P2P Based Distributed Virtual Reality

– a DVE Architecture and Implementation

terrapeer

Henrik Gehrmann

Thesis Document
IMM, DTU 2004

Submitted in partial fulfillment of the
requirements for the degree of

Master Of Science In Computer Science

at the
Institute of Informatics and Mathematical Modeling
Technical University of Denmark (DTU)
February 2004

by
Henrik Gehrmann
Student ID s948179
gehrmann@earthlink.net

Supervisor for this project was
Niels Jørgen Christensen, IMM, DTU

Abstract

This thesis document represents the architecture and implementation of a distributed virtual environment application – TerraPeer.

The idea of this project is to design an interface to a multi-user virtual space, which runs on an absolute decentralized, server-independent network. The application is built with specific technological choices, including a user interface framework on top of the Java3D API, the JXTA peer-to-peer platform, and a XML-based protocol.

By examining existing systems and current research in relation to three-dimensional virtual reality and distributed networks, this project aims to assemble the various parts that are required to create such an interface..

Keywords

Distributed Virtual Environments (DVE), Peer-to-Peer (P2P), 3D User Interfaces, Avatars, Virtual Spaces and Zones, Distributed Networks, Online Games and Worlds, Cyberspace, User Rights, Trust and Access

Website http://www.student.dtu.dk/~s948179/master/

Last updated 26.02.2004

Afleveret som en del af eksamensprojektopgaven
til afslutning af uddannelsen til

Civilingeniør

ved
Institut for Informatik og Mathematisk Modellering
Danmarks Tekniske Universitet (DTU)
Februar 2004

af
Henrik Gehrmann
Student ID s948179
gehrmann@earthlink.net

Vejleder for dette projekt var
Niels Jørgen Christensen, IMM, DTU

Sammendrag

Dette eksamensprojekt dokument repræsenterer arkitekturen og implementeringen af en distribueret virtuel verden applikation – TerraPeer.

Ideen bag projektet er at udvikle et system til et fler-brugers virtuelt rum, som indrettes i et server-uafhængigt netværk. TerraPeer er konstrueret med specifikke teknologiske valg, såsom et grænseflade framework der ligger i toppen af Java3D API`en, peer-to-peer platformen JXTA og et XML-baseret protokol.

Ved at undersøge eksisterende prototyper og relevante forskningsområder i relation til tredimensionale virtuelle verdener og distribuerede netværk, er det projektets mål at samle de forskellige dele der er krævet for at bygge et brugervenligt system.

Nøgleord

Distributed Virtual Environments (DVE), Peer-to-Peer (P2P), 3D User Interfaces, Avatars, Virtual Spaces and Zones, Distributed Networks, Online Games and Worlds, Cyberspace, User Rights, Trust and Access

Website http://www.student.dtu.dk/~s948179/master/

Sidst Opdateret 26.02.2004

Foreword

It is with pleasure that I am allowed to present this paper, and attached software, which was mainly constructed over a period of time between 2002 and 2003.

The intricate theme of the TerraPeer project is the architecture and implementation of a distributed virtual environment that in its core employs a multi-user 3D graphical world running on a peer-to-peer network. The design realizes an elementary set of functionalities, and emphasizes the construct itself, promoted by summarizing initial research in related areas.

The technical aspects of the application primarily involve the J2SE and Java3D API's, the peer-to-peer JXTA protocol, and XML.

To fully appreciate the content of this document, I recommend having the appendices handy. As most areas in software engineering, understanding and contextualizing the text requires not only insight into the technologies, but also good access to glossaries, definitions and abbreviations. Appendix II should lists most of the terms used in this project.

- Henrik Gehrmann
IMM, Technical University of Denmark, February 2004

Acknowledgments

Thanks to fellow students,
the people at IMM, and other institutes at DTU.

Quick overview of Contents

Table of Contents

Illustration Index

Index of Tables

1.Introduction

Creating a multi-user distributed virtual environment is a complex endeavor. The goal of this work is to develop a design and an implementation that can serve as a supportive model for research and application development. Special attention was payed to the usability, the service abstraction, as well as the fully distributed model.

1.1.Overview

1.1.1.Objective

The objective of this project is to construct an application for a distributed virtual environment (DVE) that is based on a peer-to-peer (P2P) network.

An examination and analysis of existing systems and technologies results in the finding that DVE systems often are dependent on central entities, or focus on the distibution of 3D graphics, performance, or immersive techniques. Most often, these systems lack mechanisms for users to publish information or services.

This thesis proposes a DVE solution that is absolutely decentralized and open. Any property can be represented in 3D through services, including information publishing, sharing, and communication between peers. The application prototype is able to demonstrate the concept of independence, as well as visualization of services.

The attempt is to answer the question: How would a possible solution to visualize peer-based activity and services by users in 3D space look like?

1.1.2.Problem

The scope of the problem is to create a large-scale, multi-user 3D virtual space that visualizes network peers, information, and services in 3D. The system should be open, provide a user interface for viewing and building the virtual world, support information publishing and sharing, as well as communication. It should be based on a specific selection of technologies for the graphical interface and underlaying network. Openness means any data, and any functionality can be supported (or extended).

In a scientific context, the area of research looking into networked multi-user 3D worlds is called Distributed Virtual Environments, or DVE.

The core intention is to design and implement a working environment, which adheres to the principle of absolute independent peers. Independence means fully decentralized with no central entity (server-independent), and can be achieved through the concept of peer-to-peer networks.

Although peer-to-peer is not new per se, it's popularity a few years ago began to rise with emerging file-sharing tools. The main idea of P2P is to distribute processing power, memory and storage across the networked computers, without the overhead and dependency of a central server, where each peer acts as both client and server. The network is distributed, decentralized, and ad-hoc; sometimes coined 'edge computing'.

This projects does not aim to study the properties of 3D virtual worlds, virtual reality, or distributed peer networking technologies in particular, but rather suggest how an implementation of a DVE might look like.

1.1.3.Solution

The DVE as a combination of P2P and VE can essentially provide a platform for user collaboration and socialization, as well as for services that users can utilize.

This project defines an architecture and implements a working application. Hence, the imminent task is to narrow project boundaries, which requires deeper insight into each problem area.

To approach a solution, it is initially desirable to understand existing technologies and research areas. An examination of related topics provides the necessary knowledge. Topics to be examined are DVE systems and approaches, scalability and performance methods, distributed networks, 3D graphics to visualize networks, 3D building and interaction mechanisms, and user interface design.

A solution is presented in the form of an analysis that results in a prototype architecture and implementation. The application is based on the selection of JXTA and Java3D technologies, and demonstrates a working DVE.

1.2.Background

The internet has evolved dramatically throughout the past 10 years, mainly due to the web-platform, that enables creation of personal and business sites in a dynamic matter, but based on certain rules and protocols (like HTML and HTTP).

New network technologies and software emerges constantly (multi-tier networks, J2EE, SAN), and recent distributed peer-to-peer platforms have shown very interesting capabilities (for example software such as Gnutella, Napster, Kazaa and SETI). 3D gaming and online virtual worlds, the latter mostly rooted in social interaction, have become increasingly popular (there are countless examples: Counterstrike, Wolfenstein, Ultima, Worlds, etc.).

The internet provides unique opportunities to create 'virtual spaces', differentiated in purpose, functionality, content, and style. A virtual space could be defined as any kind of abstract environment running on computer-based systems, not necessarily supporting real user interaction. There exist numerous examples of virtual spaces, such as text-based systems, multi-user games and 3D environments, design and engineering tools, collaboration and discussion spaces, messengers or file sharing systems.

These Virtual Environments (VE's) often present overlapping characteristics, objectives and attributes, depending on the architecture of each system. Most multi-user environments, for example, usually provide essential communication tools, as well as abilities to send or share certain data.

Notably multi-user virtual spaces, abandoning physical boundaries, are subject of intense development in recent years, and are based on a range of technologies running on the top OSI-7 layers. Further, the trend towards graphical 3D environments is set, as new graphical microprocessors (GPU's) continue to grow in capacity, and especially online game engines have evolved to a state of art, presenting ever new rendering features and interaction algorithms.

The network's inherent problem is its overwhelming accumulation of data as well as traffic bandwidth when millions of users go online. When developing network-based applications, scalability, reliability, extensibility, openness and interoperability are essential questions for the designer.

Furthermore, applications that utilize the network more than ever need to focus on usability, as functionality and interaction becomes more complex. Navigation and search tools are essential aids for a use; this is reflected in the build-up of user-friendly websites, directories or web-crawlers and search-engines (examples include Google, Yahoo, and LDAP).

Previously, researchers have argued that virtual environments bring a paradigm shift in data representation and information access. The following table, Table 1.1. Technological Advancements, illustrates this assessment by technical advances [modified from PORTA01, pp.219].

Paradigm	Time-sharing	Desktop	Networks
Period	1960's-1980's	1980's-2000's	1990's-2010's
User	Specialist	Individual	Group
Interface	Text	2D, 3D	2D, 3D, DVE
Interaction	“Read and type”	“Draw and click”	“Model and navigate”

Table 1.1. Technological Advancements

Being totally immersed in a virtual reality – as the last half of a centuries studies and developments indicate – usually requires an array of technologies ranging from hardware and user interface devices to 3D software engines. The difficulties to artificially project a virtual environment onto the user, implementing a spectra of sensors, as well as enabling control and feedback mechanisms, are slowly pushed aside, and advancements in processing power, micro systems, haptic devices, and software should soon be able to reach a state where an immersive – or at least an augmented – reality is possible.

The opportunities in such an artificial reality seem unlimited, as they enhance the current virtual space – the now commonly used internet – with more dimensions, creating a true cyberspace, as William Gibson coined the virtual world (see Appendix II).

Considering the explosion of the world-wide-web after the first browsers emerged in the early 1990's [MOSAIC], the increasing research in the field of distributed virtual environment (DVE) systems might promote a new kind of a network. Development of DVE's is partly fueled by advances in computer graphics and networking technologies that provide the necessary processing power to create these environments.

1.3.Problem Scope

The problems to be examined can be distinguished by different perspectives, or 'levels', that the project can be looked upon. This rough distinction will resurface later, at the architecture of the system. The following list shows the core questions.

- *Application Level*
  - What virtualization (i.e. 3D representation) capabilities and functionality should the application feature?
  - Which technologies could support such a system, and what properties are important in choosing a particular platform or tool?
  - How should the development be attempted, and what methods applied to successfully create a demonstration of the system?
- *Visualization Level*
  - When creating a 3D multi-user environment, how should the interface be structured, from a users point-of-view?
  - How should the standard HCI/MVC design pattern be applied? What model-view-control, navigation and interaction functionality and restrictions are necessary to provide a usable interface?
  - How should the virtual space be designed? How can it account for the distributed nature of P2P networks? Where are how should users, content, and network peers be abstracted? Is it mandated to implement certain rules and restrictions?
  - What kind of objects might exist in the space, what should they represent, how could they be created, and who should have access to them?
- *Network Level*
  - How can the user's disorientation in a P2P environment, with regards to both services and relationships be circumvented?
  - Which basic building blocks should enhance the virtualization of a distributed P2P system?

These levels highlight the scope of the problem. Each require further investigation, which leads us to the theoretical scope of this project in the next chapter. Before diving into the research and technological aspects, though, the remainder of this chapter will shortly outline the prerequisites and general approach.

1.4.Prerequisites

Major prerequisites to attempt an architecture and implementation include knowledge of software design and engineering topics. They should encompass software modeling and object-oriented design patterns, graphical design algorithms and methods, virtual environment architectures, and network protocols.

The creation of application of the described type requires theoretical knowledge about DVE projects, existing prototypes, and related research. To this extend, an examination of distributed system designs, 3D graphics, environment sharing and networking technologies, multi-user settings, performance and optimization issues, as well as network topologies is necessary.

As will be described later, the architecture and implementation is created through specific choices. This project aims to build upon existing experiences and reuse frameworks as needed in order to be able to concentrate on the central aspects.

1.5.Approach

This thesis consists of seven chapters. Chapter 2 begins by looking in more depth into the associated research, and develops an understanding of existing DVE systems. Based on existing architectures, studies and projects, a map of the DVE landscape and related issues is derived.

The chapter continues to describe that landscape to some extend, but emphasizes specific technologies. In particular, a 3D User Interface Framework on top of the Java3D API for usability enhancements, and the JXTA P2P platform for distributed networking, are examined.

Chapter 3 analyzes the research areas, projects and technologies. From this platform, an outline of a suitable solution, and an approach for the design are described.

Chapter 4 presents the architecture. The focus in this chapter is moved to the application itself, where the selected technologies are integrated into the application design. The major design issues for the architecture are explained, starting from high-level principles to specific models.

Chapter 5 is more separate in that it describes the actual prototype implementation, rather than specifying or analyzing certain methodologies. Not all source-code is listed, but distinct methods are emphasized.

Chapter 6 discusses the results of the project work, and the thesis finishes in Chapter 7 with conclusions and directions for future work.

A chronological path of the document is given below, which lays out how this project is approached by briefly highlighting each major section:

Chapter 2. Theoretical Scope

- 2.1.Research Overview – About the current state of research on related topics such as graphics, games, networks, and virtual environments
- 2.2.Distributed Virtual Environments – Detailed examination of DVE research
- 2.3.3D Technologies – Emphasis on Java3D
- 2.4.Network Technologies – Emphasis on JXTA

2.Theoretical Scope

The intention of this chapter is to lay out the theory behind the application by illuminating the related fields, formulating a preliminary design, and thereby prepare for the actual architecture and implementation that will be described in the following chapters.

Through research on associated subjects, the chapter aims to bridge the gap between the initial goal introduced in chapter 1, and the specific architecture in chapter 3.

Starting with section 2.1. about the current state of research on related topics, the second section 2.2. looks into the particular area of distributed virtual environments, and introduces the essential methodologies. Then, section 2.3 dives into the assumptions behind the project, and considers the scope of the application.

Alas, this thread should prepare the reader with the concepts and scope of the application, and pave the road to the architecture in the next chapter.

