Challenges in modern Distributed Interactive Application design

D. Marshall, D. Delaney, S. McLoone and T. Ward

January 2004
Technical Report: NUIM-CS-TR2004-02

Abstract

This technical report gives an introduction to Distributed Interactive Applications (DIA), including a brief background with some examples. An outline of the three classes of challenges that drive the design of DIA architectures - the users, the physical restrictions, and the required capabilities - is given. The manner in which the designer addresses these challenges, namely dynamic extensibility, scalability, interactibility and interoperability, is explored in detail in terms of the software and network architectures.

1 Introduction

The creation of the Head Mounted Display at Harvard University in 1968 [1] and the publication of Neuromancer by William Gibson in 1984 [2], inspired people to dream of a single, fully immersive alternate world that features thousands, even millions, of simultaneously interacting users. Since then, a class of application called Distributed Interactive Applications have become more widely accepted. These are software systems in which multiple users interact with each other in real-time, in a virtual environment, even though those users may be located in geographically dispersed locations around the world.

Current technology does not allow for scenarios such as fully evolved artificial intelligence as detailed in William Gibson's prophetic work. Despite this, people still enjoy connecting up with other like-minded players in games such as EverQuest and the Quake series. In addition, the three main games consoles of today, Xbox (www.microsoft.com), PlayStation 2 (www.sony.com) and GameCube (www.nintendo.com), all boast the ability for online connections. The three companies that manufacture these consoles, Microsoft, Sony and Nintendo, have described online connectivity as the greatest asset in current and future hardware and software. It is estimated that the multiplayer market alone will be worth $1.8 billion by 2005 [3].

Distributed Interactive Applications (DIAs) aim to provide a shared sense of space, presence, and consistency to users. They have become very popular with the current trend of creating "massively multiplayer" computer games, and the need for collaborative systems for areas such as engineering, education and architecture. This is discussed in section 2. With this rise in popularity, there are a number of challenges that face designers as they strive to satisfy ever-increasing user expectations. These are introduced in section 3. To meet these challenges, DIA designers focus on the software and network architectures. Section 4 gives an overview of the most important of the innovations made in the software and network architectures.

2 Background

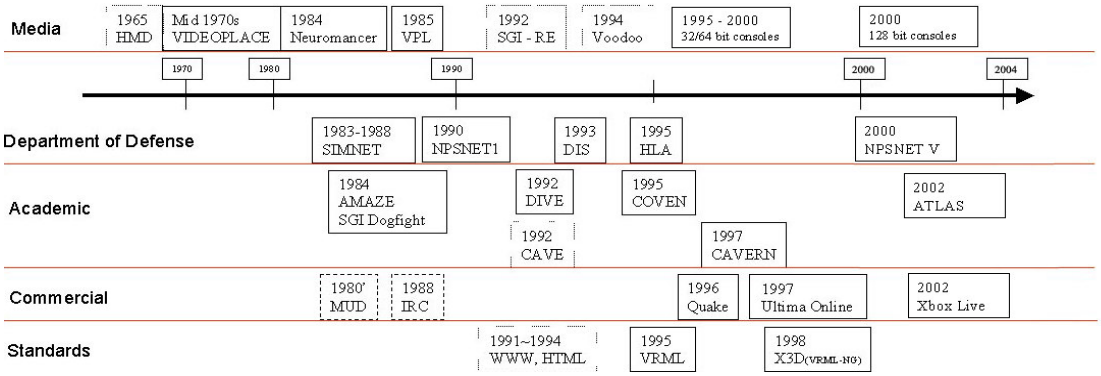

Figure 1. History of DIA's

The history of the computer industry has witnessed parallel developments in hardware speed, software development techniques and network capabilities – see Figure 1. DIA's are an amalgamation of the current state of the art of all three sectors and represent some of the most cutting edge research and development found in the interactive entertainment industry. So it is only recently that they have enjoyed wide-scale popularity.

An interesting point to note about DIA development is that its history has seen parallel growth of interest in different sectors, without the presence of much inter-sector correlation. In this sense, the DIA community consists of a number of cliques and a brief history of each of these will be given here.

2.1 US Department of Defense

The first main development from the US DoD was SIMNET, a project that began in 1983, and was finally delivered in 1990. Coming from the words SIMulation NETwork, the primary goals of SIMNET were to create a high-quality, low-cost simulator, and to network multiple simulators into one seamless DIA. The most important points to note about SIMNET was that it introduced object-event architecture, which means that objects generate update events, and that it also introduced the idea of dead-reckoning to reduce the number of these packet updates

The problem with SIMNET is that it was not generic enough to be used in other types of simulations. The Distributed Interactive Simulation Network (DIS) architecture, which was delivered in 1993, attempted to resolve this issue. It was built using the same ideals as SIMNET, but introduced the idea of a protocol data unit (PDU). A PDU is a message that is generated upon an event, for example fire, and collision. There are a set number of PDUs, but they are designed to be generic enough to deal with updates from all types of participants in the environment. DIS also extended the dead reckoning mechanisms, and the DIS standard defines nine dead reckoning algorithms.

2.2 Networked Games

One of the first games that incorporated networking was Flight, development of which began in 1983, although networking was not added until 1984. Flight ran on SGI workstations, and was developed at Silicon Graphics Inc. Users could see each other, but not interact. This issue was resolved with the release of Dogfight, which was a modified version of Flight.

ID Software (www.idsoftware.com) has been largely responsible for the current interest in network gaming, with the release of Doom, and more importantly the Quake series. Released in 1996, Quake was the first game to provide a true six-degrees of freedom environment for users to interact in. Combined with the blistering pace of the so-called "deathmatch", it became an instant success.

Currently, interest focuses on multiplayer games which have spawned a new genre known as Massively Multiplayer Online Role Playing Games (MMORPG) [4]. In these games, players cooperate with thousands

of others in one environment to perform tasks. Character development is one of the more important issues to players of these games, but some games such as Ultima Online (www.origin.com), have a social structure, in which people can join guilds, form relationships and learn a useful trade.

2.3 Academic

The academic arena has also seen the development of a number of distributed interactive applications. Like the Department of Defense simulations, these systems tend to concentrate more on the software implementation issues, such as consistency management, scalability, rather than the graphics, sound and control aspects that are of more interest to commercial developers. One of the more important, and longest running, academic groups is the NPSNET Research Group. This group covers all areas of research, including consistency, scalability, dynamic extensibility, interoperability and composability [5]. The latest incarnation of their project is NPSNET V. Other academic implementations of note include: SPLINE, developed at Mitsubishi Research Laboratories, which introduced the idea of subdividing the world into smaller, more manageable areas known as locales [6]; DIVE, from the Swedish Institute of Computer Science, concentrates on consistency and concurrency [7] ; and from the Collaborative Distributed Systems and Network Laboratory we have ATLAS II, which introduces the ideas of personalised information filtering and self-reconfigurability [8].

