A Distributed Platform for Simulating Peer-to-Peer Distributed Virtual Environments

Silvia Rueda, Pedro Morillo, Juan Manuel Orduña[1]

Resumen— **The current expansion of multi-player online games has promoted the growth of large scale distributed virtual environments (DVEs). In these systems, peer-to-peer architectures have been proved as an efficient scheme for supporting massively multi-player applications. In order to research on this type of architecture, stand-alone simulators do not take into account inconsistencies due to network latency, and it is necessary to develop a distributed tool that allows to simulate large-scale DVEs in an efficient way. In this paper, we propose a distributed platform for simulating the behavior of Peer-To-Peer DVEs. This simulator is implemented following a modular architecture. It is capable of providing the main performance metrics in distributed systems, and it contains all the elements involved in real DVE simulations like the awareness method and the graphic interface. As a result, this tool can be used in real simulations of Peer-to-Peer DVEs, becoming an invaluable tool for capturing the behavior of this kind of systems.**

Palabras clave— **Peer-to-Peer architectures, Distributed Virtual Environments.**

I. Introducción

NO wadays, the extensive use of multi-player online games has promoted the use of large scale distributed virtual environments (DVEs). Users in these systems share a 3D virtual world and can interact among them and with the elements of the virtual scene. Usually, each system user is represented inside the virtual world by an entity called *avatar*. Users control their avatars through a client computer, which should render the images of the virtual 3D environment that the user would see if he was located at that point of the virtual world. Currently, large scale DVEs can simultaneously support thousands and even hundreds of thousands clients. Clients can connect to these systems through different networks, and usually through Internet. Although DVE systems are present in many different applications [1], such as civil and military distributed training, collaborative design and e-learning, the most extensive example of DVE systems are commercial, massively multi-player online games (MMOG) [2], [3], [4].

Peer-to-peer architectures were proposed some years ago for DVE systems [5], [6]. In classic peer-to-peer architectures, each client computer is also a system server, and the control of the simulation is distributed among all the client computers. In hybrid peer-to-peer architectures, only some of the client computers act as system servers.

Peer-to-peer architectures seem to be an efficient scheme for supporting Distributed Virtual Environments [7]. However, in order to allow an in-depth research on P2P DVE, it is necessary to develop an efficient simulation tool capable of managing simulations and evaluating the performance of medium and large-scale DVEs.

Currently, in other fields of computer science like image compression or algorithmic analysis there are standardized methodologies for performance evaluation [8]. However, the field of Distributed Virtual Environments still lacks a standardized evaluation methodology. As a result, the literature on this subject shows a great heterogeneity in the way that these systems are evaluated [9], [10].

In this paper, we propose a distributed peer-to-peer simulator of P2P DVEs. The distributed nature of the simulator allows taking into account distributed features that stand-alone simulators cannot evaluate, like inconsistencies due to network latency, computer delays, clock drifts, etc. This simulator is based on the methodology proposed in [10], and it takes into account crucial parameters like the number of avatars in the system, their distribution in the virtual world and their movement pattern, among other aspects. The simulator includes mechanisms for evaluation and data acquisition in real time, and it accurately reproduces the behavior of a Distributed Virtual Reality system. This simulator has been used for researching on the saturation of P2P DVE systems [11], providing the necessary evaluation data for the research on these systems.

The rest of this paper is organized as follows: Section II describes the the main modules and functionalities of the proposed simulator. Section III shows some examples of use of the peer-to-peer simulator, as well as some performance evaluation results of different DVE configurations. Section IV shows some conclusions and future work to be done.

