# Network Software Architectures for Real-Time Massively-Multiplayer Online Games

Roger Delano Paul McFarlane

Degree of Master of Science

School of Computer Science

McGill University

Montreal, Quebec, Canada

Feb. 02, 2005

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of Science.

## ACKNOWLEDGMENTS

First and foremost, I would like to thank my wife, Abaynesta, for her support and encouragement. Undertaking a full-time graduate degree while maintaining full-time employment and trying to be a full-time husband would have been an impossible challenge without your love, patience, and understanding. I love you.

To my supervisor, Dr. Jörg Kienzle, I would like to express my thanks for taking on a student with a research area only somewhat related to your own, Software Fault Tolerance, and granting me the latitude to freely explore massively multiplayer game infrastructure.

In the course of pursuing my graduate degree and writing this thesis, I was employed by two supportive organizations. To the management of Zero-Knowledge Systems Inc., most notably co-founders Austin Hill and Hammie Hill, I express my profound gratitude for your support and encouragement to pursue my academic goals. Thank you for allowing me the flexibility of schedule to take on a full-time course load and for your understanding and encouragement when the time came for me to move on to other opportunities.

I would also like to thank the management of the Ubi.com group at Ubisoft Entertainment Inc. for recognizing the potential of this graduate student with no direct gaming experience and giving me the opportunity to learn from and contribute to your game projects while completing my thesis.

# TABLE OF CONTENTS

# LIST OF FIGURES

## ABSTRACT

A real-time massively multiplayer online game (MMOG) is a networked computer or video game in which tens of thousands to hundreds of thousands of consumers may interact with one another in real-time in a shared environment, even though these users may be separated by vast geographic distances. Game industry analysis highlights trends indicating that online game usage and market penetration will grow significantly over the next five to ten years. As such, game developers and entertainment companies seek to offer subscription based mass-market online games. However, the risks, costs and complexity involved in the successful development and operation of a scalable online game service are high, in part due to lack of well established and understood models for the network software architecture of such a product. This thesis explores the literature and research regarding distributed military simulation, academic networked virtual environments, and commercial online gaming in search of patterns for network software architectures which are applicable to massively multiplayer online games. It is the hope of the author to contribute to this cross pollination of ideas by providing a thorough review of the techniques and approaches for the design and implementation of large scale distributed systems having properties similar to those found in a massively multiplayer online game system. In this way, perhaps the cost, complexity, and risk involved in building a massively multiplayer online game service can be reduced.

## ABRÉGÉ

Un jeu en ligne massivement multi joueurs en temps réel est un jeu vidéo ou d'ordinateur géré en réseau dans lequel des dizaines à des centaines de milliers de consommateurs peuvent interagir entre eux en temps réel dans un environnement partagé, et ce même s’ils sont répartis dans des régions géographiques très distantes. Les analyses de l’industrie du jeu démontrent que l'utilisation et la pénétration de marché du jeu en ligne se développeront de manière significative au cours des cinq à dix prochaines années. Ceci explique que les développeurs de jeu et les compagnies de divertissement cherchent à offrir à un marché grand public des jeux en ligne basés sur un abonnement. Cependant, les risques, les coûts et la complexité impliqués dans le développement et l'opération d’un service de jeu en ligne sont élevés, dû en partie au manque de modèles bien établis et compris pour l'architecture de logiciels de réseau de tels produits. Cette thèse explore la littérature et la recherche concernant la simulation militaire distribuée, les environnements académiques virtuels gérés en réseau, et le jeu en ligne commercial à la recherche de modèles pour les architectures de logiciels de réseau qui sont applicables aux jeux en ligne massivement multi joueurs. C'est l'espoir de l'auteur de contribuer à cette pollinisation d’idées en fournissant un examen complet des techniques et des approches utilisés dans la conception et l'implémentation de systèmes répartis à grande échelle ayant des propriétés semblables à celles que l’on retrouve dans les systèmes de jeu en ligne massivement multi joueurs.

De cette façon, peut-être, le coût, la complexité et le risque impliqués dans la réalisation d’un service de jeu en ligne massivement multi joueur pourront être réduits.

# Introduction

Already earning more revenue than the movie industry, media and entertainment analysts predict the earnings of the video game industry will surpass that of the music industry, the current revenue leader in the global entertainment and media industry, by the year 2006 [159]. Other industry observers predict that worldwide growth for video games and personal computer (PC) games to be between 37% to 45% between 2002 and 2007 [52]. The global video game market is expected to expand from its current value of $21.2 billion in 2002 to a market value between $28.4 billion and $35.8 billion in 2007, growing at an 11% compound annual rate [52, 158]. None of the aforementioned figures take into consideration revenue based on the sale of accessories, game rentals and the resale of used games, which some analysts estimate will account for an additional $5 billion or more in annual consumer spending [52].

Globally, industry watchers predict that the largest video game market will be the Asia/Pacific market, growing 8.5% annually from a spending level of $8.4 billion in 2002 to $12.6 billion by 2007 [158]. The United States market will experience similar growth, reaching a spending level of $12.3 billion in 2007; this represents an 11.3% annual growth average from $7.2 billion in 2002. The market representing Europe, the Middle East, and the rest of Asia will experience 13.8% annual growth to reach $9.3 billion by 2007 from a level of $4.9 billion in 2002. At the same time, the Latin American and Canadian markets will experience the fastest growth, expanding by a 16.1% yearly to $312 million in 2007 from $148 million in 2002 and $1.2 billion in 2007 from $575 million in 2002, respectively.

Alongside the growth of the gaming market will be increased household penetration of broadband Internet connectivity. Game industry analysts predict that the worldwide number of households with broadband will grow to 190 million by 2008 [51]. Nearly 70% of South Korean households, for example, already have broadband connections and the number of broadband enabled households in Europe increased by 140% in 2002 [51]. As broadband services continue to become ubiquitous in these markets as well as in the rest of Asia and in North America, the video game industry has new opportunities to take their product offerings online. As a specific example, Korean video game developer and publisher NCsoft became the first online game vendor to earn over $100 million per year with Lineage [141] its highly successful massively-multiplayer online game.

The success observed in Korea, as well as the success in North America and Europe of massively multiplayer online games such as Sony Online Entertainment's Everquest [179] and Lucas Arts Entertainment's Starwars Galaxies [121] is an indicator of the connection between the growth of broadband and the growth of online games. Industry analysts forecast that the number of worldwide online games played will increase from 73 million in 2002 to 198 million by 2008, with online game usage growing to 35 billion hours per year [51]. In North America, adults representing more than 42 million households currently use their personal computers to play games for about 4.8 hours per week either online or offline [198]; by 2007, one quarter of North American households, representing approximately 70 million people, will be playing games online [117].

Finally, the latest generation of game consoles are now “online-capable” and represent a large installed customer base that is yet to transition to online gaming and, thus, a further growth opportunity for online gaming [51]. By 2006 there will be 23.4 million online console gamers worldwide; however, even given predicted growth of 20% by 2008, online console gamers will still represent only 20% of all online gamers [50].

All indications are that online game usage and penetration will grow significantly; however, a number of challenges will make if difficult for companies in the online game market to generate profits from their products [51]. Firstly, the costs and complexity involved in the development and operation of an online game service are high [5, 43, 51, 176]. Secondly, consumers are reluctant to pay for gaming content on a monthly basis, as is currently the industry norm [51, 67]. Additionally, the overwhelming majority of online games that are presently in development or operation are narrowly clustered in the fantasy role-playing genre, which has not proven accessible to the mass-market audience [67, 78, 134]. A game developer must overcome these and other challenges in order to field a commercially successful online game service.

## Motivation

As mentioned above, developing a massively multiplayer online game service is a high cost and high complexity endeavour. A part of this difficulty arises from the fact that most commercial online game services are based on proprietary architectures and designs and there has been relatively little reuse of the lessons learned from other, similar, projects. This thesis explores the literature and research regarding distributed military simulation, networked virtual environments, and commercial online gaming in search of patterns for network software architectures which are applicable to massively multiplayer online games. As we will see, there exists a large body of knowledge from the military and academic simulation communities that is slowly being appropriated by the developers of commercial video games. It is the hope of the author to contribute to this cross pollination of ideas by providing a thorough review of the techniques and approaches for the design and implementation of large scale distributed systems having properties similar to those found in a massively multiplayer online game system. In this way, perhaps the cost, complexity, and risk involved in building a massively multiplayer online game service can be reduced.

## Overview

The body of this thesis is broken down into four main sections. The first is an introduction to massively multiplayer online games; we present a definition for massively multiplayer games and provide a brief overview of the history of such games to the present. Secondly, we explore the technical challenges inherent in building and operating a massively multiplayer online game. We identify the similarities and differences between the challenges considered by military and academic community versus those facing the gaming community, particularly in those cases where the communities make very different assumptions about the operating environment, needs, or behaviour of the participants in the online environment.

Thirdly, we survey the related research and literature from the military, academic, and commercial gaming spaces. This section, comprising the majority of this thesis, explores lessons learned from significant projects in each of these domains and considers each project's network software architecture and systems for appropriateness, or lack thereof, to massively multiplayer online game systems. Lastly, we draw upon the surveyed material to consider several client/server and peer-to-peer network software architectures that can be used to construct a real-time massively multiplayer online game.

# Massively Multiplayer Online Games

Massively Multiplayer Game (MMP and/or MMPG), Massively Multiplayer Online Game (MMO, MMOG, and/or MMPOG), Massively Multiplayer Online Role-Playing Game (MMORPG), and Massively Multiplayer Online Persistent World (MMOPW) are names which are used interchangeably to refer to networked video and computer games in which a large number of simultaneous players may each assume an identity and interact in a shared environment. For consistency, this paper uses the name Massively Multiplayer Online Game and the acronym MMOG throughout. The reader will encounter all of these terms, and perhaps more, in the referenced literature. This chapter explores the nature of such games as well as the market and history behind them.

## A Definition

Many computer and video games feature multiplayer capabilities. These capabilities range from two players sitting together in front of a gaming station to thousands of players interacting across the Internet. For the purposes of this discussion we define a massively multiplayer online game (MMOG) as a networked computer or video game in which tens of thousands to hundreds of thousands of consumers may interact with one another in real-time in a shared environment, even though these users may be separated by vast geographic distances. This definition is an adaptation of that for a Networked Virtual Environment as stated in [176] and embodies the following characteristics, the first five of which are also from [176]:

- ***A shared sense of space:*** All players participate and interact in a common virtual world. Each participant observes the same characteristics and attributes of the environment, and is able to observe other participants within the environment.
- ***A shared sense of presence:*** Each participant is represented by a virtual persona, commonly called an avatar, within the environment. An avatar conveys to other participants a physical form, animated motions, expressions, and so forth on behalf of the controlling participant.
- ***A shared sense of time:*** Participants are able to observe the behaviour of other participants in the environment as it occurs.
- ***A way to communicate:*** Participants are able to convey information to other participants through some communication medium, be it gestures, textual messages, or audible voice.
- ***A way to share:*** Participants interact not only with one another, but also with the environment. The environment reflects the actions of the participants where appropriate, and enforces the necessary constraints applicable to the represented locale.
- ***A massive concurrent user population:*** The number of simultaneous participants is in the tens of thousands to hundreds of thousands. As we will see, the challenges inherent in constructing

an online game to support such a large number of simultaneous users increase dramatically over those found in supporting four to thirty-two players, the amounts commonly supported for current multiplayer games.

- ***A sense of fairness:*** We assume that, given the opportunity, some portion of the user population will endeavour to manipulate the system so as to gain some advantage over other participants [67, 88, 102, 110, 160, 170, 178, 192, 193]. With their roots in the military and academic fields, the majority of systems we will survey tend to presume that participants are honest and will not cheat [57, 143]. As the military is able (in principle) to maintain somewhat tight controls on the computing hardware of the participants, these systems assume that participants are constrained to the rules dictated by the environment. Instead, the security considerations in these domains are primarily concerned with preventing the updates reported about an avatar from leaking sensitive information about any real-world entity that avatar may represent (for instance, the performance characteristics of a new fighter plane) to other participants (who might represent a foreign military force or competing industrial group).

  In contrast, the commercial gaming domain is primarily concerned with preventing a subset of users or an electronic attacker from spoiling the gaming experience for paying customers, either by performing actions outside the bounds of allowed game-play, or by preventing a paying user from safely accessing the game environment. This can directly jeopardize a games revenue, as customers tend to abandon games where cheating is common and not controlled [48].

- ***A consumer-friendly cost of entry:*** Computer and network resources are presumed expensive. Network bandwidth and computing hardware is costly to the operator of the game; and, the average user has moderate computer hardware and network connectivity. With the rapid transference of high-end technology into the consumer-space, the terms expensive", "costly" and "moderate" are deliberately non-specific. The point to be taken is that that target user is a "typical" home user, not a military or academic entity having access to facilities beyond the reach of the average consumer.

The goal of the massively multiplayer online game developer is to use the techniques and technologies at her disposal to achieve not only a game system meeting these characteristics, but to also provide an environment that is richly detailed, highly interactive, and "fun" for the user. For the purposes of this discussion, we further consider that the MMOG developer desires to field the environment as a commercial service offering; therefore, the resulting game system must be amenable to sustainable operations by a commercial MMOG service provider.

## A Historical Perspective

The genesis of massively multiplayer online games is generally accepted to have begun with the development of multi-user dungeons (MUDs) by Roy Trubshar and Richard Bartle in 1979 [36, 67, 76, 108, 186]. This text based game and the countless variations that followed it allowed thousands of players to congregate on their favourite servers to explore dungeons and kill monsters together in a Dungeons & Dragons [9] style adventure. These games ran, by and large, on mainframe and mini-computers until well into the 1990s and used a single world database located in shared memory or on disk [9, 36]. The various incarnations of the MUD source code have, over the years, been rewritten several times and migrated to the C and C++ programming languages for the UNIX and Linux platforms; MUD service providers still enjoy a small but loyal following [186].

By the mid-to-late 1980s much of the technology was in place to offer graphical interfaces to multiplayer online games. In 1985, for example, Randy Farmer and Chip Morningstar created Habitat, a virtual online world that Commodore-64 users could access through QuantumLink; similarly, in the late 1980s QuantumLink, Sierra Online and Genie offered Rabbitjack's Casino, Yserbius and Air Warrior on their proprietary networks [108]. These games supported hundreds of simultaneous players at a time when the average multiplayer game supported up to 16.

The early 1990s brought other large scale multiplayer games like Neverwinter Nights on America Online's (AOL's) proprietary network [84]. The first Internet based online-game, and the first game to use massively multiplayer as a marketing term, was 3DO's Meridian 59 [108, 142] which was released in 1996. Essentially the first fully graphical MUD, Meridian 59 established many of the game-play and business model trends that continue in MMO gaming today. For example, the in-game systems used for chat and character customization, as well as the flat fee pricing model were industry firsts which have become industry norms.

In 1997, massively multiplayer online games entered the mainstream in the western world with Origin Systems' release of Ultima Online [108, 148]. With a user population that has, at times, exceeded two hundred thousand simultaneous customers [109], Ultima Online was the first game to demonstrate the market potential for MMO games. Indeed, at the time of this writing, nearly seven years after its original release, Ultima Online continues to operate in a market now crowded with MMO games.

Outside of North America, particularly in Korea and Taiwan, the success of massively multiplayer games has been even more impressive. In 1996, around the same time 3DO was releasing Meridian 59 in the United States, Nexon released Kingdom of the Winds [144] in Korea, which went on to attract over one million subscribers [109]. In 1998 another Korean publisher, NCsoft, released Lineage [141] which now enjoys a worldwide subscriber base approaching four million users [109]. In comparison, Sony Online Entertainment’s Everquest [179], the most popular MMO game in the west is approaching five hundred thousand subscribers worldwide [109].

The success of these titles paved the way for the development of more recent titles such as the Asheron’s Call [130, 187] series by Microsoft/Turbine, Dark Age of Camelot [139] by Mythic Entertainment, Star Wars Galaxies [121] by Sony/LucasArts, The Sims Online [56] by Electronic Arts, Anarchy Online [71] by Funcom, and Shadowbane [196] by Wolfpack/Ubisoft. Imminent on the horizon are Worlds of Warcraft [18] by Blizzard, Middle-Earth Online [188] by Turbine/Vivendi, The Matrix Online [135] by Monolith/Warner-Bros, and many others. A quick search on 22 February 2004 in the Multiplayer Online Games Directory [138] returned one hundred eighty-two (182) massively multiplayer titles currently operating or in development.

An extremely interesting question, which is beyond the scope of this paper, considers the size of the world-wide market for MMOGs. Game industry analysts estimate that the online gaming market can support twenty or fewer massively multiplayer titles over the next few years [51]. Unlike traditional single and multiplayer games, the average consumer will not subscribe to more than one MMOG at a time [67] due to the investment of both time and money required. Additionally, the costs and complexity involved in launching and subsequently operating MMOG titles make them one of the most expensive interactive entertainment product offerings a company can add to its portfolio [51].

# The Technical Challenges

Massively multiplayer online games are difficult to design and implement. Computer Games in general, and networked computer games in particular, are a complex combination of problems from many domains of computer science and engineering: three-dimensional graphics, concurrent systems, real-time systems, distributed systems, human user interaction, physics, artificial intelligence, graph theory, database systems, digital signal processing for audio, data compression, electronic security, and cryptographic protocols are all tools employed by the MMOG developer. This chapter outlines the many technical challenges faced by the developer of a massively multiplayer online game.

## Network Bandwidth

Massively multiplayer online games utilize the network to exchange information about the virtual world between participating nodes. The more dynamically rich the environment, the more information is required to inform participants about the changes in the world. Additionally, as the size of the participant set increases, the total amount of information generated by the system increases. Similarly, increasingly rich environments with increasingly large user populations result in an increasingly large volume of information which must be conveyed to each participant. For a commercial service offering, we assume that each participant has a finite bandwidth capacity, typically that of a consumer-grade high-speed modem or broadband connection. Further, we presume that the bandwidth available to the massively multiplayer online game operator is not only finite, but charged to the operator based on usage. The MMOG developer is thus challenged to limit the bandwidth consumption of the game while providing the richest experience possible.