2.4 Summary

This brief introduction highlights the fact that each DIA research group concentrated its efforts on different problems, driven by particular end-user requirements. However, there are a number of common challenges that affect all DIA designers. These challenges will be introduced in the next section and it is shown how they affect the design of all DIA's regardless of implementation requirements.

3 The DIA Design Predicament

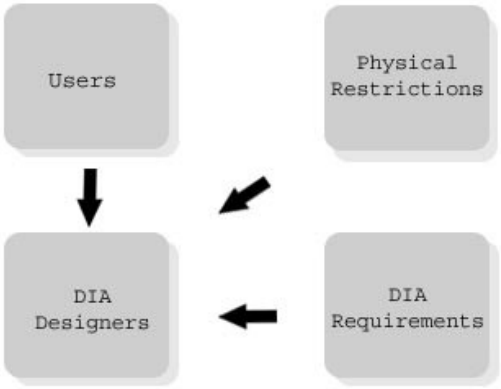

Figure 2. The challenges facing designers are multi-faceted

Designers of DIA's face challenges from three distinct sources: the users, the physical system,, and the basic requirements of the DIA itself - see Figure 2. These challenges drive the creative and technical options available to designers of current and future systems alike. In the following paragraphs we will explore each of the challenges in detail.

3.1 Users

Impressionism changed people's appreciation of art from admiration of the technical achievement of painstaking realism to an appreciation of an artist's style and mood. In the same manner, users now look towards systems such as DIA's to provide an experience rather than just admiration of the visual and aural accomplishments. These users have certain expectations of how the system should behave and the level of performance the system should offer. In short, users have five main requirements from a DIA.

(i) Awareness
Users need to be aware of other participants, the environment, and objects within that environment so as to experience a shared sense of space with others.

(ii) Responsiveness
Responses to their commands should be immediate and should also be consistent with their expectations. This provides a shared sense of time.

(iii) Consistency
Users need to be sure that what they are experiencing is consistent with others, in order to give them a shared sense of presence.

(iv) Security
Users should be sure that any communication that is taking place within the environment is secure. This security deals with issues ranging from users masquerading as others, to secure transmission of personal data.

(v) Fidelity
Users have a certain expectancy of the fidelity of the representation of a virtual environment that the application delivers. For example, if the environment is a representation of a real world locale,

then the look and feel of the environment should be consistent with users' previous experiences with such an environment.

Although not a standard metric, there are some who consider the end-user's experiences to be the yardstick by which computer systems can be evaluated [9]. The purpose of any interactive application is to provide an experience within a virtual environment, and if any of the above user requirements are not taken into account, then the system could be considered a failure.

3.2 Physical Restrictions

(i) The Network

The network, which is at the core of a DIA, is one of the most prominent restrictions. Bandwidth, the rate at which the network can deliver data, is limited. The delay between sending the information and receiving it at the other end of the connection, which is known as latency, can be the thorn in the side of many designers. The variation in latency times, known as jitter, makes any solution to the latency problem even more difficult to implement. During transmission, packets can be dropped or lost by the network. The measure of packet loss is known as reliability, and an unreliable network can lead to inconsistencies, thus hampering the user's sense of shared space and presence, and to extra delays caused by retransmissions.

(ii) Heterogeneous Nodes

Designers also have to deal with systems that are heterogeneous in terms of operating system and processing power. This problem is similar to that relating latency and jitter, in that if all participating nodes had similar capabilities, then the architecture could be constructed around this limitation. However, given the potential disparity of computing resources among participants, the system architecture has to deal with providing experiences of varying fidelity to users.

(iii) The Users

As previously discussed in section 3.1, users have certain expectations of the system, and these expectations can be considered a physical limitation to the system. Designers have to deal with users that can, and will, react intelligently to what they perceive in the simulation. Given certain situations, one can almost guarantee that although some users will react similarly in the long term, the majority of short-term inputs will be random. This makes user behaviour very hard to model, and therefore, difficult to provide for in the design of the system.

3.3 DIA Requirements

There are four basic requirements that have become standard for all DIAs and are now all active research areas. The Naval Postgraduate School, creators of the NPSNET project, has identified extensibility, scalability, and composability (which incorporates interoperability), as the main infrastructural requirements of a large-scale, persistent online virtual-world [10]. We also believe that interactibility is another essential component that must be implemented correctly and efficiently in order to provide for complete user satisfaction.

(i) Interoperability

These are the mechanisms that allow heterogeneous implementations of DIAs and homogeneous implementations of DIAs on heterogeneous operating systems, to interoperate.

(ii) Interactibility

The interactibility components provide the user with means to perceive and interact with the environment around them. More than just providing aural and visual feedback, these components deal with management of events outside of the user's control.

(iii) Scalability
The potential for a system, or an aspect of the system, to continue to function effectively as the processing requirements and the number of simultaneously participating users of that system increases is known as scalability.

(iv) Dynamic Extensibility
Dynamic extensibility is the ability of a system to add or change components at run time without the need to take the system offline, or recompile it in its entirety.

3.4 Summary

We have outlined the challenges and restrictions imposed on the designers of DIA systems. But what can the designer do to create a system that satisfies user requirements in the presence of such limitations? If a designer were to produce a system that is used exclusively by other DIA designers, what main components would be included? The next section will detail how designers meet the conflicting challenges imposed by users, by the physical system and by the basic design requirements in the development of a DIA.

4 Current Solutions to the DIA Design Predicament

Much research has been devoted to the development of techniques for graphics, physics, networking and user input, with the aim of providing the user with a truly believable and fully interactive virtual environment. To this end, many teams are developing all-in-one engines that allow other developers to deal with the design of actual game play. This should lead to a better game playing experience, rather than spend time and money on technology development. Examples of these teams include ID Software, with the Quake series, and Epic Games with the Unreal engine (www.epicgames.com).

These have already been the focus of much previous attention and have been extensively documented [11] [12]. This technical report will therefore concentrate on those topics that we have identified as the key requirements of future DIA's: Dynamic Extensibility, Interoperability, Interactibility, and Scalability. The following sections will discuss how designers have striven to satisfy these requirements by manipulating the software and network architectures.