2.1.Research Overview and Technology Areas

This section provides a basic foundation for the TerraPeer project by examining the technological and research areas that are related to the subject. It is the wealth of interesting and inspiring ideas that have created the foundation for this thesis.

2.1.1.Related Research and Technologies

The main focus areas of research surround the following topics, sub-topics that to some degree are associated, and related platforms, standards, and technologies. See Table 2.1. Related Areas of Research and Technologies.

Central Areas	Associated Areas	Related Technologies
Distributed Virtual 3D Environments (DVE)	3D Frameworks and Protocols	Java3D, OpenGL, X3D/VRML, SDL, VRTP
	Multi-user Game Engines	
	Virtual Worlds	
	3D User Interfaces	
	Virtual Reality (VR), Augmented Reality (AR)	
	Avatars and User Representation	
	Event Driven Architectures	
	Computer Graphics	
P2P and Client/Server Network Architectures	Distributed Architectures	JXTA, HTTP, XML, Schemas, RDB/ODB, Gnutella, FTP, SMTP, WebServices/SOAP
	Process-, Web- and File-sharing	
	Multi-agent systems (MAS), Autonomous Agents, Bots	
Dynamic Network Visualization	GUI Design, Usability	HCI
	Information search, Information maps	
	Navigable information spaces	
	Web-crawling, ranking engines	
Object-Oriented building blocks (network implicit)		OOAD, UML, Java (J2SE/J2EE), CORBA, DCOM

Central Areas	Associated Areas	Related Technologies
Collaboration	CSCW	
	MUD's, MOO's	
	User awareness	
AI	Neural Networks, Pattern Recognition	
Trust	Trust Feedback and Reputation systems	Encryption/PKI, P3P, MS Passport, “Poblano” (JXTA)
	Policies and Privacy, Certificates	

Table 2.1. Related Areas of Research and Technologies

Considering the rapid growth of each of these technologies, one might imagine an alluring future, where computer graphics on special mobile haptic devices can produce reality-like environments, which combined with the expanding interconnectivity of everything from kitchen devices to satellites, truly would create a cyberspace as Gibson imagined it. While there are many hurdles to be overcome before this goal can be reached, laboratories, institutes, and businesses around the world are racing ahead with new theories, designs, and applications.

The evolution of 3D hardware has been impressive. Game engine developers, for instance, have to constantly keep up to date with the 3D card industry. The original Voodoo 1 graphic-card had 2 MB on-board memory, the Riva TNT increased its memory to 16 MB, then the GeForce and ATI Rage supplied 32 MB, and today the GeForce 4 and Radeon have 64 MB to 128 MB on-board memory.

The GeForce graphical chip itself well outperforms some of the recent standard CPU's on the market. Mobile services provided by DoCoMo in Japan have reached broadband-like speeds, capable of streaming video. New P2P file-sharing applications manage to download the same file from multiple sources simultaneously. Software, such as the latest Half-life 3D game engine performs highly efficient, while providing a very dynamic multi-user experience.

Multi-user online gaming using 3D graphical engines has reached proportions that cannot be disregarded when considering large scale virtual environments. The Half-life game mod 'Counter Strike' alone has a user-base of some 30.000 in Europe, running on a total of around 7.000 servers (see Appendix X.).

2.1.2.Computer Graphics

Computer graphics has been studied ever since the first digital machines emerged, when pioneers such as Weiner, Whitney, Sutherland, and Freeman thought about the possibilities of implementing and using graphical interfaces, image processing, and animation. Representation of data on devices capable of showing 2D and 3D graphics have proven mostly valuable.

Today, any given PC or mobile system could not exist without the important graphical user interface (GUI). HCI, simulation and animation, Computer Games, CAD systems, Desktop Publishing, Web Design and Usability, 3D graphics, Virtual as well as Augmented Reality, are all topics that have evolved out of, and are build upon computer graphics; each is now a subject of study. 3D graphics is being used extensively in a broad range of areas and industries, taking advantage of the increasing processing power, speed, and specialized CPU's.

Using a virtual environment based graphical interface can only be valuable when viewing information in 3D supersedes viewing it in 2D. The extra dimension should enhance the orientation and representation of data to the user, or it just unnecessarily worsens the interface. This is often not the case, as the step to the next dimension also involves new challenges, for example navigational difficulties, cluttered views and object overlaying. The common hardware interface devices are still limited: the monitor screen, keyboard, and mouse. Obviously, in most settings, the 3D space still is displayed on a 2D screen.

3D spaces exist in various forms, both networked and 'local' (non-distributed), as toolkits or frameworks. The OpenGL and DirectX technologies are the de facto graphical API's for low-level graphical calculations. OpenGL is used on different OS platforms, and is often able to utilize specific hardware routines. Two examples of typical 3D spaces follow.

The OpenGL Performer by SGI (which is running on IRIX, Intel and Linux) is a toolkit that uses OpenGL as a low-level interface. Performer creates a non-distributed 3D space (virtual environment), and is often used at universities, in research and commercial software. It supports multi-threading, has no event model, and stores data in a binary format (.PBF). Application language is C/C++.

The VRML/X3D specification on the other hand has no low-level graphics API, but provides event routes between nodes, and stores data in XML and binary formats.

Both frameworks support SG with nested transformations and SG persistence. Other local 3D spaces include Open Inventor and VR Juggler (see appendix IX).

2.1.3.Game Engines

2.1.3.1.Game Engines

At the core of high-end 3D games lies the game engine. This term has only existed a few years, but defines the essential attributes of a game. A 'game engine' is an extensible and modularized design concept, which allows developers to create new or modify existing game models, scenery, and sounds.

"The engine can be defined as all the non-game specific technology. The game part would be all the content (models, animations, sounds, AI, and physics) which are called 'assets', and the code required specifically to make that game work" [GE02]. A modern game is composed of many elements; see the following Table 2.2. Game Engine Structure.

Component	Description
Renderer	Visualizes the scene for the player, requires extensive processing, and implements the 3D Pipeline, culling methods (remove not visible polygons), BSP trees, lighting, etc.
Bump mapping	Remodeling of surface to create light effects.
Fogging	Fading out distant parts of the scene (visual range crossing the far clipping plane). Volumetric fogging involves enclosed areas.
Anti-aliasing	Smoothing edges.
Inverse Kinematics (IK)	Implicit repositioning of the joints of a model. Forward Kinematics works in reverse to IK (among other issues, knowledge about the motion-range that a joint can go through is important).
BSP trees and PVS	Visibility and occlusion methods.
Game Physics	Simulating physical properties in the world environment.
Effects Systems	Special effects embedded in the scene.
Sound Systems	Background music and spatial sounds. OpenAL is an API that supports sound (a software interface to audio hardware, providing high-quality multi-channel output).
Game Networking	Multiple clients
Game Control Mechanisms and Scripting Systems	Special methods embedded in the scene to manipulate certain objects or otherwise control the environment.
Entities and Cameras	User view.
Artificial Intelligence (AI)	Game non-player characters (NPC) and rules that the computer controls in an intelligent manner.
World Navigation	Navigation functionality.
Game Rules	Besides normal rules, there is so-called emergent game play (no coded rule for every possible board play scenario; Chess).
Front-End	User interface on top of the scene for visual control and feedback (HUD).

Table 2.2. Game Engine Structure

2.1.3.2.Rendering

Graphical triangle calculations is the most basic level in scene rendering, and the number of triangles generated (the "tris count" is often cited in relation with a game's framerate) is an important taxonomy. Triangles and polygons describe 3D surfaces. 'Patches' are higher-order surfaces describe geometry with a mathematical expression, which are used to generate a mesh of polygons from the equation 'on the fly'. Triangles and polygons, combined with textures, and manipulated through filters or shaders, are fundamentals of the graphical rendering process.

To counter hardware and pipeline limitations, the rendering process has to be optimized. Texture compression, for example, reduces the memory footprint and bandwidth demands of textures. "The technique of MIP mapping involves preprocessing a texture to create multiple copies, where each successive copy is one-half the size of the prior copy" [GE02]. Modern multi-piped 3D accelerators allow a single rendering pass when applying multiple textures. Texture caching is another tool to increase performance.

Another common method is depth testing, where occluded pixels are discarded. This involves 'overdrawing' - the number of times one pixel location is drawn, which is based on the number of elements existing in the Z (depth) dimension.

Vertex shaders are used to calculate and perform effects on vertexes before submitting them for rendering. Pixel shaders are used for each pixel when the texture is rendered to create special effects (out of focus, haze, internal reflection, etc.).

The 'Mod' communities are growing. Most of the game engines enable gamers to modify the original game by supplying modules to the original core, and thereby in effect creating new games. The Quake 3 engine, for example, is used by Quake III Arena, Quake III: Team Arena, Return To Castle Wolfenstein, Enemy Territory, Jedi Knight II: Jedi Outcast, Soldier Of Fortune II, and Star Trek Voyager: Elite Force. The Half-life engine is used by Counter-Strike and Ricochet among others.

2.1.4.Games and Worlds

3D games and worlds do hardly require an introduction. The popularity of both is asserted by millions of users spending hours of their time inside these virtual environments, and though the study of social interaction comes into mind, the focus here is placed on the technological aspects.

Virtual worlds are usually created differently from games, which is attributable to different target audiences. Whereas a world typically implements chat, emphasizes large groups, and is build upon text-based, VRML-like protocols [VRML97], online games emphasize smaller groups, and use high-performance 3D engines that use binary protocols [Quake-BSP]. The intention of a world is to socialize, that of a game to entertain.

A merger of these intentions and supporting technologies in the near future, is quite realistic.

Many 'worlds' that exist on the internet today, are based on different engines that commonly are available as plug-ins to a browser. In contrast, games usually require separate installations that are by far larger in size (and demand).

The following sections describe a few of the current solutions.

2.1.4.1.Quake, Half-life, EverQuest, and OpenWorlds

The usual setting for games like Quake and Half-life are seperately running game-servers for clients to join. Each game server maintains a unique game state, which is not shared among the servers. These client–server systems are not real multi-server systems in that each game-server is represented by their own unique environment.

In contrast, the game EverQuest divides the entire DVE into distinct zones, and maintains these zones by individual game servers [DVE-PERF02]. This setting enables a continuous space that all participants share, and could be considered a 'real' DVE system. Whereas the games mentioned above represent a shared environment, the scale of such systems is narrow both in terms of number of clients (in a scale 5-20), and spatial extension (relatively contained space).

OpenWorlds is a X3D-compatible system that provides immersive Web 3D graphics, multimedia, animation, and VR capabilities [OW].

OpenWorlds is a toolkit that supports standards Web 3D, and is used to develop applications or as a browser plug-in. The toolkit provides script interfaces to add any scripting language to VRML script (C++ and JavaScript API's), and allows custom extensions. Further, it has a built-in node interface that is not based on any particular rendering system, i.e. OpenWorlds features a platform-specific rendering system (supported are Perfomer [OpenGL-Perf], OpenGL, and Optimizer for the SGI and OpenGL and Optimizer for the Windows platforms). It is also possible to select a GUI toolkit (MFC, Console, or Tcl/Tk based).

2.1.4.2.Crystal Space

The open source project "Crystal Space" is a 3D graphics engine with a large, extensible API. "The framework features approximately 700 files of C++ source code and 1,000 header files containing more than 500,000 lines of code. The basic framework design is centered around an external object model called Shared Class Facility (SCF). A typical Crystal Space application instantiates and uses several SCF objects providing services like 3D rendering, physics, collision detection, mesh creation, etc." [CRYSTAL].

The architecture of Crystal Space offers a 6DOF engine with arbitrary sloped convex polygons, a large plug in system (modules include scripting languages, Python, Perl, and Java), the SCF for communication between layers, true-color and multi-resolution support, and command line arguments.

The interesting part about Crystal Space is it's open source code, which as with many other projects allow free usage and development on the software, as well as being constructed by a broad base of contributers from around the world.

2.1.4.3.CryEngine

One of the most recent advances in game engine technology is currently displayed by the so-called CryEngine (a preview of the game 'FarCry' is astonishing to watch, and seems to leave competing engines a generation behind). "Real-time editing, bump-mapping, static lights, network system, integrated physics system, shaders, shadows and a dynamic music system are just some of the state of-the-art features that the CryENGINE offers" [FARCRY].

This game engine implements the following parts (from the FarCry website): A renderer, a physics system, Character Inverse Kinematics & Animation Blending, Network Client and Server System, Shaders, Terrain, Lightening and Shadow mechanisms, Fogging mechanism, Polybump (rendering quality), and other features to increase performance.

2.1.5.Networks

2.1.5.1.Overview

The web-platform not only enables creation of personal, organizational and business sites in a dynamic matter, but also provides a platform that is based on certain rules and protocols (such as HTTP and HTML).

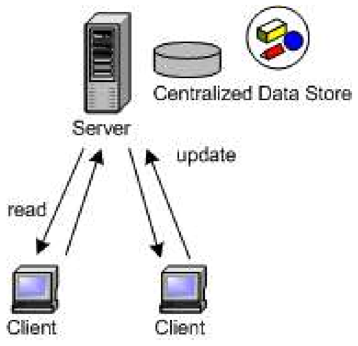


Illustration 2.1. Typical Client/Server network

This is quite important to acknowledge, as it creates a common standard and narrows the users options, thereby simplifying authoring, which in turn promotes the standard further.

New hardware, network devices, and software emerges in a constant pace. Examples of recent developments include gigahertz processors, mobile devices using Java, Beowolf clusters, large-scale multi-tier applications, and very capable client/server 3D game engines. The recent distributed peer-to-peer (P2P) platform has shown very interesting capabilities (Kazaa, SETI). 3D gaming and online virtual worlds, mostly rooted in social interaction have become increasingly popular (Ultima, Worlds, etc.).

At the core of this evolution is the network; it enables people to interact in seemingly inexhaustible ways, most of them not conceivable or practical in the 'real world'. Interaction is now possible via e-mail, peer-to-peer messaging, IRC, forums and newsgroups, blogs, VoIP, MUD's and online games, web-based collaboration and management tools, and many more.