Military and academic simulation builders also face bandwidth issues. Unlike commercial game providers, however, the cost of bandwidth is generally not a consideration for their projects; instead, their challenge is to remain within the physical bandwidth limitations of their local and wide area network resources.

### Network Latency

Network Latency is the time taken to transmit a message from one node to another and represents one of the largest challenges facing massively multiplayer online game developers [176]. When one network application transmits data over a network, the network's latency determines when the data becomes available for consumption by the receiving application. As players expect to interact with each other and with the environment in real-time [6, 67, 82, 118, 149, 152, 176], the MMOG developer wishes to give the user the illusion that the entire game world is located on their local machine [176]. Network latency means that incoming data about the state of the game environment is already somewhat out-of-date by the time it arrives for processing.

Unfortunately for the MMOG developer, latency is an inherent part of network communications [30]. Electronic signals take some non-zero amount of time to travel from one location to another. A light-based signal traveling on a fibre-optic link, for instance, would take over 21 milliseconds to travel from the western to eastern coast of the United States [30]. On top of this fundamental delay, every piece of network hardware (i.e., routers, gateways, modems and network adapters) through which the signal passes has its own capacity and latency for the processing of network traffic. It further takes time for an operating system to take data from an application and deliver it to the network hardware for transmission, and vice versa for delivery of data to an application. Thus, network latency is an unavoidable consideration for MMOG developers.

The issue is further complicated by the variability of network latency across the Internet. Latency varies with the physical distance traveled by the data, the current load on each piece of network hardware, the current processing load on the operating system, and other factors. These factors are beyond the control of the game service provider. Fortunately, the military and academic communities have developed techniques to compensate for, and hide the effects of, network latency. For the most part, the gaming community has already appropriated these techniques into its repertoire.

### Distributed Consistency

Latency and distributed communications make it difficult to maintain consistent world interpretations for each participant. The MMOG developer wishes to present users with a single shared environment, which is challenging if the messages communicating changes in the environment arrive at different times at each node. Further, if participants act based on differing views of the environment, some means must be provided to determine which resulting version of the environment is correct. Players expect consistency in areas such as player-to-player and player-to-environment collision detection and in-game economic transactions (if applicable) [67]. A lack of consistency hampers the level of immersion of the player, taking away from their enjoyment of the game and decreasing their desire to play. Fortunately, the military, academic, and gaming communities have all found that, for many types of interactions, human players are often satisfied given the appearance of consistency.

Overall, maintaining consistency in an online game has proven to be a difficult task for game developers; players commonly uncover defects that violate the consistency of the game world, many of which may be exploited to gain an unfair advantage over other players [12, 23, 75, 101, 110, 136 2002, 145, 160, 171, 178, 193, 197]. These consistency violations tend to stem from implementation or design errors. Achieving true consistency in a distributed environment is often quite complex, having subtle failure modes and timing or ordering considerations.

## Fault Tolerance

A massively multiplayer online game system should insulate its users from the failure of one or more of its components. Ideally, failure and failure recover are transparent to the user population. User visible failures can be expensive not only in with respect to the cost of replacing the failed component(s), but also in terms of the reputation and image of the game, customer service and support burdens, and, ultimately, user satisfaction [29].

As in any distributed system, there are a number of areas in which failure can occur. In this section we consider random failures in network or computer equipment. For the moment, we assume that software components operate correctly in accordance to their specifications given correctly operating network and computing resources. A more in-depth classification of failure modes and stability can be found in [61].

### Network Failure

Computer networks, particularly the Internet, are subject to myriad failure conditions. For example, network messages may not be delivered if network connectivity is lost due to power failure, damaged communication lines, faulty network equipment, overloaded telecommunications systems due to peak usage, or any number of other random failure causes. An MMOG provider should take measures to ensure that network failures are unable to prevent a substantial portion of the paying customer base from using the game environment. Clearly, it is impractical, if not impossible, to guarantee that no permutation of network failures prevents any customer from using the game environment. Rather, a MMOG considers the likeliness and consequences of failure and fields a network system in which the most likely anticipated failure modes affect the least number of customers.

For most network systems, both in and out of the gaming community, resilience to network failure is accomplished with redundant network connections, redundant network equipment with failover, and disaster recovery plans. The provision and maintenance of robust network systems represents an established problem that organizations deal with on a daily basis and is not directly related to the software architecture for a massively multiplayer game. As such, this topic falls outside the scope of this work and will not be discussed further.

### Hardware Failure

A component may also fail due to hardware issues in the computing environment. Hardware failure may be an underlying cause of one of the network failures mentioned above; however, for the purposes of this discussion, we consider hardware failures in the computing resources used by the MMOG developer to provide game services to its customers. This includes game and database servers which may operate incorrectly due to failure of a power supply, hard disk, or other electronic component. The distribution of game tasks and responsibilities should be such that, at worst, the loss of a server due to a hardware failure affects only a localized subset of players and, at best, the loss and recovery of a server is entirely transparent to players.

### Participant Failure

The computing device of a player may fail for any number of reasons. As in military and academic simulation, the massively multiplayer game developer seeks to isolate the effects of participant failure such that other players are not affected. In an MMOG where player state is persistent, the developer also wants to minimize the effects on the players who experience the failure. For example, a player will be unhappy to subsequently resume playing the game and find that their character's inventory has inexplicably gone missing. Game systems lacking built-in mechanisms for recovering from participant failure may find their customer service representatives spending significant portions of their time searching game event logs to determine whether or not a user's claim to have lost items due to disconnection are valid or fabricated [29].

### Administration and Live Production

Unlike most types of commercial game or software products, the delivery of a massively multiplayer online game does not finish when the boxed software hits store shelves or the package is made available to customers for download [43, 67, 170]. Beyond its release to the public, a commercial MMOG is ultimately a 24x7 (i.e., continuous) live service offering; as a product it will require continual administration, new features, additional content, and customer support. Ideally, all of this will be provided without user-visible interruption of service. Thus, it behoves the MMOG developer to build a system which, in addition to all of the other technical requirements, is easy and inexpensive to maintain, administer, and extend in a live environment. Generally, for military and academic simulation systems these ongoing requirements are not present, as the participants come together at some established time to launch the environment, execute the training or demonstration exercise, and shut-down the environment.

## Cheating

Cheating is widespread in multiplayer games [12, 13, 35, 48, 67, 88, 102, 110, 160, 170, 178, 192, 193] and can have disastrous consequences for an organization providing massively multiplayer online game service. Most cheating occurs in order to gain some undue advantage for the cheating participant [48, 153]. As mentioned previously, honest players tend to leave a game in which they perceive themselves to be at a disadvantage due to widespread cheating [48]. A much smaller group of players, commonly referred to as griefers, will cheat and engage in other anti-social behaviour purely for the pleasure of it. Representing only 3% of the average multiplayer game user population, MMOG providers have found that their customer support representatives spend upwards of 40% of their time dealing with griefer related issues [153].

## Security of Customer Information

As a part of operating a commercial service, a massively multiplayer online game provider will often require personal or financial information about its users. For example, the user’s name, address, and credit card information might be required for billing purposes. In turn, the users of an

MMOG service expect that any personal or financial information they entrust to the service provider will be handled in such a way as to protect the privacy and confidentiality of the user [67]. Indeed, depending on the location in which the game provider operates, there may be a legal obligation to ensure the privacy of personal and financial information collected about its users. Many European countries, for instance, have had progressive privacy and data protection laws in place for some time. More recently in Canada, the Personal Information Protection and Electronic Documents Act (PIPEDA) [79] governs the collection, use, and disclosure of personal information for national, international, or inter-provincial commercial activities and, in the United States, the Children's Online Privacy Protection Act (COPPA) [59] defines the privacy framework with which a service provider must comply when offering internet-based services to children.

While a philosophical and technical discussion around the use and protection of customer and financial information would be quite interesting, it is beyond the scope of this work and will not be discussed further.

### Scalability

Ultimately, the challenge for the massively multiplayer online game developer is to design and implement a system that addresses each of the other technical challenges in the presence of large numbers of concurrent users. The MMOG developer seeks to partition and distribute computational tasks and game state such that (1) each participant is aware of any and all game information which is relevant to that participant; (2) each computing device participating in the game is not over-burdened; (3) the capacity of the network resources available to the game is not exceeded; and (4) the size of the concurrent user set, and of the virtual environment, can be increased without requiring architectural changes to the game.

# Literature Review

Massively-multiplayer gaming systems are one particular example of a more general application model, the Networked Virtual Environment (net-VE). A net-VE is defined [176] as “a software system in which multiple users interact with each other in real-time, even though those users may be located around the world.” This chapter provides an historical overview of networked virtual environment technologies from the military, academic, and commercial gaming fields; in each exploring the most significant systems developed.

## Military Modeling and Simulation

The United States Department of Defence (DOD) spends in excess of $1.5 billion per year on modeling and simulation for a variety of purposes, such as to train individual soldiers, conduct joint training operations, develop doctrine and tactics, formulate operational plans, assess war-fighting situations, evaluate new or upgraded systems, and analyze alternative force structures [32]. The features and technologies of many of these systems are strikingly similar to those found in massively-multiplayer online gaming. In particular, many of the technical challenges facing developers of massively-multiplayer games today have been encountered in one form or another by the military training community [128]. Indeed, the DOD was the first to do work on large scale networked virtual environments [176] and is thus a rich source of information for today’s game developer.

This section provides a summary of the most significant military networked virtual environment based training systems and their potential contribution to massively multiplayer online gaming. Specifically, we will explore Simulator Networking (SIMNET), Distributed Interactive Simulation (DIS), Aggregate Level Simulation Protocol (ALSP), and the High-Level Architecture (HLA) for Modeling and Simulation.

## SIMNET – Simulator Networking

Originally developed for the Defence Advanced Research Projects Agency (DARPA) by Bold, Beranek and Newman (BBN), Perceptronics, and Delta Graphics from 1983 through 1990 [154], SIMNET is a distributed simulation system intended to provide a “low-cost” net-VE for training military units (tanks, helicopters, command posts, etc.) in co-ordinated combat and assault tactics [128, 176]. The SIMNET network software architecture has three core elements: (1) an object-event architecture; (2) autonomous simulation nodes; and, (3) a set of predictive modeling algorithms [176].

### Object-Event Architecture

The object-event architecture models the world as a collection of objects which interact by means of events. Objects are used to represent vehicles (a helicopter, for example), munitions (a missile, for instance) and other entities within the environment. An event is a message broadcast to the simulation network indicating a change in world or object state (for instance, the explosion of said missile upon striking the aforementioned helicopter).

### Peer-to-Peer / Replicated Data Model

Each simulation node maintains its own repository of information describing the state of every object in the virtual environment and is autonomously responsible for maintaining the state of one or more objects in the virtual world (Figure 1). Object responsibility entails placing event messages onto the network such that the current state of the object is accurately represented to the other nodes participating in the environment. It also entails processing received event messages as a part of calculating the controlled objects' new state. Beyond transmitting and receiving event messages, nodes do not interact with each-other or with the net-VE. The use of autonomous nodes, each maintaining its own world representation, means that there is no central server, nor central point of failure. It also allows nodes to enter and leave the simulation at any time.

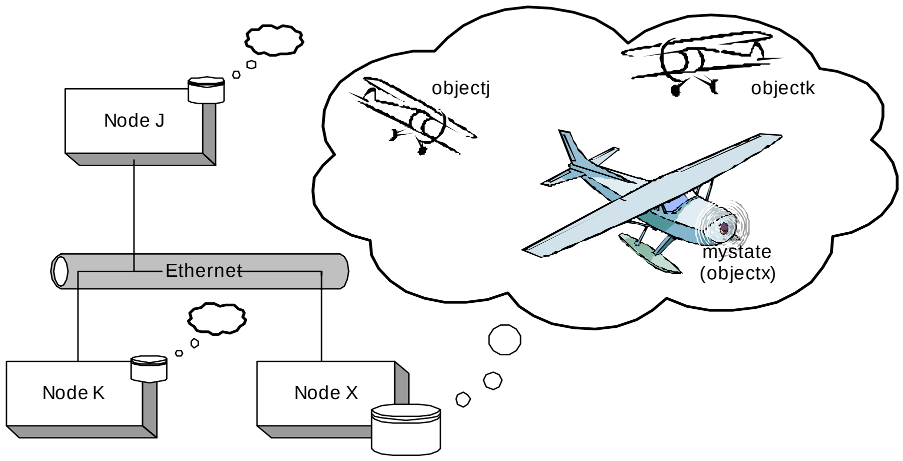

**Figure 1. SIMNET Fully Replicated Data Model.**

## Weakly-Consistent Data Model / Dead-Reckoning

Perhaps the most important design concept for the massively multiplayer on-line game developer is the Weakly-Consistent Data Model [128]. There exists a broad set of game and simulation scenarios in which no participant ever requires a complete and fully accurate real-time representation of the shared state. Rather, in between updates for a given object, each participant may extrapolate some of the object's state using a set of predictive algorithms collectively referred to as dead-reckoning [133], or more generally as predictive contracts [127]. In order to reduce the number of messages transmitted, hide the effects of message latency, and alleviate the effects of lost messages, the developers of SIMNET introduced the objects and ghosts paradigm. In this model, the node responsible for an object (its home node) is the only node having the authoritative version of the object's state; all other nodes maintain a ghost copy of the object and track the rate of change of the object's state (Figure 2). The home node also tracks the rate of change in the object's state. An update event is only broadcast when the predictive model for the object's state differs from the authoritative model for the object's state by some pre-established threshold. When an update is received for a ghost object, the object's state is corrected to the new values and dead-reckoning begins again. A node also transmits heartbeat messages periodically to inform other simulation participants that it is still actively involved in the simulation. Typically, dead-reckoning is applied to positional information about an object, such as location and orientation, but the concept can be applied to any state whose values and changes in values exhibit continuity (i.e., no sudden jumps or gaps) over time.

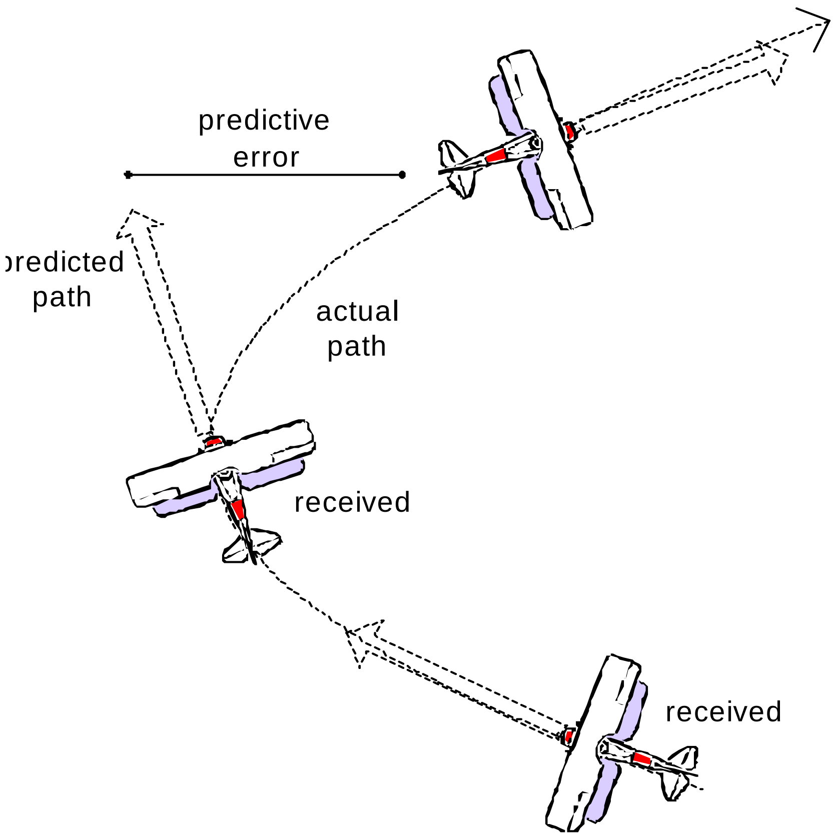

**Figure 2. A Dead Reckoning Example.**

The ideas embodied by dead-reckoning have been widely adopted by the training simulation community as well as the network gaming community. Dead-reckoning protocols trade accuracy of shared state for reduced network traffic, thus allowing a networked virtual environment to support more participants [176]. Dead-reckoning protocols generally consist of a prediction technique and a convergence scheme. The most popular prediction technique employs derivative polynomials (i.e., based on the current value, rate of change and/or acceleration in the rate of change for a state variable) to approximate the current state of a remote entity, usually by transmitting the instantaneous rates of change for the value. For example, a first-order derivative polynomial scheme for approximating object position $p'$ would predict, given the object's position $p_0$, velocity $v_0$, and acceleration $a_0$ at some time $t$, that the position after time lapse $\Delta t$ would be $p' = p_0 + v_0 \cdot \Delta t$. A second order derivative polynomial scheme would further consider the acceleration of the object and predict that $p' = p_0 + v_0 \cdot \Delta t + \frac{1}{2} a_0 \cdot \Delta t^2$.

A convergence scheme is required in order to adjust the current approximation to more correctly represent a newly received update. The simplest form of convergence simply updates the local representation to match the updated values, possibly resulting in discontinuity of the local state. Linear convergence involves the selection of a future convergence point and then linearly adjusting the local state over time in the direction of the convergence vector. For example, the correction of the position of the aircraft in Figure 2 might follow a convergence path along the line indicated by the prediction error. More sophisticated approaches involve correcting the local approximation along quadratic, cubic, quaternion or spline based convergence vectors.