4.1 Software Architecture

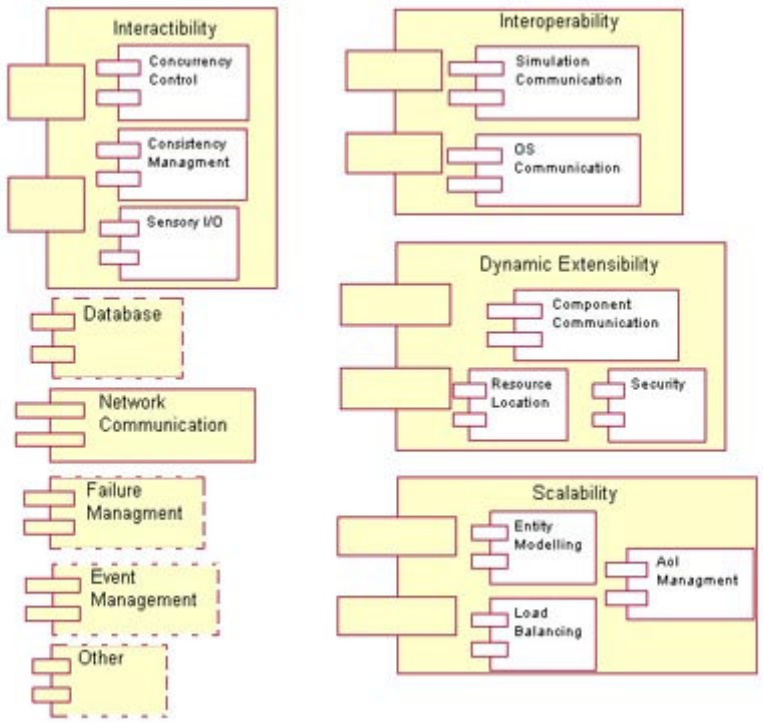

Figure 3. The Software Architecture

The Software Architecture of any system refers to the "structural elements and their interfaces of which the system is composed" [13]. Examples of a structural element in a DIA include: rendering, audio, physics, and networking. This software architecture is that which resides on the host machine and uses the network architecture to communicate with other participating software architectures.

It is difficult to describe software architecture of DIA's in generic terms, as it is always application specific, although efforts have been made to categorise essential components in order to provide a taxonomy of existing architectures [14]. The software architecture can vary from the micro-kernel of NPSNET V, with all components loaded dynamically at run time, to a system like DIVE [15] or MASSIVE [16], where every component is hard coded.

Figure 3 shows a generic software architecture of a DIA. As already mentioned, many areas of the software architecture, such as those with dashed outlines in figure 3, have been, and continue to be, the focus of much research and discussion. Hence these elements will not be the focus of this technical report. Instead, the next sections will detail dynamic extensibility, interoperability, interactibility and scalability as well as the associated software and network architecture components required for their implementation.

4.1.1 Dynamic Extensibility

Due to their open-ended nature and the fact that collaboration amongst users rather than achievement of a final goal is one of their primary uses, the lifespan of a DIA is usually far greater than that of a typical virtual environment. In the future, it may be possible that popular DIA's will run over the course of many decades. Even today, Ultima Online, which was released in 1997, still has a large user base.

Over time, technology becomes outdated. Ultima Online is six years old. It cannot begin to compare to modern leaders in the genre graphically and aurally and many users complain about the effects of network latency and reliability on the game play. Modern methods would surely help to deal with these issues but this product, like many others, is considered static or averse to change.

A common technique currently used for updating software is the so-called 'patch'. A patch is a revision of a software product that is generally used to resolve issues that were not dealt with before the release of the product. But with possibly thousands of players partaking in a simulation at once it would seem unreasonable to pause services so as to perform updates and fixes, add new components, or propagate the patches around the user base.

Any software system that consists of components would be considered to provide a limited degree of extensibility. If these components are well designed, then they could be interchanged for ones that have different functionality. For example, by keeping the rendering component separate from the main design, various rendering methods can be used.

Other systems offer dynamic extensibility in the form of updateable data and entities. The SPLINE architecture utilises URLs to let participants download descriptions of new, previously unencountered, entities [17]. By updating locales in MASSIVE, the environment can be changed at run time [8].

The examples above cannot be viewed as components of the software architecture as they rely on adapting user perceptions of the virtual environment, rather than extending and changing core components of the system. For a system to be considered truly extensible, mechanisms must be provided which allow for most, if not all, of the software components to be extendable at run time.

Dynamic extensibility opens up a new realm of opportunity to DIA designers. Time constraints will no longer hold back the development of features for the systems, and the lifespan of these systems will be greatly increased. It could also lead to a better user based community for applications, with users taking the time to develop their own modifications for systems in order to improve on its original implementation.

Below, the core sub-components of dynamic extensibility are detailed.

(i) Resource Location

Resource location is the retrieval of information or modules that the system needs to interact and operate within the environment. These resources could be stored locally, for example on another permanent storage device such as a hard drive or on another user's machine or a database located on the network.

In a dynamically extensible system there must exist a method for participants to discover components that need to be loaded and registered. The best manner of achieving this is to have a central repository, for example using LDAP (Lightweight Directory Access Protocol) found in NPSNET V, which is available to all the users [5]. This repository would either store all available components or keep a reference to the location of components.

A resource-locating component is fundamental to the operation of the dynamic extensibility system. If it failed to function then no new components could be loaded and systems would be unable to find the necessary bootstrapping components. For this reason, the repository must be easily accessible and have provisions for fault tolerance and load balancing.

(ii) Component Security

In a completely dynamically extensible system all components would be loaded at run time. Sometimes, these could be loaded from a locally cached copy. However, if they are being remotely accessed, how can

the system be sure that a rogue component is not masquerading as one officially recognised by the system. The malicious components could perform actions ranging from adversely affecting user performance to stealing user's personal information. These components could also be indirectly malicious, due to malfunctioning code or an inability to communicate effectively with other components, thus causing system performance degradation.

Therefore, a security component must be used to perform validation checks on components that are being dynamically loaded in order to check that they operate in an expected manner with no failures, and will not cause any damage to the user's system or the user's experience of the simulation.

(iii) Component Communication

This component allows for all components of the architecture to communicate with each other. Using the building analogy again, this component is the 'cement' of the system. This communication module falls in the realm of dynamic extensibility as there needs to be a common interface for all member components to interact with each other. This is important in a static system but is vital in a dynamically extensible one. Resource location and error and failure checking of the resource are costly processes in terms of processing and time. Therefore, when a component is dynamically added, one needs to be sure that it can interact with others efficiently and correctly.

(iv) Examples

Perhaps the most pertinent research in this area is taking place at the Navy Postgraduate School with NPSNET V. The NPSNET project has been running for many years, with NPSNET V as its latest iteration. Based entirely upon the Java (www.sun.com) platform, the most important aspect of this architecture is that at its root lies an invariant microkernel, and the abstract base class Module, that all components must extend [18]. By the use of configuration files written in XML, the microkernel can load modules at runtime. These configuration documents act as a 'glue' to bind components together, concisely describing the relationships between components.

The extensibility component of ATLAS-II is also one that merits discussion. The group at CDS&N labs have identified extensibility as an important part of a DIA's architecture but consider it essential that the system be able to extend itself automatically, rather than with explicit human intervention. This technique is known as self-tunability [8]. When system degradation is detected or a system reconfiguration message, written in XML, is received the ATLAS II architecture, using a component known as the Resource Discovery Manager, locates the new component to be loaded. This component may be found locally or on external peers or servers. The component is initiated and then registered with the proper ATLAS manager.

4.1.2 Interoperability

Interoperability refers to the ability to exchange and use information in a large heterogeneous network. There are two main classes of interoperability:

- The ability *of heterogeneous implementations* of DIA's to communicate.
- The ability of single implementations of DIA's running on *heterogeneous operating systems* to communicate.

