A Scalable Architecture for Supporting Interactive Games on the Internet

Wentong CAI   Percival XAVIER   Stephen J. TURNER   Bu-Sung LEE
Parallel & Distributed Processing Laboratory
School of Computer Engineering
Nanyang Technological University
Singapore 639798

Abstract

This paper presents a scalable architecture for supporting large-scale interactive Internet games. In order to support a large number of participants and to divide the workload, the virtual world is divided into partitions. Each partition is then assigned to a server. A client (i.e., a player or a participant) will join a server according to the position of the avatar it controls. Compared to a centralized architecture, this distributed client-server architecture is more scalable. In addition, compared to a fully distributed, peer-to-peer architecture, it also provides a means for detecting cheating in distributed games. Since interactions and accounting information must be forwarded directly to one of the servers for qualification and verification, cheating amongst distributed players of the game will be minimized. 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 communication among servers is enabled by the Run-Time Infrastructure (RTI) services. The High Level Architecture (HLA) 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. In this paper, the design detail of the architecture will be presented. An experimental interactive Internet game realized using the architecture will be also described in the paper.

Keywords: Large-scale Interactive Games, Scalability, Distributed Client-Server Architecture, High Level Architecture (HLA), Run-Time Infrastructure (RTI), Ownership and Data Distribution Management.

1 Introduction

Interactive multi-user Internet games have their origins from programs sharing a common heritage known variously as MUGs (Multi-User Games), MUDs (Multi-User Dungeons or Multi-User Dimensions) and MUAs (Multi-User Adventures). They have gained significant popularity due to their entertainment value. 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 [9]:

- 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. A 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.

Proceedings of the 16th Workshop on Parallel and Distributed Simulation (PADS'02)
1087-4097/02 $17.00 © 2002 **IEEE**

Most of the currently available, interactive multi-user Internet games are based on a *centralized architecture* (i.e., *client-server* architecture), where all the clients (i.e., players) are connected to a centralized server. The communication between the clients will go through the server and the sever maintains a consistent game view of all the clients. So, the major problem of this architecture is that the server will become the bottleneck in terms of both communication and computation, thus limiting the scalability. A logical solution to this problem is to have a *fully-distributed architecture*, where each client computes its own view of the game state and communicates with other clients without the intervention of a server. An example application using the fully-distributed architecture is the MiMaze which is a distributed multiplayer interactive game developed using IP multicast [3]. In addition, the High Level Architecture (HLA) [2, 4] also provides a framework for constructing such fully-distributed, interactive, multi-user Internet games.

Another important area of concern in interactive multi-user Internet games is security. One major aspect of security is the prevention of cheating among clients participating in the game. In a centralized architecture, information reaches its destination through the server. Therefore, a consistent state of scoring and accounting can be easily maintained and game companies, for example, can charge players based on the duration of their participation. However, in a fully-distributed architecture, since interactions and accounting information are not qualified and verified by a server, cheating amongst distributed players of the game is possible. Detection of cheating in a fully-distributed architecture is also more difficult than that in a centralized architecture.

Hence, the objective of this paper is to develop a *distributed client-server architecture* for interactive Internet games, which combines the advantages of both centralized and fully-distributed architectures. The virtual world is divided into partitions. Each partition is then assigned to a server. A client will join a server according to the position of the avatar[1] it controls. An overview of the architecture will be presented in Section 2.

To preserve interactivity of the game, a fast response from the server is required. The strategy applied is to incorporate a hybrid communication mechanism depending on the nature of the information that is passed between clients and the server. In Section 3, a detailed account of the design strategy of the front-end client-server communication will be discussed.

The communication among servers is enabled by the services provided by the Run-Time Infrastructure (RTI) of the HLA. Our work is in-line with the areas involving distributed virtual environments [10], networked virtual environments [9] and distributed simulations based on the HLA [6]. A common technique used by these applications to limit the amount of data packets being transmitted in the network is data filtering (or interest management). HLA Data Distribution Management (DDM) services are employed to limit the amount of communication between the servers. The issues concerning the design of the back-end servers will be studied in Section 4.

In Section 5, an experimental interactive Internet game realized using the architecture will be also described. The conclusions and the description of future work will be given in Section 6.