The technical core of a network are the layered protocols (OSI model; see Table 2.3. Typical OSI Network Layers). Typically, the layers to be considered when designing a DVE system are above the IP-layer. Mainly two options are available on layer 4: the TCP over IP and the UDP over IP protocols.

OSI Layer	Protocols	
7: Application	SMTP, FTP	DNS, SNMP, NFS
6: Presentation	MPEG, GIF, JPEG	
5: Session		
4: Transport	TCP	UDP
3: Network	IP	
2: Data Link	Ethernet	
1: Physical		

Table 2.3. Typical OSI Network Layers

The Transmission Control Protocol (TCP) allows communication between systems with the following features:

- reliable transmission (connection-oriented, end-to-end reliable packet delivery through an internetwork)
- stream data transfer (unstructured stream of bytes identified by sequence numbers)
- efficient flow control (avoid internal buffer overflow)
- full-duplex operation (both send and receive at the same time)
- multiplexing (numerous simultaneous upper-layer conversations can be multiplexed over a single connection)

The User Datagram Protocol (UDP) is connectionless, and provides no reliability, flow-control, or error-recovery, but also consumes less network overhead than TCP.

2.1.5.2.Client/Server

Multi-user virtual environments are often built on simple client/server architecture (see Illustration 2.1. Typical Client/Server network). There exist several such applications that both provide navigation, modeling, data representation, and user interaction functionality. When multiple users model on the same scene, or shoot at the same monsters in a game, each node would connect to the server, and all information would be calculated centrally. Though, this isn't an easy task by far, and many optimization routines and design patterns have evolved to enhance these networks, the centralized architecture still simplifies the multi-user environment.

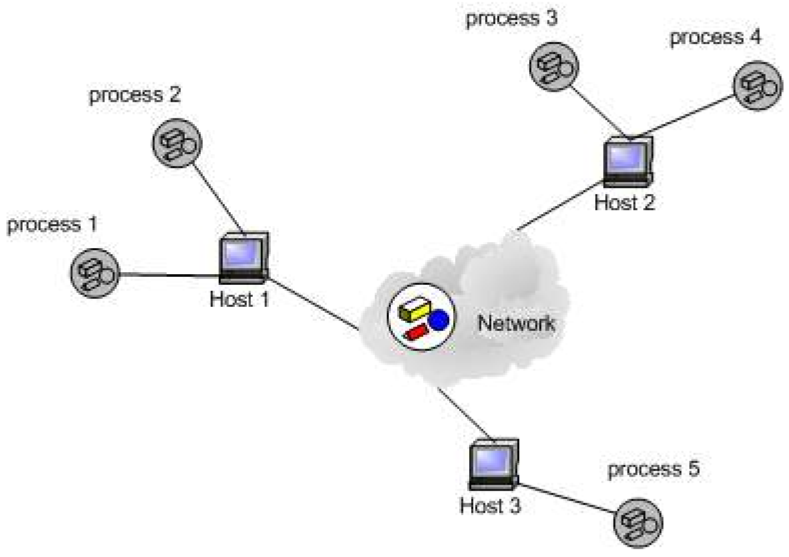


Illustration 2.2. Typical P2P network with distributed processes

In a totally distributed P2P network, the architecture is decentralized (an example is Illustration 2.2. Typical P2P network with distributed processes). These networks have no central entity, and each node – or peer – thus has to maintain it's own state of the current environment. It should be noted, that the distinction between network architectures is not always clear, and though P2P became popular in recent years, it is not a 'new' technology per se. A peer can be seen as a server and a client combined. Some P2P networks rely on central servers as well.

Several advantages and disadvantages exist between a client/server and peer-to-peer network architecture. While the former obviously retains information and control at a single place, the latter often can ease network traffic and does not depend on a single server. P2P has proved most successful in resource-sharing, file-sharing, and messaging applications, whereas the relative common client/server and multi-tier network is used in almost all database, web, business, and communication applications.

2.1.5.3.Peer-to-Peer

Peer-to-peer (P2P) networking is a technology, that by itself is not much different from Client/Server (C/S) networks. P2P is more a label than a definition. Simply put, a peer (node) acts as both server and client, thus enabling a decentralized distributed network.

The definition of peer-to-peer networks is often vague. Taken literally, servers talking to one another are peer-to-peer. The game Doom is peer-to-peer. Napster is not peer-to-peer in the strictest sense, because it uses a centralized server to store pointers and resolve addresses.

A common example for a C/S system is the well-known Web-server/Browser setting. Similarly, a typical 3-tier system usually consists of a backend database, a middle-tier application server, and a thin front-end GUI. Compared to these systems, P2P differs in the sense of where the workload is located. Whereas servers process and store the bulk of data in C/S networks, such central location does not exist in P2P networks. A peer has to manage, process, and store all relevant data locally.

The disadvantage of this setting is that data and functionality cannot be managed at one central location, which makes it difficult to maintain a concurrent state between peers or easily control the distribution of data. The advantage of P2P networks is the ability to heavily distribute processing, replicate data, and promote 'edge-computing'. Common examples include distributed calculation (SETI), file-sharing (Kazaa, eMule), and collaboration applications (ICQ, Groove).

What makes P2P distinctive? Applications using P2P take advantage of resources -- storage, cycles, content, human presence -- available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P nodes must operate outside the DNS system and have significant or total autonomy from central servers. P2P is a way of decentralizing features, costs and administration [P2P-Shirky].

> Asking “What is P2P...” a few years ago (in 2000), Clay Shirky described the evolution of networks as following [P2P-Shirky]:
>
> The launch of ICQ in 1996 marked the first time those intermittently connected PCs became directly addressable by average users. Faced with the challenge of establishing portable presence, ICQ bypassed DNS in favor of creating its own directory of protocol-specific addresses that could update IP addresses in real time, a trick followed by Groove, Napster, and NetMeeting as well. (Not all P2P systems use this trick. Gnutella and Freenet, for example, bypass DNS the old-fashioned way, by relying on numeric IP addresses. Popular Power and SETI@Home bypass it by giving the nodes scheduled times to contact fixed addresses, thus delivering their current IP address at the time of the connection.)
>
> Whois counts 23 million domain names, built up in the 16 years since the inception of IP addresses in 1984. Napster alone has created more than 23 million non-DNS addresses in 16 months, and when you add in all the non-DNS Instant Messaging addresses, the number of P2P addresses designed to reach dynamic IPs tops 200 million. Even if you assume that the average DNS host has 10 additional addresses of the form foo.host.com, the total number of P2P addresses now equals the total number of DNS addresses after only 4 years, and is growing faster than the DNS universe today.

As can be abstracted from the box above, one of the primary features of P2P is evidently it's distributed and decentralized nature, which makes it possible to create networks that are truly independent of any central (controlling) entity.

This sometimes political question of where to place control becomes imperative for the designer of the system as well. In his book 'Code', the author Lawrence Lessig analyzes this very important topic to a great detail.

2.1.5.4.Distributed Networks

Distributed applications and infrastructure research is currently examining several issues. The contrast between tradition client/server versus peer-to-peer networks, as well as challenges in P2P designs is shortly outlined below.

While centralized systems are evolving toward decentralization, decentralized systems are evolving toward centralization. Both in a response to growth and the need to scale upward [P2P-ACAD]. For example the hosts file on the internet evolved towards the DNS, while Gnutella evolved towards a system with superpeers, Freenet provides gateways, and JXTA search creates a hierarchy of servers.

Naming and resource discovery is an essential part of any networked system, unless anonymity is important, to find particular individuals or repositories for information. Many P2P systems, for example IM services, achieve this with identities stored in a strictly centralized repository. Although IPv6 might resolve some of the related problems in identification and resource discovery, there will still be open questions about how to balance centralization and decentralization.

One of the strengths of P2P networks, such as Gnutella and Freenet, is that they are able to provide content independent of its location. This stands in contrast to the client/server approach, which fluctuates less in terms of networking dynamics, but requires clients to be dependent on a single entity.

Delivering services is usually focused on web-based services (HTTP, XML-RPC, SOAP, etc.) as most services today are offered through the web layers (avoiding firewalls). This is not very efficient for P2P communication, though. Specific protocols exists to take advantage of lower network layers, including JXTA, the SCTP transport-level protocol, and the BEEP application-level protocol.

"By decentralizing data and therefore redirecting users so they download data directly from other users' computers, Napster reduced the load on its servers to the point where it could cheaply support tens of millions of users" [P2P-ACAD]. P2P can take advantage of this principle; it can distribute the burden of supporting network connections, and bottlenecks are eliminated at central sites, though overall bandwidth will be similar.

2.1.5.5.P2P Socket

The Peer-to-Peer (P2P) Sockets Project [P2PS03] reimplements Java's standard Socket, ServerSocket, and InetAddress classes to work on the JXTA peer-to-peer network, rather than on the standard TCP/IP network.

According to their website, the P2P Sockets project is designed for developers interested in:

- Returning the end-to-end principle to the Internet.
- An alternative peer-to-peer domain name system that bypasses ICANN and Verisign, is completely decentralized, and responds to updates much quicker than standard DNS.
- An Internet where everyone can create and consume network services, even if they have a dynamic IP address or no IP address, are behind a Network Address Translation (NAT) device, or blocked by an ISP's firewall.
- A Web where every peer can automatically start a web server, host an XML-RPC service, and more, and quickly make these available to other peers.
- Easily adding peer-to-peer functionality to Java socket and server socket applications.
- Having servlets and Java Server Pages work on a peer-to-peer network for increased reliability, easier maintenance, and exciting new end-user functionality.

P2P Socket is a new project that has been included here to highlight the on-going development of network protocols that might be of interest for DVE system designs.

2.1.6.Virtual Environments for Multiple Users

A generally used term for virtual worlds or 3D spaces is 'Virtual Environment' (VE). The history of virtual worlds began with the first MUD's (Multi-User Dungeons), which enables participants to meet in a virtual, though text-based, environment where they could communicate with each other. Since these days, VE technologies have evolved dramatically, especially considering 3D rendering techniques and the performance of underlaying hardware.

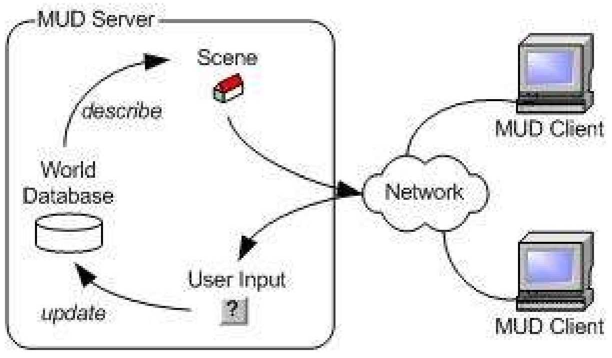


Illustration 2.3. MUD Client/Server Architecture

The step from VE to DVE is ambiguous. Research has not strictly enforced boundaries on the distinction, and though single- versus (networked) multi-user systems are easily set apart, the latter is most often discussed. Literally speaking, VE's should represent single-user 3D spaces that are not shared. Distributed virtual worlds are engineered on a networked architecture.

DVE design is naturally complex as it is comprised of many sub-systems, with each their individual problems and interfaces. Creating and optimizing the 3D environment is one issue, the user interface interaction and feedback mechanism a second, the network synchronization and efficiency another. Robustness and scalability extend the complexity further. An interesting taxonomy of the problems one may encounter when dealing with large networked virtual environments can be found in [DVE-TAX97].

2.1.6.1.3D multi-user Virtual Space systems

DVE's have been implemented using a range of frameworks that simplify the work of creating a necessary distributed infrastructure from scratch. Relative standardized technologies include J2EE, DCOM, CORBA, VRML/X3D, VRTP, and a long list of specific research-related DVE platforms (will be described later). Other implementations are middleware systems [FLEXI99], mobile services [VOY02], agent or mobile scripting [AGLETS02] systems.

Studies on 3D multiuser virtual space systems have focused on a large collection of areas: 3D graphics (culling, spatial representations) and VR technologies (user interfaces, 3D representations), game engines, communication and collaboration between users in VE, methods to interact with avatars and virtual objects, network efficiency, massive distribution (scalability) and synchronization, etc.

A couple of interesting research papers are summarized in the following sections, and a more comprehensive listing of VE/DVE related projects will be given.

2.1.6.2.The Porta Susa Project

In a large-scale project in Italy, a virtual information system was implemented. A 2001 paper describes the "Porta Susa Project" [PORTA01] and the implementation of their Shared Virtual Reality (SVR) system.

The main purpose of the SVR was to support the design and construction of a large (300M Euro in investments) central urban area, namely the railway junction of Porta Susa and the surrounding urban area in the city centre of Turin, Italy (see Illustration 2.4. Screenshot of Porta Susa Urban Map).

The project also targeted "to renew the overall system of communications of the city towards an integrated system of exchange between different means of transport" [PORTA01, pp.218].

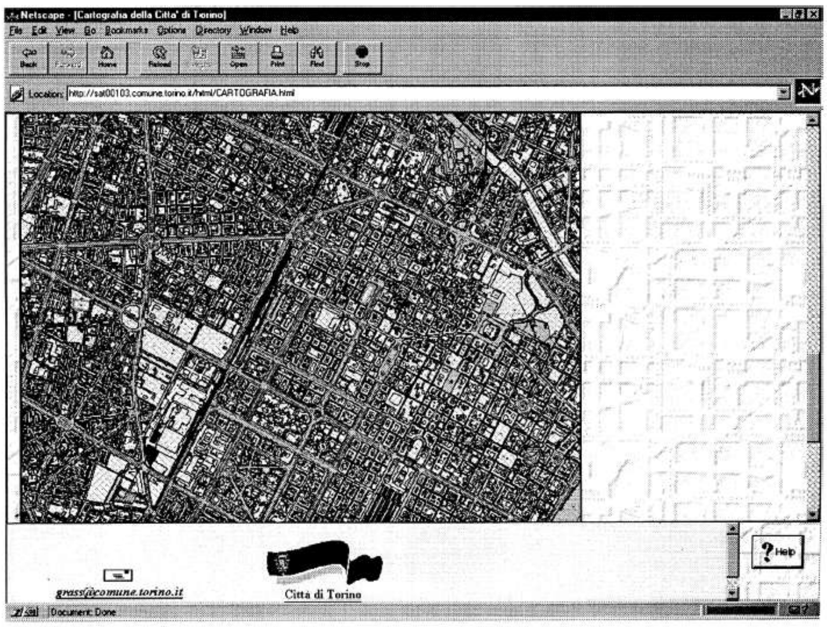

Netscape - [Cartografia della Citta' di Torino]
Location: http://sat00103.comune.torino.it/html/CARTOGRAFIA.html
grass@comune.torino.it
Città di Torino
Help
Document: Done

Illustration 2.4. Screenshot of Porta Susa Urban Map

The SVR system was distributed to be accessible location-independent, and was designed to be flexible, scalable and simple. Further, the aim of SVG was to allow users to enter and explore complex data through a spatial representation through computer interactivity (the paper refers to [VRBC]). A client/server approach, as well as the employment of plug-ins for web-browsers created the basic structure of SVG. In 1996–1997, hardware and software put an upper limit to the complexity of the models, as only one third of the PCs offered state of the art CPUs and more than 32MB of RAM; 3D graphics accelerator cards were limited.