With the increased use of non-uniform operating systems, DIA's will have to be designed to cater for each operating system. Most current software comes with some element of connectivity, so this problem is not just exclusive to DIA's but is apparent in all areas of computer science research and industry.

We will now discuss the implementation of components to deal with the two classes of interoperability.

(i) Heterogeneous Simulation Communication

This component is considered the less important of the two. In this implementation, a common interface would be needed between all systems to enable them to communicate reliably with each other. In most cases this could be unfeasible and not very worthwhile as it would limit what designers could achieve with their system and most commercial applications would have no desire to communicate with each other in

any case. The only really practical use for this component would be for a large organisation with many departments to provide a common means for applications in each department to communicate.

A prime example of this is the High Level Architecture [19] , developed in the US department of defence (DoD). The DoD features many departments, for example the Navy, Army and Air Force. Each department had its own distributed interactive system for simulations. Maintaining all of these separate simulations proved costly in terms of money and time. The Defense, Modelling and Simulation Office (www.dmso.mil) was founded, whose goal was to develop an architecture that promoted interoperability and re-use. It is based on the premise that no one simulation can satisfy all uses. Therefore, a composable set of interacting simulations should be used. Under the HLA definition, a single simulation is known as a *federate*. Groups of federates that can interoperate form a *federation.* Each federation also makes use of what are known as "Object Model Documents". Object models are descriptions of the essential elements of the federation in terms of the participating objects. Any federate which wishes to interoperate with a specific federation, must comply with the implementations described in these object model documents.

(ii) Heterogeneous Operating System Communication

With an increasing number of people using operating systems other than Microsoft Windows (www.microsoft.com), a new challenge is presented to DIA designers. If systems are to be created which are cross-platform, yet aim to interoperate with each other, then a common communications interface must be created. This means that even though implementations of the simulation, implementation programming language and methods of message encoding are different due to the architecture of the platform, there must be a common format that allows heterogeneous platforms to interact.

This component deals with providing methods for these operating systems to interact. Mechanisms must be in place to convert network messages to the proprietary format used by various operating systems. This must be performed reliably and in a consistent manner amongst the various operating systems, to ensure that participating systems interpret equal meanings from equivalent messages.

The most popular format in use for this type of application is known as extensible mark-up language, or XML. Already operational in ATLAS II and NPSNET V, XML is what is known as a meta-language, or a language about languages. It is a method of describing data using plain text, rather than using a platform dependant binary representation. One of the more important developments from XML is SOAP, which stands for Simple Object Access Protocol [20].

SOAP uses XML and HTTP to provide a means of transmitting data across a network. As SOAP deals with the conversion of data from XML to a format that is usable by the environment, designers only have to worry about *what* data to send, rather than *how* to send it.

Efforts are being made to integrate SOAP-like mechanisms into DIA systems. The Dynamic Behaviour Protocol [5], found in NPSNET V, facilitates the dynamic loading of network protocols. XML is used as a method of describing the syntax of the new protocol. At runtime, an engine reads this XML file and uses the gathered information to extract data from the binary format network messages.

4.1.3 Interactibility

The interactibility components aim to provide users with a satisfactory means of interacting with the environment. Interactibility includes both the visual and aural outputs from the system and the mechanisms by which users provide inputs to the system. In addition this component deals with more complex issues such as concurrency and consistency control.

The interactibility component underlies the experience of the users in the DIA. If interactability issues were implemented haphazardly, then the user's shared sense of space, presence and time would be lost and the system would be unresponsive, unusable and impractical.

(i) Sensory I/O

Containing drivers for graphics, audio, haptic output and user input, this component deals with providing users with a view of the current environment, as well as allowing the user to react and issue changes upon what they perceive. All modern games consoles feature a form of haptic output with force feedback controllers and provide aural immersion with three dimensional surround sound. For example, ID

Software has constantly redefined what users can expect graphically from computer games with the Quake series and the upcoming Doom III.

(ii) Concurrency control

Concurrency control is "the proper management of simultaneous data updates when multiple users or multiple tasking occurs"[21]. The concurrency control component allows for the resolution of simultaneous actions or what is known as conflict resolution. For example, in a virtual environment, when two separate participants are contending for one object, decisions have to made as to which participant should become the rightful owner. Most distributed interactive applications employ either a pessimistic or optimistic concurrency control scheme. The former guarantees consistency at the expense of high communication delays while the latter allows updates without conflict checks, which may lead to the undoing and redoing of previous user actions [22].

This component requires the presence of some time maintenance mechanism, such as Network Time Protocol, virtual time or, at the very least, a scheme that ensures that each communicating participant has a clock that can be synchronised intermittently with other participants. As with consistency below, concurrency control can have a direct negative impact on the scalability of a system. If a system is fully concurrent it means that all messages have to be delivered in the correct order to all users. This requires the use of a protocol such as TCP/IP, which guarantees eventual delivery. In a fully concurrent system, the simulation cannot continue until all users have received all update messages, which can lead to delays for all users involved.

(iii) State consistency management

Due to physical restrictions such as latency and poor reliability, all DIAs are inherently inconsistent. The focus is therefore on best effort consistency, or what is known as controlled consistency. Therefore, the main objective of this component is to provide users with a uniform view of the environment, rather than a fully consistent one. For example, in an inconsistent environment, entities may appear in an area, when in fact they have moved or even been removed from the environment. It ruins the sense of awareness for the user and also makes the system incoherent. However, maintaining consistency can have a dramatic effect on scalability of the system and also can have implications on latency performance. It leads to a fundamental rule about DIA shared state, known as the *Consistency-Throughput Tradeoff* [12]:

> It is impossible to allow dynamic shared state to change frequently and guarantee that all hosts simultaneously access identical versions of that state.

In simple terms, this means that it is impossible to have a dynamic environment with high update rates and also have guaranteed consistency amongst users. An example of this is found in the SPLINE architecture. Under this system, known as the relativity model, users will receive updates about events some time after the event actually occurs. This system uses only approximate consistency to maintain a high level of communication between participants.

Another part of this component is data replication, which is inherently linked with state consistency management. As mentioned previously, a DIA aims to provide users with a shared sense of space, presence and time. To achieve this, each user must at least have the same representation of the environment available to them. Data replication deals with faithfully transporting all relevant details of the environment to all interested users.

4.1.4 Scalability

Scalability is a measure of the number of entities that may participate simultaneously in a system. When a DIA system is considered scalable, it means that it can handle an increasing number of users with little or no performance degradation. Being scalable is a very important feature of any multi-user environment. For example, although the DIVE architecture is very capable in certain areas such as concurrency, its use is limited by the fact that it difficult to scale beyond 16 to 32 participants. Scalability depends on a wide range of factors.