II. A Distributed Peer-to-peer Simulator

The literature on P2P DVEs does not describe any distributed simulator for this kind of systems. Therefore, we have developed a simulator that is capable of performing the evaluation, parameterization and result acquisition in real time during the simulation of DVEs. This simulator accurately reproduce the behavior of a peer-to-peer DVE composed of N avatars (for the sake of shortness, in the rest of the paper we will refer to avatars also to denote the client computer controlling the avatars) interconnected through a network that uses point-to-point TCP/IP communications implemented upon BSD socket APIs. Additionally, there is a single central entity, denoted as the *Loader*, that manages the clients to join the system. The simulator follows the

[1]Dpto. de Informática, Universidad de Valencia, e-mail: {Silvia.Rueda,Pedro Morillo,Juan.Orduna}@uv.es

main standards for modeling collaborative virtual environments [12], [13].

A. Simulator Description

Since the purpose of the performance evaluation is to study how the system behavior evolves when it is working, not how clients join the system, the simulator uses some configuration files for system initialization, the joining of new avatars to the simulation and also the initial interconnection of avatars with their neighbors. These files include informations such as the initial location of avatars in the virtual world, data about the neighborhood and IP addresses of all clients in the simulations. All the computers involved in the simulation should have the same copy of these configuration files.

In a generic DVE system, each avatar should exchange messages at least with all the other avatars within its Area of Interest (AOI) [1]. These messages contain the location of the avatars in the virtual world, as well as the changes in the state of other (usually static) elements of the AOI. For the sake of simplicity, the clients in the simulator exclusively move with the same movement rate, and they do not change the state of any other elements in the AOI. This limitation makes easier the system simulation and evaluation, since all the messages contain the same kind of information and it is possible to control when each movements are made. In its turn, this control allows stopping the simulation and making any correction if necessary.

A simulation is defined in this simulator as the set of 100 iterations. Each iteration, in its turn, is defined as a single movement of all the avatars in the virtual world. The virtual scene consists of a 2D square whose sides are 200 meters long. We have chosen this size because in this way (taking into account the number of iterations and the maximum distance traveled by the avatar in a single movement) an avatar can go in a single simulation from the center of the world to any of the vertices of the world.

During the simulation, each time that a given avatar i makes a movement, it reports about its movements to all its neighbors by sending updating messages. Among other informations, these updating messages contain a timestamp indicating the instant when the message was generated. When an avatar receives an updating message, it returns an acknowledgment message to the sending avatar. When the sender avatar receives the acknowledgment it can compute the round-trip delay for the message by simply subtracting the timestamp in the message from the current time. Since both instants (the instant when the updating message is generated and the instant when the acknowledgment arrives to its destination) are time-stamped in the same computer, clock shifts between different computers are avoided. We have denoted the round-trip delays for all the messages sent by a client as the *Average System Response (ASR)* for that client. Among other output results, the simulator provides not only the ASR for each client, but also the average value of the ASR provided to all the clients in the simulation.

The simulator is composed of two different kind of applications, written in C++. One of them implements the clients, and the other one implements the central manager or *Loader* [14] to whom the rest of the clients should connect to in order to join the system. Both type of applications use different threads for managing the connections that should be established. As indicated above, the communications are implemented by means of sockets.

Each client has a main thread that manages the actions required by the user. Additionally, for each of its neighbors each client has also two different threads, one for listening and one for sending information. In the same way, the central *Loader* has two communication threads for each client in the system, and also a main thread. The Loader do not represent a system bottleneck, because once an avatar has joined the system it no longer needs to exchange information with the Loader.

Each client initially has (by means of a configuration file) the IP addresses of those other clients that are going to be its neighbors initially. That is, each client needs a configuration file with its initial location in the virtual world, the list of its initial neighbors, and also the IP addresses and the listening ports of these neighbors. Once all the simulation clients have connected to the Loader, the Loader itself broadcasts a synchronizing message for starting the simulation. From that instant, avatars can move within the virtual world.

