD 2002*, Chennai, India, Dec. 2002.

[146] T.Murata, "Petri nets: Properties, analysis and applications," *Proc. of the IEEE*, vol.77, no.4, pp. 541–580, Apr. 1989.

VITA

Sharath Ramaraj, the third and last son of Mr. Ramaraj Krishnamurthy and Mrs. Padmavathy Ramaraj, was born on August 6th, 1979, in Chennai, India. He graduated from the University of Madras, India in May 2001 with a Bachelor of Engineering degree in Computer Science. He came to the United States of America to study at Virginia Tech in August 2001. This thesis completes his Masters degree in Computer Science.