Capacity of the network: It needs to be able to handle the throughput of information on the network. The bandwidth requirements may vary across the DIA; for example a server requires a high bandwidth but a single user will need much less. Several methods exist for reducing the number of transmitted packets. These will be described below.

Capacity of the participant: The participants hardware, for example processing, rendering, and connection speeds, needs to be adequate to deal with the information been parsed by the client. However, in a similar fashion to the network example above, if the network transport is handled intelligently, then the connection speed is not as much of an issue.

Capacity of servers: Their speed and throughput must be adequate to deal with the number of users required on the system. If the load exceeds their means, provisions must be in place so that the load can be spread amongst other servers.

Designing a fully scalable system is not easy. How can a system be scaled indefinitely when resources such as bandwidth, computational power and money are limited? When designers wish to start guaranteeing consistency and concurrency between users, which is vital in some cases, it also leads to scalability problems. Scalability will always remain an issue in DIA's. However, there exist a number of novel and important mechanisms that can help dramatically with creating a scalable system, some of which are now detailed.

(i) Area of Interest Management

DIA environments can be very large. For example, each zone in EverQuest is between five and eight simulated kilometres squared. In the Playstation 2 version of the game, there are in the region of 160 zones.

Although dealing with environments of these dimensions may seem like an insurmountable task, it can actually be of benefit to the designers. In any large environment, real or virtual, a participant is only interested in a subset of the environment. Participants are only interested in updates that directly affect them at that time - for example, someone entering their field of vision, a sound from behind.

By using this property in a scheme known as relevance filtering or area of interest filtering, designers can greatly reduce the volume of data transmitted between users of the environment. Many methods exist but some are more application specific than others [23]. The most common metrics of relevance are:

Geographical location: The user is interested in data being transmitted in its geographical area in the virtual environment.

Visibility: When a new entity enters a user's field of vision, updates need to be sent regarding that entity to the user

Audibility: In a similar fashion to the visibility metric, users need to receive updates from entities that they could possibly hear.

Another novel method is the use of *interest expressions* [24] which allows users to specify their current area of interest using a set syntax. Interest managers accept these expressions and use them to filter messages in accordance with user needs. This allows for a high granularity of relevance filtering. For example, an aircraft could express interest in receiving updates from all other aircraft and large ships but not from soldiers, within a certain region.

(ii) Entity Modelling

Some entities within the virtual environment are bound by the rules of the environment. These are usually simulation-controlled objects, or what are known as Non-Player Characters. Even though they may use sophisticated artificial intelligence mechanisms, their knowledge and behaviour is still limited. This makes these entities very easy to model because given a starting condition, or a certain situation, it is easy to model the reaction of an entity and predict the future actions of that entity. For example, consider a

rocket fired in Quake. It will only travel in a straight line from initial point to where it finally explodes on contact with an object or player. Therefore, all that theoretically needs to be sent to other users is the starting point of the rocket and its direction. The only other information that is required is if the rocket comes into contact with a dynamic entity, for example another user. Otherwise, each participant will correctly model the path of the rocket from firing to collision with another static object e.g. a wall.

(iii) Dead Reckoning

First used by sea-faring adventurers for accurate navigation, the dead reckoning method is an important subset of entity modelling.

Dead reckoning literally means "navigation without the aid of celestial observations". To make the definition more relevant the words "celestial observations" could be replaced with "packet updates". In the dead reckoning method, participants use information from previously received packets to predict future behaviour of other participants. Update packets are only transmitted when the error between the actual participant behaviour and the predicted behaviour differ by a certain threshold amount. This means that update packets need only be sent when necessary, resulting in far less network traffic. Pioneered in SIMNET, dead reckoning consists of two main elements, namely prediction and convergence.

***Prediction**:*

This is the method used to compute the current state of an entity using previous information relating to the entity e.g. velocity, direction. The most common method of achieving this is via *derivative polynomials* [12], although methods exist which are less generic but perform better in specific situations. e.g. the phugoid scheme for modelling aircraft [12]

A derivative polynomial can be formed using a formula involving various derivatives of the entity's current position. For example, the first derivative of an object's position is velocity, and the second is acceleration.

Second order derivatives are the most popular technique in use, as they are easy to understand, fast to compute, and provide sufficient predictions of entity position. DIS, the follow up to SIMNET, uses second-order prediction for its dead reckoning algorithms.

Research has shown that using higher-order terms exemplifies the law of diminishing returns - that more effort provides progressively less impact on the overall effectiveness of a particular technique. As terms are added to the polynomial, more calculations are needed, and more information needs to be sent in order to provide correct prediction. This is in direct conflict with the goal of the method – reducing network traffic.

Convergence:

Convergence defines how the state of an entity is corrected on deviation from the computed prediction. Better convergence algorithms correct the variation in position quickly without creating noticeable visual distortion to the user.

Convergence is achieved using curves of varying complexity. Zero-order convergence is known as "snap convergence". The idea is that the modelled entity's position is immediately transformed to the correct position. Although simple to implement and compute, it performs worst in the area of creating visual anomalies.

In linear convergence, the first-order method, a *convergence point* is picked, based on the entity's correct state. The convergence point lies somewhere along the trajectory of the predicted path of the modelled entity. The entity then moves in a linear path towards this point. Although not as visually jarring as snap convergence, this method can result in strange movement, as the entity will suddenly change direction to move to the convergence point, and change again to follow the newly predicted path.

The second order method uses a quadratic curve to show motion between the incorrect path and the convergence point. It helps alleviate some of the problems of the linear method, but it can result in visual distortion when the entity reaches the convergence point, and suddenly changes direction.

This problem can solved using a cubic spline. A point is picked a short time before the current position on the incorrect path, and a short time after the convergence point on the newly predicted path. The entity then travels along the cubic spline created using these points, resulting in smooth motion

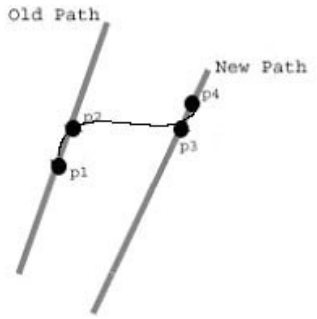

Figure 4. Cubic spline convergence

In figure 4, p2 represents the current point, p4 the convergence point, and p1 and p3 represent the point before the current point, and the point before convergence point respectively. The entity moves along the smooth cubic spline path between all the points.

(iv) Strategy Modelling

Users are much harder to model than computer-controlled entities such as that described in the above paragraph. An intelligent reaction is what differs users from these entities. However, research has shown, that given a set goal in an environment, most users will follow the same set of strategies to reach that goal [25]. By observing the strategies used in this environment, a library of possible strategies can be created based on these observations.

During normal simulation users are monitored as they move around the environment. Their behaviour is compared with the stored models, which are available to all participants, and a particular strategy is chosen from the library. As long as the user follows the strategy, there is no need for updates to be sent to other users, as the behaviour can be successfully predicted. Updates need only be sent when users change strategy or use no recognisable strategy at all.

