School of Computing Science,
University of Newcastle upon Tyne

Scalable Massively Multiplayer Online Games

Graham Morgan

Technical Report Series

CS-TR-888

February 2005

Scalable Massively Multiplayer Online Games

Graham Morgan

School of Computing Science

Newcastle University, Newcastle upon Tyne, NE1 7RU, UK

Telephone: + 44 191 222 7983, Fax: + 44 191 222 8232

E-mail: Graham.Morgan@newcastle.ac.uk

Abstract

We describe an approach for satisfying the message dissemination requirements of distributed virtual environments. We assume a distributed virtual environment is deployed using client/server architecture typical of commercial massively multiplayer online games. We exploit the scalability provided by the clustering of servers in the development of a message dissemination scheme that may scale to satisfy the requirements of many thousands of players.

1. Introduction

A distributed virtual environment (DVE) may be defined as a computer simulation subject to user manipulation in real-time that is presented via multimedia technologies (e.g., graphics, sound) to more than one user at the same time. The concern of this paper are those DVE applications designed for social play that provide a simulation that approximates a 3D geographic model populated with objects representing a virtual world providing players with a sense of realism. For example, landscapes that are similar to those found in the real world (e.g., land, water, sky, buildings) that are governed by laws similar to that found in classical physics (e.g., solid objects may not share the same space at the same time).

The use of DVE applications for social play has increased in popularity in recent years (e.g., Star Wars Galaxies^{TM} [6]). This, in part, is due to the proliferation of broadband ISP access into the home of game players together with the latest game consoles designed with network connectivity in mind (e.g., Xbox, PS2). Commercial success of an online game is not measured only in terms of extended shelf life (the length of time a game remains popular with buyers), but by an ability to charge players to play. In such scenarios, commercial success may be approximately measured by the number of registered players that “pay-to-play”. A massively multi-player online game (MMOG), where registered players may be measured in the hundreds of thousands, presents the gaming industry with the most appropriate approach for commercial success.

MMOGs are deployed using a client/server based architecture. Players register directly with a server of a game provider, allowing a game provider to manage the pay-to-play aspect of an online game centrally. The existence of a server also allows a vendor to manage a game instance (allowing persistence of virtual world game state) and monitor player activity (a requirement to identify inappropriate player behaviour such as cheating). Persistent virtual worlds may maintain player interest over substantial periods of time as periodic introduction of game enhancements by a provider and the evolving of a player’s game character introduce a constantly changing gaming environment. The ability to maintain player interest is a desirable property for a commercial MMOG that generates revenue from players paying to play via regular subscription fees (which is the basis of the economic model used in many commercial MMOGs).

Due to the interactive nature of an MMOG, the requirements that need to be satisfied to provide a gaming environment that may scale to support large complex worlds with many thousands of players are challenging:

- **Timeliness** – state change events in the virtual world must be propagated to all players in a timely manner to allow the desired player interaction to occur.

- **Consistency** – interacting players must share a view of the virtual world that is sufficiently mutually consistent to ensure a fair gaming environment.

The scalability issues addressed by an MMOG relate, primarily, to the number of players and the complexity of the game (e.g., number of objects populating the virtual world). An increase in the number of players may lead to an increase in network traffic as a player's console attempts to propagate local game state updates (related to player events) to other players. If an attempt is not made to identify appropriate message recipients and so inhibit the sending and processing of unnecessary messages, then providing participation for many hundreds or thousands of players will not be possible. The process of identifying appropriate message recipients, referred to as *interest management*, introduces further computation and must be successfully completed in a timely manner to ensure timeliness and consistency requirements are satisfied.