Since one of the main issues in peer-to-peer DVEs is the awareness method [15], the simulator should be provided with an awareness mechanism. Concretely, we have implemented the VON method [16], the method proposed in [17], and also the COVER method [18] as a simulator option. In order to achieve this feature, the clients also have an additional thread for eventually performing supernode tasks if necessary (as the COVER method requires). Since the behavior of the threads in the system is independent, we have solved the concurrency concerns by creating in each peer client a queue of messages and a specific thread for processing this queue. In each peer client, all the listening threads (one for each current neighbor) drop the received messages either on the avatar queue or on the supernode queue, depending of the type of message. The main thread of the avatar is the one in charge of processing the messages in the queue of incoming messages. In the same way, the main thread of the supernode processes the messages in the queue of incoming messages for the supernode. An analog scheme is used for sending messages. The main thread of the client adds messages to the different sending queues (one for each current neighbor). The corresponding sending thread is in charge of processing the messages (sending them in order to the neighbor to whom the thread is connected). This scheme allows sending in-order messages without blocking the sending thread.

In order to guarantee correct concurrent accesses to the queues, we have used locks to implement the critical sections of these accesses. The same mechanism is also used for implementing the communications in the Loader. In order to illustrate these mechanisms, Figure 1 shows the communications among the different client applications in the proposed peer-to-peer simulator.

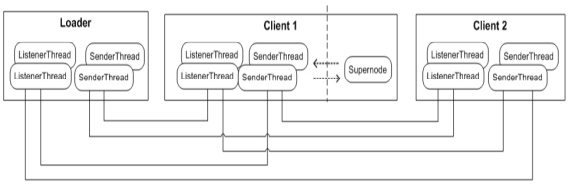

Fig. 1. Communications between different clients in the simulator

As Figure 1 shows, each client application can perform client functions as well as supernode functions (if the COVER awareness method is used). The communications between a client application i (running on a given client computer) and other client j in the simulation are performed by means of two threads, one for listening from the socket and one for sending to the socket communicating i and j. A socket is established between i and j because of one of these reasons: i needs to communicate with j because j is the Loader, because j is a neighbor within the AOI of i or because j is an uncovered avatar and i is a supernode of j (when the COVER awareness method is used). Although each client requires multiple threads, the peer-to-peer scheme allows this workload, as shown in section III.

A simulation in the proposed simulator consists of a given number of iterations. Each iteration consists of all avatars making a new movement. When a given avatar arrives to the 101th iteration (it has already performed 100 movements) it leaves the simulation, notifying about it to both the Loader and its current neighbors. Additionally, the simulator generates a simulation log with all the relevant information produced during the simulation, for a detailed off-line analysis.

Although in real systems there are no communications between the Loader and each client after the client has joined the system, the simulator implements a monitoring algorithm that allows to measure the awareness rate (the percentage of clients that have correctly computed which other clients are their current neighbors [18], [15]) in real time. In order to achieve this feature, each client has two different phases in its cycle time. In the first phase, the client moves the avatar and notifies its new position in the virtual world to all its neighbors. In the second phase, the client sends the Loader information containing its new position and which other clients it considers as its current neighbors. In this way, the Loader can compute the percentage of clients that have correctly computed which other avatars are its neighbors (the awareness rate). Additionally, this monitoring algorithm also measures the *awareness delay* of avatars. These performance measurement consists of the time interval between the instant when a new neighbor enters the AOI of a given avatar i and the instant when the own avatar i considers that neighbor as its neighbor.

The simulator allows a high number of configuration options, in order to study a wide range of situation that can arise in P2P DVEs. Concretely, the configurable attributes of the client application in the proposed simulator are the following ones:

- *Avatar identifier:* Every avatar should have a unique identifier. However, this identifier can be changed *ad-hoc*.
- *Awareness monitoring:* If this option is active, then the awareness rate is monitored by the Loader (the avatar should notify the Loader its current position and which other clients it considers as its neighbors).
- *Awareness period:* Delay between each movement of the avatar and the instant when the client notifies the Loader its current neighbors.
- *AOI size:* Size of the AOI for the avatar controlled by the client. This parameter is directly related with the *presence factor* [19] (the number of neighbors that can see a given avatar currently), and in its turn this parameter has an effect on the workload generated to the hosting client.
- *Quad-tree size:* Minimum region size (COVER awareness method).
- *Uncovered threshold:* Maximum number of uncovered avatars in a region (COVER awareness method).
- *Number of AE:* Number of active entities (awareness method shown in [17]).
- *Iterations:* Number of iterations in the simulation.