(v) Load balancing

In general, load balancing is heavily dependant on provisions made in the network architecture, for example the presence of backup servers. However, components must exist in the software architecture to provide for load balancing mechanisms. These components are concerned with distributing the load of a system among the available resources in a manner that improves overall system performance and maximises resource utilisation. This includes performance degradation detection, idleness detection so as to offer any unused resources to loaded systems, and the seamless transfer of users between servers.

This must be performed seamlessly to avoid upsetting the users' experience of the system. Servers could be used to compress data, so as to reduce packet size, or perform packet bundling. This is a method of reducing the amount of messages being sent on the network and also delivers smoother packet rates.

4.1.5 Summary

The above sections have discussed the challenges facing DIA designers in terms of the software architecture. We have chosen to represent each of the four designer requirements as components of the software architecture, and described in detail the sub-components that provide the functionality of these requirements. The next section will detail the elements present in the network architecture that can deal with providing these four requirements.

4.2 Network Architecture

The network architecture is the communication aspect of a DIA. It reflects events at the communication level in the application, or software architecture, level [26]. The choice of architecture is vital, as usually the design of the rest of the application is dictated by it. It is important to note that although the network architecture implies the existence of a physical network, it can be used as a model to describe the interaction of components that reside on one machine. For example, many computer games that feature

both single and multiplayer aspects can use the client/server architecture for both implementations. In the single player experience both the client and server reside on a single machine with the 'network' being the function calls between the two. The game can then be easily ported to a network with the server residing on another machine.

The next sections will discuss various aspects of the network architecture and the manner in which they can be used to satisfy the designer's requirements detailed in section 3.3. The suitability of each network architecture in addressing the four key DIA requirements is indicated by an 'X' to indicate 'not suitable' and a tick to indicate 'suitable' at the start of each paragraph.

4.2.1 Client/Server

Dynamic Extensibility	Scalability	Interactibility	Interoperability
✘	✘	✔	✘

The client/server model is a classic approach to network architecture (see Figure 5) and is used extensively in most popular multiplayer computer games. The architecture itself consists of a single server, with all inter-client communication taking place through this server. The server remains the final arbiter of all simulation [27], which makes it very useful for commercial developers who wish to maintain total control over the environment in order to prevent cheating.

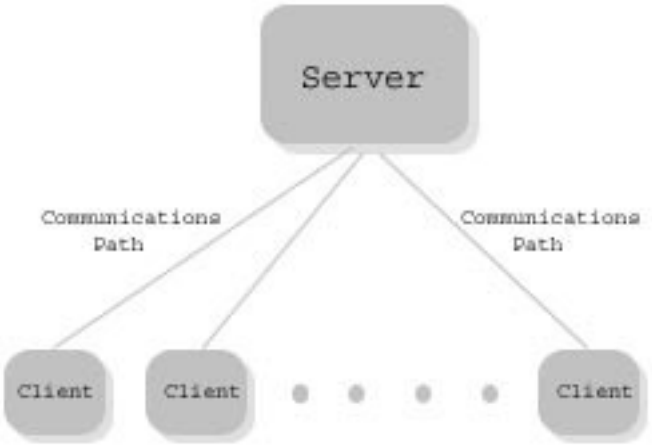

Figure 5. Client / Server Model

As the server is considered to maintain absolute truth by the clients the consistency and concurrency features of interactibility can be implemented with relative ease. In every time frame, the server receives all user updates, considers them in relation to the simulation and then decides whether or not a particular update can be made and the order in which the updates are carried out. This information is then relayed back to each client. Even if the client had a different representation of the environment, due to prediction mechanisms, their depiction must be altered so as to correspond with that of the server's.

The architecture, however, is not considered to be very scalable. The server is the single point through which all traffic flows and, therefore, is the likely source of a bottleneck leading to performance degradation for all users. Testament to this fact is that modern day computer games that rely on this architecture can only handle in the region of 8 - 64 players, depending on game genre.

4.2.2 Peer to Peer

Dynamic Extensibility	Scalability	Interactibility	Interoperability
✘	✘	✘	✘

In a peer-to-peer architecture servers are eschewed – see Figure 6. Each host directly communicates its update information to all other participants in the virtual environment. Usually with this method hosts must keep a full or partial copy of the current state of the environment. Since there is no one member with an authoritative representation of the environment (e.g. the server in the client/server model), interactibility mechanisms can be difficult to implement and usually require some form of global time keeping mechanism, such as the Network Time Protocol, or at the very least, a reliable time-stamping method for update messages.

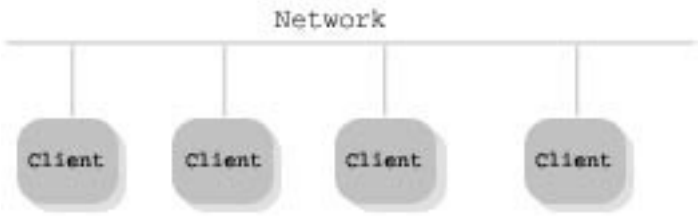

Figure 6. Peer to Peer architecture

This architecture can be more efficient when implemented using multicasting. Multicast uses unreliable UDP packets to communicate with multiple users simultaneously. However, support for multicast communication in the existing Internet is poor [7]. Peer to peer architectures also suffer from scalability problems; overall bandwidth use scales with N^2, where N is the number of participants [10].

4.2.3 Multiple Server Systems

Dynamic Extensibility	Scalability	Interactibility	Interoperability
✘	✔	✔	✘

Similar to the client/server architecture, this model, as the name suggests, features multiple servers for the clients to communicate with – see Figure 7. As with the client/server model these servers maintain the most up to date representation of the environment and therefore this architecture provides for good interactibility.

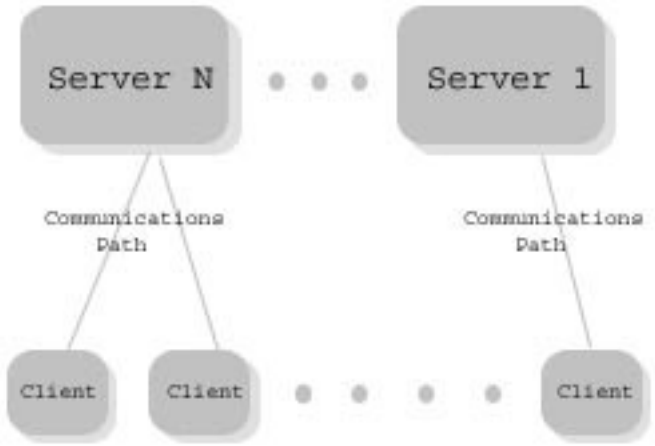

Figure 7. Multiple Server Architecture

If used intelligently, this system can provide a far more scalable solution than its single server predecessor. One of the most common implementations using this architecture is the use of both primary

and backup servers in the system. If the primary server suffers a system failure then the backup could be used in its stead, providing more reliability and scalability than the client/server model.