The ability to provide game server technologies that may scale to satisfy the requirements of many thousands of players is essential to ensure the commercial success of an MMOG. An approach to server side scalability is via the clustering of servers. Existing server cluster technology typically utilises standard personal computers (PCs), acting as servers, that are co-located on the same local area network (LAN). The responsibility for satisfying client requests is distributed across servers in a cluster in an effort to utilise the combined processing resources offered by servers. In such a scheme, an increase in process resource requirements that exceed available processing recourses may be satisfied via the addition of servers to a cluster with minimum disruption to service provision. An interest management scheme must be deployed in a manner that may benefit from the scalability offered by clustered server technologies if scalable MMOGs are to be realised.

In this paper we describe an approach to interest management implementation that is suitable for deployment over a cluster of servers. By utilising server clustering, we aim to satisfy the timeliness and consistency requirements of an MMOG while providing complex virtual worlds that may scale to support many thousands of players.

2. Interest Management

Interest management is the term commonly used to describe restricted message dissemination using virtual space division in an effort to limit the sending of messages between objects to only those required to ensure mutually consistent views of the virtual world for all players. Interest management schemes may be classified into two broad categories based on their approach to virtual space sub-division:

- Region - virtual world is divided into well defined regions that are static in nature (defined at virtual world creation time). The recipient (object) of a message is limited to only interested participants (i.e., reside within the same, or neighboring, region as the sender object). This approach is illustrated by NPSNET [1], where the terrain of the virtual world is divided into hexagonal cells.
- Aura - each object is associated to an aura that defines an area of the virtual world over which an object may exert influence. An aura may be simply modeled as a sphere that shares its centre with the positional vector of the object it is associated with. An object may potentially communicate their actions only to objects that fall within their aura. This approach is illustrated by the Model, Architecture and System for Spatial Interaction in Virtual Environments projects (MASSIVE 1 & 2) [2].

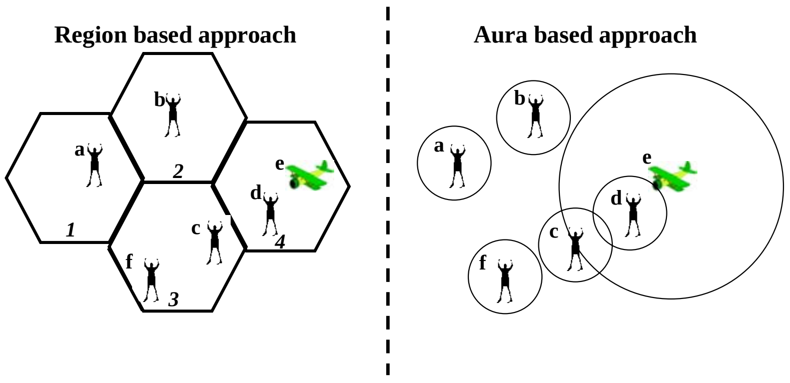

Figure 1 - Region and aura based approaches to interest management.

The way a virtual world is sub-divided in an interest management scheme determines message recipients and influences the scalability of an MMOG: if an area of sub-division is too large then there is a possibility that unnecessary messages may be received by some objects; if an area of sub-division is too small then there is a possibility that interest management may not be resolved in a timely manner to ensure purposeful message dissemination may occur.

In the region based approach, region size may not be suitable for all types of objects. For example, if region size is decided when considering the top speed of a fighter aircraft then soldiers traveling on foot may give rise to unnecessary message exchange between foot soldier objects that are separated by a distance too great for such objects to influence each other. Alternatively, if region size is more suited to foot soldier objects then a fighter aircraft may traverse region boundaries with such frequency that there is a possibility that an object can traverse a region in less time than it takes to realize regional membership changes (preventing meaningful message dissemination).

The aura based approach to interest management provides a more accurate model of object interaction on which to base message exchange than the region based approach. Figure 1 describes a virtual world scene using region and aura based interest management. Using auras we may determine that object *e* (plane) may influence objects *c* and *d* (due to aura overlap). However, in the region based approach object *e* may only influence object *d* (as *c* is in another region). We could expand the area of influence of object *e* to additional regions and so allow *e* to influence *c*, but this would result in object *e* appearing to influence object *f* (messages unnecessarily sent from *e* to *f*).