2 An Overview of the Distributed Client-Server Architecture

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 [1]:

- *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. An example application using participant subdivision can be

[1] We use the word "client" or "player" to refer to a physical participant in the interactive Internet game and the word "avatar" to refer to the graphical embodiment representing the participant in the virtual environment.

found in [7]. In this paper, we adopt the virtual world subdivision approach.

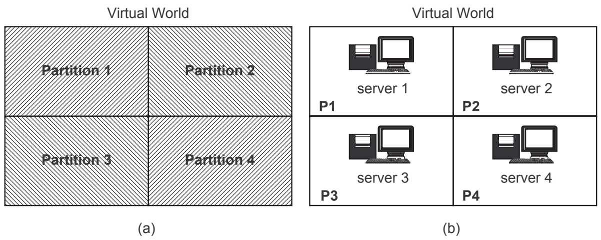

Figure 1. Spatial Partition of Virtual World

An optimum choice for distributing load amongst the servers may depend on the application, network topology and other design decisions. In [8], it suggests three possibilities for partitioning a virtual 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.

In this paper, the spatial partitioning approach is adopted. As shown in Figure 1(a), the virtual world is spatially divided into partitions. A server is then assigned to each partition, managing a group of players (clients) who have avatars in the partition (Figure 1(b)). Thus, each server is responsible for managing only a portion of participating players in the entire virtual environment.

Figure 2 shows the overall communication infrastructure of the distributed client-server architecture. In this infrastructure, the servers comply with the HLA rules and communicate with each other through the RTI. The clients connected to the same server are represented by the server as a federate in the federation. Communication and interaction between clients and the server is via socket connections.

Figure 3 shows a more detailed design of the distributed client-server architecture. It consists of two main parts: the *front-end client-server* and the *distributed back-end servers*. Each server consists of three modules: *server back-end*, *message queue* and *server front-end*. The main role of the server back-end module is to process interactions, accounting information and update messages of the participating players and to keep state information of all the avatars in the partition. Server back-end modules are in fact the federates participating in the federation. They together with the RTI form the distributed back-end servers. The main aim of using an HLA-based implementation is to enhance interoperability and scalability. When an avatar moves from one partition to another, the HLA Ownership Management (OM) services are used to migrate the corresponding avatar between the two servers involved. In order to reduce the amount of data transmitted among the servers, the HLA DDM services are employed so that a server only subscribes to the state updates of remote avatars near the edge of its partition. The functionality of the distributed back-end servers will be further explained in Section 4.

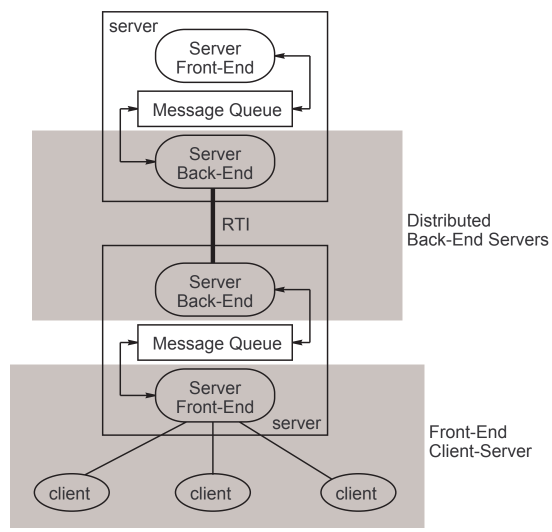

Figure 3. Design of Distributed Client-Server Architecture

The front-end client-server structure consists of the server front-end module and the associated clients. The main task of the server front-end module is to handle the arrival of joining clients and to provide a mechanism for the server to communicate with the clients. The server back-end and front-end modules communicate with each other through a message queue. The development of the front-end client-server structure will