Position-History Based Dead-reckoning [174], or PHBDR, an extension to the derivative polynomial approach, chooses between first and second order polynomials based on the objects state over previous updates. Rather than transmitting instantaneous values for the rates of change, the rate of change is derived from the previous observed states and an approximation formula chosen based on threshold values in the rates at which the value appears to be changing. Similarly, the convergence algorithm used to correct approximation errors is dynamically selected based on the significance of the predictive error.

Auto-adaptive dead-reckoning [24] further extends these ideas by varying the rate at which updates are transmitted to an observer based on the relationship between the object and its observer. For example, an observer that is far away from an object may have lower accuracy requirements for the approximated state than an observer that is close to the observed object. Based on threshold values describing the relationship between object and observer, a different error tolerance value may be used to determine when an update should be transmitted, serving to lower the rate at which updates are transmitted over the network to “distant” observers.

More generally, a predictive contract [127] is any specification for deriving a reasonable estimate of an object’s state over time. Consumers of a predictive contract may extrapolate the current state of an object based on its initial state and the predictive contract. Dead-reckoning techniques are specific examples of predictive contracts. Other predictive contracts would include statements like:

- traveling along waypoints A, B, and C
- flying in Delta formation
- following an arc having radius R with velocity V

The network gaming community has found dead-reckoning to be a very useful technique to appropriate from the military simulation community [6, 21, 87, 128, 129, 152]. Evaluations of dead-reckoning techniques in computer games have found them able to reduce the impact of network transmission delay across a variety of game genres [150] and, when combined with synchronized clocks across hosts, to provide a fairly accurate approximation of the state of remote objects [4]. In spite of the computational cost of prediction as well as the additional implementation complexity, dead-reckoning has been successfully applied to first person shooters [15], flight/space simulators [118], racing games [150], sports games [150], and massively-multiplayer online games [21, 112, 114]. The use of other types of predictive contracts has not yet been adopted by the game development community. When queried by the author, the participants of a network gaming roundtable [195] at the 2004 Game Developer's conference were largely unaware of predictive contracts; those developers who were aware of predictive contracts, came from the military simulation domain.

Dead-reckoning and predictive contracts are not without drawbacks. It is important to note that a weakly-consistent data model is vulnerable to time-related cheating [12, 13, 35, 48]. A dishonest participant can pretend to have greater latency in order to see the actions of other players before sending out their own updates, with timestamps set in the past. This effectively, allows a cheater to see slightly into the future. A cheater could alternatively elect not to send updates as regularly as the game expects; instead, the cheater deliberately drops as many updates as tolerated by the game (we presume that after some threshold of dropped updates the game will determine that the participant is no longer playing) before sending an update that accords the cheater some advantage. As long as the cheating party periodically sends plausible updates, one cannot discern a cheater from an honest player having a poor network connection. A recently suggested approach to securing dead-reckoning is to introduce delay in the processing of messages until all participants have committed to their choice of action for the current time-slice of the game using a cryptographic hash [35]. The delay with which updates are applied to the world representation is managed as a function of the observed latencies of each player. In order to avoid stalling the game, players do not transmit their moves in advance. Game state is updated by dead-reckoning if players must wait for a participant to commit to their choice of action. The effect of this approach on the perceived responsiveness of a game to the user's commands is not clear.

### SIMNET Shortcomings

Several aspects of SIMNET render it an inappropriate technology on which to base a massively multiplayer online game. Firstly, as a development of the United States Military, SIMNET defines an overly specific and specialized set of message packets [124, 176]; the introduction of new types of simulation entities is not directly supported. Secondly, the SIMNET network protocol is explicitly tied to Ethernet multicast and requires network bridges for simulations which span Ethernet LANs, at the cost of latency [124]. Thirdly, SIMNET was not developed as an open or interoperable solution; it is insufficiently documented for easy use by parties outside of those involved in the original project [176]. These drawbacks made SIMNET less than ideal for a long-lived military standard for networked virtual environments.

SIMNET also has some deficiencies that, while acceptable for the military simulation community, deter from its usefulness as an MMOG platform. The level of trust accorded to each autonomous simulation node poses a security risk in a consumer environment, where fair-play cannot be mandated by policy or oath, and customer retention is jeopardized by cheating participants. For example, a dishonest node may grant unfair capabilities to objects under its control, teleporting vehicles or ignoring requests to apply damage from enemy munitions. Alternatively, a node may make unfair use of the information at its disposal, requesting that enemy units apply damage when they have not been hit, or making use of positional knowledge of enemy units that should not be visible to the player.

The network communications model and platform of SIMNET, Ethernet multicast to all simulation participants, is not appropriate for massively multiplayer online games. SIMNET effectively broadcasts all events to the entire simulation network, regardless of the relevance of a particular event to a given node. With this model, the SIMNET network software architecture proved to be somewhat scalable (a five site simulation involving some 850 objects was conducted in 1990 [133]), but not to the level required for an MMOG.

### DIS – Distributed Interactive Simulation

Attempting to overcome the shortcomings of SIMNET, in 1989 the United Stated Department of Defence and supporting industries initiated a process to formalize and extend the SIMNET protocols to support a wider range of military small unit simulation requirements. These efforts culminated in the ratification of the IEEE Distributed Interactive Simulation (DIS) standard in 1993 [90, 91, 92, 93, 94, 95].

The DIS network software architecture has the same basic components and structure as SIMNET but is designed to support fully distributed heterogeneous simulations having fewer than three hundred (300) participants over a UDP/IP local area network (LAN) [124, 176]. For the purposes of this discussion, the core differences between DIS and SIMNET are: (1) a larger set of predefined simulation entities, the DIS terminology for SIMNET objects; (2) a larger suite of message definitions, including an entity specific “data” message; and (3) the use of UDP/IP broadcast instead of Ethernet multicast as a network transport facility.

DIS was well received and supported by the training community; over one hundred DIS compatible systems were developed [176]. DIS was capable of effectively supporting up to the design target of 300 participants; however, DIS so successfully met its DOD objectives that it inspired a demand from the training community for simulations supporting a larger number of entities. Extensions to the DIS protocol and hierarchical hardware architectures were introduced in the late 1990 for the Synthetic Theater of War (STOW) [25] and Joint Precision Strike Demonstration (JPSD) [106] exercises, which scaled to support 3000 and 30000 entities, respectively.

One approach for extending the scalability of DIS across LAN boundaries for the STOW and JPSD exercises was to introduce gateway servers to act as the interface between the LAN and a wide area network (WAN). For the STOW exercises, the Application Gateway (AG) serves as a bridge between DIS participant sites, allowing them to operate as a single DIS session. Tasks performed by an AG include packet aggregation, delta compression, relevance filtering [25, 124, 165, 190], each of which is discussed below. For the JPSD exercises, the Run-Time Gateway (RGW) bridges simulation sites using interest management (IM) to ensure that each simulation receives relevant information [156]. JPSD is further discussed in the section on relevance filtering.

### Packet Aggregation

In order to reduce the number of message packets transmitted across the WAN an application gateway would bundle multiple protocol data units (PDUs) from the subnet into larger message packets before transmission. In the STOW environment, this technique was specifically useful in order to accommodate the Ethernet frame size and rate requirements of the Network Encryption System (NES) used to secure communications across the WAN [124]; however, packet aggregation is a useful technique for reducing bandwidth requirements even in the absence of the constraints introduced by the NES.

The user datagram protocol (UDP) [155] and transmission control protocol (TCP) [46] of the internet protocol (IP) suite [47] each have a header of twenty-eight (28) and forty (40) bytes respectively.  For networked virtual environments having traffic composed of frequent small messages, a large portion of the required bandwidth may be devoted to header information for the underlying network protocol.  Merging messages into a single packet eliminates packet headers.  Depending on the environment and the amount of data contained in each packet, aggregation may eliminate up to 50% of the bandwidth requirements for a networked virtual environment [176].   Clearly it pays to transmit as much information per packet as possible, to reduce the amount of bandwidth wasted on network protocol overhead.

Packet aggregation often requires a trade-off between bandwidth and timeliness. The transmission of a data message may be delayed until sufficient messages can be aggregated, thus increasing the latency of message delivery. Additionally, delayed data may lose its relevancy to the receiver, or even be superseded by a subsequent update before being transmitted. In the ideal case, large sets of update messages become available to the transmitting host at some regular interval, making packet aggregation a rather trivial process. More generally, updates become available in an unpredictable manner and some transmission policy for packet aggregation manages the trade-off between message delay and message size [175]. For example, in a time-based transmission policy a packet may be sent after some fixed period following the availability of the first update message; this provides an upper bound on the delay introduced by the packet aggregator. In a quorum-based transmission policy a packet is transmitted when it contains some minimum number of updates; this ensures a particular reduction in bandwidth and packet rates. One could also employ a hybrid approach where a packet is sent when the period expires or when the quorum is reached, whichever comes first; this approach can be used to dynamically balance the aggregation trade-off in response to object update rates.

Packet aggregation is particularly suited to client/server network architectures in which a game server host is responsible for multiple game objects, or large portions of the game environment. Indeed, developers of client/server multiplayer games typically have the server send out aggregate state update messages to the clients rather than sending out an update message per object [20, 86, 87, 118, 146, 178].

### Delta Compression

In order to decrease the number of bytes required to transmit data from one application gateway to another, each message undergoes delta compression using the Protocol Independent Compression Algorithm (PICA) [190]. PICA, and other delta compression schemes, exploits the observation that much of the data describing an entity’s state changes infrequently. Rather than always sending the full state of an entity, it is often sufficient to regularly transmit only the portions of the state which have changed, and less frequently retransmit the full state. PICA is “protocol independent” because it performs this compression at the bit level, removing redundant bits from the previously transmitted state representation, without any regard for the semantics or structure of the message content. In addition to PICA, each application gateway also offers a Quiescent Entity Service (QES) which detects and tracks stationary simulation entities. The QES informs other application gateways that an entities state is no longer changing and that heartbeat updates for that entity will not be transmitted over the WAN; based on the last known state of that entity, each AG can then generate the appropriate heartbeat updates for its local subnet.

PICA requires a mechanism for reliable message delivery, as each message represents the change in state from the previous message. The lost of a message introduces errors in the derived local state until the next full state update is received. The magnitude of the error is compounded if packet aggregation is further employed. One approach is for the recipient of a delta compressed message to detect a lost packet (usually by assigning a sequence number to each packet) and issue a retransmit request, or negative acknowledgement (NAK) to the sender [124, 176]. In the worst case, a lost packet results in a NAK from many other application gateways, leading to network congestion and more lost packets [124].

The Quake 3 network model [87] demonstrates a technique for applying delta compression to purely unreliable protocols. Rather than waiting for a NAK from a client, the game server continuously sends out delta compressed state from the last positively acknowledged state. This approach transparently handles packet loss at the expense of more frequent message transmission. Because a client's positive acknowledgement message is also delivered unreliably, it further requires each client to cache all past states later than that on which the most recently received update is based.

### Relevance Filtering

In order to decrease the number of messages processed by each host on a LAN, its application gateway discards updates which are not relevant to the entities it serves. An AG uses Grid Filtering to discard updates for entities which are outside of the geographic area covered by its local subnet. An AG also applies Culling to discard updates for entity types which are not of interest to its local subnet; for example, updates for seaborne vessels may be discarded by an application gateway serving a subnet composed of ground vehicles [124].

The designers of the JPSD simulation architecture modified the DIS implementation of participating simulators to broadcast their updates on a local multicast channel and introduced the run-time gateway to manage inter-simulation messaging on behalf of its clients [156]. Each RGW, based on a priori knowledge of the characteristics of its clients distributes interest expressions to all of the other RGW hosts. When an update is received from a client simulation, its RGW reads the update and forwards it to all other RGW nodes having an interest expression which is satisfied by the contents of the update.

### DIS Shortcomings

Like SIMNET, DIS was developed for small unit training and is not suitable as a massively multiplayer online game platform. The sizes of DIS messages are slightly larger than those for SIMNET, requiring more network bandwidth per message. In order for the simulation to be completely decentralized and nodes fully autonomous, DIS requires that every entity maintain its own complete representation of the virtual world which, in turn, requires that every message must be received and processed by every participant. DIS further requires that all entities periodically broadcast their entire state for the benefit of new participants, as well as still-alive messages if their update rate slows. Given these requirements, a simulation having 100,000 players would have network connectivity requirements of up to 375 Mbit/s of bandwidth to each player [120], a figure well in excess of modern broadband access. This is in addition to the large computational and space requirements for maintaining the state of each entity by dead-reckoning.

The decentralized nature of DIS also has the same MMOG security and trust issues found in SIMNET. For example, the anecdotal Mega-Death program cheated by collecting the positions of enemy players and then simultaneously issuing a detonation event for powerful munitions next to each one [176]. The inability to detect or prevent cheating, when combined with unreliable UDP/IP broadcast, has led to disputes in DIS exercises where a player that should have “died” turns around to shoot the player that just “killed” him [176].

### ALSP – Aggregate Level Simulation Protocol

The operational goal for the STOW program was the ability to run real-time simulations having 100,000 to 300,000 participants, a value on the order of the size of Operation Desert Storm in the early 1990s [176]. Therefore, in early 1990, DARPA sponsored MITRE to explore the design of a general interface by which large simulation exercises, such as those based on DIS, could cooperate, increasing the functionality and value of the existing combat simulations developed by various training organizations [194]. This effort resulted in the development of the Aggregate Level Simulation Protocol (ALSP) [105] which uses synchronization algorithms from the analytic simulation community [69] to join multiple independent simulations to form a larger, seamless simulation covering a larger scope than intended or possible by the original simulation designs [194]. Using the ALSP to aggregate platform-level entities into simplistic groups of forces, the Joint Training Confederation and similar groups commonly simulate battlefields with 20,000 to 100,000 entities [128].

While the ALSP is not directly applicable to massively multiplayer real-time gaming, it does provide some ideas which may be helpful to MMOG developers. In particular, it introduced an infrastructure layer into the distributed simulation framework. This common software component, composed of a translator and gateway layer, is responsible for creating and transmitting update events, receiving updates and modifying local ghost objects, controlling local simulation time, and converting between the internal and external representations of entity attribute values.

### Time Management

That ALSP provided a test platform for many of the ideas that would later become part of the High Level Architecture for Modeling and Simulation. The time management services of the ALSP and of the HLA are a prime example of this. The time management system of the ALSP utilized a conservative execution model based on the Chandy-Misra time synchronization algorithm [28, 194] to manage the advance of simulation time between the concurrent processes participating in the distributed simulation. Conservative execution algorithms prevent inconsistency by having each simulator advance its simulation clock only when it can safely assert that no unknown action taken by another entity could possibly cause its perceived state to differ from the correct state. In essence, participants that interact with one another synchronize and execute the simulation in lock-step.

Optimistic execution algorithms such as Time Warp [103], on the other hand, allow participants to extrapolate the state of the environment; if the extrapolation proves to be invalid then some corrective measure is taken. In Time Warp algorithms, the simulation state and time revert to the last known valid state and time, possibly causing entities to move discontinuously. Further advances in time management are discussed in the section on the HLA Time Management service.

### Infrastructure

As a precursor to HLA Run-Time Infrastructure the Aggregate Level Simulation Protocol introduced a common software infrastructure facilitating the distributed interaction of simulators. The ALSP software component is composed of translators and gateways. Each translator handles the semantic conversion between the internal and external representations of simulation data and prevents the local simulator from advancing its simulation clock until it is safe to do so [194]. Each gateway is responsible for applying the Chandy-Misra time synchronization algorithm. Operating in a peer-to-peer manner, the gateways provide an event transport service which guarantees that no participating simulator will receive a late event. Gateways also allow simulations to join and resign from the simulator in a time coherent manner [194].

#### ALSP Shortcomings

The requirement to link numerous, possibly dissimilar, simulations into one aggregated system while maintaining the high-fidelity desired by the training community, introduced additional complexity into the management of time within the system. In the ALSP, time is not tied to wall-clock time. Instead, the system executes as fast as possible with the requirement that all events be processed in timestamp order by each participant. This incurs significant processing and network overhead, which may result in delayed time advancement in real-time modelling systems; conversely, it also means that hours to days of simulation time may pass in minutes to hours of wall-clock time, for faster than real-time modeling systems [128].

### HLA – High-Level Architecture

In 1995 the Defence Modeling and Simulation Office (DMSO) [55] initiated the High Level Architecture for Modeling and Simulation (HLA) project to define a general purpose simulation framework facilitating interoperability and reusability of large numbers of distributed heterogeneous simulators. In the HLA, an individual federate (simulator) or set of federates developed for one purpose can be combined with other federates and applied to some other purpose as a federation: a composable set of interacting simulations [41]. The HLA builds on the lessons learned from the previous real-time distributed simulation projects described above, as well as that of the analytical simulation community, to define a simulation model and common run-time infrastructure by which to aggregate diverse simulation applications. It extends the ambitions of the ALSP to not only aggregate simulators based on the traditional real-time and DIS models, but to simultaneously support other simulation models within the same simulation, particularly those based on discrete-event simulation formalisms. The HLA was accepted as an IEEE standard in 2000 [96, 97, 98, 99].

The high level architecture is formally defined by the federate interface specification [97], the object model template specification [98], and the HLA framework and rules specification [96]. For our purposes, the key innovation of the HLA is the introduction of a standardized Run-Time Infrastructure (RTI) for distributed simulation, defined in the federate interface specification. The HLA Object Model allows a structured specification of each federate, its attributes, and the overall federation to be published to all participants. However, the HLA object models are largely descriptive; the primary consumers of an HLA object model are the human developers attempting to integrate a federate into a federation. Finally, the HLA framework and rules specification describes the design principles, constraints, and requirements that a federate and federation must adhere to in order to be HLA compliant.

Functionally, the core elements of an HLA federation (Figure 3) are federates, the federate interfaces to the RTI, and the RTI itself [39, 40, 41, 42]. The remainder of this section describes the run-time infrastructure and the services it provides. It further discusses the applicability of the services and concepts of the RTI to massively multiplayer online gaming.