Additionally, the simulator has a global configuration file for establishing global simulation options. These options are the following ones:

- *DVE size:* Number of clients in the simulation.
- *Log file:* Name of the log file for the simulation.
- *Updating period:* Period between iterations, that is, the time between two consecutive movements of the avatars.
- *Awareness algorithm:* Awareness method to be used in the simulation (COVER,VON, or the one shown in [17]).
- *Initial distribution:* This parameter determines the initial location of the avatars in the virtual world. The current options allowed are the uniform distribution of avatars, the skewed distribution and the clustered distribution [10].
- *Movement pattern:* This parameter determines the different paths that each avatar can follow in the simulation. Concretely, the simulator supports three movement patterns: Changing Circular Pattern (CCP), Hot-Points-ALL (HPA), and also Hot-Point-Near (HPN). CCP considers that all avatars in the virtual world move ran-

domly around the virtual scene following circular trajectories. HPA considers that there exists certain "hot points" where all avatars approach sooner or later. This movement pattern is typical of multiuser games, where users must get resources (as weapons, energy, vehicles, bonus points, etc,) that are located at certain locations in the virtual world. Finally, HPN also considers these hot-points, but only avatars located within a given radius of the hot-points approach these locations. In order to illustrate these movement patterns, Figure 2 shows the final distribution of avatars that a 2-D virtual world would show if these movement patterns were applied to an uniform initial distribution of avatars. The nine combinations of the initial distributions and movement patterns cover any possible situation in a virtual world.

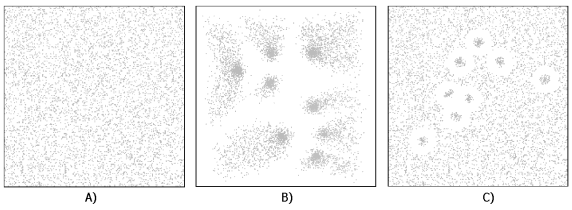

Fig. 2. Movement patterns supported by the simulator: a) CCP, b) HPN, and c) HPA

B. Internal Implementation

Since the purpose of the simulator is to become a peer-to-peer platform for simulating peer-to-peer DVEs, the implementation of the simulator consists of the software to perform the tasks that a peer node should potentially perform: standard node tasks, supernode tasks (COVER awareness method) and Loader tasks. The software architecture used to implement a peer node is modular, and each module has been performed as a C++ class. Figure 3 shows a XML scheme of all the modules implemented, as well as the interactions among them.

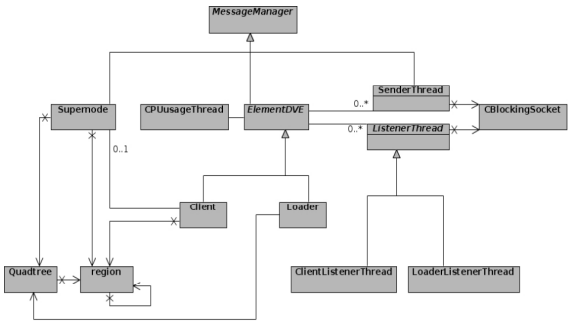

Fig. 3. Modular architecture of a peer node

The classes implemented for this scheme are the following ones:

MessageManager class: This is a generic class that includes a thread for processing requests and also a FIFO queue where other objects can drop their requests.

CPUusageThread class: Thread in charge of periodically monitoring the utilization of the CPU as well as other resources (number of messages sent, etc.).

Quad-tree class: This class represent the hierarchical division of the virtual world. It contains information about the current scene tree and the client identification of the supernode of each region.

Node class: This class represents each one of the regions in the quad-tree.

CBlockingSocket class: A communication class that is based on sockets for reading and writing messages, listening for connections requests, etc..