An alternative architecture is that used by SPLINE [17]. Although communication between nodes is peer-to-peer there are four types of server used in the architecture: a session server, to handle new connections; a server to handle users with slow connections and interact with them using the client/server model; a locale update server to provide information to users regarding the new locale they have entered; and a name server for the easy location of entities within the world. By delegating responsibility of these tasks amongst dedicated servers that are separate to the main simulation scalability is increased.

Due to its foundation in the client/server model described above, the multiple server architecture suffers from the same inherent flaws as the single server model, namely bottlenecks and single points of failure. Although it can be more scalable and reliable than the single server implementation, problems can still arise from these issues.

4.2.4 Coordinated Multiple Servers

Dynamic Extensibility	Scalability	Interactibility	Interoperability
✗	✓	✓	✓

The main feature of this architecture is that, in addition to client-server communication, each of the servers is able to communicate with each other – see Figure 8. Allowing communication between servers offers the most freedom to designers in terms of the future challenges. For example, scalability is greatly improved if each server governs a geographical section of the virtual environment. When a user wishes to move from one area to another the relevant server is given the details of the new arrival from the old server and the user can seamlessly move from one section of the map to another. A better implementation of this architecture is found in Asheron's Call (www.microsoft.com). In the example above, if all the users were congregated on one server then the others would be just idle while one server is overloaded. In the Asheron's Call architecture, if the area governed by a server is overloaded then the area load is distributed among multiple servers, thus making the system more scalable and reliable while not wasting any resources on idle servers [28].

This architecture is one of the only means of providing for a level of interoperability, other than with protocols, on the network level. If each server is responsible for a single simulation then each simulation could be handled independently whilst still allowing for communications between simulations via the servers.

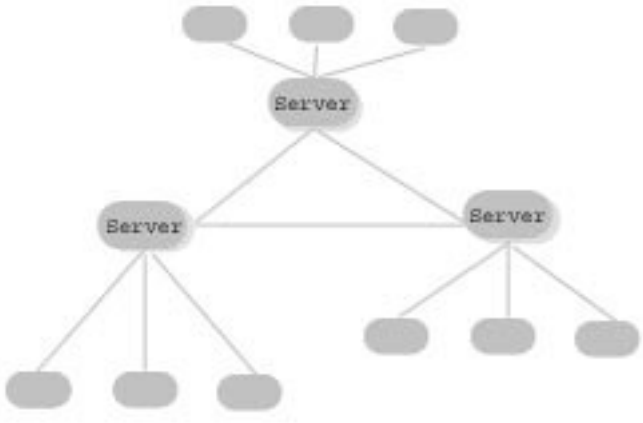

Figure 8. Coordinated Multiple Server Architecture

4.2.5 Quality of Service

Dynamic Extensibility	Scalability	Interactibility	Interoperability
✘	✔	✔	✘

Each network service can be characterised by a Quality of Service (QoS) [29]. When a provider promises a certain QoS, they are providing certain guarantees relating to the quality of network connection that a user of that service may expect, including high bandwidth and low latency and jitter. If the provider fails on this agreement there can be financial repercussions.

The performance of a system in relation to scalability and interactibility is directly related to the network performance. So if the system is analysed and the minimal requirements for the system to provide a scalable and interactible solution are identified, a certain quality of service could be agreed upon with service providers, which will have a direct implication on overall system performance.

4.2.6 Active Networks

Dynamic Extensibility	Scalability	Interactibility	Interoperability
✘	✔	✘	✘

In general, the network is considered to be a 'dumb' system. Its primary function is end-to-end routing of traffic between users with all computational resources located outside the core of the network. The idea behind active networks is that by placing computational resources directly within the network to support user processing, increased performance can be achieved [30].

This can lead to better system scalability. For example, if routers are endowed with abilities to process information, rather than simply forward it, mechanisms such as area of interest management can be performed at a network level, which would lighten the processing load at host and server machines.

4.2.7 Protocols

Dynamic Extensibility	Scalability	Interactibility	Interoperability
✘	✔	✔	✔

A network protocol describes the set of rules that applications use to communicate with each other [12]. It consists of three main components

> ***Packet format:*** This describes the layout of each type of packet. It tells the sender what the packet should consist of and informs the receiver how to parse the incoming packet.
>
> ***Packet semantics:*** Typically described using a finite state machine, packet semantics detail the appropriate action the recipient takes upon the delivery of a packet. For example, in the SPLINE architecture, a potential participant contacts the session manager to enter the virtual environment, upon which the server responds with a ticket granting access to the world.
>
> ***Error behaviour:*** Linked with the failure management and security components described above, error behaviour details the actions of the recipient or sender given an erroneous message, or problems limiting the participant's ability to communicate effectively. Given the nature of this area, it can be described using a finite state machine, similar to packet semantics.

Although thousands of protocols can be used in a DIA, the most useful are those that can be utilised for the transmission of information from the environment. The four best examples of these are Transmission Control Protocol (TCP), User Datagram Protocol (UDP), IP Multicast and IP Broadcast. Table 1 describes the main characteristics of these protocols and how they can be used to meet the designer requirements.

Protocol	Characteristics	Strengths	Limitations
TCP	Point to point transmission, reliable transmission, provides idea of a connection between two end points.	**Interoperability** – TCP is a widely recognised protocol, so nearly ubiquitous acceptance. **Interactibility** – Reliable transmission means TCP is useful for guaranteeing concurrency.	**Scalability** – TCP has a large bandwidth overhead. The guaranteeing of ordering may delay the deliverance of packets.
UDP	Unreliable, lightweight packet size	**Interoperability** – UDP is a widely recognised protocol, so nearly ubiquitous acceptance. **Scalability** – Small bandwidth overhead. Immediate delivery as there is no error checking.	**Interactibility** – UDP has no reliability or ordering guarantees. Corruption of data is possible due to lack of a checksum for error checking.
IP Multicasting	Unreliable transmission to multiple hosts whose individual identities are anonymous to the transmitter	**Scalability** - Small bandwidth overhead. Immediate delivery as there is no error checking. Can deliver to multiple internet hosts using the minimal number of messages. **Interoperability** – Uses UDP, which is a widely recognised protocol, so nearly ubiquitous acceptance.	**Interactibility** - has no reliability or ordering guarantees. Corruption of data is possible due to lack of a checksum for error checking. **Interoperability** - Only available on hosts connected to the Mbone.
IP Broadcasting	Unreliable transmission to all hosts on a Local Area Network	**Scalability** - Small bandwidth overhead. Immediate delivery as there is no error checking. Can deliver to multiple LAN hosts using the minimal number of messages. **Interoperability** – Uses UDP, which is a widely recognised protocol, so nearly ubiquitous acceptance.	**Interactibility** - has no reliability or ordering guarantees. Corruption of data is possible due to lack of a checksum for error checking. **Interoperability** - Can only deliver to local networks.