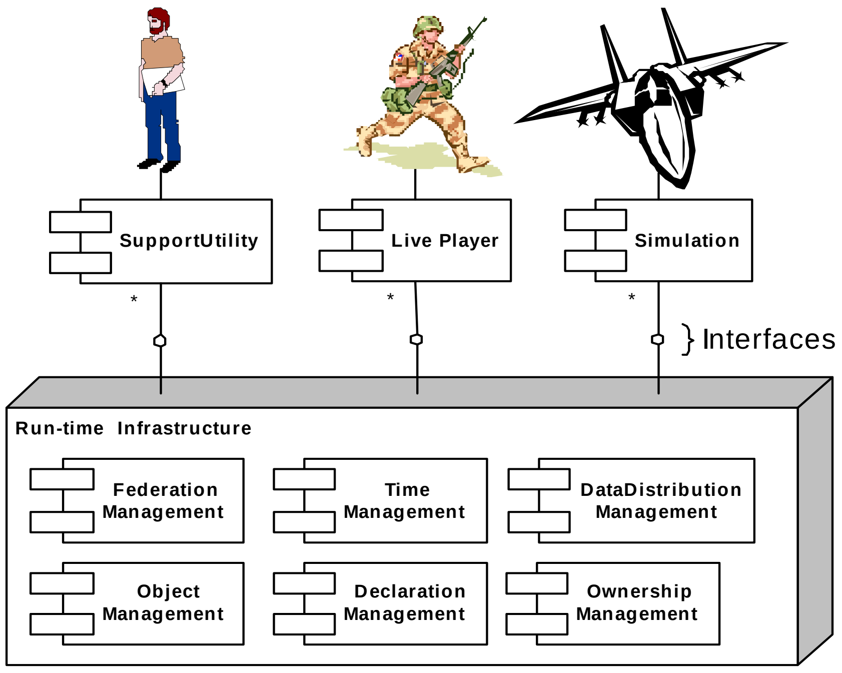

**Figure 3. Functional Components of the High Level Architecture.**

## Run-Time Infrastructure

The RTI provides six core service classes. The federation management service facilitates the creation and operation of a federation. The declaration management service allows a federate to specify the data it intends to produce and declare the data it expects to consume. The object management service is used to create, delete, identify, and otherwise manage simulation objects. The ownership management service allows one object to dynamically transfer control of attributes to another object during a federation execution. The time management service maintains temporal consistency throughout the distributed simulation. The data distribution management service provides efficient routing and exchange of messages between federates. The HLA federate interface specification [97] defines both the functionality provided by these services as well as the application programming interface (API) used by software developers to access these services.

### RTI – Federation Management

The Federation Management service facilitates the creation, dynamic control, modification and deletion of a federation execution [97]. This service manages the complete lifecycle of the federation and its federate members. It supports:

- the creation and initialization of the federation based on data found in the Federation Object Model (FOM) document data, or FDD [98];
- federates joining or leaving the federation;
- saving and restoring the state of a federation execution to/from persistent storage;
- the registration and management of synchronization points for federates and for the federation; and,
- the destruction of a federation execution.

The Federation Management facilities provide a pattern for system administration and service discovery that may be of interest to the massively multiplayer online game developer.

### RTI – Declaration Management

The Declaration Management (DM) service allows joined federates to declare the manner in which they intend to participate in the federation [97]. It further allows federates to discover the object instances, attributes, interactions, and service which are available within the federation. The DM service provides a pattern for providing data-driven and meta-data-driven (schema) distributed systems. Data-driven techniques and systems are becoming a standard part of the game developer's arsenal of tools and are a natural fit for massively multiplayer online game systems [115, 116, 167]. Data-driven techniques have been used by game developers from all game genres, from role-playing games to real-time strategy games and first person shooters, to provide flexibility and extensibility during development and to allow the user community to build their own game modifications, or mods [164].

### RTI – Object Management

The Object Management (OM) service provides a remote object interaction paradigm similar to that of Microsoft's Distributed Component Object Model (D-COM) [131] and the Common Object Request Broker Architecture (CORBA) [147]. These object-oriented models for distributed computation provide a layer of abstraction between the invocation of a procedure on an object and the physical location at which the object resides and at which the execution of the procedure takes place. They further provide facilities for object and service discovery. These models, particularly when object methods are defined as asynchronous, may be of value to an MMOG developer as it allows him or her to focus independently on the interaction of objects and the distribution of objects.

#### RTI – Data Distribution Management

The Data Distribution Management (DDM) facilitates the transfer and delivery of simulation updates between federates using a producer/consumer model [97]. The DDM service applies and extends the distribution techniques of previous military and academic simulations, making use of area-of-interest management, packet aggregation, hierarchical distribution, and various network topologies. The DDM service abstracts the notion of multicast, interest, and regions away from the network using the concept of routing spaces: a mechanism for expressing and applying interest to network updates [97, 128]. In the commercial game space, middleware companies are adapting these ideas for multiplayer and massively multiplayer online games [162].

#### RTI – Ownership Management

The Ownership Management service of the Run-Time Infrastructure allows joined federates and the RTI to transfer ownership of objects instances and instance attributes among joined federates [97]. This supports the cooperative modeling of an entity across multiple hosts. It further provides a means for the control of an object or an attribute to migrate from one host to another.  In the HLA a federate may divest its ownership to the RTI, allowing the RTI to assign ownership to any interested federate, or a set of federates may coordinate an exchange of ownership among themselves.

An ownership management service may be quite useful for a massively-multiplayer online game environment.  MMOG systems offering seamless worlds using client/server architectures will require a mechanism by which entities are transferred from one server to another as they move through the environment.  If the MMOG system provides support for the reassignment of ownership without the participation of the current owner, an ownership management service can also be used as a fault-tolerance measure; objects managed by a failed host can be transparently reassigned to another host.

### RTI - Time Management

Each federate maintains two local clocks.  Scaled-wallclock-time is used to synchronize federates execution with humans and live entities.  Logical-time, which is synonymous with simulated-time, is used to ensure that messages are delivered, and events occur, in the proper order.  The Time Management service provides a federation execution with mechanisms to order the delivery and application of messages throughout the federation execution [97].  In providing this service, the RTI extends the ASLP and its other predecessors to support [68, 70]:

- federates having different event ordering requirements (e.g., DIS and ALSP)
- federates using different time advance mechanisms (e.g., time-stepped or event driven)
- real-time, scaled real-time, and as-fast-as-possible simulators
- federates using conservative synchronization
- federates using optimistic synchronization
- federates having different event transportation requirements (e.g., reliable and best effort message delivery)
- federates having different message ordering and processing requirements (e.g., receive order, priority order, causal order, etc)

To meet these requirements, the time management service is responsible for coordinating the message transportation system (i.e., the Data Distribution Management, or DDM, system) of the federation and the time advance mechanisms employed by each federate. The message delivery capabilities of the DDM system are categorized according to (1) the reliability of message delivery and (2) message ordering [68]. With regard to reliability, best effort message delivery means the transportation system will attempt but not guarantee to deliver the message; while reliable message delivery means the transportation will utilize mechanisms such as retransmission to ensure that the message is delivered. These semantics are equivalent to those of User Datagram Protocol (UDP) [155] and the Reliable Data Protocol (RDP) [151, 191], respectively.

With regard to message ordering, the HLA DDM provides five schemes [68] by which messages will be delivered to a federate:

- ***Receive Order***. Incoming messages are placed into a first-in-first-out (FIFO) queue. The RTI thus delivers messages to the federate in the order in which they were received. This is the simplest ordering mechanisms and has the least latency.
- ***Priority Order***. Incoming messages are placed into a priority queue, where precedence is given to the message having the lowest time stamp. This mechanism is also quite simple and exhibits fairly low queuing latency.
- ***Causal Order***. This ordering guarantees that a federate receiving messages for two events E and F, where E causally precedes [113] F, will have the message for event E delivered to it before the message for event F. For example, event E might be the firing of a weapon and event F might be the subsequent destruction of the weapon's target.
- ***Causal and Total Order***. This ordering extends the Causal Order mechanism to additionally guarantee that messages for events having no causal relationship (i.e., messages pertaining to concurrent events) will be delivered to all federates in the same order. That is, for concurrent events P and Q, this ordering guarantees that all federates will observe P before Q or all federates will observe Q before P.

- ***Time Stamp Order***. This message delivery order guarantees that messages are delivered to the federate fully ordered based on their time stamp. As a consequence, the RTI further ensures, using conservative synchronization techniques, that no message is delivered to a federate “in its past” (i.e., no time stamp ordered message is delivered having a time stamp lower than the current time of the federate).

To support the time stamp order delivery of messages, the RTI time management service also provides a time advance mechanism. The time management service permits a federate to advance its logical time to T only when it can guarantee that the federate will not receive a time stamp order message having a time stamp less than or equal to T. To support this, each federate specifies a look-ahead value denoting the minimum distance into the future that it will generate a time stamp ordered event. A federate may advance its logical time to the lowest logical time plus look-ahead taken over all participating federates. This means that a federate may not process any local event until the federate’s logical time has advanced to the time of the event. The federate may also request that run-time infrastructure only deliver time stamp ordered events to the federate when the federate’s logical time reaches that of the event, for conservatively synchronized federates; optimistically synchronized federates may relax this constraint.

Based on these facilities and requirements the HLA Federate Interface Specification [97] defines two broad ways in which a federate can interact with the time management system. A joined federate either is, or is not, time-constrained. Time-constrained federates receive and process time stamp ordered messages and require that their logical time be regulated by the run-time infrastructure. A joined federate may also be time-regulating. Time-regulating federates generate time stamp ordered messages which in turn control the advance of logical time for time-constrained federates.

#### HLA Shortcomings

The High Level Architecture for Modeling and Simulation provides a wealth of patterns and techniques for the massively multiplayer online game developer. However, it is not, as a reference system, well suited for use as the underlying implementation of an MMOG environment. The HLA, with its design objectives focused on interoperability, multiple hardware and language interfaces, and legacy system support is more general than required for an MMOG [128]. For game development, particularly on game consoles and other constrained platforms, the memory and computation requirements to support an interface richer than required become a concern. Further, the HLA provides computationally expensive correctness and repeatability guarantees that exceed the requirements of most online games. Although correctness and repeatability are highly desirable, primarily for testing and quality assurance purposes, an operational MMOG only needs to provide its participants with the appearance of consistency and fairness [60, 63].

Rather than using a fully HLA compliant subsystem for implementing an MMOG, a game developer may use the HLA concepts to build a system more specifically tailored to the needs of online games. Over the course of the developing the HLA Run-Time Infrastructure the Defence Modeling and Simulation Office funded exploratory options for game constructed via the RTI [128]. The OpenSkies system [38], from Cybernet Systems Corporation, is an online gaming framework which is extensible to massively multiplayer online games [37] and is directly derived from the HLA. Similarly, Quazal's Eterna [162, 163], a middleware infrastructure for massively multiplayer online games, also has its roots in the High Level Architecture.

### Academic Research

While the majority of the impetus and funding for the development of large scale virtual networked environments has historically emanated from the Department of Defence, much of the knowledge and expertise developed by DOD projects failed to transition outside of each individual project [176]. Thus, a great deal of the technology developed under the auspices of the DOD was subsequently reinvented, published, and extended by the academic community. This section covers the most significant of the early academic contributions to networked virtual environment, and in turn MMOG, development.

### NPSNET

The NPSNET Research Group [140] of the Naval Postgraduate School focuses on developing networked virtual environment technology for use by the Department of Defence. In particular, this group developed the Naval Postgraduate School Network (NPSNET) [124, 157] Simulator as a test bed for exploring the construction of large scale virtual environments. This effort culminated in NPSNET-IV [123], which extended the DIS protocols using IP multicasting and human interface design techniques in order to provide an virtual environment in which fully articulated human players could interact with almost all types of ground, air, and subsurface military vehicles [176] .

The NPSNET Research Group explored the use of internetworking technologies, particularly those used by the Internet, as a communications medium for building large scale distributed virtual environments [123]. They were the first group to exploit the real-world characteristics of spatial, temporal, and functional locality within a large scale environment by introducing an entity-local area-of-interest-manager (AOIM) and using IP multicasting to restrict and focus local processing and network resources [125].

#### IP Multicasting

Instead of using server software to perform point-to-point delivery of updates from the source entity to receiving entities, NPSNET used the emerging facilities of IP based inter-networks to simultaneous multicast packets to a set of subscribers [125]. The IP Multicast Protocol [45] provides one-to-many and many-to-many unreliable message delivery over the Internet. This places the burden of propagating messages from publisher to subscriber on the network transport system instead of the application. This can allow for extremely efficient utilization of network bandwidth and computational resources; only one message need be transmitted over the network for an arbitrarily sized recipient set [189]. There are however, a number of obstacles to using IP multicast for a massively-multiplayer online game. Firstly, IP multicast is not ubiquitously supported across the Internet [10, 165]. A number of multicast based systems have reintroduced specific server processes which emulate network multicast services for regions of the network where it is not supported [11, 66] . Secondly, where IP multicast is supported, the sustainable number of multicast groups is insufficient for very large entity populations [10, 128]. Thirdly, the MMOG developer must select an appropriate strategy for the creation, management, and assignment of multicast groups [2, 73, 165]. For this, we turn to area of interest management.

#### Area of Interest Management

In an area of interest management scheme, entities transmit updates to a set of subscription managers who take responsibility for delivering the updates to all interested parties. In NPSNET, the area of interest (AOI) manager uses spatial, temporal, and functional relationships between entities in order to partition the virtual environment into smaller environments and/or classes in which an entity can express interest [123, 124, 125, 126]. While the primary AOI criteria used was spatial, based on the hexagonal grid inhabited by an entity, area of interest management has been adopted and expanded by a number of other projects [1, 2, 3, 58, 83, 119, 127, 137, 156, 165, 177, 184, 185, 201].

One way to specify interest is to introduce an interest expression language by which an entity can describe the information it would like to receive [127, 156, 176]. Using predicates, each entity declares a sequence of filters or assertions that an update must pass in order to be considered relevant to the entity. In this model, server processes generally manage the filtering and delivery of messages to the interested parties, as the resulting delivery patterns for filtering are application based and do not correlate with network multicasting [176].

There exist other schemes for interest management that do not explicitly rely on application data; these schemes are more easily mapped to message delivery optimizations such as network multicasting. One such approach assigns a different multicast address to each entity in the virtual environment [2, 177]. Each entity then subscribes to the set of multicast groups corresponding to those entities in which it is interested. This scheme requires some means for each entity to discover the simulation objects that are in its general vicinity or otherwise worthy of its attention. Directory, or beacon, servers provide a means for an entity to discover the members of its interest set, and the corresponding multicast group for each member [8]. This approach may further be extended to support multiple levels of fidelity or multiple channels of information by assigning more than one multicast group to an entity [176]; clearly, a multicast-group-per-entity approach can consume a large number of multicast addresses for simulations have large entity populations.

Another scheme for mapping interest management onto network multicast protocols is to assign a multicast address to each region of the simulation [73, 125, 165, 176]. In this approach, an entity transmits its updates to the multicast group corresponding to the region of the virtual environment in which it currently resides. In comparison to a multicast-group-per-entity approach, a multicast-group-per-region approach may significantly reduce the number of multicast groups required, but may significantly increase the load on the network routing system to manage the frequent changes in multicast group subscriptions [73].

Hybrid schemes have also been proposed, where both group-per-entity and group-per-region approaches are employed [1, 2, 3, 176, 177]. These hierarchical interest management systems use locality, entity type, information type, and/or area-of-interest filters to control the information received by an entity, either by directing the entity to one or more multicast groups, or with application server software which is responsible for message delivery. For massively multiplayer online games having client/server architectures, hybrid approaches are particularly interesting as the server may also perform message aggregation on the receiver’s behalf and the network topology can be such that a client remains connected to a single server, which can provide (more) stable message latencies than having a client connection migrate from host to host during a game.

## PARADISE

At Stanford University, the Distributed Systems Group developed the Performance Architecture for Advanced Distributed Interactive Simulation Environments (PARADISE) to explore network software architecture for building large-scale multiplayer three-dimensional simulations running over a wide area network [54]. Like several other academic projects we will survey, the contributions of the PARADISE network were directed at DIS or DIS-like simulation environments. Several of those contributions have already been mentioned in previous sections. For example, Position History Based Dead-reckoning [174, 175] was a product of this group's research. As this extension to dead-reckoning has already been covered, it will not be discussed further in this section. Some of the group's other contributions include projection aggregation, hierarchical area-of-interest servers, and exploration of the design and use of reliable multicast protocols for data distribution.

### Projection Aggregation

As the number of entities within a simulation environment increases, so too does the volume of updates each participating host must process.

Given a large enough population of "interesting" entities, the volume of messages may overburden a host or saturate a network connection. Techniques such as dead-reckoning and interest-based filtering of messages serve to decrease the number of messages received by each host. We have also seen approaches to message aggregation which reduce the number of messages by bundling them based on organizational information, such as the type of object the message is about, or based on the location of the entity in the virtual environment. The PARADISE group developed Projection Aggregation to apply both organization and locality information when creating aggregations [176]. More generally, the approach extends to arbitrary dimensions along which one would like to project [177].

Unlike packet aggregation, which saves bandwidth overhead by combining multiple messages into a single packet, projection aggregation introduces new container entities which provide summary information for their members. For example, a projection aggregation might transmit the number of entities in the projection, a focal point, and information about how the entities are distributed about that point. This would allow a distant observer to construct a low-fidelity approximation of the projection, which may be sufficient for the specific purposes of that simulation environment [177]. An approach such as this would be useful to model crowded areas, for example, where detailed information about entities in one’s immediate vicinity is required, but where crowd density information is sufficient for representing the entities beyond one's range of direct interaction.