SenderThread class: This class is derived from the MessageManager class, and it is in charge of taking out and sending the messages in the MessageManager FIFO queue. In order to sned the messages, it uses an object belonging to the CBlockingSocket class.

Supernode class: This thread is derived from the MessageManager class, and it processes the messages sent to the client when that client is the supernode of a region.

ElementDVE class: This is a generic class. Both the clients and the Loader derive from this class, that contains the common methods and attributes of both applications: list of clients in the system, methods for obtaining the initial configuration, etc..

Loader class: It is the main class for the Loader client, and it derives from the MessageManager class. It processes the requests that the listening threads (one for each avatar in the simulation) receive. This processing consists of computing the awareness rate in real time, synchronizing the start of the simulation, etc.

LoaderThread class: This class is composed of listening threads used for receiving the clients requests. It exists one object of this class for each client in the simulation. In order to receive a message, it uses an object of the CBlockingSocket class.

Client class: This is the main class of the client application. It processes all the requests that the listening threads (connected either to the Loader or to other clients) receive and drop in its queue. It contain methods for moving the avatar, computing the neighbors, sending data to the Loader, etc. It uses objects belonging to the SenderThread class for sending information to the rest of clients (including the Loader). It also includes an object of the Supernode class.

ClientThread class: This class is composed of listening threads that are used for receiving requests either from the clients or from the Loader. It exists an object of this class for each neighbor client and another additional object for

the Loader. When these objects receive a message they drop the message either on the client queue or on the supernode queue.

III. Examples of use

We have used the proposed simulator for evaluating the performance of peer-to-peer DVEs. Also, we have used it to evaluate the performance of different awareness methods [17], [16], [18]. These methods had been previously evaluated on sequential systems. However, there are a lot of situations in a real distributed system that do not arise in a stand-alone (sequential) simulator, it is executed on a single computer and therefore time-space inconsistencies due to network latency, computer delays, clock drifts, etc. do not actually exist. Only a distributed simulator like the one proposed in this paper is able to reproduce such situations.

Figure 4 shows the ASR values (latency values) obtained when using the considered awareness method in a given DVE configuration (consisting of 100 avatars using an AOI radius of 10 meters). In this Figure, the X-axis shows the iteration number, and the Y-axis shows the average ASR value obtained for that iteration. Concretely, in this case we have considered the awareness method proposed in [17] with two different sizes of the list of Active Entities, 10 and 20 neighbors. The plots corresponding to these awareness methods are labeled as "KW10" and "KW20", respectively. The awareness method proposed in [16] is labeled as "VON", and the awareness method proposed in [18] is labeled as COVER. In this case, the simulation results show that the awareness method that provides the lowest ASR values is the COVER method.

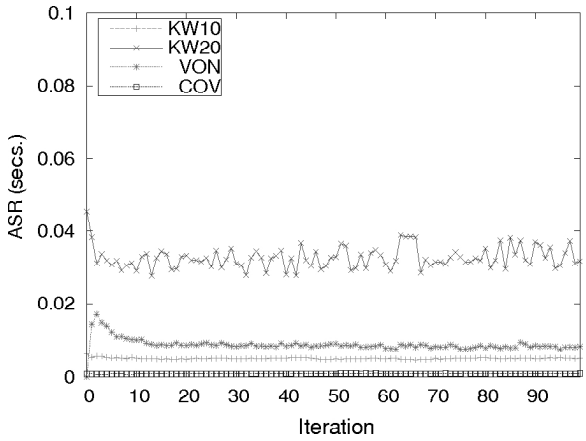

Fig. 4. ASR values obtained with different awareness methods.

Figure 5 shows the awareness delays obtained by the considered awareness methods for the same DVE configuration. In this case, the Y-axis shows the awareness delays provided by the simulator. These values show that the COVER method provides the best awareness delays during the whole simulation.