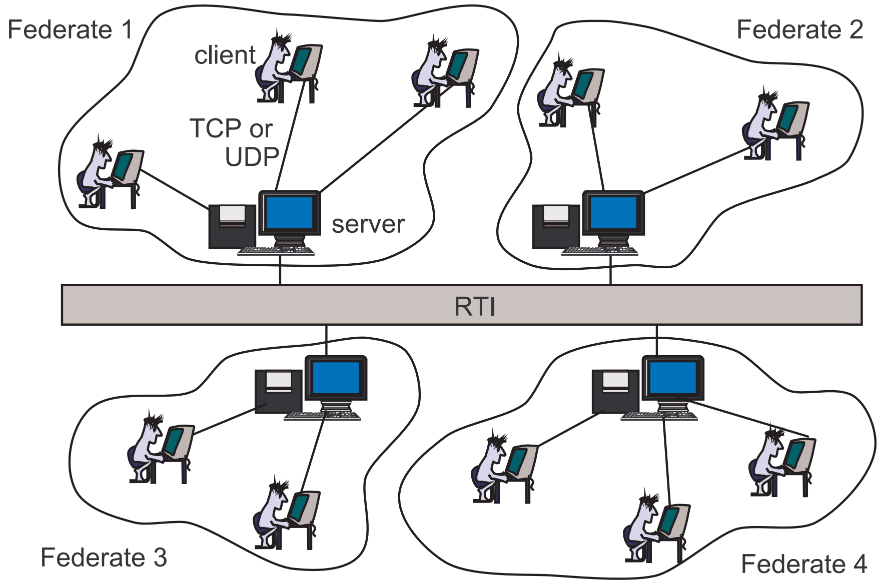

Figure 2. Overview of Distributed Client-Server Architecture

be further explained in the next section.

3 Front-end Client-Server Structure

A communication architecture is required for information exchange between clients and the server. There are three types of message-passing in a front-end client-server structure: *position updates*, *interactions* and *accounting information*. Clients need to issue position updates which will be sent to the server and will also be propagated to other clients in the structure. The server will use these updates to maintain the state of the virtual environment so as to verify clients' interactions. Clients must also forward interactions (e.g., attacking actions), when attempting to interact with the virtual environment, to the server. The server will qualify the interactions and sends results back to the client. The accounting information is sent between the server and clients for session management and scoring. Table 3 gives a summary of the relative frequency of these three types of client-server message-passing.

Type	*Relative Frequency*
Position Updates	High
Interactions	Medium
Accounting	Low

Table 1. Classification of Client-Server Communication

To ensure the security of data, a connection-oriented approach is required for handling interactions and accounting information. Thus, a TCP connection is established between each client and the server. IP multicast relies on a connectionless, datagram-based approach to route packets between network entities. Since no acknowledgment packet from the receiving entity is required, information passing between network entities can take place at a faster rate than TCP. With this advantage, IP multicast is deployed to handle position updates of clients because these updates are required to be transmitted at a higher frequency as compared to interactions and accounting information. A major problem with IP multicast is the possibility of lost packets. This issue is however ignored because the occasional loss of position updates does not severely affect the simulation of remote avatars.

Figure 4 shows a logical view of the communication scheme used in the front-end client-server structure. The server front-end module is multithreaded. There is a thread for each TCP connection and a separate thread for handling position updates. These threads act as proxies between the distributed back-end servers and the clients, relaying messages in both directions. The message queue is used to buffer the messages be-

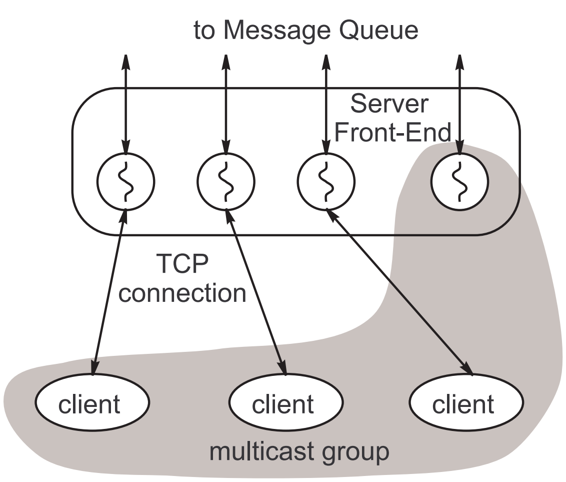

Figure 4. Communication Architecture for Front-end Client-Server

between the front-end client and the distributed back-end servers. The shaded area in Figure 4 represents a multicast group which includes all the clients connected to the same server and the thread responsible for position updates. As explained above, this multicast group is used for transmitting position updates.

In summary, to support secured communication for interactions and accounting information as well as to speedup periodic update messages (e.g., position updates), a hybrid communication architecture using both TCP and IP multicast is used in the front-end client-server structure.