Arranged in a hierarchy, projection aggregation also provides a general model for providing variable level-of-detail in update messages and hierarchical area of interest management [177]. In this model, a parent aggregation represents summary information for its member entities, which may be the parts of a tank, a formation of fighter planes, or a distant fleet of warships. As an observer requires a higher-fidelity view of an aggregate object, it simply subscribes to updates from the child aggregates instead of the parent.

Dead-reckoning techniques can be further applied to the summary information, with updated summaries being transmitted when the actual distribution of the aggregated entities differs by some threshold from the predicted distribution. The PARADISE team observed a 40% reduction in the number of entities in the interest set of each host and a 72% reduction in packet rate [177].

### Reliable Multicast

To further reduce the number of update messages required by the DIS protocols, the PARADISE group explored the use of reliable multicast protocols to eliminate the need for frequent heartbeat messages from entities having stable state [176]. Log-Based Receiver-reliable Multicast (LBRM) provides a reliable multicast and persistence system with low-latency recovery from packet loss [85]. This technique adds log servers which maintain a history of recently transmitted messages from one or more hosts. If a log server does not receive an acknowledgement from each of a designated list of Designated Ackers then the message is multicast again, otherwise, the message is unicast on receipt of a negative acknowledgement from any other subscribers who notice that they failed to receive a particular message. With this scheme, the team was able to support terrain as a first-class entity while reducing heartbeat packet overhead by a factor of fifty and lowering the time required by a new participant to learn the current state of the environment [85].

## MASSIVE

The Communications Research Group at the University of Nottingham created the Model, Architecture, and System for Spatial Interaction in Virtual Environments (MASSIVE) teleconferencing environment to explore interaction and awareness models in collaborative virtual environments [80]. The group looked not only at network software architecture, but also at integrating heterogeneous user-interfaces, multimedia, and common patterns of social interaction into the virtual environment. The MASSIVE network software architecture has evolved from a peer-to-peer system in its early incarnations to a client-server model with a fully replicated distributed database and emulated multicast [81]; server processes maintain centralized representations of the world state and distribute updates from the otherwise autonomous client process that owns a given entity to all observers of that entity. MASSIVE is particularly interesting for its spatial model of interaction and its division of the virtual environment into locales.

### Spatial Model of Interaction

As its name suggests, this model uses spatial properties, such as proximity, as a means for mediating entity interaction. Objects can move about the virtual environment in order to from groups with other entities and hold conversations within these groups [80]. Similar to the DIVE system, every entity has an aura around it which defines the area in which it can meaningfully interact, a nimbus defining the area in which the object's properties may be observed, and a focus defining the area in which it is interested. Two objects may interact when their auras intersect and one's focus intersects the other's nimbus. More concretely, the intersection of auras implies that it may be possible for the objects to interact while the interaction of focus and nimbus implies that one object is aware of the other.

The specifics of the interaction is determined by the level of awareness one object has of another. For example, in an environment supporting audio, a player who is close to a speaking player and is "looking" at that speaker would hear that speaker more clearly or loudly than another speaker at the periphery of our avatar's senses. Similarly, entities in the center of one's field of vision would be rendered with greater fidelity than objects at the edge or outside of one's field of vision. This model rather intuitively approximates the real-world behaviour of the human senses.

Unfortunately, a pure aura-nimbus-focus implementation of the spatial model of interaction does not scale to large numbers of entities [176]. The pair-wise management of spatial interactivity requires considerable processing resources; moreover, the specialization of updates based on the degree of interaction between pairs of entities results in messages that are customized for each particular recipient. A system based on the spatial model of interaction cannot easily take advantage of network efficiency optimizations such as multicasting.

### Locales

A locale, introduced by the creators of the Spline system [8], encapsulates a region within the virtual environment, such as a room, corridor, open space, or vehicle with it's own local coordinate system and a set of boundaries through which entities might pass into other locales. The approach is strikingly similar to the manner in which portal based graphics engines [122] represent the world and compute a potential visibility set (PVS); a portal is analogous to a boundary between a locale. The third generation of the MASSIVE system extended this idea to integrate awareness and projection aggregations, which the authors referred to as abstractions [161].

Locales allow a virtual environment to effectively support non-Euclidian structures such as virtual mirrors, worm-holes, and buildings whose interior is larger than their exterior, within a three dimensional framework. Combining focus with locales/portals or binary space partition (BSP) trees is also an effective way to perform interest management based not only on proximity, but on what is actually visible to the participant. This level of granularity and accuracy, while desirable, is counter-balanced by the need for low-latency discovery about new entities that enter a participant's field of vision. If, in order to maintain sufficient interactivity, a client must be informed about an entity the player cannot currently see, but has the potential to see in the immediate future, then it may not be worthwhile to support fine-grained visibility culling at all.

## Other Systems

This section describes several other academic projects whose contributions have already been mentioned in the context of extending or enhancing a previously mentioned technique or approach. Rather than restating each contribution here, we briefly summarize each project and highlight any contributions not previously mentioned. Inclusion of a project in this section, as opposed to its own larger section, does not relate in any way to the magnitude or importance of the contribution of the project; it is purely a side-effect of the organization of this work.

### DIVE - Distributed Databases

The Distributed Interactive Virtual Environment (DIVE), developed by the Swedish Institute of Computer Science, uses a peer-to-peer distribution model to simulate a concurrent shared memory which is used by distributed processes to provide a shared world database [26, 27]. DIVE's shared memory model implements an active, partially-replicated distributed database. Each participating application process maintains its own local copy of the world database while sending and receiving updates to the database via reliable IP multicast and multicast proxy servers, where IP multicast is not supported [66]. The DIVE database is also dynamic, meaning that entities and entity types can be added to the environment at run-time.

### RING – Network Topologies

RING is a client-server system for networked virtual environments [72]. In RING every entity is managed by exactly one client host and communication between clients is managed by servers, which filter and/or aggregate messages and simulate multicast. Like the locale system used by MASSIVE, a RING environment is composed of rooms and corridors connected by doors, which are analogous to portals/boundaries; this allows RING to perform line of sight visibility tests to filter messages. As these concepts are discussed elsewhere in this thesis, they will not be repeated here. For the purposes of this section, the key contribution of the RING project is the group's experimentation with, and insight into, network topologies and messaging protocols [73], which will be further explored in the chapter on network software architectures.

### BrickNet – Client/Server Architectures

BrickNet [172, 173] is a virtual environment toolkit that was among the first to explore client/server architectures [176]. Rather than replicating the world database (as done in its contemporary projects SIMNET, DIS, and

DIVE), BrickNet partitioned its data model among the client hosts, and used servers to broker and mediate the exchange of information among the clients. Derived from BrickNet, a PC based gaming library called NetEffect has been used to implement HistoryCity, a virtual world in which children can explore the history of Singapore [44, 176].

## Commercial Computer Games

Companies and developers within the video and computer game industry have, in the last eight to twelve years, become extremely open with regard to sharing the technologies and approaches used in their products. Developer conferences, trade magazines, and books have been available for quite some time, mostly targeted to game developers and/or prospective game developers. More recently, crossover and transfer of both technology and individuals between the military/academic community and the gaming community has become more common [128, 129, 150, 168, 183]. Computer and video game development has become a topic for study at specialized schools, as well as mainstream colleges and universities [74]. Increasingly, companies and industry groups actively pursue collaborations with academic institutions [65, 100]. Indeed, the distinction between military, academic, and commercial research and development in the multiplayer and massively-multiplayer environments may no longer be appropriate.

This section briefly summarizes the massively-multiplayer online game literature emanating or derived from the commercial video and computer game industry. The many instances where commercial games have adopted or extended techniques originating in the military and/or academic space are not repeated here; rather, those examples are mentioned in the preceding sections in the context of the extended technique(s).

### First Person Shooters

The popular series of Quake games, from Id Software [89], have proven to be successful not only as commercial products, but also as foundations for research into the development of multiplayer online games. The Quake games make use of dead-reckoning, delta compression, and client/server network software architectures [86, 87]. Id Software has released the source code for Quake, Quake II and Quake III Arena under open-source and non-commercial use licenses, allowing students and enthusiasts alike to modify and extend the games. For example, a group of students at the University of Washington used Quake as a test-bed to explore the game play effects of different choices of transport and network layer strategies [20]. Another group of students, at the University of Michigan, extended Quake to support the use of distributed game servers employing full world database replication and optimistic synchronization [33, 34].

### Real-Time Strategy

Real-Time Strategy (RTS) games, such as Blizzard Entertainment's Warcraft series [19] or Microsoft Corporation's Age of Empires series [132], allow a small group of players to command tens, if not hundreds, of semiautonomous units against one another, dispersed across the game environment. The networking models for this style of play have different requirements from those of action or simulation games, in which the player typically controls exactly one character with much smaller visibility into the game environment. RTS games largely use a synchronized-parallel model of distributed simulation, in which identical copies of the game environment are simulated in lock-step on each node [16, 82, 181]. Rather than transmitting state updates, individual hosts transmit and synchronize on command updates; the game engine is such that given an identical command stream, the evolution of the simulation is entirely deterministic across all hosts. The network model does not scale effectively beyond support for several dozen players [181]. Indeed, it is not clear if the RTS genre, as currently defined, is well suited to larger numbers of simultaneous players.

### Massively Multiplayer Online Games

As discussed previously, a number of game development and publishing companies have entered the massively multiplayer online game market. In addition, several game technology companies have taken up the challenge to produce middleware and development platforms for massively multiplayer online games. This section explores the network software architectures currently exploited by MMOG developers.

## Shards

One common technique for distributing the computational requirements of client/server massively multiplayer online games is to partition the player space across more than one logical instance, or shard [49, 53, 67], of the game world (Figure 4). An MMOG using this technique might offer (1) several identical shards of the game world, where each acts as a sort of parallel universe to the others [139, 148, 179], (2) several distinct game worlds, representing disjoint regions of the world or catering to different styles of play [196], or (3) a set of identical shards for a set of worlds [121]. Each shard typically corresponds to a game server, or cluster of game servers, allowing the MMOG provider to scale the system as the player population grows.

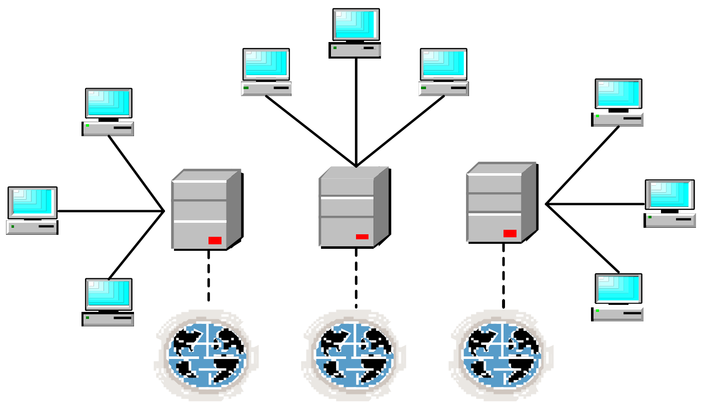

**Figure 4. Client/Server Shards.**

In shard-based game systems, a player in one shard cannot interact with a player in another shard, even if the two players are in the same location in the logical game world. In this regard, a shard based game system does not strictly conform to our previously given definition of a massively multiplayer online game; players in the “same” game do not necessarily inhabit a common environment, and may not be able to interact with one another. For example, a common scenario in shard-based games is for a user to join an MMOG only to find herself isolated from her friends, who happen to reside on another shard; this scenario is only grudgingly tolerated by players [49, 53]. The primary attraction for most MMOG players is the opportunity to interact with others, particular with their friends [67].

A shard based world, where each shard corresponds to a server, has a number of benefits.

- Compared to the other approaches we will consider in this section, shards are the least complex [14].
- Maintaining the consistency of the game world on a single shard server is fairly straightforward; the server’s representation of the world state is correct, by definition.
- Each shard is independent of the others; communication between shard servers is not, in general, required.
- A shard represents a complete world, all game resources (art, sounds, and models) are localized to that world. This can simplify the loading and management of resources on the client, as well as the management of game assets by the game’s artists and developers.

- When the player population grows, one or more additional shards may be introduced to which new players are assigned.

There are two principal drawbacks to using shards. Firstly, as discussed above, shards result in the segregation of the player population. Secondly, for systems where each shard corresponds to a single game server, individual shards are not scalable. For a given shard, the server on which it resides will support some maximum number of concurrent users. Once this limit is reached, there is no way to increase the capacity of a shard other than replacing or upgrading the server hardware.

### Zoned Worlds

Another approach, closely related to shards, is to divide the world into distinct regions, or zones (Figure 5), and allow the player some explicit means for travelling from one zone to another [14]. Each zone is managed by a different server or server process, allowing the player capacity of the massively multiplayer online game environment to scale by the addition of new zones. Zones also allow the effects of server failure to be isolated to the players using the failed server. In essence, a zone is a region-oriented shard system that explicitly allows player migration. This approach can be used both in a single world environment, where there exists a single cluster of zone servers representing the game world, or in a shard-based environment, where there exist multiple clusters, each representing a shard.

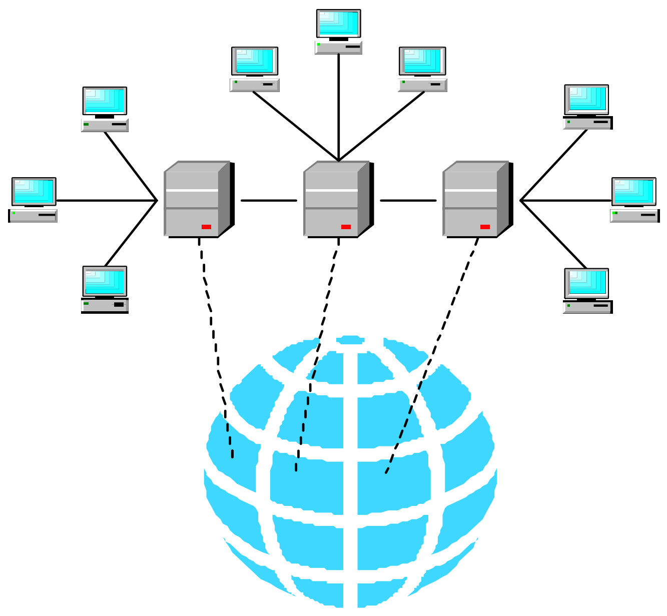

**Figure 5. Zoned World.**

A massively multiplayer online game that is presented to the user as a single zoned world provides many of the benefits of shards (distributing the user population across multiple servers) without the forced segregation of users. As mentioned above, an MMOG can also be presented as a set of zoned shards; this allows the developer to support a larger number of simultaneous players in a single shard, thus reducing the required number of shards and decreasing the degree of user segregation. This approach, where each shard is supported by a server cluster implementing the zones of the shard, is the most prevalent network software architecture employed by the MMOG products on the market at the time of this writing [14, 49, 53, 62, 63].

The main drawback to zoned worlds is the explicit discontinuity between zones. Players dislike the interruptions incurred as their game client transitions to a new zone, unloading the game resources for the previous zone and loading the resources for the new zone [49, 53]. Ironically, these explicit transition points are quite attractive to MMOG developers as they serve to greatly limit the complexity of building large game environments [14]. A second drawback, which is similar to that of shards, is the difficulty in scaling the capacity of an individual zone. For a given zone, the server on which it resides will support some maximum number of concurrent users. Once this limit is reached, there is no way to increase the capacity of a zone other than replacing or upgrading the server hardware. This issue has the potential to be more problematic in a zone-based system than in a shard-based system. Limiting the number of players in a popular shard distributes the players across multiple copies of the same game world; the player does not, in principle, miss out on any part of the game experience. Limiting the number of players in a zone may prevent players from freely exploring the entirety of the online environment.

#### Seamless Worlds

A seamless world is a zoned world in which the boundaries and transitions between zones are transparent to the player [14, 49, 53]. This model is almost identical to the a zoned world model; however, the servers for neighbouring zones must interact in order to notify each other about objects that are ‘close’ to the zone borders, and in order to transfer ownership of objects as they cross zone boundaries [14]; more ambitious seamless world models might employ dynamic zone boundaries, growing and reducing the extent of a zone in order to maintain zone populations that do not overburden the computing resources of any game servers.

Seamless worlds offer a number of benefits [14]. Firstly, a seamless world can present the player with a much larger contiguous environment in which to play. Secondly, a sophisticated seamless world environment might dynamically manage the location of zone boundaries in order to more evenly distribute the player population across the available servers; this provides a mechanism for greater scalability, as well as for fault tolerance. Thirdly, a seamless world insulates the client from the load and transition times associated with zoned worlds; the game client must dynamically anticipate and load required data in the background as appropriate during game play. Lastly, a seamless world does not segregate the player population.

The benefits of seamless worlds come at the expense of considerable complexity. Whereas a zoned world allows a single server to manage all local interactions between players, a seamless world must support interactions between players straddling the boundaries between neighbouring zones. As we have seen, maintaining distributed consistency is quite difficult and error prone. Further, care must be taken in designing game content (in-game assets such as maps, models and art) such that they respect server boundaries and the radii of awareness for players [14]. For seamless worlds having dynamic zone boundaries, this complexity further increases.

Despite these challenges, the massively multiplayer online game developers and middle-ware vendors are actively pursuing the benefits of seamless worlds. At the time of this writing, a number of current and upcoming MMOG products provide seamless world support [104, 135, 187, 188]. Additionally, several MMOG infrastructure vendors have introduced middleware supporting seamless worlds [7, 17, 200].

### Grid Computing

Butterfly.net [7, 22] extends the multiple server model described above by using a grid computing framework for resource management and allocation. Grid computing (also called utility computing) is a model in which the computational and storage capacities of connected devices are viewed as a managed pool of resources. Participants in the grid can request and use these resources as their needs and service agreement dictate. This model has several attractive features. Firstly, it can be mapped rather naturally onto a fault-tolerance by fail-over model. If a server becomes overloaded it can request that some of its responsibilities be reassigned to another resource. If server failure is detected, an administrative process could automatically assign the tasks of the failed server to another resource. Open-source grid computing toolkits [182] are available for use by massively multiplayer online game developers.