To facilitate message exchange a communication sub-system must exist that may identify message recipients in a manner suitable for implementing an interest management scheme. In the region based approach message exchange may be achieved by associating each region with an identifier that may be used to send/receive messages to/from. For example, each region may be assigned an IP-multicast address. If an object traverses a region boundary, say region *1* to region *2,* an object simply subscribes itself as a sender/receiver for region *2*'s IP-multicast address and unsubscribes from region *1*'s IP-multicast address. As there are well defined regions that do not change throughout the lifetime of a virtual world, objects need only be aware of which region they are in to enable message passing to occur. There is no need for individual objects to contact each other directly to determine message recipients.

Due to the lack of static regions, implementing message exchange in an aura based interest management scheme requires periodic message passing to determine aura collisions as and when they happen. The frequency of such message exchange must be sufficient to ensure that aura collision may be determined in time to allow objects to purposely disseminate messages. If message exchange frequency is not sufficient there is the possibility that objects are unaware of aura collision as such a collision may not last for a sufficient amount of time to enable appropriate message recipients to be identified before objects move away from each other (aura collision no longer exists). We term this situation the *missed interaction problem*. Increasing message frequency to prevent the missed interaction problem has a negative impact on scalability: an object will typically only receive a small percentage of messages that relate to collisions that involve its own aura (unnecessary message exchange).

From our discussion it is clear that auras provide the potential to model object interaction in a more appropriate manner than regions. However, regions require less processing and message passing requirements when determining message recipients. The question posed is:

"Is it possible to implement interest management that provides the detailed modeling of interaction associated with auras in a scalable way for use in MMOGs?"

Research into interest management has not considered client/server style implementations suitable for MMOGs. Therefore, our research goals may be stated as:

- Develop aura based interest management scheme that reduces the possibility of missed interactions.
- Implement our interest management scheme using clustered server technologies to provide scalability that is suitable for MMOGs.

The next two sections provide summary descriptions of our interest management scheme and its implementation.

3. Predictive Interest Management

In this section we describe an interest management scheme based on the aura approach that attempts to overcome the missed interaction problem by using a three step approach to message exchange: (i) low frequency message exchange used for object discovery; (ii) variable message exchange, with frequency of messages between objects related to the probability of interaction of such objects in the future; (iii) high frequency message exchange between interacting objects. We present here only a brief overview of the technique to enable a reader to gain the basic underlying theory of the work. For a more complete description of predicted interest management and associated sphere collision detection techniques the reader is referred to [3] [4].

3.1 Identifying Scope of Interest

The aura of an object describes an area of the virtual world enclosed by a sphere (Figure 2). The radius of an aura is specified on a per object basis and is fixed at object creation time. Objects have the ability to influence each other when their auras collide via the exchange of messages.

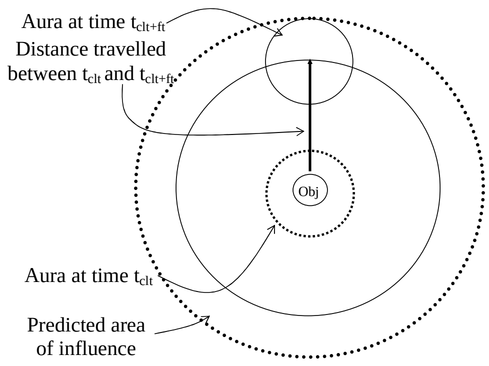
Aura at time t_{clt+ft}
Distance travelled between t_{clt} and t_{clt+ft}
Obj
Aura at time t_{clt}
Predicted area of influence

Figure 2 – Defining Predicted Area of Influence (PAI).