4 Distributed Back-end Servers

Distributed back-end servers consist of server back-end modules, each of which is built as a federate and is responsible for maintaining state updates of all avatars it owns and handling interactions and accounting information. As discussed in Section 2, the spatial partitioning scheme is adopted in the design of the distributed client-server architecture. The selection is based on the assumption that at any time, the participating avatars are likely to be uniformly distributed in the virtual world since they are able to move around in the virtual world autonomously. One problem in the spatial partitioning scheme is the migration of an avatar from one server[2] to another when it changes its partition.

[2] Server in this section actually refers to the server back-end module of the server.

This is illustrated in Figure 5 where the virtual world is spatially divided into four partitions. When an avatar changes from partition P_1 to partition P_3, the ownership of the avatar should be transferred from server 1 to server 3. In addition, the client which controls the avatar should also be reconnected to server 3. The HLA's Ownership Management (OM) services are used to solve this problem.

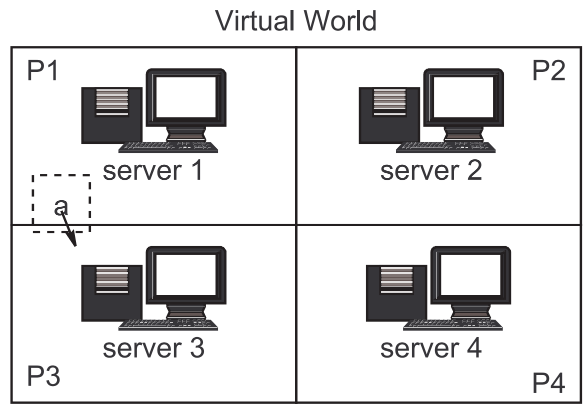

Figure 5. Migration of Client and Avatar

In order to ensure a seamless virtual environment, an avatar located at the edge of the partition must be able to "see" the close-by avatars belonging to other partitions so that they can interact with each other. So, another problem is how to maintain a seamless virtual environment without resulting in too much inter-server traffic. To solve this problem, the HLA's Data Distribution Management (DDM) services are employed.

4.1 Transferring Ownership of Avatars

The HLA's OM services are deployed to transfer ownership of avatars when they cross the boundaries of the partition to which they currently belong. In order to enforce a scheme for ownership management of avatars, each server must be able to perform a real-time query of each avatar's position to determine whether an avatar has moved out of the server's partition. The interaction diagram in Figure 6 shows how the transfer of avatar ownership is achieved. Dotted lines represent communications in the distributed back-end severs and solid lines represent communications in the front-end client-server structure.

The ownership of each avatar is divested based on the ownership push scheme. Ownership push suggests that a federate that owns update responsibility

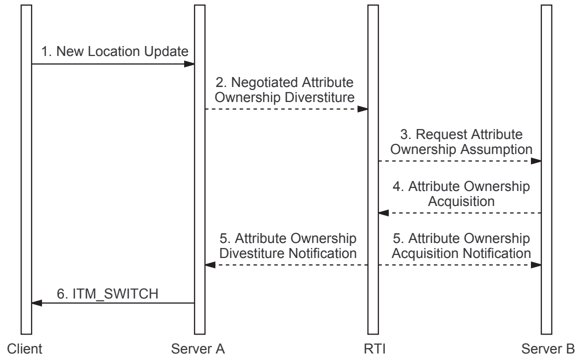

Figure 6. Protocol for Transferring Ownership of Avatar

of and/or the privilege to delete instance attributes wishes to transfer ownership of the attributes to another federate. The ownership may be surrendered *unconditionally* or *by negotiation*. Unconditional push releases a federate from attribute update and/or deletion responsibility without any commitment from other federates to assume these responsibilities. Negotiated push is a formal exchange where a federate retains responsibility until a new owner is identified and a formal exchange process is completed. The negotiated push scheme is adopted in our design.