An organization having such a grid could run several massively multiplayer online game products using the same network and hardware. The load distribution properties of the grid may be used to dynamically assign and balance the use of computing resources to each MMOG as it needs. Such a service could be offered to MMOG developers using a public-utility model in which the MMOG provider pays for the computing resources they use each month. This would serve to lower the initial costs of deploying and operating a new massively multiplayer online game. Conversely, an MMOG provider offering multiple MMOG products on its own grid could, in principle, run them all on the same infrastructure. This would avoid much of the difficulty of provisioning infrastructure for a new MMOG product and allow the game provider to leverage their hardware investment over multiple products.

# Network Software Architectures

This chapter considers models for distributing simulation tasks and world representation for a massively multiplayer online game network. Categorized broadly, there are two basic structures for creating a distributed game network: client/server and peer-to-peer. In the following sections, we will briefly examine several variations and combinations of these two basic architectural patterns, discussing the benefits and challenges of the basic approaches.

## Client/Server

In a client/server system, one or more server nodes are tasked with providing services to the player (client) workstations. Clients do not communicate directly with other clients; rather, they communicate directly with one or more servers, which will in turn communicate with the other clients (and possibly servers) participating in the system. In the simplest client/server architecture, a single server provides the entire game environment for all clients or some subset of clients (Figure 4). Multi-server architectures allow the game designer to (1) partition the clients across several server hosts and/or (2) to partition the environment across multiple hosts [176].

In a replicated client/server system (Figure 6) clients are partitioned across multiple servers. This architecture is similar to the architecture for shard world. Each client sends and receives updates via a single server for the duration of the game session and the servers communicate among themselves using peer-to-peer protocols, much as in DIS or SIMNET [72, 176]. When a server receives an update from one of its clients, it is responsible for forwarding the update to other interested clients to which it is connected, as well as to each of the other servers. As opposed to having a single server managing the entire game environment, this architecture distributes the workload of handling the clients by dividing the clients among the servers. Because each server will likely have clients which are widely scattered throughout the virtual environment, the level of inter-server communications and per-server processing required to support the clients can be quite high [73, 176]. Architectures of this form are able to support networked virtual environments in which the client is trusted to authoritatively compute and distribute game state. In these architectures, the servers primarily route messages; however, these architectures have been shown less scalable than others we will discuss [73]. For virtual environments in which the client is not trusted, these architectures require the servers to shoulder the additional burden of observing, arbitrating, and/or computing game state.

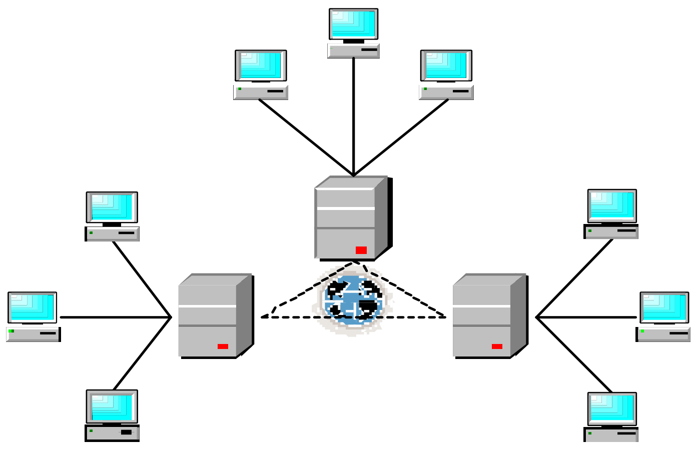

**Figure 6. A replicated multi-server client/server archictecture.**

As mentioned previously, another approach for distributing the computational and client load across multiple servers is to divide the game world into zones and assign each zone to a server (Figure 7). In this approach, each server is responsible for managing the state of the objects within its zone(s) as well as for maintaining appropriate communications with clients currently within its zone(s). As the object(s) controlled by a client move through the environment, the client communicates with the appropriate game server for the object's current zone. As such, each zone-oriented game server provides a natural location from which to provide area-of-interest-management and message aggregation when distributing game state updates to interested clients. Compared to replicated server approaches, partitioning the game environment and client population across multiple servers can reduce inter-server messaging requirements by as much as 95% [176] and has been shown to scale more effectively to large client populations [73]. Inter-server communication is not eliminated entirely as client crossing zone boundaries (whether seamless or not) must be transferred from the server for the objects current zone to the server for the neighbouring zone. Additionally, administration and implementation of zone based servers can be quite complex and error prone [14, 176], as can managing and redirecting clients to the appropriate game server as they move throughout the environment.

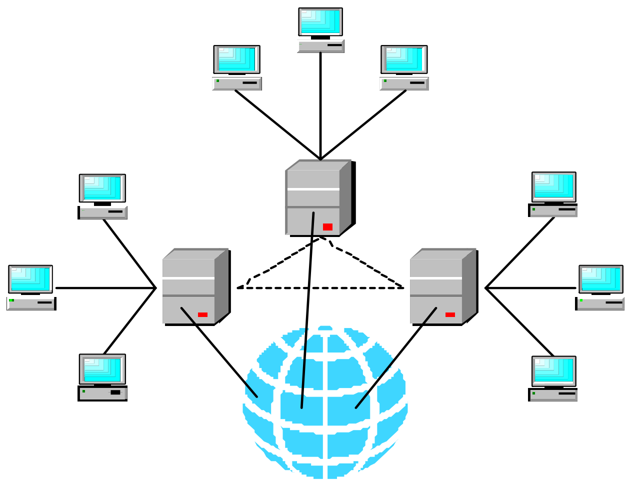

**Figure 7. Zoned world multi-server client/server architecture.**

The network interface that a game presents to its clients can be simplified by combining the two previous approaches. Rather than having the clients communicate directly with the zone servers, clients can instead communicate with gateway, or front-end, servers (Figure 8) that route messages to the appropriate back-end game server [14, 22, 199]. To the client, the game service appears as a set of replicated servers, each capable of acting as its entry point into the virtual environment. Internally, however, the game service provider is able to transparently partition the game world and distribute the computational load across the “real” game servers. This additional level of indirection can be used to facilitate:

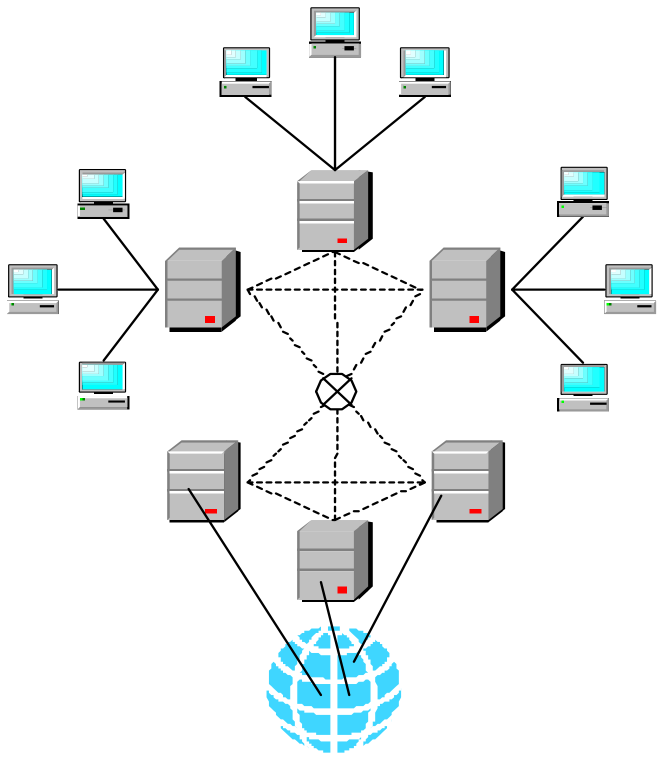

**Figure 8. A Hierarchical Client/Server Architecture.**

- fault tolerance: a failed server can be dynamically replaced without affecting its clients
- dynamic zone boundaries: an over-populated zone might be divided and a portion of its clients transparently reassigned to another server
- scalability: additional servers and/or zones can be added to the world without affecting the client interface

A massively multiplayer online game developer may also distribute the computations for a game zone across multiple servers (Figure 9). For example [14]:

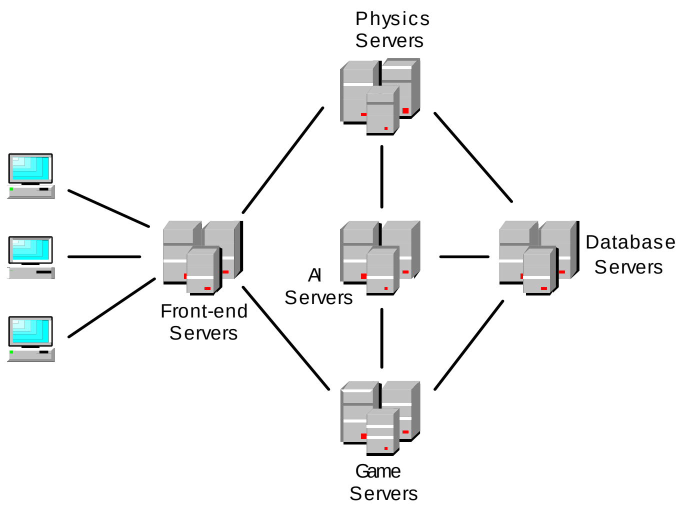

**Figure 9. Distributing computational tasks.**

- a “physics server” might compute changes in game state that evolve over time
- a “game server” might manage interactions between players (e.g., combat, commerce, etc)
- a “database server” might manage persistent state and transactional semantics

Assuming that the network bandwidth available to the massively multiplayer online game provider’s infrastructure is scalable, an MMOG system utilizing distributed computation to model a zoned world can, in principle, scale indefinitely [73, 199]. Grid computing frameworks can be used to help manage the complexity of assigning tasks and provisioning computational resources within the game network [22, 31, 64, 182].

## Benefits

The primary benefit of client/server architectures is that they are well understood by the gaming and research industries. Client/server architectures have fairly straightforward administrative, security, and consistency models [43, 110, 114, 128, 166]. Centralized control and administration of server resources also simplifies the live operation of a massively multiplayer online game service. Additionally, having a controlled hardware and software environment on the servers facilitates easier development and deployment of an MMOG [43]. Hierarchical client/server models have been shown to be, in principle, scalable to large client populations [73] and client/server architectures are easily amenable to grid computing [22].

## Challenges

There are a number of reasons why MMOG developers and providers are dissatisfied with client/server architectures. Firstly, implementing client/server architectures for an MMOG can require significant investment in game server hardware and network infrastructure [29, 107]. Managing this investment is further complicated by difficulties in estimating the initial popularity of an MMOG so as to calculate the resources required in order to provide sufficient capacity. Recurring utility, hosting, and/or bandwidth costs for server facilities also increase as the number of servers increase. In particular, nearly all game updates will pass through the game provider’s network, causing their bandwidth consumption to be quite high, and possibly quite expensive. Additionally, latency in message delivery is higher for messages passing through a server than for messages sent directly from peer-to-peer.

## Peer-to-Peer

One general family of models for distributing a simulation across multiple nodes is to consider every node an equal peer to the others (Figure 10-a). In a peer-to-peer system design the workstation of each player may communicate directly with any other player's workstation. Further, no single node is solely responsible for some function without which the system ceases to operate. Providing many technological benefits, it has been argued that peer-to-peer architectures represent the future of online gaming [62, 63, 111, 163]. This section briefly considers peer-to-peer architectures for massively multiplayer online games, discussing their benefits and drawbacks.

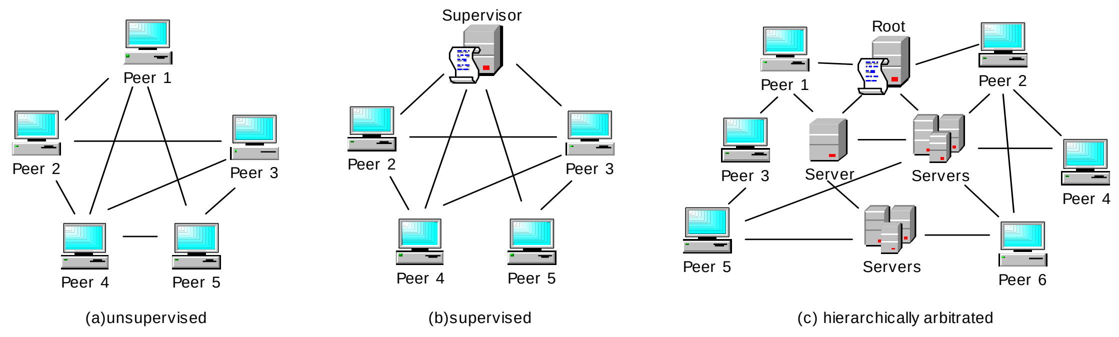

**Figure 10. Peer-to-Peer Architectures.**

The application of peer-to-peer architectures for massively multiplayer online games is an active research area [12, 13, 37, 77, 111, 163]. Hybrid systems utilizing peer-servers for game computations, and/or hierarchical routing through peer-servers for message delivery, are attractive ways for a game developer to lower hardware and bandwidth costs.  Key issues include the management and application of locality of interest [111], and the development of a suitable security model for trusting client hosts with authoritative game state.  Open-source peer-to-peer toolkits [111, 169, 180] are freely available, allowing a MMOG developer to readily explore peer-to-peer architectures.

### Benefits

Peer-to-peer architectures are attractive to MMOG service providers for a number of reasons, not least of which is their potential for substantial reduction on operating and hardware costs.  A system which leverages the computational capacity of its users may require a significantly lower hardware investment on the part of the game service provider [29, 62, 63, 107, 111, 168].  Quality game server hardware is expensive; estimating the popularity of the game in order to provide sufficient capacity, without waste, is difficult [29]; and, recurring utility or hosting costs for server facilities reach significant proportions over time [107].  Additionally, MMOG providers seek to reduce recurring bandwidth costs which, over the lifetime of a successful MMOG product, might approach or even exceed the hardware investment [107].

In a peer-to-peer architecture bandwidth costs are largely assumed by the user population; indeed, it is quite possible that the bandwidth consumed by a given user in a peer-to-peer context will not exceed any threshold at which the user's Internet service provider charges additional fees. For an MMOG service provider, the ideal system architecture would seem to be a peer-to-peer system in which the capacity of the system increases as the utilization of the system increases, particularly if utilization does not overtake capacity [62].

Peer-to-peer systems also offer a number of technical advantages over client/server architectures. Firstly, direct routing of messages between peers can provide lower network latency than passing messages through intermediate hosts [73]; for hierarchical routing of messages through peer-servers, this advantage is moot [111]. Peer-to-peer overlay systems have also been shown to be resilient to node failures [111, 169, 180], possibly providing the fault-tolerance properties required of an MMOG system.

### Challenges

Peer-to-peer architectures also present a number of challenges. Firstly, a pure peer-to-peer system does not require that any particular subset of participants remain active in order for the system to continue to function; in particular, this would imply that the game would continue to operate without the participation of the MMOG service provider, which weakens the revenue model. Additionally, the security and privacy constraints surrounding some game related data, such as a user's personal and financial information, make it unattractive to locate that data on hosts outside of the strict control of the game provider [111, 163]. For the remainder of this discussion, we will assume a hybrid peer-to-peer model in which some set of required hosts are controlled by the MMOG service provider (see (Figure 10-b and Figure 10-c).

Another concern a massively multiplayer online game developer faces when considering a peer-to-peer architecture is that of game security [12, 13, 43, 63, 77, 111, 163]. A peer-to-peer system might elect to place decision making authority on user machines, providing opportunity to cheat. The alternative, using distributed decision and/or transactional techniques, introduces considerable complexity and provides other opportunities to cheat [14]. Peer-to-peer systems also reveal the network address of each user to some subset of other users, exposing users to possible network attack from their peers. Researchers and game technology companies are actively exploring solutions to these issues [13, 35, 48, 63, 77, 111, 160, 163].

Peer-to-peer security models based on reputation capital have been proposed [63, 163] in which a node A decides whether or not to trust another node B based on the past performance of B. The degree of trust which A accords B is based on the opinions expressed by other nodes that have previously interacted with B. This approach can be combined with run-time verification [48], a general validation approach in which a set of rules defining correct/incorrect transaction semantics is evaluated by nodes to determine if the constraints of the game system have been violated. This is much like a post-condition assertion or invariant following the execution of the transaction.  Peers, or server based arbiters, can use the results of run-time verification as a basis for forming their opinions.

Another challenge is that the administration and maintenance of a peer-to-peer system is less well understood than for client/server systems, the de facto standard architecture presently used for massively multiplayer online games [63].  One of the defining properties of a peer-to-peer system is that it is adaptively self-organizing [62], which makes understanding, debugging, and patching the system more complex than for a client/server model.  Perhaps a supervised or arbitrated peer-to-peer system [63] could utilize resource allocation and management techniques from the grid-computing domain [64] in order to ease the facilitate administration of the distributed system.  Grid technology may also provide mechanisms to reduce the complexity of balancing load across the heterogeneous computing resources provided by the users.

A final challenge is the variation in quality of service as peer relationships change over time.  The game play experience of a given user might change dramatically with the capabilities of their current peer set as some peers will have better connectivity or computing resources than others [111].

# Conclusion

Game industry analysts highlight trends indicating that online game usage and market penetration will grow significantly over the next five to ten years; however, the risks, costs and complexity involved in the successful development and operation of a scalable online game service are high, in part due to lack of well established and understood models for the network software architecture of such a product. In this thesis, we have explored the technological landscape from which the massively multiplayer online game developer might select techniques and approaches for the network software architecture of a massively multiplayer online game.