Architectured CAD 3D models were converted to VR models using VRML, which enabled participants to view spatial representation directly. The SVR made it possible to "enhance the understanding of interference between flows of different transport systems", and integrated vocal messages, Java scripts, avatars with limited capabilities, and a taxonomy of VRML.

The system distinguished between a construction model to store documents, with meta-data, position, classification, and decomposition information, and a document model to present connecting documents, directories and servers through links, as can be seen in Illustration 2.5. Screenshot of SVR Document Linking.

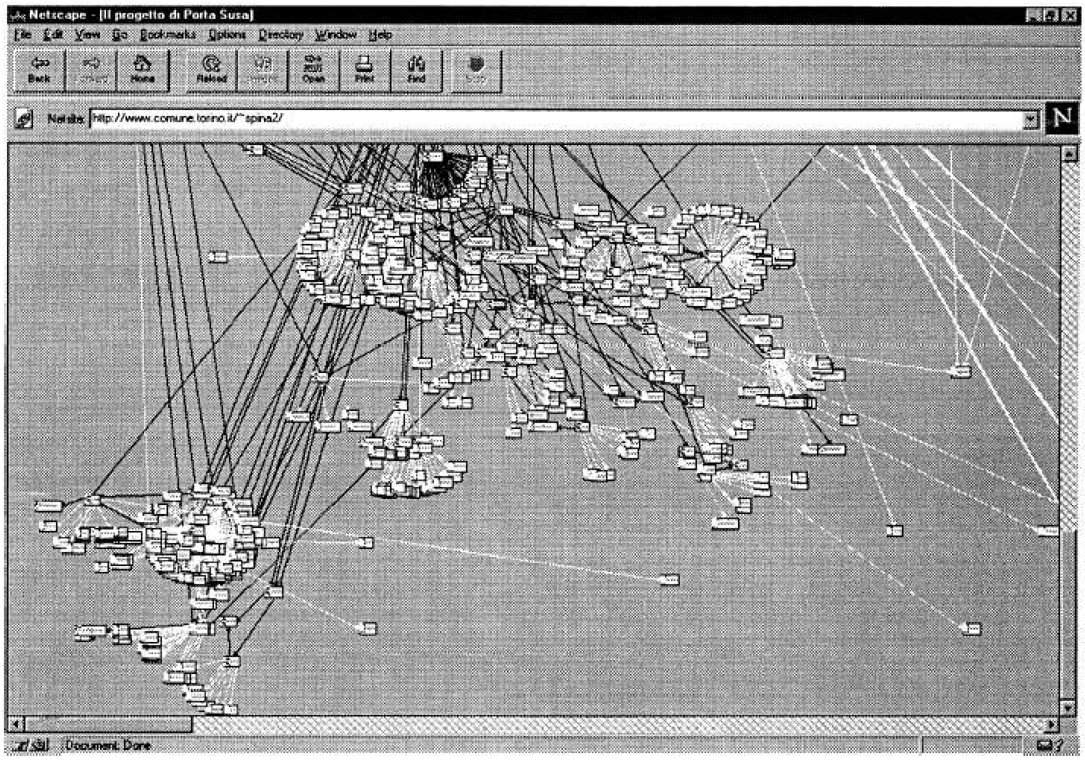

Netscape - [Il progetto di Porta Susa]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Netsite: http://www.comune.torino.it/~spina2/
Document: Done

Illustration 2.5. Screenshot of SVR Document Linking

The virtual environment of SVR was build up by 'Stands' (sites by participating contractors, subcontractors, firms, suppliers and other trading partners), 'Resources' (documents or drawings explicitly represented by 3D icons), and 'Broadcasts' (to communicate with visiting citizens). This VE could solve the problems of accessing massive information (some 20.000 documents, reports, specs, and drawings) and multi-user collaboration simply by creating distributed places: "SVR opens virtual places in Internet, cyberspaces" [PORTA01, pp.220].

All taken into consideration, the Porta Susa implementation seemed quite successful. Interviews to users reported a satisfaction rate between 68% and 76%, depending on previous experience [PORTA01, pp.226].

2.1.6.3.Scene Graph Distribution

A project called SOFT is used as a basis for a study about distributed scene graphs [DISTR-SG99]. The paper examines network architectures to solve the complexity of sharing a common view in distributed virtual realities. Most interestingly, the authors were using a different approach for the distribution of graphical data where scene graphs are applied as a bus.

Primarily, though, they tested two settings: A centralized Java server with serialization mechanisms, and the Dial-a-Behavior (DaBP) protocol. The result of this study was that DaBP in comparison reduces the server's overhead, and provides implementation flexibility [DISTR-SG99, pp.89].

2.1.6.4.Avatars

Avatars are 3D representations of users that are engaged in a virtual environment (see Appendix II).

In recent years, avatar-based communication has attracted a great deal of attention. Several methodologies for avatar motion and interaction have been examined:

- methods to create human body motion in real time by using motion-capture systems [Kalra98]

- to reuse generated motion through combinations [AOHA00]
- techniques for storing acquired motions [ADHA99]
- methods for realizing interaction between virtual humans [Capin99, Toshiya97, and Moser99]
- methods for realizing interaction between a virtual human and its surrounding objects [Joslin01 and Honda95]

Information about specified motions is often stored in advance on the client side, which restricts performable movements and makes it difficult to introduce new motions. A different approach is taken in [DMVSS]; see next section.

Commonly, users select avatars from a limited palette to represent themselves in VE's. As more users use the same avatar, it becomes difficult to distinguish between users. Nakao and Ogawa [AVAT02] propose an Open Avatar Architecture (OAA), which enables users to utilize various virtual spaces using their own avatar.

2.1.6.5.Avatars, the HORB, and Quality-of-Service

In an IEEE article from 2003 [DMVSS], Endo, Yasuda, and Yokoi from Nagoya University in Japan write about a distributed multiuser virtual space system. They describe the usual avatar interaction between users, and examine techniques to express emotions across a network. Their main idea of their paper is to present a method of transferring avatar gestures, such as movements, in a direct matter, contrary to the usual approach of using locally stored action libraries that get triggered when only coded names are transmitted. Although the latter is more efficient for network communication, it limits the users actions.

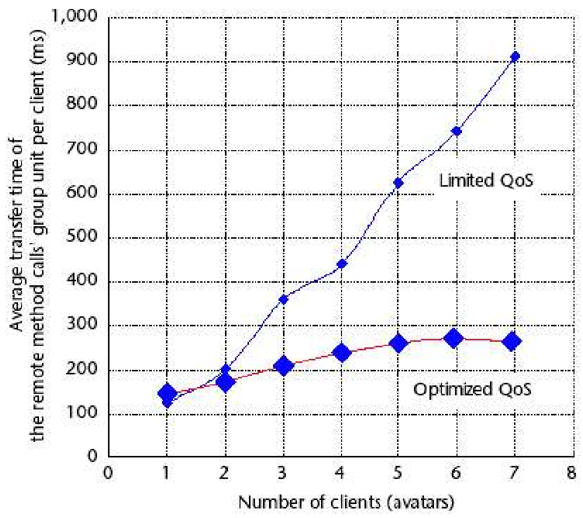


Illustration 2.6. QoS of a DVE

Existing 3D virtual spaces are most often based on single servers that manages all client activity. The problem with this architecture, though, occurs when a growing number of users connect to the space. Their environment is based on HORB [HORB], which allows much greater scalability. HORB consists of 'numerous clients and a set of two network-independent servers' [DMVSS pp.50], provides security as well as communication quality features, and is more flexible than CORBA or RMI.

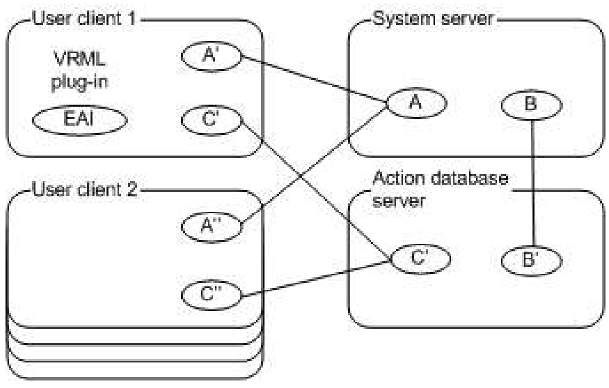


Illustration 2.7. Dual-server DVE System

Through several approaches, and testing of the Quality-of-Service (QoS), they where able to create a system, that was comparably faster (see Illustration 2.6. QoS of a DVE). Optimization techniques included prioritized connections, timeout methods, the HORB's dynamic remote object creation technique, and asynchronous remote methods.

One of the two servers in the system - the action database server - is unique in that it contains a library of avatars and motion information, which is build on the H-Anim specification (a Web 3D Consortium project), and specifies a Level-of-Articulation (LOA) to determine the level of detail of avatars. The system is accessible through any VRML client due to independent library. See Illustration 2.7. Dual-server DVE System.

In large multiuser virtual spaces, the massive volume of information often causes a decrease in client performance [DMVSS pp.52]. With optimized action command information multicasting, motion can be processed in real-time over the network. By controlling the QoS, and applying movement interpolation, network latency and client fluctuations can be avoided. QoS and performance was tested extensively experimenting on a VE with a range of avatars [see graphs on DMVSS pp.55], and the overall conclusion was that their system proved quite efficient.

2.1.6.6.Avatar Navigation, Zones, and Performance

Avatar navigation is discussed in a document by Sung and Park, "The Avatar Navigation of Distributed Virtual Environment By Using Multiview Client" [DVE-AVAT]. They introduce a multi-user virtual environment (see Illustration 2.8. AOI DVE Network) that implements a network structure composed of heterogeneous multi-servers, and multiview clients. The protocols VSTP, which runs on top of TCP/IP, guarantees the connection, and together with HTTP enables navigation in the VE and the web. VSTP is the Virtual Space Transfer Protocol.

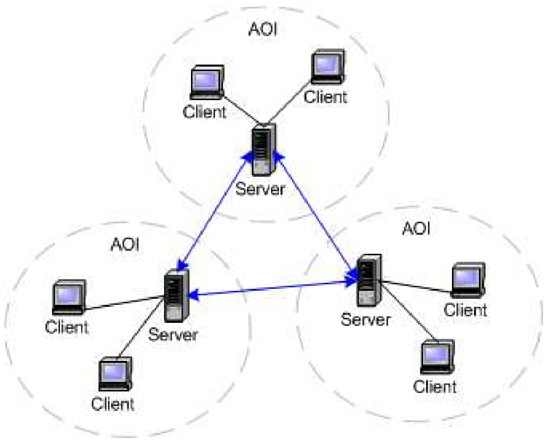


Illustration 2.8. AOI DVE Network

A very interesting part of their DVE network structure are 'Zones'. The network architecture defines a mechanism to reduce network traffic among VE users by dividing the VE space into Zones. Zones are also called "Area of Interest" (AOI), and can effectively be administrated by several servers [DVE97]. The network design restricts the VE to a virtual "room" that corresponds to an AOI; i.e. the VE is partitioned into several rooms, and thus makes the system more efficient (it allows occlusion filtering). Each room could provide different purposes.

The network follows an approach where AOI's are distributed on multiple servers and multiple clients. Both the virtual world and avatar are represented by scripting files, and are rendered client-side thus significantly reducing the server-load.

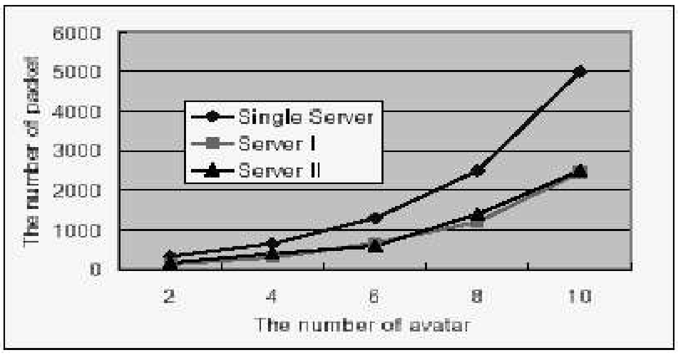
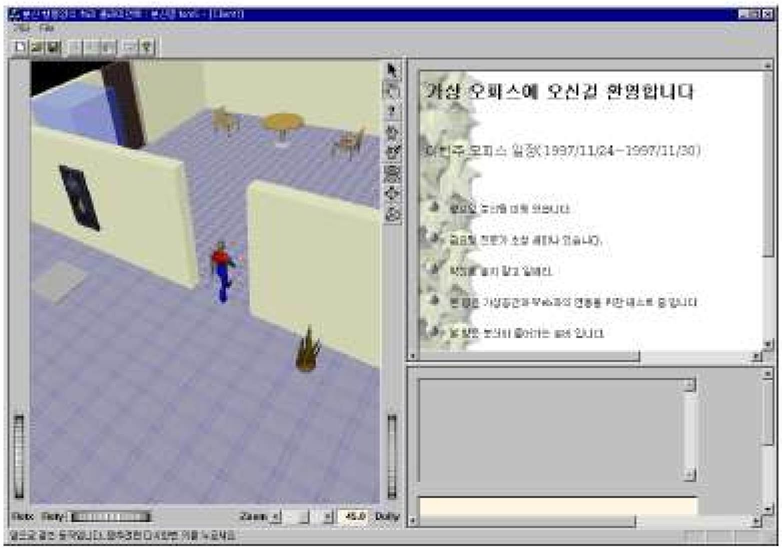


Illustration 2.9. Multi-Server Avatar Statistic

The VSTP protocol allows user representations - avatars - to interact in virtual spaces. In this system, avatars have nine behaviors that are build up by a forward kinematics technique. An avatar server is charge of packet routing and avatar position.