As shown in Figure 6, a server wishing to let go the responsibilities calls the RTIambassador method *negotiatedAttributeOwnershipDivestiture*(). As only one server may acquire the ownership of an avatar, the divesting server will specify which server shall be the new owner of its divested avatar. The specification is made through the tag data when *negotiatedAttributeOwnershipDivestiture*() is called. Given that the other servers are capable of updating any or all of the attributes being given away, they are therefore, notified via their FederateAmbassador method *requestAttributeOwnershipAssumption*(). A server wishing to acquire one or more of the offered attributes indicates its interest using the method *attributeOwnershipAcquisition*(). If any server is found to assume the responsibilities being given away, the server that initiated the push receives *attributeOwnershipDivestitureNotification*(). The server gaining the responsibilities is informed with *attributeOwnershipAcquisitionNotification*(). Once the transaction for ownership management is done, the server which originally owned the avatar instance will send a message ITM_SWITCH to the client to inform it of the need to join another multicast group. It is one type of the interaction messages sent between the server and clients [11]. The IP address of the new server and the multicast group address are provided in the ITM_SWITCH message.

4.2 Provision of State Updates of Edge Avatars

The HLA's DDM services are used to provide state updates of the avatars near the edge of a partition. Servers exchange avatar state information by updating and reflecting object attribute values. And they obtain only relevant state updates from neighboring partitions by defining update and subscription regions.

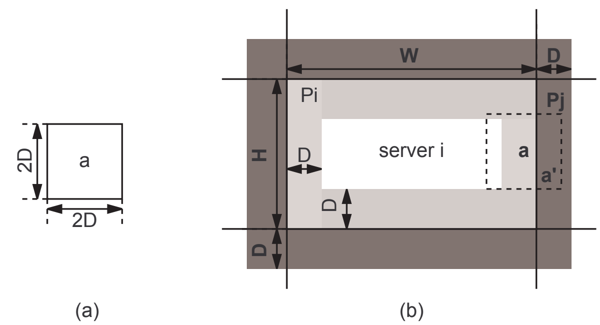

Figure 7. Update and Subscription Regions

The extent to which an avatar can see another is defined by the avatar's view radius D. Factors affecting this value depend on the virtual world's environmental conditions (e.g., fog and obstructions) as well as the speed of the avatar. Figure 7(a) shows the dimensions of the part of the virtual world displayed for each player. Figure 7(b) shows a server's update and subscription regions. The update region of a server is the entire partition assigned to the server. In this case, the update region of server i is the partition P_i itself. The subscription region of a server is the region surrounding its partition. In Figure 7(b), the subscription region of server i is represented by the darkly-shaded area. DDM services *associateRegionForUpdates*() and *subscribeObjectClassAttributesWithRegion*() are used. A server only updates the state of an avatar, by *updateAttributeValues*(), if it is in the edge of the partition (for server i in Figure 7(b), it is represented by the lightly-shaded area.

For example, as shown in Figure 7(b), when avatar a' moves into the darkly-shaded area in partition P_j, server j will update its position. The update region of a' is the partition P_j itself and the subscription region of the server i is the darkly-shaded area. So, there is an overlap between update and subscription region. The position update of a' will be sent to server i. Therefore, avatar a in partition P_i will be able to see avatar a' in partition P_j in its displayed virtual world.

Obviously, the advantage of using the DDM services is the reduction on inter-server communication. Now, each server only receives the updates of relevant avatars from other servers. The other advantage of the above definition of update and subscription regions is that they are constant and need not be dynamically recreated. Thus, the overhead on calculating the overlap between regions is minimized.

5 An Experimental Internet Game

An experimental interactive Internet game has been constructed using the distributed client-server architecture. The game scenario is reminiscent of "Pacman" in which each player has a 3D representation of his/her view of the game (with respect to Figure 7(a)). The game involves a group of avatars that navigate through a virtual 3D maze with the objective of finding as many items as possible. Items are generated by the virtual environment dynamically at random positions. Avatars may group into teams (according to their colors). Avatars may also attack one another to "steal" items. However, an avatar that attacks its own team member will result in a penalty. Figure 8 shows the view of a player. In this case, there are two remote avatars and one static item in the view. The self-representation is ignored in the view since each player is assumed to view the virtual world from his/her own angle.