We first introduced the concept of a massively multiplayer online game, presenting a definition and historical overview of MMOG projects and products. We considered the technical challenges inherent in building and operating a massively multiplayer online game, commenting on the similarities and differences between the constraints of military/academic simulation and those of commercial game development. With our context thus established, we surveyed the literature from the military, academic, and commercial gaming domains, identifying the techniques and approaches of interest to the MMOG developer. Lastly, we drew upon the surveyed material to consider the pros and cons of client/server and peer-to-peer network software architectures for implementing a real-time massively multiplayer online game.

# LIST OF REFERENCES

1. Aarhus, L., K. Holmqvist, and M. Kirkengen. Generalized Two-Tier Relevance Filtering of Computer Game Update Events. in Proceedings of the 1st Workshop on Network and System Support for Games. 2002. Bruanschweig, Germany: ACM Press.
2. Abrams, H., K. Watsen, and M. Zyda. Three-Tiered Interest Management for Large-Scale Virtual Environments. in Proceedings of the ACM Symposium on Virtual Reality Software and Technology. 1998. Taipei, Taiwan: ACM Press.
3. Abrams, H.A., Extensible Interest Management for Scalable Persistent Distributed Virtual Environments. Ph.D. 1999. Department of Computer Science, Naval Postgraduate School.
4. Aggarwal, S., et al., Accuracy in Dead-Reckoning based Distributed Multi-Player Games. in ACM SIGCOMM 2004 Conference on Network and System Support for Games. 2004. http://ww2.cs.fsu.edu/~khandelw/research/paper.pdf
5. Alexander, T., A Flexible Simulation Architecture for Massively Multiplayer Games, in Game Programming Gems 3, D. Treglia, Editor. 2002, Charles River Media, Inc.: Hingham, Massachusetts. p. 506-519.
6. Aronson, J., Dead Reckoning: Latency Hiding for Networked Games. 1997, Gamasutra. http://www.gamasutra.com/features/19970919/aronson_01.htm

7. Ballbach, M. and M. Ferguson, Product Review: Massively Multiplayer Online Game Middleware. 2003, Gamasutra. http://www.gamasutra.com/features/20030115/ferguson_01.htm

8. Barrus, J.W., R.C. Waters, and D.B. Anderson, Locales: Supporting Large Multiuser Virtual Environments. IEEE Computer Graphics and Applications, 1996. **16**(6): p. 50-57.

9. Bartle, R.A., Incarnations of MUD. 2004. http://www.mud.co.uk/richard/incarns.htm

10. Bassiouni, M.A., et al., Performance and Reliability Analysis of Relevance Filtering for Scalable Distributed Interactive Simulation, in ACM Transactions on Modeling and Computer Simulation (TOMACS). 1997, ACM Press. p. 293 - 331.

11. Bauer, D. and S. Rooney. The Performance of Software Multicast-Reflector Implementations for Multi-player Online Games. in Proeedings of the 5th COST264 Internaltion Workshop of Networked Group Communication (NGC 2003). 2003. Munich, GR: Springer.

12. Baughman, N. and B.N. Levine. Cheat-Proof Playout for Centralized and Distributed Online Games. in Proceedings of the Twentieth IEEE Computer and Communication Society INFOCOM Conference. 2001: IEEE Computer Society.

13. Baughman, N., M. Liberatore, and B.N. Levine, Cheat-Proof Playout for Centralized and Serverless Online Games. in Technical report 04-35. 2004. http://signl.cs.umass.edu/pubs/

14. Beardsley, J., Seamless Servers: The Case For and Against, in Massively Multiplayer Game Development, T. Alexander, Editor. 2003, Charles River Media, Inc.: Hingham, Massachusetts. p. 213-227.

15. Bernier, Y. Latency Compensation Methods in Client/Server In-game Protocol Design and Optimization. in Proceedings of the 2001 Game Developers Conference. 2001. San Jose, CA: CMP Media LLC.

16. Bettner, P. and M. Terrano, 1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond. 2001, Gamasutra. http://www.gamasutra.com/features/20010322/terrano_01.htm

17. BigWorld Pty, BigWorld Technology. 2004. http://www.bigworldtech.com

18. Blizzard Entertainment, World of Warcraft. 2004. http://www.blizzard.com/wow

19. Blizzard Entertainment, Homepage. 2004. http://www.blizzard.com

20. Bonham, S., et al., Quake: An Example Multi-User Network Application - Problems and Solutions in Distributed Interactive Simulations. in CSE 561 Term Project Report. 2000, University of Washington. http://www.cs.washington.edu/homes/grossman/projects/561projects/quake

21. Brockington, M., Client-Side Movement Prediction, in Massively Multiplayer Game Development, T. Alexander, Editor. 2003, Charles River Media, Inc.: Hingham, Massachusetts. p. 293-303.

22. Butterfly.net, The Butterfly Grid, in *http://www.butterfly.net*. 2004: Martinsburg, WV.

23. Byrne, G., Security Issues of Online Gaming. 2004, GameDev.net. http://www.gamedev.net/reference/articles/article2062.asp

24. Cai, W., F.B.S. Lee, and L. Chen. An Auto-adaptive Dead Reckoning Algorithm for Distributed Interactive Simulation. in Proceedings of the Thirteenth Workshop on Parallel and Distributed Simulation. 1999. Atlanta, GA: IEEE Compuer Society.

25. Calvin, J.O., et al. STOW Realtime Information Transfer and Networking System Architecture. in Proceedings of the 12th Workshop on Distributed Interactive Simulation. 1995.

26. Carlsson, C. and O. Hagsand. DIVE - A Multi-user virtual reality system. in Proceedings of the 1993 Virtual Reality Annual International Symposium. 1993. Seattle, WA: IEEE Computer Society Press.

27. Carlsson, C. and O. Hagsand. The Distributed Interactive Virtual Environments - Architecture and Applications. in Proceedings of the IEE Colloquium on Distributed Virtual Reality. 1993. London, UK: Institution of Electrical Engineers.

28. Chandy, K.M. and J. Misra, Distributed Simulation: A Cast Study in Design and Verification of Distributed Programs. IEEE Transactions on Software Engineering, 1979. **SE-5**(5): p. 440-452.

29. Chappe, O., Personal communication. 2004, Ubi.com Shanghai: Montreal, Canada. p. (Former) Director of Operations.

30. Cheshire, S., Latency and the Quest for Interactivity. in Synchronous Person-to-Person Interactive Computing Environments Meeting. 1996, Volpe Welty Asset Management, LLC: San Francisco, CA. http://www.stuartcheshire.org/papers/LatencyQuest.html

31. Coddington, P.D., et al. Extensible Job Managers for Grid Computing. in Proceedings of the 26th Australasian Computer Science Conference in Research and Practice in Information Technology. 2003. Adelaide, Australia: Australian Computer Society, Inc.

32. Committee on Modeling and Simulation, et al., Modeling and Simulation: Linking Entertainment and Defense. 1997, Washington D.C.: National Academy Press, http://www.nap.edu/readingroom/books/modeling/.

33. Cronin, E., B. Filstrup, and A. Kurc, A Distributed Multiplayer Game Server System. 2001, Electrical Engineering and Computer Science Department, University of Michigan.

34. Cronin, E., et al. An Efficient Synchronization Mechanism for Mirrored Game Architectures. in Proceedings of the First Workshop on Network and System Support for Games. 2002. Bruanschweig, Germany: ACM Press.

35. Cronin, E., B. Filstrup, and S. Jamin. Cheat-Proofing Dead Reckoned Multiplayer Games. in Proceedings of the 2nd International Conference on Application and Development of Computer Games (ADCOG'03). 2003. Hong Kong.

36. Cuciz, D., The History of MUDs: Part II. 2001, Gamespy Industries Inc. http://www.gamespy.com/articles/january01/muds1/index4.shtm

37. Cybernet Systems Corporation, OpenSkies Massive Multiplayer Onling Gaming (MMPOG) SDK. in OpenSkies Documentation. 2004. http://www.openskies.net/files/Openskies_MMPOG.pdf

38. Cybernet Systems Corporation, OpenSkies. 2004. http://www.openskies.net

39. Dahmann, J.S., R.M. Fujimoto, and R.M. Weatherly. The Department of Defence High Level Architecture. in Proceedings of the 1997 Winter Simulation Conference. 1997.

40. Dahmann, J.S., R.M. Fujimoto, and R.M. Weatherly. The Department of Defence High Level Architecture: An Update. in Proceedings of the 1998 Winter Simulation Conference. 1998.

41. Dahmann, J.S. The High Level Architecture and Beyond: Technology Challenges. in Proceedings of the 13th Workshop on Parallel and Distributed Simulation (PADS). 1999. Atlanta, GA: IEEE Computer Society.

42. Dahmann, J.S., J.O. Calvin, and R.M. Weatherly, A Reusable Architecture for Simulations. Communications of the ACM, 1999. **42**(9): p. 79-84.

43. Dalton, W., MMP Server Development and Maintenance, in Massively Multiplayer Game Development, T. Alexander, Editor. 2003, Charles River Media, Inc.: Hingham, Massachusetts. p. 236-243.

44. Das, T.K., et al. Developing Social Virtual Worlds Using NetEffect. in Proceedings of the Sixth IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises. 1997. Cambridge, MA: IEEE Computer Society.

45. Deering, S., RFC-1112: Host Extensions for IP Multicasting. 1989, Internet Engineering Task Force (IETF). http://www.ietf.org/rfc/rfc1112.txt

46. Defence Advanced Research Projects Agency (DARPA), RFC 793 - Transmission Control Protocol. 1981, Internet Engineering Task Force (IETF). http://www.ietf.org/rfc/rfc793.txt

47. Defence Advanced Research Projects Agency (DARPA), RFC 791 - Internet Protocol. 1981, Internet Engineering Task Force. http://www.ietf.org/rfc/rfc791.txt

48. DeLap, M., et al. Is Runtime Verification Applicable to Cheat Detection? in SIGCOMM '04 - NETGAMES Workshop. 2004. Portland, OR: ACM Press.

49. Delio, M., Multiplayer Games: Shards Unite! in Wired News. 2004, Lycos, Inc. http://www.wired.com/news/games/0,2101,62736,00.html

50. DFC Intelligence, Online Gaming on the Video Game Systems. 2002. http://www.dfcint.com/game_article/aug02article.html

51. DFC Intelligence, Challenges and Opportunities in the Online Game Market. 2003. http://www.dfcint.com/game_article/june03article.html

52. DFC Intelligence, Worldwide Market Forecasts for the Video Game and Interactive Entertainment Industry. 2003. http://www.dfcint.com/game_report/games_reports_toc.html

53. Dibbell, J., What's in a Shard? in Terra Nova "Blogs". 2004. http://terranova.blogs.com/terra_nova/2004/04/whats_in_a_shar.html

54. Distributed Systems Group, The PARADISE Project. 2004, Stanford University: Stanford, CA. http://www-dsg.stanford.edu/paradise.html

55. DMSO, Defence Modeling and Simulation Office. 2004. http://www.dmso.mil

56. Electronic Arts, The Sims Online. 2004. http://www.thesimsonline.com

57. Elkins, A., J.W. Wilson, and D. Gracanin. Security Issues in High Level Architecture Based Distributed Simulation. in Proceedings of the 33rd Winter Simulation Conference. 2001. Arlington, VA: IEEE Computer Society.

58. Eugster, P.T., et al., The Many Faces of Publish/Subscribe, in ACM Computing Surveys (CSUR). 2003, ACM Press. p. 114 - 131.

59. Federal Trade Commission, Children's Online Privacy Protection Act of 1998. 15 U.S.C. §§ 6501-6506, P.L. No. 105-277, 112 Stat. 2681-728. 1998: United States Government.

60. Fitch, C., Part 4 - Foundations II. in Cyberspace in the 21st Century. 2000, Gamasutra - CMP Media LLC. http://www.gamasutra.com/features/20001229/fitch_01.htm

61. Fitch, C., Part 5 - Stability Before Security. in Cyberspace in the 21st Century. 2001, Gamasutra - CMP Media LLC. http://www.gamasutra.com/features/20011226/fitch_02.htm

62. Fitch, C., Part 6 - Scalability with a Big 'S'. in Cyberspace in the 21st Century. 2001, Gamasutra - CMP Media LLC. http://www.gamasutra.com/features/20010226/fitch_01.htm

63. Fitch, C., Part 7 - Security is Relative. in Cyberspace in the 21st Century. 2002, Gamasutra - CMP Media LLC. http://www.gamasutra.com/features/20020805/fitch_01.htm

64. Foster, I., C. Kesselman, and S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of Supercomputer Applications, 2001. **15**(3).

65. Frasca, G. Practical Game Theories: Academics Fragging Developers. in Roundtable Sessions of the 2004 Game Developers Conference. 2004. San José, CA: CMP Media LLC.

66. Frécon, E. and M. Stenius, DIVE: a scaleable network architecture for distributed virtual environments. Distributed Systems Engineering, 1998. **5**(3): p. 91-100.

67. Friedl, M., Online Game Interactivity Theory. 2002, Hingham: Charles River Media.

68. Fujimoto, R.M. and R.M. Weatherly. Time management in the DoD high level architecture. in Proceedings of the 10th Workshop on Parallel and Distributed Simulation (PADS). 1996. Philadelphia, PA: IEEE Computer Society.

69. Fujimoto, R.M., Parallel and Distributed Simulation Systems. 1999: John Wiley & Sons Inc.

70. Fujimoto, R.M. Parallel and Distributed Simulation Systems. in Proceedings of the 33rd Winter simulation Conference. 2001. Arlington, VA: IEEE Computer Society.

71. Funcom Inc., Anarchy Online. 2004. http://www.anarchy-online.com/

72. Funkhouser, T.A. RING: A Client-Server System for Multi-User Virtual Environments. in ACM SIGGRAPH Special Issue on 1995 Symposium on Interactive 3D Graphics. 1995. Monterey, CA: ACM Press.

73. Funkhouser, T.A. Network Topologies for Scalable Multi-User Virtual Environments. in Proceedings of the 1996 IEEE Virtual Reality Annual International Symposium (VRAIS'96). 1996. San José, CA.

74. Gamasutra, Game Development Education Portal. 2004, CMP Media LLC. http://www.gamasutra.com/education/

75. Gamer-Talk.net, Champions of Norrath: Realms of EverQuest Cheat Codes for Playstation2. 2004. http://www.gamer-talk.net/cheat311.html

76. Gamespy, #19: Roy and Richard Play in the MUD. in 25 Smartest Moments in Gaming. 2003, Gamespy Industries Inc. http://www.gamespy.com/articles/july03/25smartest/index8.shtml

77. GauthierDickey, C., et al. Low Latency and Cheat-Proof Event Ordering for Peer-to-Peer Games. in NOSSDAV'04. 2004. Cork, Ireland: ACM Press.

78. Goslin, M., J. Shochet, and J. Schell, Toontown Online: Building a Massively Multiplayer Game for the Masses, in Massively Multiplayer Online Game Development, T. Alexander, Editor. 2003, Charles River Media: Hingam, MA. p. 3-19.

79. Government of Canada, Bill C-6: The Personal Information Protection and Electronic Documents Act. Statutes of Canada, Second Session, Thirty-Sixth Parliament. 2000: Public Works and Government Services Canada - Publishing.

80. Greenhalgh, C. and S. Benford, MASSIVE: a collaborative virtual environment for teleconferencing, in ACM Transactions on Computer-Human Interaction (TOCHI). 1995, ACM Press: New York, NY. p. 239-261.

81. Greenhalgh, C., J. Purbrick, and D. Snowdon. Inside MASSIVE-3: flexible support for data consistency and world structuring. in Proceedings of the third international conference on Collaborative virtual environments. 2000. San Francisco, CA: ACM Press.

82. Greer, J. and Z.B. Simpson, Minimizing Latency in Real-Time Strategy Games, in Game Programming Gems 3, D. Treglia, Editor. 2002, Charles River Media, Inc.: Hingham, Massachusetts. p. 488-495.

83. Han, S., M. Lim, and D. Lee. Scalable Interest Management Using Interest Group Based Filtering For Large Networked Virtual Environments. in Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST). 2000. Seoul, Korea: ACM Press.

84. Harrison, D., Mac, PC and Linux Players Unite! : Players can romp together in the worlds of Neverwinter Nights. 2004, Digital Game Developer. http://www.digitalgamedeveloper.com/cgi-bin/getframeletter.cgi?/headlines/gametalk.htm

85. Holbrook, H.W., S.K. Singhal, and D.R. Cheriton. Log-Based Receiver-Reliable Multicast Distributed Interactive Simulation. in SIGCOMM '95. 1995. Cambridge, MA: ACM Press.

86. Hook, B., Introduction of Multiplayer Game Programming. in Book of Hook. 2003. http://www.bookofhook.com/Article/GameDevelopment/MultiplayerProgramming.html

87. Hook, B., The Quake 3 Networking Model. in Book of Hook. 2004. http://www.bookofhook.com/Article/GameDevelopment/TheQuake3NetworkingModel.html

88. Howard, M. and D. LeBlanc, Writing Secure Code: Practical Strategies and Techniques for Secure Application Coding in a Networked World. 2nd ed. 2003, Redmond: Microsoft Press.

89. id Software, id Software Homepage. 2004. http://www.idsoftware.com/

90. IEEE Computer Society, IEEE Std 1278-1993: IEEE standard for information technology - protocols for distributed interactive simulations applications - Entity information and interaction. 1993: New York.

91. IEEE Computer Society, IEEE Std 1278.2-1995: IEEE standard for distributed interactive simulation communication services and profiles. 1995: New York.

92. IEEE Computer Society, IEEE Std 1278.1-1995: IEEE standard for distributed interactive simulation - application protocols. 1995: New York.

93. IEEE Computer Society, IEEE Std 1278.3-1996: IEEE recommended practice for Distributed Interactive Simulation exercise management and feedback. 1996: New York.

94. IEEE Computer Society, IEEE Std 1278.4-1997: IEEE trial-use recommended practice for distributed interactive simulation - verification, validation, and accreditation. 1997: New York.