A predicted area of influence (*PAI*) identifies the extent of an object's aura over a period of time given the distance an object may travel in a straight line in any direction (figure 2). The period of time used to identify a *PAI* is bounded by current time, say t_{clt}, and some future time (t_{clt+ft}, where *ft* is a positive number and is defined system wide). By this method the distance an object, say obj_1, travels in a straight line identifies the radius of a sphere that encloses all the areas of a virtual space reachable by obj_1 between t_{clt} and t_{clt+ft} with the position vector of obj_1 at time t_{clt} defining the centre of this sphere. Extending this radius by the radius of obj_1's aura defines a sphere that describes the *PAI* for obj_1. When determining a *PAI* we assume an object is travelling at its highest speed (defined on a per object basis) in a straight line at time t_{clt} and continues at this speed and direction until t_{clt+ft}. This presents a *PAI* that is guaranteed to contain all possible auras of an object between t_{clt} and t_{clt+ft} irrelevant of an object's velocity. Assuming the highest speed remains constant for an object throughout its lifetime allows a *PAI* to be calculated and fixed at object creation time.

When the *PAI*s of two objects collide, but not their auras, there is a possibility that such objects may influence each other and subsequently exchange messages at some point in the near future. A period of time (*window of collision*) may be defined within which the auras of such objects may collide. Using collision detection techniques based on the intersection of spheres we may identify the separating distance between two objects as *sd*. If a collision window exists between two objects then *sd* may be used to determine an approximate upper bound value (*AUBV*) indicating the time taken for the auras of these objects to collide assuming they are travelling towards each other at their respective full speeds. *AUBV* provides a basis for determining frequency of message exchange between two objects within the same collision window.

3.2 Message Exchange

An object is responsible for sending a *positional update message* (*PUM*), identifying its position vector. *PUMs* are sent frequently and form the basic mechanism for message exchange between objects that are influencing each other. To determine if *PAI*s overlap objects must send an *admin PUM* (*APUM*) containing aura radius, *PAI* radius and vector position information. *APUM*s are sent less frequently than *PUM*s and form the basic mechanism for identifying when objects should exchange *PUM*s.

We assume the existence of a communications sub-system capable of providing reliable FIFO channels that may facilitate inter-object communications via the sending and receiving of *PUM*s and *APUM*s:

- **Admin**: Used to disseminate *APUM*s to all objects.
- **Local**: Created on a per object basis to provide mechanism for passing *APUM*s and *PUM*s between objects without the need to send messages to all objects.

Three local timers, associated on a per object basis, are responsible for regulating the frequency of publishing *APUM*s on the admin channel (*ta*), *APUM*s on a local channel (*tal*) and *PUM*s on a local channel (*tp*) respectively. The value of *ta* should be set to a value that ensures *APUM*s may be received and processed by all objects in time to determine potential aura overlaps. The time interval *tal* used to define the frequency a node publishes *APUM*s on the local event channel is determined by *AUBV*. The time interval *tp* is defined on a per object basis at a developer's discretion (message exchange associated to interacting objects).

The modelling of variable frequency message exchange via admin and local channels is achieved as follows:

- **Low frequency** – all objects periodically send *APUM*s to all other objects via the admin channel at a frequency determined by *ta*.
- **Variable frequency** – an object, say O_i, sends *APUM*s to another object, say O_j, using O_i's local channel at a frequency determined by *tal* if O_j determines that interaction between O_i and O_j may occur in the near future.
- **High frequency** – an object, say O_i, sends *PUM*s to another object, say O_j, using O_i's local channel at a frequency determined by *tp* if O_j determines that interaction between O_i and O_j is occurring now.

An object decides on subscription and *APUM* frequency based only on local information (derived from *PUM*s and *APUM*s). Therefore, it may be possible for an object, say O_i, to subscribe to another object's, say O_j's, local channel without O_j subscribing to O_i's local channel. However, if such objects are moving closer to each other then we may assume that O_j will receive the appropriate *APUM* on the admin channel and subscribe to O_i's local channel before *PUM* message exchange is required. If objects are moving apart then it may be that O_i will remove its subscription from O_j's local channel with O_j never subscribing to O_i's local channel. This is acceptable behaviour and an overhead which is a result of attempting to prevent missed interactions.