For simplified navigation for the user, the application GUI (see Illustration 2.10. A DVE GUI using 3D and Web Viewer) is split up into views that enable to view both 3D world and web-pages at the same time (this princip is also used in the OpenWorlds project). The render viewer and the web viewer can cooperate with each other to support the better understanding the virtual space.



가상 오피스에 오신걸 환영합니다

이번주 오피스 일정(1997/11/24~1997/11/30)

Illustration 2.10. A DVE GUI using 3D and Web Viewer

The paper also describes how the application setting was tested for performance by comparing the average number of packet transmitted between server and clients with an increasing number of participants. The result is shown in Illustration 2.9. Multi-Server Avatar Statistic.

2.1.7.Cooperative Work

Computer supported cooperative work (CSCW or Groupware) is an area of research, where recent studies have worked towards an integration with virtual environments. Although collaboration can occur on several levels, such as text-based messaging or shared whiteboards, the combination of 3D spaces with the option to increase visualization of users as well as objects would overall enhance the collaborative experience.

In CSCW session management is essential. Session management allows group members to be aware of who is in, who is doing what with whom, how to coordinate interactions, etc. Using an an algorithm for automatic session management, it is possible to enhance the dynamics of cooperation, i.e. members are not constrained, and can easily access shared objects.

So-called 'rule driven' sessions are important to users of a CSCW environment, in that they need clear awareness of group activity in order to collaborate efficiently [Rodden96].

In "Automatic Management of Sessions in Shared Spaces", G. Texier describes a session model suited for cooperative work in shared virtual 3D spaces [AMSSS03]. The session model structures interactions on shared objects, without requiring explicit session preparation.

As cooperative applications are distributed by essence, maintaining data consistency is a difficult issue, this session model and its rules defines groups of objects which then can be used by a distributed consistency management service.

2.1.8.Distributed Computing

The SETI screen saver was (and still is) one of the widest used applications that harnesses idle computer processing power. By distributing the enormous amount of calculations to thousands of desktops connected to the internet, it is possible to gain computational power equivalent to modern super computers (or dedicated 'cluster' computing).

Since then, many organizations have created similar 'render farms' to plow through bulks of data. Finding the human genome, searching for medical compositions, or predicting weather are a few examples of where this processor-sharing model is very effective.

Philip K. Dick's novel "Do Androids Dream of Electric Sheep?", which the movie Blade Runner was based on, describes almost-human androids..

A distributed screen saver with the name “Electric Sheep” has the purpose of animating and evolving artificial life forms. “The project is an attention vortex. It illustrates the process by which the longer and closer one studies something, the more detail and structure appears” [Sheep03].

A shared visual space is rendered on each node, and frames uploaded to a central server, which in turn creates an animation of fractal flames. This 'phenotype of an artificial organism' is called an "electric sheep", and its shape is specified by a string of 120 real numbers — a 'genetic code'.

The client node downloads the electric sheep, and displays it as a continuous sequence on the screen. Distribution occurs every 5 minutes, and a sheep's life is finite dependent on disk quota. Popular sheep - selected by user votes - live longer, and are more likely to reproduce. "Users' preferences provide the fitness function for an aesthetic evolutionary algorithm (an idea first realized by Karl Sims)" [Sheep03].

2.2.Distributed Virtual Environments

2.2.1.Introduction

Distributed Virtual Environments (DVE's) strive to create a realistic, semi-immersive "virtual world" experience for users by incorporating 3D graphics and stereo sound [DVE-CS03]. DVE's act as a base for interactively sharing information and manipulating objects by a network of users.

They can potentially be used in a variety of areas, including entertainment, education, engineering, design, and commerce. In recent years, research of large-scale DVE's has been growing significantly.

DVE research actually diverts quite broadly, as the following loosely-ordered listing illustrates:

- **Architectures**
  - Projects [DIVE98][NPSNET94][DVE-P98][PARADISE][NVE-D&I99][Singal99][MASSIVE95][OW][LSVR99]
  - C/S System [RING95][NPSNET94][MASSIVE]
  - Frameworks [DVE-Frame02][DVE-MNG00][MSVW02][DVE-VJC01]
  - Object Description Language [NPSOFF94]
  - Interactivity VE's [DIVE96]
  - Performance [DVE-PERF02][DVE-PLIM]

- **Large-scale**
  - Projects [SPLINE][DVE-SA95][NPSNET94][HIVE][MASSIVE-3]
  - Multicast Groups [MCG95]
  - Mobile Objects [MOBJ95]
  - Massive Distribution [MASSIVE99][MaDViSPE][MaDVi02]

- **Networking**
  - Protocols [Kawakami98][DWTP98][DVE-VRML][NOMAD01][JXTA-P03][P2P-ACAD][Sheep03][ARA00][X3D]
  - Active Services [SRM00]
  - Active Networks [AN-Survey97]
  - Handling Heterogeneity [HET03]
  - Interacting [CVE95]
  - Taxonomy [DVE-TAX97]
  - Perception Filter [DVE-Filter02]
  - Fidelity Optimization [DVE-Fid00]

- **Communication**
  - Communication System [DVE-CS03]
  - Multicast Framework [SRM97]
  - Message Filtering [3DDMF01]
  - Direct Conversation [Kawakami98][Lea97]
  - Multimedia Communication [DVE-MM99]
  - Session Management [AMSSS03]
  - Interest Management [EIM-SDVE99]
  - Sharing Attractions [Joslin01]

- **Building and Visualization**
  - Navigation [CITY94][NavISpace]
  - Environment Manager For Building DVE [EM95]
  - Virtual Bodies [VBVS01]
  - Virtual Object Interaction [Singal99]
  - Urban Planning [PORTA01]
  - Virtual Construction and Manufacturing [VRBC][VSVM01]

- **Social and Entertainment**
  - Virtual Society [VSCollab96][Honda95]
  - Education and Learning [DVE-E00][Learn99]
  - VE games [DEE98][Game98]
  - Social Aspects [Lea97]
  - Collaborative Sculpting [COLVS01]

- **Graphics**
  - Animation [Kalra98][AOHA00][Toshiya97][Moser99]
  - Distributed Scene Graphs [DISTR-SG99]

- **Avatars**
  - Studies [Capin99][AVAT02][DVE-AVAT]
  - Gestures [DMVSS]
  - Action Database [ADHA99]
  - Awareness [Rodden96]

This list of research papers is not extensive. An overview of these and other resources is available in appendix I and on this project's website.

Research of DVE shows that the evolutionary stage of such systems has moved quite far; many rather explicit subjects are being illuminated by scientists and engineers.

A DVE system has four basic components [NVE-D&I99]:

- Graphics engines and displays
- Communication and control devices
- Processing systems
- A data network

Each of them can be composed of of specific techniques, platforms, or sub-systems. This complexity is inherent in most designs of distributed environments, as it requires a somehow wide range of underlying mechanisms.

2.2.2.Challenges

The challenges raised by DVE's [DVE-Future02]:

- How to provide network support for the mixed traffic types demanded by DVEs, typically voice, video and data, with their very different characteristics with regard to network properties such as delay, jitter and loss rate?
- How to manage group participation in a DVE? Not all data are relevant to all participants in a DVE.
- How to manage persistent state within a and manage state updates?

One of the main difficulties of implementing DVE systems, besides overcoming the complexity, lies in its lack of efficiency, especially when scaling to a large number of nodes. Limitations to be considered are network-implicit data losses, limited bandwidth, and delay.

As numbers of users grow, a number of factors will increase [DIVE98]:

- Geographical distance between participants
- Network distance between participants
- Total user population
- Number of simultaneous participants
- Scope of participant awareness, i.e. the fraction of the total environment to which a single participant has access at any moment
- Complexity of the virtual environment
- Richness of communication, e.g. number of media, level of detail or fidelity
- Variability of delivery platforms

Each user or participant in a DVE is usually represented graphically by an avatar, which logical state has to be updated to prevent inconsistent representations of the VE among different hosts. The same is true for object interaction.

This constant state exchange results in a large network traffic volume, and requires optimization to employ bandwidth most efficiently. Filtering data using a participant's interaction scope (i.e. not all participants need to receive all updates) is one approach to optimize traffic.

The necessity to update the display at reasonable rates is a major bottleneck in graphical applications. Thus, research has been devoted to solve the problems, i.e. by building networked rendering clusters.

For example, Illustration 2.11. Parallel graphics systems shows a classification according to the point in the processing pipeline at which data is redistributed to multiple subsystems. N is a node, and I-S-R-D represents the processing pipeline. The four examples to the right show normal distributed input data, a replicated application state, parallel renderers, and parallel renderers with image composition [AVOCADO, pp.27].

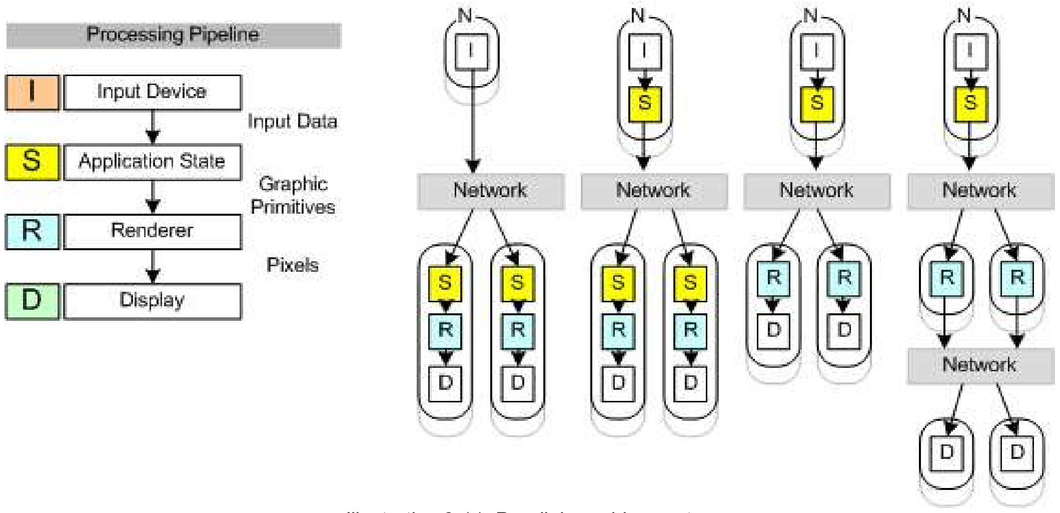


Illustration 2.11. Parallel graphics systems

Other difficulties concern the spatial design, user interaction and navigation, synchronization, and communication inside virtual environments. Comparing a DVE system to a car metaphor, it requires both the underlying infrastructure, such as highways (network protocols), traffic (scalability and performance), and traffic-lights (latency), as well as an engine (3D visualization), wheels (communication platform), steering and screen (a usable interface), road-signs (navigation), and of course a purpose to drive somewhere in the first place.

The car metaphor primarily highlights the networking aspects. The two main approaches in network architectures for DVE's are client/server and peer-to-peer, but hybrid systems that take the best from both worlds are emerging, especially considering that both extremes are characterized by increasing difficulties.

But this is merely half the story, as a car's only purpose usually is to transport someone from A to B. The metaphor is not adequate to describe the entire level of user interaction and collaboration, the visualization of information, the user interface, access control and sharing, or building virtual world's. A different metaphor might be useful here: that of a city [see also CITY94, NavISpace, PORTA01, and Rodden96].

2.2.3.Client/Server Architecture

A single server manages the state of the virtual environment. A client sends an update to the server which is then propagated to other clients. All communication goes through the server, which becomes both a bottleneck and a single point of failure.

As in Web applications, client caching strategies also cause difficulties and require sophisticated protocols to maintain consistency and freshness of state. However, the client/server approach is relatively simple to implement and provides a perfectly satisfactory solution for DVE's up to some size limit.

This approach is also taken in internet game servers such as Kali, TEN and Mplayer [DVE-Future02, pp.2].

2.2.4.Peer-to-Peer Architecture

The P2P network distributes the state amongst all clients. Each client has a partial copy of the state. When a change occurs, a client has to send the changes to all other clients. If a client fails or leaves the environment, its data are lost, but other clients can continue without it (under some assumptions about how data are distributed). Thus, peer-to-peer communication is a more promising approach to large scale DVE's than the client-server approach.

The key idea in making such an architecture scalable is the use of multicast groups [CN96]. Multicast protocols enable a sender to send data to a set of hosts in a multicast group, avoiding unnecessary duplication of transmission by sending only one copy of the data along each arc in the routing tree between sender and receivers [DVE-Future02, pp.2].

The DIVE - Distributed Interaction Virtual Environment [DIVE98], and the MASSIVE [MASSIVE99] systems are both based on P2P architectures. Multicast delivery systems were first used for the delivery of time critical data such as video and audio streams, where it is pointless to retransmit lost packets.

2.2.5.Hybrids

There are hybrid architectures constructed from both extremes, which for example focuses on servers communicating in a peer-to-peer manner [RING95]. The server bottleneck that occurs when the number of clients increase can thus be eliminated.

A different approach, the 'Peer/Server' architecture, focuses on communicating using peer-to-peer with some nodes and through a server with some other nodes. An example of this system is the "Distributed World's Transfer and Communication Protocol" (DWTP) [DWTP98].

The "Virtual Reality Transfer Protocol" (VRTP) is a project by the Web3D consortium, which aims to enhance the networking of shared 3D worlds, and to keep pace with the subsequent explosion of network demand by large-scale DVEs [VRTP].