95. IEEE Computer Society, IEEE Std 1278.1a-1998: IEEE standard for distributed interactive simulation - application protocols. 1998: New York.

96. IEEE Computer Society, IEEE Std 1516-2000: IEEE standard for modeling and simulation (M&S) high level architecture (HLA) - framework and rules. 2000: New York. p. i-22.

97. IEEE Computer Society, IEEE Std 1516.1-2000: IEEE Standard for Modeling and Simulation [M and S] High Level Architecture [HLA] - Federate Interface Specification. 2001: New York. p. i-467.

98. IEEE Computer Society, IEEE Std 1516.2-2000: IEEE standard for modeling and simulation (M&S) high level architecture (HLA)-object model template (OMT) specification. 2001: New York. p. i-130.

99. IEEE Computer Society, IEEE Std 1516.3-2003: IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP). 2003: New York. p. i-32.

100. IGDA, International Game Developers Association (IGDA) Website. in Adacemic/Student Relations. 2004. http://www.igda.org/academia/

101. IMG News, DiabloII.Net Recaps Real Issues. in Inside Mac Games. 2002. http://www.insidemacgames.com/news/story.php?ArticleID=4741

102. Isensee, P., Secure Sockets, in Game Programming Gems 3, D. Treglia, Editor. 2002, Charles River Media, Inc.: Hingham, Massachusetts. p. 546-556.

103. Jefferson, D.R., Virtaul Time. ACM Transactions on Programming Languages and Systems, 1985. **7**(3): p. 404-425.

104. Johnson, J., Massively-Multiplayer Engineering. in Game Developers Conference 2004 - Roundtables Sessions. 2004. http://www.gdconf.com/archives/2004/johnson_jeff.doc

105. Joint Training Confederation, Aggregate Level Simulation Protocol (ALSP) Web-Site. 2004. http://alsp.ie.org/alsp/

106. JPSD Project Office, Joint Precision Strike Demonstration. 2004. https://peoiewswebinfo.monmouth.army.mil/JPSD/

107. Julien, M., Personal communication. 2004, Ubi.com: Montreal, Canada. p. Director of Operations.

108. Kent, S.L., Week 1 - From MUDs to Mainstream: The History of MMOGs. in Massively Multiplayer Online Games: The Past, the Present, and the Future. 2003, Gamespy Industries Inc. http://www.gamespy.com/amdmmog/week1/

109. Kent, S.L., Week 3 - Designing for the Hordes. in Massively Multiplayer Online Games: The Past, the Present, and the Future. 2003, Gamespy Industries Inc. http://www.gamespy.com/amdmmog/week3/

110. Kirmse, A. and C. Kirmse, Security in Online Games. 1997. http://www.gamasutra.com/features/19970707/security.htm

111. Knutsson, B., et al. Peer-to-Peer Support for Massively Multiplayer Games. in INFOCOMM '04. 2004. Hong Kong, China.

112. Lambright, R., Distributing Object State for Networked Games Using Object Views, in Game Developer Magazine. 2002. p. 30-39.

113. Lamport, L., Time, Clocks, and the Ordering of Events in a Distributed System, in Communications of the ACM. 1978, ACM Press. p. 558 - 565.

114. Lee, J., Considerations for Movement and Physics in MMP Games, in Massively Multiplayer Game Development, T. Alexander, Editor. 2003, Charles River Media, Inc.: Hingham, Massachusetts. p. 275-289.

115. Lee, J., Leveraging Relational Database Management Systems to Data-Drive MMP Gameplay, in Massively Multiplayer Game Development, T. Alexander, Editor. 2003, Charles River Media, Inc.: Hingham, Massachusetts. p. 372-384.

116. Lee, J., Relational Database Guidelines For MMOGs. in The Development of Massively Multiplayer Online Games Resource Guide. 2003, Gamasutra. http://www.gamasutra.com/resource_guide/20030916/lee_01.shtml

117. Li, C., et al., Connectivity Splits the Gaming Industry. 2003, Forrester Research.

118. Lincroft, P., The Internet Sucks: Or, What I Learned Coding X-Wing vs. TIE Fighter. 1999, Gamasutra. http://www.gamasutra.com/features/19990903/lincroft_01.htm

119. Logan, B. and G. Theodoropoulos. Dynamic Interest Management in the Distributed Simulation of Agent-Based Systems. in Proceedings of 10th AI, Simulation and Planning Conference. 2000. Tucson, AZ: Society for Computer Simulation International.

120. Loral Systems Company, Technical Report: Strawman Distributed Interactie Simulation Architecture Description Document Volume 1. 1992, Advanced Distributed Simulation Technology Program Office: Orlando, FL.

121. LucasArts Entertainment Company and Sony Online Entertainment, Star Wars Galaxies. 2004. http://www.starwarsgalaxies.com

122. Luebke, D. and C. Georges. Portals and mirrors: simple, fast evaluation of potentially visible sets. in Proceedings of the 1995 symposium on Interactive 3D graphics. 1995. Monterey, CA: ACM Press.

123. Macedonia, M.R., et al., NPSNET: A Network Software Architecture for Larege Scale Virtual Environments. Presence, 1994. **3**(4).

124. Macedonia, M.R., A Network Architecture for Large Scale Virtual Environments. Ph.D Dissertation. 1995. Computer Science Department, Naval Postgraduate School: Monterey, CA.

125. Macedonia, M.R., et al. Exploiting Reality with Multicast Groups: A Network Architecture for Large-Scale Virtual Environments. in Proceedings of the 1995 IEEE Virtual Reality Annual International Symposium (VRAIS'95). 1995.

126. Macedonia, M.R., A Taxonomy for Networked Virtual Environments. IEEE Multimedia, 1997. **4**(1): p. 48-56.

127. Mellon, L. and D. West. Architectural Optimizations to Advanced Distributed Simulation. in Proceedings of the 1995 Winter Simulation Conference. 1995. Arlington, VA: ACM Press.

128. Mellon, L., J. Aronson, and D. West. Applying the Technology of Distributed Training Simulations to Internet Gaming. in Proceedings of 1999 Game Developer's Conference. 1999. San José, CA: CMP Game Group.

129. Mellon, L. Game Develoment and the Research Community: How might they overlayp? in PADS 2003. 2003.

130. Microsoft Corporation, Asheron's Call. 2004. http://www.microsoft.com/games/zone/asheronscall/

131. Microsoft Corporation, Microsoft Component Object Model (COM) Technology Website. 2004. http://www.microsoft.com/com

132. Microsoft Inc., Age of Empires Homepage. 2004. http://www.microsoft.com/games/empires/

133. Miller, D. and J.A. Thorpe, SIMNET: The Advent of Simulator Networking. Proceedings of the IEEE, 1995. **83**(8): p. 1114-1123.

134. Mine, M.R., J. Shochet, and R. Hughston, Building a massively multiplayer game for the million: Disney's Toontown Online. ACM Computers in Entertainment, 2003. **1**(1).

135. Monolith Productions Inc. and Warner Bros. Entertainment Inc., The Matrix Online. 2004. http://www.thematrixonline.com

136. Moran, J., Cheating Poses Potential Problem to a Burgeoning Online Gaming Market. December 12, 2002. http://www.gamemarketwatch.com/news/item.asp?nid=2611

137. Morse, K.L., Interest Management in Large-Scale Distributed Simulations. 1996, University of California, Irvine.

138. MPOGD Inc., Multiplayer Online Games Directory. 2004. http://www.mpogd.com

139. Mythic Entertainment Inc., Dark Age of Camelot. 2004. http://www.darkageofcamelot.com/

140. Naval Postgraduate School, NPSNET Homepage. 2004. http://www.npsnet.nps.navy.mil/npsnet

141. NCsoft, Lineage. 2004. http://www.lineage.com

142. Near Death Studios Inc., Meridian 59 Official Website. 2004. http://meridian59.neardeathstudios.com/

143. Neely, R.B. Security Architecture Development and Results for a Distributed Modeling and Simulation System. in 15th Annual Computer Security Applications Conference. 1999. Pheonix, AZ: IEEE Computer Society.

144. Nexon Games, Kingdom of the Winds. 2004. http://www.nexustk.com

145. O'Brien, M. and G. Gray, Game Cheats and Cheat Prevention. 2002, ArenaNet Inc. http://www.arena.net/news/articles/mikearticle040802.html

146. Olsen, J.M., Server-Side Object Refresh Rates, in Massively Multiplayer Game Development, T. Alexander, Editor. 2003, Charles River Media, Inc.: Hingham, Massachusetts. p. 228-235.

147. OMG, Common Object Request Broker Architecture (CORBA) Homepate. 2004, Object Management Group. http://www.corba.org

148. Origin Systems and Electronic Arts, Ultima Online. 2004. http://www.uo.com

149. Pantel, L. and L.C. Wolf. Network Issues for Video and Games: On the impact of delay on real-time multiplayer games. in 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video. 2002. Miami, FL: ACM Press.

150. Pantel, L. and L.C. Wolf. On the Suitability of Dead Reckoning Schemes for Games. in Proceedings of the 1st Workshop on Network and System Support for Games. 2002. Bruanschweig, Germany: ACM Press.

151. Partridge, C. and R. Hinden, RFC-1151: Version 2 of the Reliable Data Protocol. 1990, Internet Engineering Task Force (IETF). http://www.ietf.org/rfc/rfc1151.txt

152. Patterson, J., Keep it Smooth: Asynchronous Clients and Time Travel, in Massively Multiplayer Game Development, T. Alexander, Editor. 2003, Charles River Media, Inc.: Hingham, Massachusetts. p. 304-313.

153. Pizer, P., Social Game Systems: Cultivating Player Socialization and Providing Alternate Routes to Game Rewards, in Massively Multiplayer Game Development, T. Alexander, Editor. 2003, Charles River Media: Hingham, Massachusetts. p. 427-441.

154. Pope, A., The SIMNET network and protocols, in Technical Report 7102. 1989, BBN  Systems and Technologies: Cambridge, MA.

155. Postel, J., RFC 768 - User Datagram Protocol. 1980, Internet Engineering Task Force (IETF). http://www.ietf.org/rfc/rfc0768.txt

156. Powell, E.T., et al. Joint Precision Strike Demonstration (JPSD) Simulations Architecture. in Proceedings of the 14th Workshop on Standards for the Interoperability of Distributed Simulations. 1996.

157. Pratt, D.R., A Software Architecture for the Construction and Management of Real-Time Virtual Worlds. Ph. D Dissertation. 1993. Computer Science Department, Naval Postgraduate School: Monterey, CA.

158. PricewaterhouseCoopers LLP, Entertainment and Media Outlook: 2003-2007, Global Overview. 2003.

159. PricewaterhouseCoopers LLP, Global Entertainment and Media Outlook: 2004-2008. 2004.

160. Pritchard, M., How to Hurt the Hackers: The Scoop on Internet Cheating and How You Can Combat It. 2000, Gamasutra. http://www.gamasutra.com/features/20000724/pritchard_01.htm

161. Purbrick, J. and C. Greehalgh. Extending Locales: Awareness Management in MASSIVE-3. in Proceedings of Virtuality 2000. 2000.

162. Quazal, Duplication Spaces. in Quazal Eterna Documentation. 2002: Montreal, Canada. http://www.quazal.com

163. Quazal, Game Security, Hybrid Architectures, and Quazal Eterna. in Quazal Eterna Documentation. 2003: Montreal, Canada. http://www.quazal.com

164. Rabin, S., The Magic of Data-Driven Design, in Game Programming Gems, M. DeLoura, Editor. 2000, Charles River Media: Hingham, Massachusetts.

165. Rak, S.J. and D.J. Van Hook. Evaluation of Grid-Based Relevance Filtering for Multicast Group Assignment. in Proceedings of the 14th Workshop on Standards for the Interoperability of Distributed Simulations. 1996.

166. Randall, J., Scaling Multiplayer Servers, in Game Programming Gems 3, D. Treglia, Editor. 2002, Charles River Media, Inc.: Hingham, Massachusetts. p. 520-533.

167. Riley, S., Data-Driven Systems for MMP Games, in Massively Multiplayer Game Development, T. Alexander, Editor. 2003, Charles River Media, Inc.: Hingham, Massachusetts. p. 385-396.

168. Rooney, S., D. Bauer, and R. Deydier, A Federated Peer-to-Peer Network Game Architecture, in IEEE Communications Magazine. 2004. p. 114-122.

169. Rowstron, A. and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. in Proceedings of SOSP-01. 2001. New York, NY: ACM Press.

170. Sage, P.D., Customer Support and Player Reputation: It's All About Trust, in Massively Multiplayer Game Development, T. Alexander, Editor. 2003, Charles River Media, Inc.: Hingham, Massachusetts. p. 90-99.

171. Shachtman, N., 'Blizzard' of Cheaters Banned. 2002, Wired News. http://www.wired.com/news/games/0,2101,55092,00.html

172. Singh, G., et al., BrickNet: A software toolkit for network-based virtual worlds. Presence: Teleoperators and Virtual Environments, 1994. **3**(1): p. 19-34.

173. Singh, G., et al. BrickNet: Sharing Object Behaviors on the Net. in Proceedings of the Virtual Reality Annual International Symposium (VRAIS'95). 1995: IEEE Computer Society.

174. Singhal, S. and D.R. Cheriton, Exploiting Position History for Efficient Remote Rendering in Networked Virtual Reality. Presence: Teleoperators and Virtual Environments, 1995. **4**(2): p. 169-193.

175. Singhal, S., Effective Remote modeling in large-scale distributed simulation and visual environments. Ph. D. Dissertation. 1996. Department of Computer Science, Stanford University: Stanford, CA.

176. Singhal, S. and M. Zyda, Networked Virtual Environments: Design and Implementation. SIGGRAPH Series, ed. S. Spencer. 1999, New York: Addison-Wesley and ACM Press.

177. Singhal, S.K. and D.R. Cheriton. Using Projection Aggregations to Support Scalability in Distributed Simulation. in Proceedings of the 1996 International Conference on Distributed Computing Systems. 1996. Hong Kong, China: IEEE Computer Society.

178. Smed, J., T. Kaukoranta, and H. Hakonen. Aspects of Networking in Multiplayer Computer Games. in Proceedings of the International Conference on Application and Development of Computer Games in the 21st Century. 2001. Hong Kong SAR, China.

179. Sony Online Entertainment, Everquest. 2004. http://www.everquest.com

180. Stoica, I., et al. Chord: A scalable Peer-to-Peer Lookup Service for Internet Application. in Proceedings of SIGCOMM-01. 2003. New York, NY: ACM Press.

181. Svarovsky, J., Real-Time Strategy Network Protocol, in Game Programming Gems 3, D. Treglia, Editor. 2002, Charles River Media, Inc.: Hingham, Massachusetts. p. 496-505.

182. The Globus Alliance, Homepage. 2004. http://www.globus.org

183. The United States Army and The MOVES Institute, America's Army PC Game: Vision and Realization. Game Scenes Exhibition, Yerba Buena Art Center, ed. M. Davis. 2004, San Francisco.

184. Theodoropoulos, G. and B. Logan. A Unified Framework for Interest Management and Dynamic Load Balancing in Distributed Simulation. in Proceedings of the Twelfth European Simulation Symposium (ESS2000). 2000. Hamburg, Germany: Society for Computer Simulation Interna.

185. Theodoropoulos, G. and B. Logan. An Approach to Interest Management and Dynamic Load Balancing in Distributed Simulation. in Proceedings of the 2001 European Simulation Interoperability Workshop (ESIW'01). 2001: Simulation Interoperability Standards Organisation and Society for Computer Simulation.

186. Toth, V.T., A Brief History of MUD2. 2004. http://www.mud2.com/history.htm

187. Turbine Entertainement Software Corporation, Asheron's Call 2. 2004. http://ac2.turbinegames.com

188. Turbine Entertainment Software Corporation and Vivendi Universal Games, Middle-Earth Online. 2004. http://www.middle-earthonline.com

189. Van Hook, D.J., Simulation Tool for Developing and Evaluating Networks and Algorithms in Support of STOW 94. 1993.

190. Van Hook, D.J. and J.O. Calvin, A Protocol Independent Compression Algorithm (PICA). 1994, MIT Lincoln Laboratory Project Memorandum.

191. Velten, D., R. Hinden, and J. Sax, RFC-908: Reliable Data Protocol. 1984, Internet Engineering Task Force (IETF). http://www.ietf.org/rfc/rfc908.txt

192. Viega, J. and G. McGraw, Building Secure Software: How to Avoid Security Problems the Right Way. Professional Computing Series, ed. B.W. Kernighan and C. Partridge. 2002, Boston: Addison-Wesley.

193. Wayner, P., Policing Online Games: Digital Currency. 2003, Gamasutra. http://www.gamasutra.com/features/20031010/wayner_01.shtml

194. Weatherly, R., D. Seidel, and J. Weissman. Aggregate Level Simulation Protocol. in Proceedings of the 1991 Summer Computer Simulation Conference. 1991. Baltimore, MD: The MITRE Corporation.

195. Weinstein, D., Multiplayer Tricks of the Trade. in Roundtables of the 2004 Game Developer's Conferences. 2004, CMP Media LLC: San José, CA. http://www.gdconf.com/archives/2004/weinstein_dave.doc

196. Wolfpack Studios and Ubisoft Entertainment, Shadowbane. 2004. http://www.shadowbane.com

197. Yan, J.J. and H.-J. Choi, Security Issues in Online Games. The Electronic Library: International Journal for the Application of Technology in Information Environments, 2002. **20**(2).

198. Yonish, S. and J. Gordon, PCs Remain Dominant Device for Gamers. 2002, Forrester Research.

199. Zona Inc., Terazona Whitepaper. 2003. http://www.zona.net

200. Zona Inc., Terazona. 2004. http://www.zona.net

201. Zou, L., M.H. Ammar, and C. Diot, An Evaluation of Grouping Techniques for State Dissemination in Networked Multi-User Games. 1999, Georgia Institute of Technology.