4. Interest Management via Clustering of Servers

The clustering of servers for the delivery of scalable services over the Internet is commonplace (e.g., e-commerce, search engines). Servers that belong to a cluster are commodity low cost PCs and are typically co-located on the same LAN. A server cluster appears to a client as a single server instance and may be manipulated during run-time with minimum disruption to the satisfying of client requests. Such manipulation may be to remove a server for maintenance or to add a server to increase processing resources.

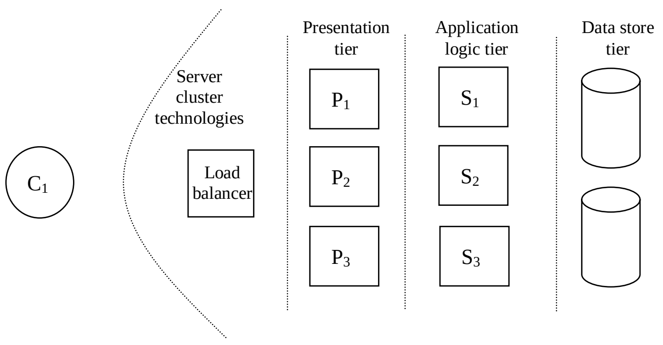

Figure 3 – Typical configuration of server cluster technologies.

Figure 3 describes a typical configuration of server cluster technology. Such systems are commonly termed n-tier systems, with each tier of a cluster associated with specific roles and responsibilities: presentation tier satisfies client interface requirements, application logic tier is where the business logic of an application exists, data store tier satisfies the persistent data requirements of a cluster. The tiers of an n-tier system are a logical view of a server cluster and may not reflect the physical configuration (components belonging to different tiers may exist on the same server). A key component to providing a scalable service and presenting a single service view of a cluster to a client is the load balancer. The load balancer is responsible for distributing client requests across the server cluster to allow appropriate utilization of processing resources.

Existing server clustering solutions typically make use of application servers [7]. Application servers deploy a variety of middleware services and provide a high level abstraction of component oriented middleware; only the business logic of an application needs to be addressed by a programmer with support services, including clustering, incorporated into the application at deployment time. Application servers provide an environment suitable for e-commerce style applications which we found not sufficiently optimised for an interest management implementation. Therefore, we based our implementation (described in figure 4) on the Common Object Request Broker Architecture (CORBA). The choice of CORBA was deemed appropriate as: (i) ease distributed application development as many useful CORBA services are available; (ii) supports C++, the programming language of choice within the games industry due to execution speed and the large number of C++ games programming tools.

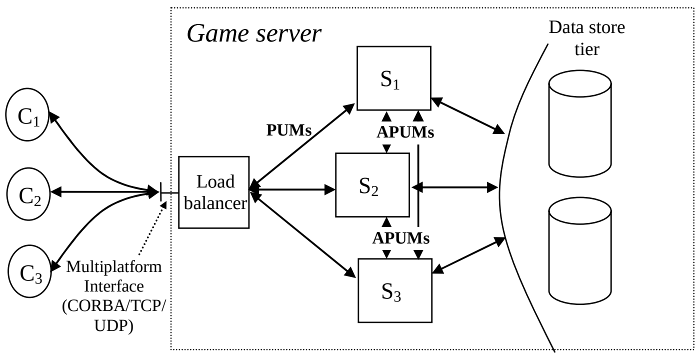

Figure 4 – Server clustering for interest management to support a game server.

Clients may communicate with our game server via a number of communication primitives (CORBA RPC, TCP and UDP). The use of CORBA allows clients to take advantage of CORBA services (e.g., security, transactions, location and discovery). We include TCP and UDP as they are commonly used, via socket programming, in current MMOG development. For brevity, we consider only clients using CORBA RPC in the remainder of this paper.