VRML supports large-scale web-based virtual environments, but since it runs on top of HTTP, it is insufficient for large-scale DVE's [VRML97]. To this extend, VRTP was created to support interlinked VRML worlds in the same manner as HTTP was designed to support interlinked HTML pages. This architecture aims to provide functionality that exists between client-server and peer-to-peer approaches.

The DIS-Java-VRML composition [DJV] has been created to establish conventions for building multicast-capable large-scale VE's, and is another Web3D working group. DIS is essentially a behavior protocol tuned for many-to-many interactions. Java is the programming language used to implement the DIS protocol, perform mathematical calculations, communicate with the network and communicate with the VRML scene. VRML 3D graphics are used to model and render both local and remote entities. Using DIS, Java and VRML can provide all necessary capabilities needed to implement VE systems.

2.2.6.Research

One of the pioneering distributed virtual environments was the **SimNet** (Simulator Networking) System by the Institute for Defense Analysis in 1990 of the American Department of Defense (DoD). SimNet is a military training DVE, and a formalized Distributed Interactive Simulation (DIS) protocol (IEEE 1278 standard) was a result of that research [IEEE1278]. This system, however, is not ideal for large-scale multi-player VE's [MCG95].

The Naval Postgraduate School created **NPSNET-IV** [see DVE-SA95, DVE-TAX97, and MCG95], which builds on DIS. This system managed to optimize data load (filtering) for nodes and network by partitioning the virtual world into classes [DVE-CS03].

DIVE - The Distributed Interactive Virtual Environment [see DIVE96 and DIVE98] is a network software architecture of a DVE platform designed to scale by using a group communication system, and a reliable multicast protocol (with NACK).

DIVE partitions the virtual world into smaller regions, avoids data filtering within a world, and implements a multicast group for each. All objects in the scene graph are replicated. A joining program receives a complete copy of the current world state (via TCP/IP). Update messages propagate changes (a fixed set of events).

The C/C++/TCL system (multi-threading) toolkit runs on SGI IRIX and Linux. The object model consists of objects, attributes, and scene graphs. The event model consists of global events and callback subscription. Persistence by saving environment state. The distributed event model implements state change and attachable notification handlers. Network transport goes via UDP (IRIS group communication), and the approach is process-grouping.

HIVE – A large-scale kernel system. "This project focuses on the development of a unified kernel, based on an underlying time parameterized environment model incorporating time based reasoning, which reduces latency by anticipation and advance communication of events" [HIVE]. See also [MASSIVE-3] and [MASSIVE95].

MASSIVE - The Model, Architecture and System for Spatial Interaction in Virtual Environments system [MASSIVE95], uses a spatial model for interaction among clients. The Massive project is distinguished by three versions.

Multiple users communicating via a combination of 3D graphics and real-time packet audio are supported in [MASSIVE-3]. The system provides scalability, world-composition, locales (originated in SPLINE), and message handling with integrated causality and advance communication of events (originated in HIVE).

Each object has an aura that determines the space within which interactions are possible. Two participants exchange state updates when their auras collide [DVE-CS03, pp260].

The original Massive is a single-threaded C system toolkit that runs on SGI IRIX. The (distributed) object model consists of objects and attributes. The event model consists of global events and callback subscription. There is no persistence. The distributed event model implements dedicated point-to-point connections between objects. Network transport goes via TCP/IP, and builds on a Client/Server architecture.

The distributed object model in CVE (MASSIVE-2) uses flat object sets and replicated objects. The distributed event model uses dedicated point-to-point connections between objects. The distributed object model in HIVEK (MASSIVE-3) uses an object hierarchy and replicated scene graphs. There is no distributed event model.

Network transport in both CVE and HIVEK goes via TCP/IP point-to-point and UDP multicast, and builds on both C/S and P2P architectures.

RING is a client-server system. The distributed object model uses flat object sets and replicated object positions. There is no distributed event model. Network transport goes via TCP/IP point-to-point, and builds on a C/S architecture. In RING, the partitioning criterion is not based on spatial position (as in SPLINE, CVE, HIVEK, and NPSNET), but on visibility.

PARADISE - The Performance Architecture for Advanced Distributed Interactive Simulation Environments at Stanford is a large-scale multi-user simulation environment over a wide-area network [PARADISE].

SPLINE - The Scalable Platform for Large Interactive Networked Environment [SPLINE96] splits the world into locales, which can be processed separately [DVE-CS03, pp.260]. Spline provides the following key capabilities [SPLINE]: Multiple users, Spoken interaction, Computer simulations, 3D graphics and sound, Run-time modifiability, and Open Interfaces.

The distributed object model uses flat object sets and replicated objects. The distributed event model uses dedicated point-to-point connections between objects. Network transport goes via TCP/IP point-to-point and UDP multicast, and builds on both C/S and P2P architecture.

MR Toolkit (no longer available) - This single-threaded C toolkit runs on SGI IRIX. The object model consists of tagged vertex lists and state variables. The distributed object model uses selected memory locations. The event model consists of global events and callback subscription. There is no persistence. The distributed event model uses event broadcast. Network transport goes via TCP/IP, and builds on a P2P architecture.

Other examples of DVE related projects are **V-Worlds**, a group at Microsoft [MSVW02], **URBI ET ORBI** [DVE-MNG00], the **NOMAD** framework [NOMAD01], Java middleware system, like **FlexiNet** [FLEXI99], mobile services like **Voyager** [VOY02], and agent technologies such as **Aglets** [AGLETS02].

It is apparent, that research projects in virtual environments and distributed virtual environments are aggregating. For a nice overview, see Appendix IX for a comparison table on VE and DVE projects by [AVOCADO].

2.2.7.Event Driven DVE

An Event-based Architecture is presented on an online paper [DVE-VRML]. This event-based notification system is a DVE being constructed of zones, i.e the DVE represents a network of zones. The authors define a zone, as a representation of a collection of information of interest to participants in the DVE [DVE-VRML]. Zones can be defined spatially, as in the SPLINE system [SPLINE].

An Event Distributor (ED) acts as the coordination point for a zones events, and runs a zone server extension to implement the zone-specific functionality. A zone server implements a generic state-sharing protocol using events. Events exist for object creation and deletion, state updates, and operations on objects among others. Events are not interpreted by the ED mechanism, but by the object receiving an event, and state update content can be defined independently.

The DVE system is based on the 'Keryx Notification System' (KNS), which employs a publish/subscribe distribution mechanism to optimize bandwidth. Participating nodes can apply filters via the ED to limit their event flows to the zones they are interested in. The mechanism is described in detail at [DVE-VRML].

2.2.8.Division of the Environment

The "impact of the communication system on distributed virtual environments" is described in a paper by Teixeira and Duarte [DVE-CS03].

Their platform model the communication requirements. Performance was evaluated via simulation, which included factors such as number of nodes, partition of the virtual environment, network node latency, number of messages exchanged, and the size of the virtual world. Their results show that the division of the environment improves overall performance.

The platform used an event driven approach, and reduces reliable communication by dividing virtual world operations into critical operations (require reliability), and non-critical operations (delivered in real time). In their simulations, the VE was divided into fixed areas, while model behavior was captured to evaluate their communication performance.

Simulation results showed, that this division of the VE proved vital for system scalability. The trade-off between interactivity gain versus interruptions generated by hand-offs were not significant [DVE-CS03, pp.276].

2.2.9.MaDViWorld

The MaDViWorld framework is a "software architecture supporting massively distributed virtual world systems" [MaDViSPE], which goal is to distribute its virtual subspaces on an arbitrarily large amount of machines without making concessions to the single server architecture. The system avoids the limitation of a few centralized servers, and instead supports a distribution model.

In their papers about the "Massively Distributed Virtual Worlds" system [see MaDViSPE and MaDVi02], Patrik Fuhrer, Ghita Kouadri Mostefaoui and Jacques Pasquier-Rocha explain: "We want to support a World Wide Web type of organization, where each individual server only maintains a very partial portion of the whole system" [MaDViSPE, pp.656].

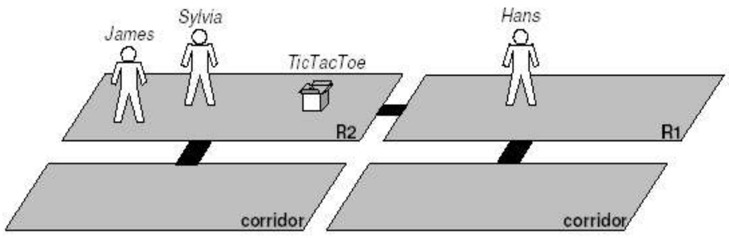


Illustration 2.12. MaDViWorld Abstraction

Fuhrer et al. describe two basic paradigms that describe networked applications: The document paradigm, where documents (often active ones) are able to react to various user actions, and are made available on one or several servers and client applications (e.g.Web browsers). And the the virtual world paradigm, where multiple users and active objects interact in the same space and therefore have a direct impact on each other [MaDViSPE, pp.646].

The first paradigm is "a metaphor of a large cross-referenced book", while the second displays much more complexity in that events have to be synchronized and forwarded in order to maintain consistency of the world. Most of the latter systems are based on architectures that implement central servers containing all data related to the VE, handling consistency and persistence for all attached clients. The paper argues that this approach has two main weaknesses: First, the whole system depends completely on the central server robustness, and secondly, it does not scale well.

The MaDViWorld layered structure [MaDViSPE, pp.660]:

- Specification layer - Avatars and Rooms, RoomFactory
- System Implementation layer
- Object Implementation layer

Subspaces are called 'rooms', which are essential to divide the world into subspaces to make it scalable. Each room can have 'doors' that work as gateways to different rooms [MaDViSPE, pp.648]. Rooms can be reached directly by a specific address. End-users avatars can explore the world transparently, without knowing on which machine the virtual room runs on (see Illustration 2.12. MaDViWorld Abstraction).

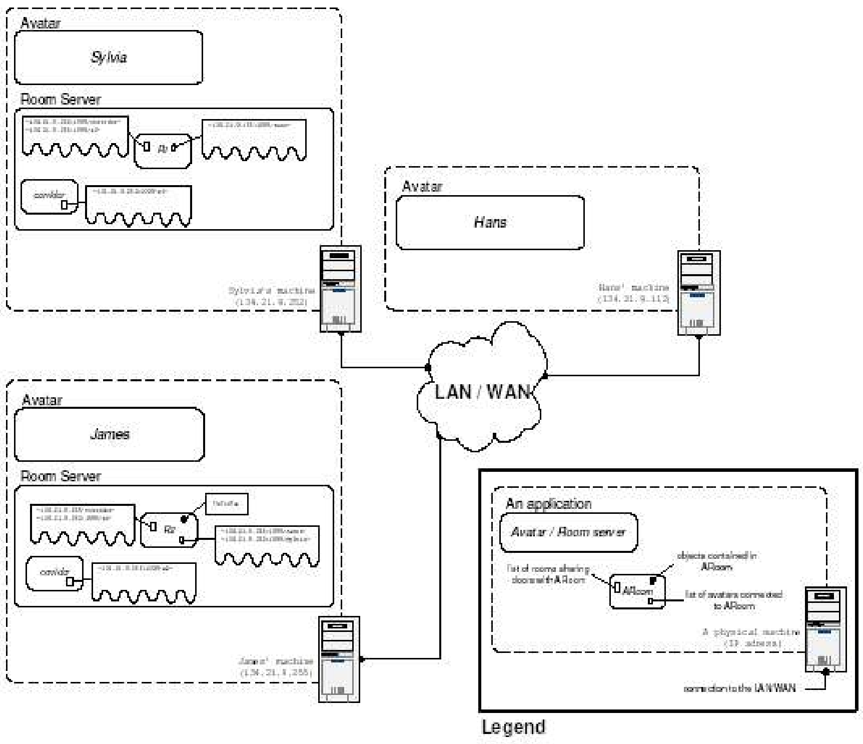

Avatar
Sylvia
Room Server
corridor
Sylvia's machine (134.21.9.252)
Avatar
Hans
Hans' machine (134.21.9.112)
LAN / WAN
Avatar
James
Room Server
corridor
James' machine (134.21.9.255)
An application
Avatar / Room server
list of rooms sharing doors with ARoom
ARoom
objects contained in ARoom
list of avatars connected to ARoom
A physical machine (IP adress)
connection to the LAN/WAN
Legend

Illustration 2.13. MaDViWorld Physical Layout

MaDViWorld can be seen from the perspectives of a user and a developer, people either purely using the DVE for their own amusement, or people further extending its functionality. The user level perspective needs to take several aspects into account: The World topology, Interactions, Active/mobile objects, World creation and extension, Persistence and recovery, and Security [MaDViSPE, pp.650]. The world developer level has to focus on aspects such as new types of objects, creating new avatars, and new types of rooms [MaDViSPE, pp.651].

Objects are distinguished into Passive objects, Reactive objects, and Active objects [MaDViSPE, pp.648]. This devision is based on how objects interact with avatars or each other.

Object mobility is the central mechanism of MaDViWorld, and is illustrated by the following virtual activity: After playing a virtual game with Sylvia inside Room 1, the user Hans "selects the game object and clicks on ‘take’. The object is automatically put into his avatar’s bag. Then he enters Room 2 and clicks on ‘put’. Sylvia joins Room 2 and both users replay the game" [MaDViSPE, pp.654].