Table 1. Network Protocols [12]

5 Conclusions and Future Work

5.1 Conclusions

This technical report has introduced the idea of a distributed interactive application and provided an insight into its background and its role in today's software industry. The main architectural features, both in software and network, were detailed in terms of the main components that would define the 'perfect' DIA system.

The current components used in today's architectures are quickly reaching the limit of what experiences they can offer the end user. Designers have to look beyond simply showing pretty pictures to users in order to deliver impressive products. Analysis of the progression of this medium tells us that future DIA's will have to provide a seamless, uniform and usable experience to all users involved, regardless of systems capabilities available to these users. In fact, at the unveiling of Microsoft's Xbox, Bill Gates mentioned that "incredible, persistent, online worlds" will become applications of the Xbox and that "we expect that a high percentage of games will have an online component". Given that Microsoft is one of the largest software companies in the world and that the Xbox is one of their flagship products, these statements emphasise the importance of this sector.

The four requirements identified in this technical report, namely dynamic extensibility, scalability, interoperability and interactibility, are the fundamental issues that need to be tackled in order to deliver an all-encompassing experience to users.

5.2 Future Work

One of the authors plans to take an existing game engine known as Torque (www.garagegames.com), and implement a test bed that will simulate and test consistency maintenance and latency reduction methods. Examples of these include some that have been mentioned previously such as dead reckoning, entity and strategy modelling, relevance filtering and interest expressions. Others include time management, which is mainly an issue for consistency maintenance, and compression and bundling of update messages.

A new protocol will be constructed to carry the data that has been encoded using the mechanisms above and this protocol will allow participants to communicate the latency reduction scheme being employed to other participants.

6 References

[1] R. Paush, P. Dennis., and G. Williams, "Quantifying Immersion in Virtual Reality," presented at ACM SIGGRAPH, 1997.

[2] W. Gibson, *Neuromancer*: Ace Books, 1984.

[3] D. Becker, "Online game makers seek key to profits," 2002, http://news.com.com/2100-1040-823258.html.

[4] GameSpy Staff, "Massively Multiplayer Online Games The Past, The Present, and The Future.," 2003, http://www.gamespy.com/amdmmog/.

[5] M. Capps, D. McGregor, D. Brutzman, and M. Zyda, "NPSNET-V: A New Beginning for Dynamically ExtensibleVirtual Environments," *IEEE Computer Graphics and Applications*, 2000.

[6] J. W. Barrus, R. C. Waters, and D. B. Anderson, "Locales: Supporting Large Multiuser Virtual Environments," *IEEE Computer Graphics and Applications*, vol. 16, pp. 50-57, 1996.

[7] E. Frécon and M. Stenius, "Dive: A Scalable Network Architecture for Distributed Virtual Environments," in *Distributed systems Engineering Journal*, vol. 5, 1998, pp. 91-100.

[8] D. Lee, M. Lim, and S. Han, "ATLAS - A Scalable Network Framework for Distributed Virtual Environments," presented at Proceedings of the 4th International Conference on Collaborative virtual environments (CVE02), Sept 30 - Oct 2, Bonn, Germany, 2002.

[9] S. Waldbusser, "Application Performance Measurement Grows Up," 2001, http://www.networkcomputing.com/1210/1210f4.html.

[10] D. McGregor, A. Kapolka, M. Zyda, and D. Brutzman, "Requirements for Large Scale Virtual Environments," 2002.

[11] A. LaMothe, *Tricks of the 3D Game Programming Gurus-Advanced 3D Graphics and Rasterization*: Sams Publishing, 2002.

[12] S. K. Singhal and M. Zyda, *Networked Virtual Environments*. New York: ACM Press, 1999.

[13] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modelling Language User Guide*, 1999.

[14] D. Snowdon, C. Greenhalgh, S. Benford, A. Bullock, and C. Brown, "A Review of Distributed Architectures for Networked Virtual Reality," *Virtual Reality: Research,Development and Applications*, vol. 2, 1996.

[15] E. Frécon and H. Olof, "Dive Architecture," 1996, http://www.scs.se/dive/manual/architecture.html.

[16] Communications Research Group, "MASSIVE-3/HIVEK Introduction," 2000, http://www.crg.cs.nott.ac.uk/research/systems/MASSIVE-3/.

[17] Richard C. Waters, David B. Anderson, John W. Barrus, David C. Brogan, Michael A. Casey, Stephan G. McKeown, Tohei Nitta, Ilene B. Sterns, and W. S. Yerazunis, "Diamond Park and Spline: A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtime Modifiability," in *Presence*, vol. 6, 1996, pp. 461-480.

[18] A. Kapolka, D. McGregor, and M. Capps, "A unified component framework for dynamically extensible virtual environments," presented at Proceedings of the 4th International Conference on Collaborative virtual environments (CVE02), Sept 30 - Oct 2, Bonn, Germany, 2002.

[19] J. Dahmann, R. M. Fujimoto, and R. M. Weatherly, "The Department of Defense High Level Architecture," presented at Winter Simulation Conference Dec 1997, 1997.

[20] T. Jepsen, "SOAP cleans up interoperability problems on the Web," in *IT Professional*, vol. 3, 2001, pp. 52-55.

[21] "Information Systems:A managment approach glossary," 2003, http://lms.thomsonelearning.com/hbcp/glossary/glossary.taf?gid=21&start=c.

[22] D. Lee, J. Yang, H. Yong Youn, C. Yu, and S. J. Hyun, "Entity-centric scalable concurrency control for distributed interactive applications," presented at Proceedings of IEEE International Performance Computing and Communications Conference (IPCCC'00), 2000.

[23] Y. Makbily, C. Gotsman, and R. Bar-Yehuda, "Geometric Algorithms for Message Filtering in Decentralized Virtual Environments," presented at ACM 1999 Symposium on Interactive 3D Graphics, Atlanta, Georgia, USA, 1999.
[24] K. L. Morse, L. Bic, and M. Dillencourt, "Interest Management in large-scale virtual environments," *Presence Teleoperators and virtual environments*, vol. 9, pp. 52-68, 2000.
[25] J. D. Delaney, T. Ward, and S. Mcloone, "On Network Latency In Distributed Interactive Applications," presented at National University of Ireland Maynooth Postgraduate Colloquium March 28, Maynooth, Ireland, 2003.
[26] M. Matijasevic, D. Gracanin, K. P. Valanis, and I. Lovrek, "A Framework for multiuser Distributed Virtual Environments," *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, vol. 32, pp. 416-429, 2002.
[27] M. Capps, "Developing Shared Virtual Environments," presented at ACM SIGGRAPH, 2000.
[28] R. Kauster, "Insubstantial Pageants: Designing Virtual Worlds," 2000.
[29] A. S. Tanenbaum, *Computer Networks*: Prentice Hall, 2003.
[30] T. Balikhina, F. Ball, and D. Duce, "Distributed Virtual Environments - An Active future," presented at The 20th Eurographics UK Conference, June 11-13, De Montfort University, Leicester, 2002.