The game server presents a wealth of functionality for use by clients, including object registration (e.g., image and game play attributes), animation blending maps (animation sequences for an object) and text message passing between players (allowing communications between players in a textual format). The data store tier is used to store information relating to player and object attributes and is implemented using propriety database technologies. Our concern, for this paper, is on clustering technologies related to predictive interest management and so we perceive the data store as a single instance of a commercial database (e.g., Oracle) and do not discuss the additional functionality beyond that of *PUM* and *APUM* exchange.

Clients periodically send *PUM*s to the load balancer. As a client may manage multiple objects (we provide flexibility in our approach in that we do not limit a client to a single object representation in the virtual world), a single message may contain multiple *PUM*s. These messages are synchronous CORBA calls, with the return part of the message containing one or more *PUM*s relating to objects that are hosted on other clients. The game server may send *PUM*s to clients that have not sent *PUM*s for a substantial length of time (i.e., due to object inactivity – timeout determined by client). Our approach to client/game server interaction eases client participation in a virtual world as a client only need send *PUM*s, not *APUM*s.

The load balancer uses a round robin approach to associate clients to a server (association lasts for as long as a client is connected to the game server). Servers exchange *APUM* messages and implement predictive interest management. *APUM* message exchange is facilitated by *Message Oriented Middleware* (MOM) (CORBA Notification Service (CORBA NS)). The central element of the CORBA NS is a notification channel that assumes the role of propagating messages from suppliers (senders) to consumers (receivers). MOM decouples communications between sender and receiver, allowing one or more suppliers to issue messages for one or more consumers. Via CORBA NS, we can represent the admin and local channels required for implementing predictive interest management. However, rather than have local channels created on a per object basis, they are created on a per server basis. Periodically, individual *APUM*s are combined into single messages and distributed on a per-server basis using local channels. Similarly, *APUM*s from a server destined for the admin channel are combined into a single message.

The load balancer updates a table indicating which objects a client is interested in whenever a server informs the load balancer of a change in object interest. This allows the load balancer to send *PUM*s to appropriate clients. The sending of *PUM*s to clients using this method is efficient: (i) PUM_a received by load balancer from a client, (ii) load balancer checks table to determine recipients for PUM_a, (iii) load balancer adds PUM_a to the appropriate client messages, (iv) load balancer sends messages to clients (as reply to synchronous call). There is no need to involve servers when returning *PUM*s to clients.

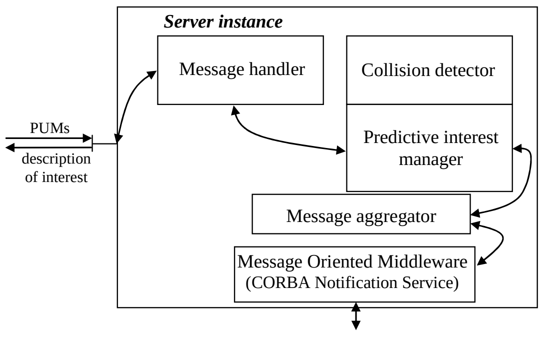

Figure 5 – Components of a server instance.

Figure 5 describes the main server components that contribute to satisfying the interest management requirements:

- **Message handler** - receives and returns messages to load balancer. If necessary, registering new object information using data store tier.
- **Predictive interest manager** - uses predictive interest management to construct appropriate *APUM* messages.
- **Collision detector** - identifies aura and *PAI* overlap to aid predictive interest manager in constructing appropriate *APUM* messages.
- **Message aggregator** - composes single messages from multiple *APUM* messages for distribution via the appropriate CORBA notification service channels.
- **Message oriented middleware (CORBA NS)** - supports message exchange between servers.