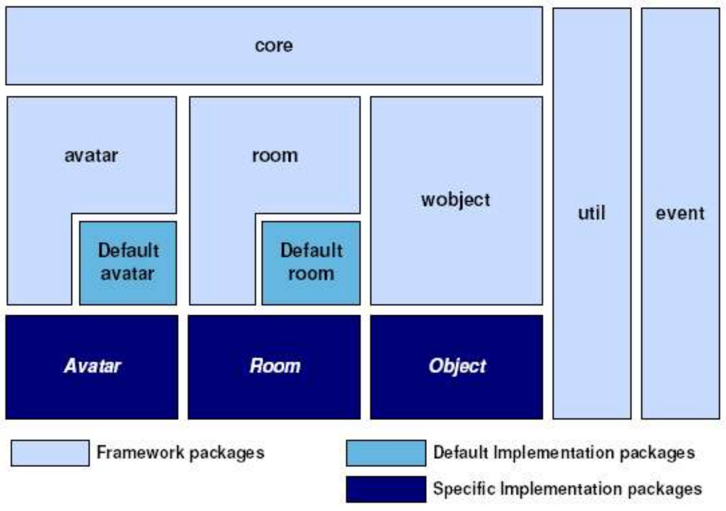
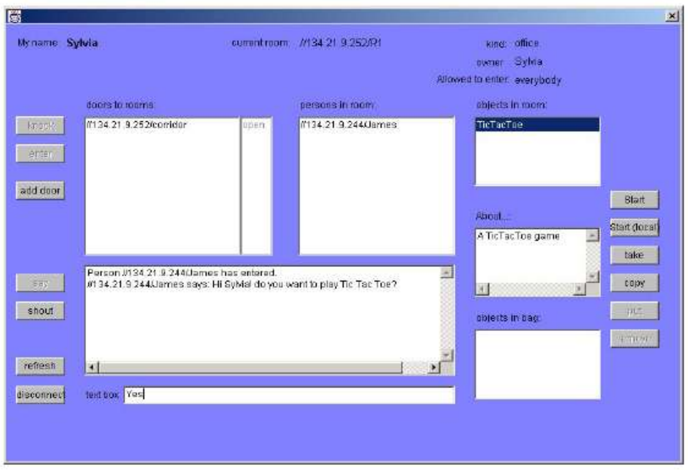


Illustration 2.14. MaDViWorld Architecture

The physical model (see Illustration 2.13. MaDViWorld Physical Layout) behind the scene is build up by:

- 2 room server applications
- 3 avatar applications running together on different machines
- a naming service for the virtual rooms and the avatars on each machine
- relations (events etc. managed by the room) between the objects within a room and the avatars connected to it

A room or the naming service might become bottlenecks, if one either populates a room with too many objects and avatars or puts too many rooms on a single server. This should be averted by the MaDViWorld design, though.



My name: Sylvia
current room: //134.21.9.252/R1
kind: office
owner: Sylvia
Allowed to enter: everybody
//134.21.9.252/corridor
//134.21.9.244/James
TicTacToe
A TicTacToe game
Person //134.21.9.244/James has entered.
//134.21.9.244/James says: Hi Sylvia! do you want to play Tic Tac Toe?
text box: Yes

Illustration 2.15. MadViWorld UI

MaDViWorld's main strengths and benefits is its 'massive distribution' of virtual world resources. According to the authors, its framework shows the following achievements, while resisting to be a new Java middleware system, a mobile service, or an agent system [MaDViSPE, pp.665]:

- pure Java, RMI-based, and distributed
- persistence mechanism
- framework is extensible
- class and object mobility facilities
- avatars in room can share the same objects

The MaDViWorld architecture is quite interesting in the context of this project, as it focuses on the 'zoning' mechanism as well as a Java-based, object-oriented, extensible approach.

2.2.10.Active Networks

Balikhina, Ball and Duce ask the question "Distributed Virtual Environments - An Active Future?" [DVE-Future02], and contemplate whether active networks might offer potential solutions to scalability issues for large scale Distributed Virtual Environments (DVE's).

Examining architectures of DVE systems and recent work in Active Network research, the authors suggests “potentially beneficial synergies” between the two fields.

Usually, computational resources only exist outside the core of conventional networks, as the primary function of the network is end-to-end routing of traffic. Challenging this assumption is the recent research of Active Networks [AN-Survey97].

Active Networks assume an unconventional approach by placing “computational resources directly within the network specifically to support end-user processing requirements, higher performance will be achievable than by conventional means, service deployment will be improved and new classes of service will become possible” [DVE-Future02, pp.3].

See Illustration 2.16. Processing with the nodes of an active network, for an example of how an event is forwarded through an active network.

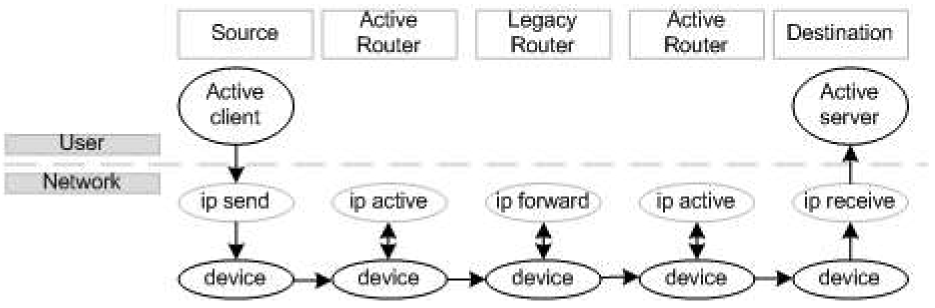


Illustration 2.16. Processing with the nodes of an active network

Harvesting the benefits by deploying Active Networks naturally requires the addressing of potential problems such as management, reliability and security. Scalable Reliable Multicast (SRM) is one class of protocol that has been shown to benefit from the presence of small amounts of session specific computation at key routing nodes, increasing throughput, decreasing latency and reducing congestion [SRM00].

The difficulty in designing scalable multicast protocols that supports reliable delivery has been met by the Scalable Reliable Multicast framework [SRM97]. The SRM framework provides basic functionality for scalable application level reliability in multicast environments. SRM is fully decentralized and handles arbitrarily large groups of participants. To achieve reliability, receivers multicast a so-called "repair request" (NACK) to ask for missing data.

Another approach, besides SRM, was suggested by Keller et al. in their "Active Router Architecture" [ARA00], which provides transcoding (digital-to-digital encoding) services within a network to address issues of bandwidth heterogeneity amongst destinations in a multicast group [DVE-Future02, pp.3]. In this approach the application selects certain transcoding processes, while the network is able to apply them.

2.2.11.DVE performance

2.2.11.1.Multi-server DVE performance

Multi-server DVE performance is tested in a document by Ng, Li, Lau, Si, and Siu [DVE-PERF01].The question they ask is, how large scale DVE's manage the workload, and how an implementation of a multi-server based DVE system could be accomplished best.

In their study, multi-server DVE systems are tested for performance by experiments under various conditions. The resulting data showed that both latency time and response time decreases exponentially with a larger number of servers, but increases with the number of objects (see fig.1 and fig.2 on Illustration 2.17. DVE Performance Study).

Further, "observe that when there is a large number of objects (>4000 objects) and a large number of clients (>64 clients), the latency time and response time start increasing super-linearly" [DVE-PERF01, pp.91].

Incrementing the number of objects does not necessarily require more bandwidth, and thus has less effect on performance. "In contrast, increasing the number of clients will contribute to a greater amount of data transmitted from the server" [DVE-PERF01, pp.91]. The bottom line is that, compared to a single-server architecture, there is a clear performance gain in applying a multi-server system.

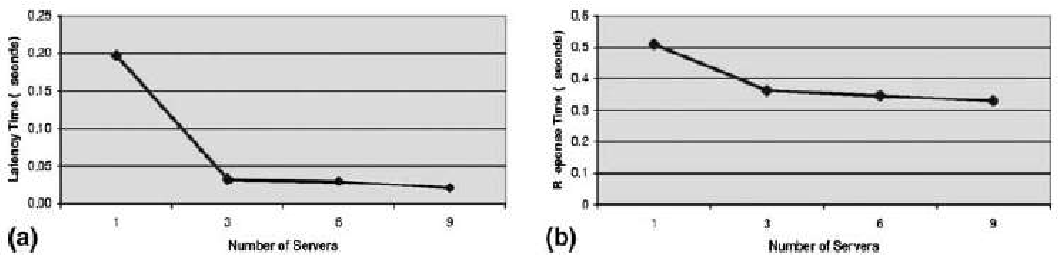


Fig. 1. The effect of number of servers on the performance of the DVE.

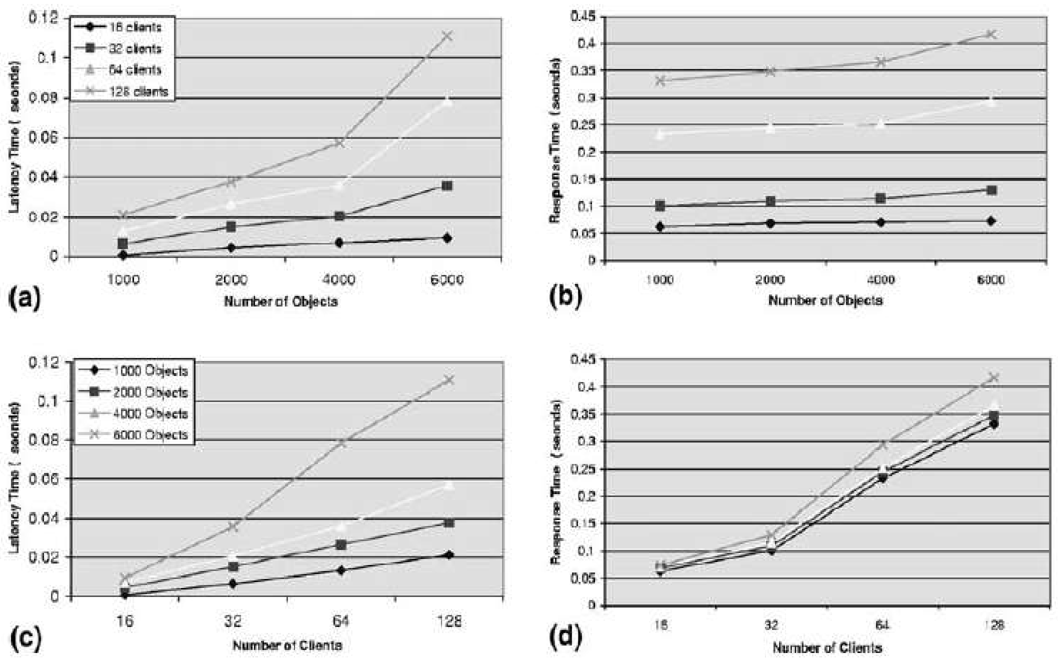


Fig. 2. The effect of number of objects and clients on the performance of the DVE.

Illustration 2.17. DVE Performance Study

2.2.11.2.3D Sub-Spaces to Increase Performance

How performance limitations affect DVE is also examined in a paper by Enser, Carraro and Edmark [DVE-PLIM]. The authors primarily describe systems that represent a VE as multiple regions in order to compensate processing limitations. Further, the paper presents a method where the view of a world scene can be 'extended' (similar to interpolation) to avoid hick-ups when delayed scene updates occur.

In a simulator called Peloton, multiple users compete within a VE. The system's display is a composition of multiple 3D sub-spaces (called regions) that each can be made up of graphical objects, still images, or video streams (see Illustration 2.18. DVE Subspaces).

The idea is that limited computation and communication resources in DVE systems can be solved by altering the visual representation. Depending on the computing and communication resources available during a simulation, the 3D spaces could be adjusted to available performance. DVE system components could be modified through both static and dynamic settings. Hence, the "graphical complexity of the regions of a virtual world can be tailored" to current performance limitations [DVE-PLIM, pp.199].

In essence, Ensor et.al. suggest that the technique of configuring different VE regions to match their performance characteristics of underlying computing and communication systems, yields better advantage of available resources.

A peculiar side-effect of their research was, that "Pyramidic panels are useful not only in multi-region virtual worlds; they can also support changing viewpoints in the more common image-based virtual worlds" [DVE-PLIM, pp.203].

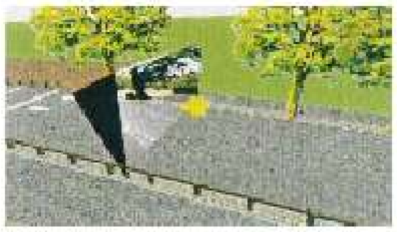


Illustration 2.18. DVE Subspaces

2.3.3D Technologies

2.3.1.Java3D API

Java3D is based on Java technology, and as such both scalable and platform-independent. "The Java 3D API provides a set of object-oriented interfaces that support a ... model to build, render, and control the behavior of 3D objects and visual environments" [Java3D].

Shortly outlining the capabilities, Java3D offers a range of 3D-features. The high-level API avoids tedious implementations such as rendering pipelines, supports a wide variety of formats and run-time loaders (VRML, CAD, interchange formats, etc.), is network-centric, portable, and scalable. Further, "the viewing model scales from a flat screen to stereo, to a tesselated wall of monitors, to fully immersive portals and caves - all without rewriting code" [Java3D].

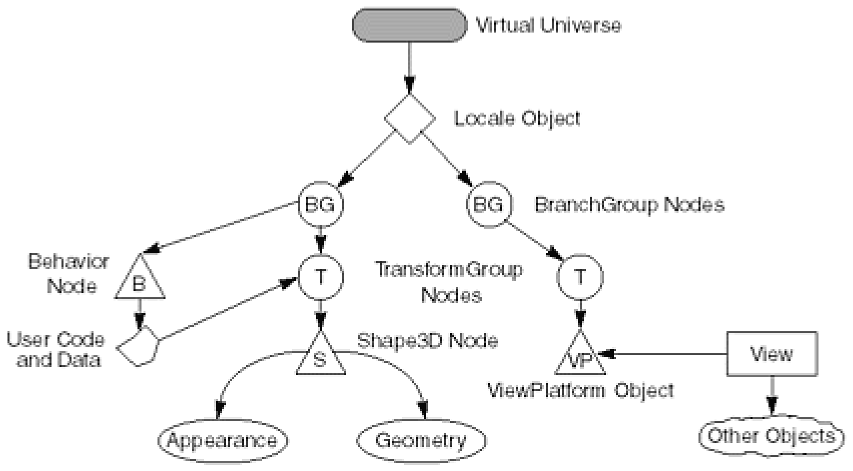


Illustration 2.19. A sample Java3D Scene Graph

The API can utilize OpenGL or DirectX, and strives to prioritize performance. Certain aspects that have been optimized, and issues that influence performance include:

- The ability to set capability bits to determine which objects may change at run time.
- The support of two different compiling methods to "compile" the data into a more efficient rendering format.
- Bounds to limit the spatial scope of a specific object (Lights, Behaviors, Fogs, Clips, Backgrounds, BoundingLeafs, Sounds, and Soundscapes). It is possible to disregard the processing of any objects that are out of the spatial scope of a target object.
- Unordered rendering of leaf nodes, which have no effect on other leaf nodes, and therefore may be rendered efficiently in any order. Usually, state required to render a specific object is defined by the direct path from the root node to the given leaf.