Table I shows another performance results that can be achieved with the proposed simulator. Concretely, it shows the awareness rates provided by

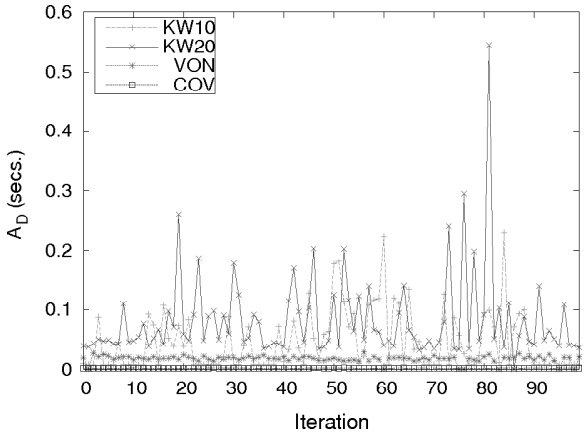

Fig. 5. Awareness delays obtained with different awareness methods.

the considered awareness methods for the same DVE configuration. This table show on each row the average awareness rates achieved in a simulation by a given method when avatars follow a given combination of initial distribution and movement pattern of avatars. Each column shows the results achieved for a different DVE size. Concretely we have considered in this example a P2P DVE of 100, 500 and 1000 avatars.

TABLA I
Awareness rates provided by different awareness methods

	KW10		
	100 av.	500 av.	1000 av.
UNF-CCP	90.9	95.9	96.1
UNF-HPA	85.5	85.9	96.0
UNF-HPN	81.8	92.0	98.2
SKW-CCP	92.7	97.5	96.8
SKW-HPA	88.7	91.8	97.9
SKW-HPN	88.7	94.5	98.0
CLS-CCP	96.4	96.8	97.6
CLS-HPA	90.9	93.5	94.4
CLS-HPN	90.9	96.4	98.7
	VON		
	100 av.	500 av.	1000 av.
UNF-CCP	90.8	94.1	92.1
UNF-HPA	79.4	63.8	92.5
UNF-HPN	89.3	81.1	93.6
SKW-CCP	98.2	92.0	96.3
SKW-HPA	90.3	83.0	88.5
SKW-HPN	89.7	79.4	91.2
CLS-CCP	85.4	87.7	94.6
CLS-HPA	88.6	77.3	89.4
CLS-HPN	92.4	84.8	94.7
	COVER		
	100 av.	500 av.	1000 av.
ALL	100	100	100

Each value in Table I has been computed as the average value of 10 different simulations with the same input parameters. For the sake of shortness, in this case we have shown in this table the results for the KW10, VON and COVER method. Also, the results for the COVER method where all the same (100%) for all the combinations of movement patterns. Therefore, we have shown only a single row for this awareness method. These results show how

two of the considered awareness methods (KW10 and VON) do not actually provide a full (100%) awareness rate (as claimed by heir authors), when non-uniform movement patterns are followed. These results can only be obtained with a distributed platform for simulating P2P DVEs.

Additionally, the proposed simulator can be used as the kernel of any P2P DVE application, due to its modular architecture. The graphical interface for the particular application can be easily linked to the Client module described above. As a result, the implementation of the graphical interface is independent of the high-level graphic library. In order to show that this feature can be used for real simulations, Figure 6 shows a snapshot of a peer-to-peer simulation where the avatars are cars and the virtual world is a city. In this case, the purpose of the simulation tool is the training of novel drivers. The graphical interface has been developed on OpenGL Performer 3.2. This figure shows how the monitoring algorithms and the computations made to evaluate the system performance do not affect the graphical quality of the simulations.

Fig. 6. Peer-to-peer simulation of cars driving inside a city.

IV. Conclusions and Future Work

In this paper, we have proposed a distributed platform for simulating the behavior of Peer-To-Peer DVEs. This simulator takes into account crucial parameters like the number of avatars in the system, their distribution in the virtual world and their movement pattern, among other aspects. It includes mechanisms for evaluation and data acquisition in real time, and it accurately reproduces the behavior of a Distributed Virtual Reality system.