The message handler receives *PUM*s from the load balancer and returns to the load balancer descriptions relating to object interests. The interest manager implements the predictive interest management scheme and calls on the collision detector to identify aura and *PAI* overlap. The collision detector implements a collision detection algorithm that we specifically designed for use with predictive interest management [5]. The interest manager constructs *APUM*s and passes them to the message aggregator, which in turn composes single messages from multiple *APUM*s on a per server basis and places such messages on the appropriate CORBA NS channels. *APUM*s are received from CORBA NS channels by the message aggregator and are passed to the interest manager to aid in determining the interest of objects. Information relating to the interest of objects is passed to the message handler by the interest manager. The message handler then informs the load balancer of updated object interests.

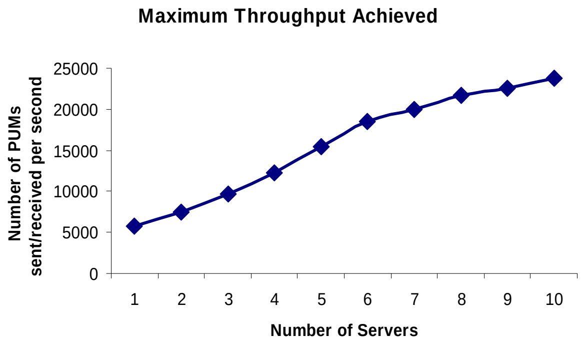

Figure 6 – Performance of cluster.

Figure 6 shows performance results indicating that our approach to predictive interest management via clustering technologies is scalable. When a server is added to a cluster the maximum throughput of messages (as perceived by clients via the sending and receiving of *PUM*s) rises. For our experiment we used a virtual world populated by objects of varying aura and *PAI* size and ensured that the cumulative volume of object *PAI*s was within 10% and 15% of the total volume of the virtual world.

5. Conclusions

We have presented an approach to interest management that aims to facilitate the accurate modeling of object interaction in DVEs while ensuring missed object interaction are avoided. This is achieved by varying message exchange between objects based on the likelihood that such objects will interact in the future. Furthermore, we have implemented our interest management service in a manner that is suitable for supporting MMOGs. By basing our deployment on server cluster technologies, we have provided a service that may scale to satisfy the requirements of MMOGs.

We expect increasing numbers of game vendors will depend on server clustering solutions to deliver highly available scalable MMOGs to their players in the future. This dependency poses hard QoS related research challenges beyond that of scalability: timeliness, inter-organisational issues, availability, metric monitoring and adaptability.

6. References

[1] M.R. Macedonia, M.J. Zyda, D.R. Pratt, P.T. Barham, S. Zeswitz, “NPSNET: A Network Software Architecture for Large Scale Virtual Environments”, MIT Presence 3(4), 1994.

[2] C. Greenhalgh, S. Benford, “MASSIVE: a distributed virtual reality system incorporating spatial trading”, Proceedings 15th International Conference on distributed computing systems (DCSc 95), Canada, June 1995

[3] G. Morgan, F. Lu, “Predictive Interest Management: An Approach to Managing Message Dissemination for Distributed Virtual Environments”, Richmedia2003, Switzerland, 2003

[4] K. Storey, F. Lu, G. Morgan, “Determining Collisions between Moving Spheres for Distributed Virtual Environments”, Computer Graphics International (CGI 2004), Crete, June 16 - 19, 2004, pp. 140-147, IEEE Computer Society Press 2004

[5] G. Morgan, K. Storey, F. Lu, “Expanding Spheres: A Collision Detection Algorithm for Interest Management in Networked Games”, In Proceedings of the Entertainment Computing – ICEC 2004: Third International Conference, Eindhoven, The Netherlands, September 1-3 Lecture Notes in Computer Science Volume 3166 pp. 435 – 440, Springer-Verlag 2004

[6] Daniel Amara, Prima Temp Authors, “Star Wars Galaxies: An Empire Divided : Prima's Official Strategy Guide”, Prima Games, July 1, 2003

[7] Sun Microsystems^{TM}, “Java™ 2 Platform Enterprise Edition Specification, v1.4”, November 24, 2003, *http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf* as viewed November 2004