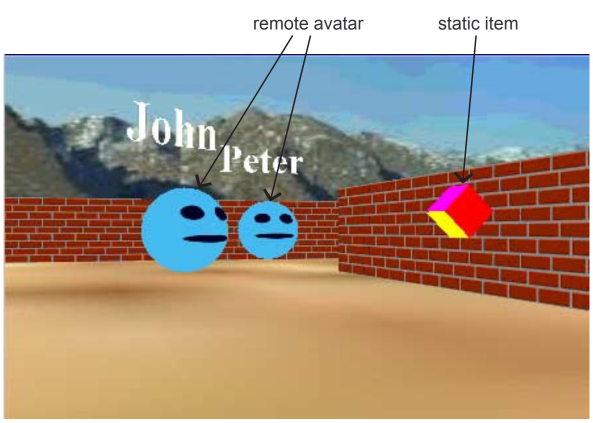

Figure 8. View of a Player

Preliminary performance analysis has been conducted using M/M/1 queue to compare centralized and distributed client-server architectures. Currently, experiments are being conducted to collect results from the experimental Internet game for further performance study.

6 Conclusions and Future Work

The development of the distributed client-server architecture is divided into two stages. The first stage is concerned with the development of a front-end client-server structure that enables client-server interaction. Both IP multicast and TCP based communication are deployed to relay different classes of information. The advantage of multicast lies with its ability to route packets at a fast rate. As such, it is employed to transmit position updates between participating clients. TCP, on the other hand, is employed to establish a secure communications channel between the client and its server to relay accounting information and interactions.

The second stage of development is involved with the construction of the distributed back-end servers. The main purpose of employing a distributed architecture is to divide the workload of managing avatars among multiple of servers. The scheme employed for scalability is based on spatial partitioning. In this scheme, there is a need to transfer the management of avatars when they migrate from the scope of interest of their current server. To maintain a seamless virtual world, avatars located at the edge of a partition need to "see" avatars belonging to other partitions. The RTI is used for the communication in the distributed back-end servers, and the HLA services are employed to address the above two issues. The HLA Ownership Management services are used to transfer the ownership of client instances. To enable the edges of partitions to be visible to neighboring partitions, the DDM management services were employed to create update and subscription regions.

Future work on this game architecture will involve an improvement on the distribution of clients. In the current scheme, the distribution of workload for managing client interactions is static and is based on the assumption that clients will be uniformly spread out in the virtual world. However, if clients are not uniformly distributed, the workload will become unbalanced. Another problem with the current design is that a client needs to migrate to a different server when it changes its partition. In case an area close to a boundary crossing becomes a hot spot, the overhead of clients constantly switching between servers might be heavy. To

Proceedings of the 16th Workshop on Parallel and Distributed Simulation (PADS'02)
1087-4097/02 $17.00 © 2002 **IEEE**

overcome this problem, a hysteresis approach can be adopted in the definition of update and subscription regions. To further improve the load balance and to minimize client migration overhead, a hybrid approach using both virtual world and participant subdivision will also be investigated.

References

[1] T. K. Capin, et. al. *Avatars in networked virtual environments* Wiley & Sons Inc, West Susses, 1999.

[2] Judith Dahmann, Frederick Kuhl and Richard Weatherly. Standards for simulation: As simple as possible but not simpler - The High Level Architecture for simulation. *Simulation*, 71:6, pp.378-387, 1998.

[3] C. Diot and L. Gautier. A distributed architecture for multiplayer interactive applications on the internet. *IEEE Network*, pp.6-15, July/August 1999.

[4] DMSO. Department of Defense High Level Architecture specification. Version 1.3 DMSO, 1999 (available at http://hla.dmos.mil).

[5] C. Heistad and S. Pietrowicz. You-build-it virtual reality. NCSA Java3D Grp., Univ. of Illinois Urbana-Campaign, 1999 (available at http://www.ncsa.uiuc.edu/SDG/Software/Java3D/).

[6] F. Kuhl, R. Weatherly and J. Dahmann. *Creating computer simulation systems - An introduction to the High Level Architecture* Prentice Hall PTR, NJ, 1999.

[7] 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.

[8] Macedonia et. al. Npsnet: A multi-player 3D virtual environment over the Internet. in Proc. Symposium on Interactive 3D Graphics, ACM, pp. 93-94, 1995.

[9] Sandeep Singhal and Michael Zyda. *Networked virtual environments: Design and implementation.* Addison-Wesley, 1999.

[10] Martin R. Stytz. Distributed virtual environments. *IEEE Computer Graphics and Applications*, pp. 19-31, May 1996.

[11] reference removed for double-blind review process