The software architecture used to implement a peer node is modular, and each module has been performed as a C++ class. In this way, the overhead added to clients for measuring the performance metrics in a peer-to-peer DVE have no significant effects on the system response. As a result, the simulator can be used in real simulations to characterize the behavior of peer-to-peer DVEs.

Acknowledgment

This work has been jointly supported by the Spanish MEC and the European Commission FEDER funds under grants Consolider-Ingenio 2010 CSD2006-00046 and TIN2006-15516-C04-04.

Referencias

[1] S. Singhal and M. Zyda, *Networked Virtual Environments*, ACM Press, 1999.
[2] ," Lineage: http://www.lineage2.com.
[3] ," Quake: http://www.idsoftware.com/games/quake.
[4] ," Anarchy Online: : http://www.anarchy-online.com.
[5] E. Frecon and M. Stenius, "Dive: A scalable network architecture for distributed virtual environments," *Distributed Systems Engineering Journal*, vol. 5, no. 3, pp. 91–100, September 1998.
[6] Michael R. Macedonia, M. Zyda, David R. Pratt, Donald P. Brutzman, and Paul T. Barham, "Exploiting reality with multicast groups: A network architecture for large-scale virtual environments," in *Proceedings of the 1995 IEEE Virtual Reality Annual Symposium*, 1995, pp. 2–10.
[7] P. Morillo, J.M. Orduña, and J. Duato, "A scalable synchronization technique for distributed virtual environments based on networked-server architectures," in *Proceedings of the 35th IEEE International Conference on Parallel Processing (ICPP'06) Workshops*. 2006, pp. 74–81, IEEE Computer Society Press.
[8] David Salomon, *A guide to data compression methods*, Springer-Verlag, 2001.
[9] John C.S. Lui and M.F. Chan, "An efficient partitioning algorithm for distributed virtual environment systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 13, 2002.
[10] P. Morillo, J. M. Orduña, M. Fernández, and J. Duato, "Improving the performance of distributed virtual environment systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 7, pp. 637–649, 2005.
[11] Silvia Rueda, Pedro Morillo, and Juan Manuel Orduña, "On the characterization of peer-to-peer distributed virtual environments," in *Proceedings of International Conference on Cyberworlds 2007 (Cyberworlds'07), Hannover, Germany*. 2007, IEEE Computer Society Press.
[12] IEEE, *1278.1 IEEE Standard for Distributed Interactive Simulation-Application Protocols (ANSI)*, 1997.
[13] F. Kuhl, R. Weatherly, and J. Dahmann, *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*, Prentice-Hall PTR, 1999.
[14] M. Oliveira, J. Crowcroft, and M. Slater, "Components for distributed virtual environments," *PRESENCE: Teleoperators and Virtual Environments*, vol. 10, no. 1, pp. 56–61, 2001.
[15] Randall B. Smith, Ronald Hixon, and Bernard Horan, *Collaborative Virtual Environments*, Springer-Verlag, 2001.
[16] S. Y. Hu and G. M. Liao, "Scalable peer-to-peer networked virtual environment," in *Proceedings ACM SIGCOMM 2004 workshops on NetGames '04*, 2004, pp. 129–133.
[17] Y. Kawahara, T. Aoyama, and H. Morikawa, "A peer-to-peer message exchange scheme for large scale networked virtual environments," *Telecommunication Systems*, vol. 25, no. 3, pp. 353–370, 2004.
[18] P. Morillo, W. Moncho, J. M. Orduña, and J. Duato, "Providing full awareness to distributed virtual environments based on peer-to-peer architectures," *Lecture Notes on Computer Science*, vol. 4035, pp. 336–347, 2006.
[19] P. Morillo, J. M. Orduña, M. Fernández, and J. Duato, "On the characterization of avatars in distributed virtual worlds," in *EUROGRAPHICS' 2003 Workshops*. The Eurographics Association, 2003, pp. 215–220.