- Using appearance bundles. Appearance being a collection of references allows the implementation to minimize state changes in the low level rendering API. A node has a reference to a Geometry and an Appearance.

Java3D is a viable choice to implement specific 3D-based user interfaces that require the flexibility and compatibility of Java. It was therefor chosen as the basic 3D environment platform for this project's application.

Java3D uses a scene graph (SG). SG's are hierarchical structured objects, which shows their relationship to each other. A local coordinate system describes the information relative to the parent object. Descending on each level, there is a grouping structure, containing objects of similar characteristics, inherited properties from parents, and the ability to move the object relative to the parent. Illustration 2.19. A sample Java3D Scene Graph shows a typical structure.

2.3.2.J3D-UI Framework

The Java3D User Interface Framework (J3DUI) is a set of packages, which utilizes the Java3D API and mainly supports manipulation of in-scene objects [J3DUI00].

J3DUI essentially provides an upper layer for intuitive user interaction that is limited to the input devices mouse and keyboard, and the 2D screen output device (in contrast, the Java3D API as J2SE itself has few restrictions on its usage and offers formidable extensions).

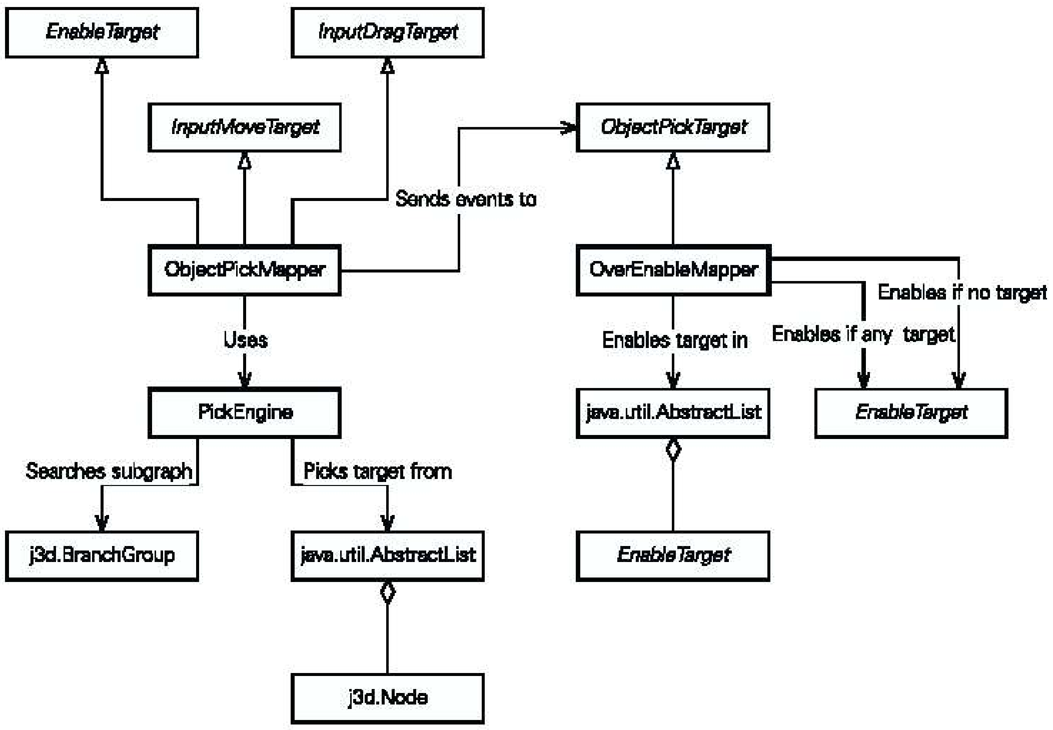


Illustration 2.20. J3DUI Framework - Object Picking

The framework contains a collection of classes and methods that are convenient when implementing 3D environments. They bundle commonly used event-chaining, input/output interfaces, vector coordinate conversions, and SG branching organization. Aiming at small-sized Java applications that want to implement a 3D user interface (i.e. to show furniture), the framework tries to simplify the programmer's task by providing those methods repetitively used.

One of the core features of the J3DUI are object picking operations (located in the sub-package `j3dui.control.mappers`). The class map for this feature is shown in Illustration 2.20. J3DUI Framework - Object Picking.

Further, J3DUI supports discrete and continuous picking, as well as bounds and geometric picking, and can be used for target control enabling, or for feedback and selection [J3DUI00].

Due to several interesting features that generally addressed the interface between the user and the 3D scene, and in particular supported object visualization, control and manipulation, as well as the conversion methods that are necessary and implicit in a desktop 2D to a world 3D translation, the framework was chosen as a major supplement in the application architecture.

2.3.3.X3D

Extensible 3D (X3D) is a software standard for “defining interactive web- and broadcast-based 3D content integrated with multimedia" [X3D].

X3D is a format for integrated 3D graphics and multimedia, and succeeds VRML with new features, additional data encoding formats, and a componentized (modular) architecture. Among other areas, the interchange format X3D was modeled to support shared virtual worlds.

The X3D feature set includes (from the web3d.org website; ISO/IEC FDIS 19775-1:200x):

3D graphics - Polygonal geometry, parametric geometry, hierarchical transformations, lighting, materials and multi-pass/multi-stage texture mapping

2D graphics - Text, 2D vector and planar shapes displayed within the 3D transformation hierarchy

Animation - Timers and interpolators to drive continuous animations; humanoid animation and morphing

Spatialized audio and video - Audiovisual sources mapped onto geometry in the scene

User interaction - Mouse-based picking and dragging; keyboard input

Navigation - Cameras; user movement within the 3D scene; collision, proximity and visibility detection

User-defined objects - Ability to extend built-in browser functionality by creating user-defined data types

Scripting - Ability to dynamically change the scene via programming and scripting languages

Networking - Ability to compose a single X3D scene out of assets located on a network; hyperlinking of objects to other scenes or assets located on the World Wide Web

Physical simulation - Humanoid animation; geospatial datasets; integration with Distributed Interactive Simulation (DIS) protocols

The X3D standard is usually implemented and used through the XML file format, which is an ideal standard for 3D SG rendering and authoring, routes, scripting, and event passing.

X3D was chosen in this project primarily due to it's large range of features and networking capabilities. Since the standard builds on previous technologies and with the above aspects in mind, it seems appropriate to assume that the Web3D consortium has created it both comprehensive and effective.

Using X3D in this architecture has several beneficial advantages. The application settles on an a priori established standard that is easily extended, build upon, and communicated. Further, X3D covers all application needs, including scene-graph handling, hierarchical node-structures that can be parsed, basic 3D geometries and text, and networking (scene hyperlinking).

As described, X3D allows a large range of 3D description and SG manipulation features that, though immediately only would be used sparsely in the initial architecture, might prove valuable when extended.

2.4.Network Technologies

2.4.1.Peer-to-Peer Network

2.4.1.1.Advantages and Disadvantages

Advantages of a P2P architecture is that the rendering operation for the environment is mostly local, which decreases network traffic. Since all necessary information about the virtual world has to be stored locally, it is relatively simple to create an initial view. The world is then build up dynamically as more peers are visible and connected.

Disadvantages of P2P are mainly synchronization, resource replication, and bandwidth management difficulties, but they are common on Client/Server networks as well.

Events have to be propagated across the network, and as more peers are involved the effort becomes more complex. These scaling problems can be avoided using optimized broadcasting techniques, such as hierarchical multicasting [Gre97].

Another issue is the utilization of peer resources (in terms of processing power). In “homogeneous networks, there is no scope for utilizing spare capacity... Increasing the complexity of the environment impacts equally on all users’ resources, and introducing a single new processor into the system would be of no benefit from this point of view” [LSVR99, pp. 93].

Maintaining 'unoccupied' environments “is peculiar to the peer-to-peer configuration... The ‘reality’ of an environment is somehow associated with and dependent upon... participants. [If] there is no world engine [to] process it... reality ceases [to] exist” [LSVR99, pp.95]. This issue, though, could arguably be a feature of the architecture rather than a problem. A VE would emerge only when users utilize it.

Client/Server architectures provides a single location for processing the environment for multiple users. This eases the overall maintenance of the VE. On the other hand, distribution of information is controlled centrally, and theoretically the autonomy might be restricted by the server.

A DVE system often is required to handle several tasks, for example client request processing, synchronization, object transmission and interaction handling. On a large scale, as multiple participants are involved in the DVE, this concurrent workload becomes significant.

"A single server based DVE system may not have enough processing power to maintain the system interactivity. To address this problem, a number of multi-server based DVE systems have been developed" [DVE-PERF01]. In a study, described in an earlier section (2.2.10. DVE Performance), the performance of multi-server DVE systems is clearly in advantage.

2.4.1.2.Requirements

There are several requirements attached to a DVE network, especially if it is P2P based. Each peer has to be both discoverable and uniquely identifiable by other peers, as there is no central place that handles these tasks.

In the 'total distribution' scenario, connections are established ad-hoc. The underlying network protocol must be able to find, connect, and send messages between peers. Further, it must be possible to publish and propagate peer data, including individual information, 3D representations, and services.

Persistence of the VE might be important for certain representations or services. In the ad-hoc network, this simulations concurrent state can seem rather difficult to maintain, but it can be argued that persistence is accomplished locally by each peer depending on the provider's preferences. This would in effect be similar to hosted websites running on servers in a stable environment with almost no 'down-time'. Persistent DVE's are defined as DVE's that are 'always on' [EIM-SDVE99].

Finally, the P2P network needs a security system that supports authentication, integrity, and confidentiality.

As this document will show shortly, the JXTA platform is a very flexible specification that cater most of these requirements.

2.4.2.JXTA

2.4.2.1.What is JXTA?

"JXTA is Sun-led project to develop an open source peer-to-peer (P2P) framework in the context of distributed computing and services" [JXTA-HELLO].

Juxtapose (or short JXTA) is a set of protocols for inter-peer communication, and offers various functionality, such as group management, content sharing and service provision.

Each protocol of the framework is modular, and provides different tasks:

- Endpoint Routing Protocol (ERP)
- Rendezvous Protocol (RVP)
- Peer Resolver Protocol (PRP)
- Peer Discovery Protocol (PDP)
- Peer Information Protocol (PIP)
- Pipe Binding Protocol (PBP)

The basic operation of JXTA is quite simple to understand, though the actual framework certainly has a higher learning-curve. The following sections cover all aspects necessary to understand the P2P platform.

2.4.2.2.JXTA Framework Overview

In JXTA, each peer is assigned a unique identifier (peer ID), and belongs to one or more peer groups. By publishing 'advertisements' that are encoded in XML format, the communication between peers is established and messages can be exchanged. Peers communicate with each other using 'pipes'.

An example JXTA Peer ID:

```
urn:jxta:uuid-59616261646162614A78746150325033F3BC76FF13C2414CBC0AB663666DA53903
```

A peer has three abstract layers. The Applications Layer contains the logic and GUI of individual applications. The Services Layer provides common library-like functionality, such as a CMS service. The Core Layer is responsible for managing the JXTA protocol. It creates an addressing space separate from IP addresses by providing each peer its own unique peer ID.

The Core Layer also provides boot-strapping mechanism for peer grouping, can locate peers in a secure/authenticated manner, if desired, and can open a pipe (simple one-way message queue) to another peer or group of peer.

This structure is also shown in Illustration 2.21. JXTA Layered Architecture below:

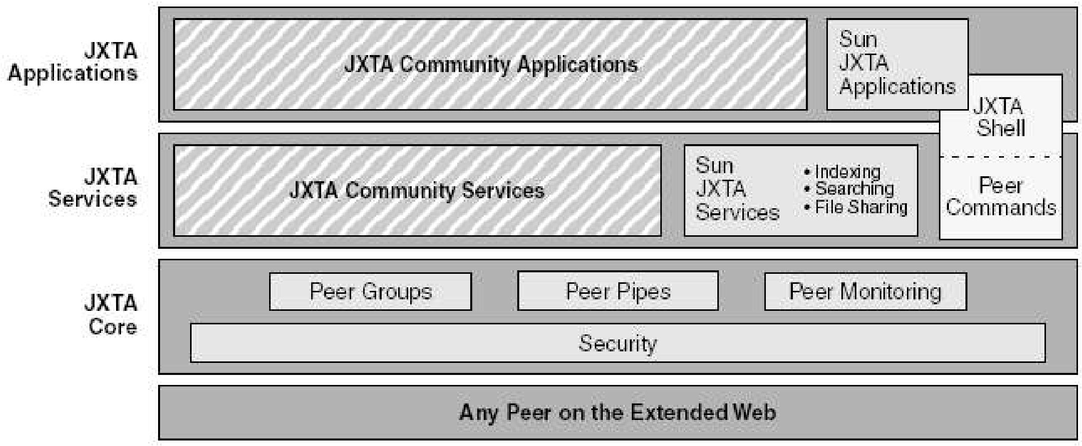


Illustration 2.21. JXTA Layered Architecture

The JXTA incorporates these protocols [specified in JXTA-P02, pp.8, and JXTA-P03, pp.19]:

- The Endpoint Routing Protocol (ERP)
  Protocol by which a peer can discover a route (sequence of hops) used to send a message to another peer. If there is no direct route between two peers, intermediary relay peer(s) are needed to route the message. ERP is used to determine the routing information. If the network topology has changed, ERP can find routes known by other peers to construct a new route.

- The Rendezvous Protocol (RVP)
  Protocol by which peers can subscribe or be a subscriber to a propagation service. Within a peergroup, peers can be rendezvous peers, or peers that are listening to rendezvous peers. RVP allows messages to be sent to all of the listeners of the service. RVP is used by the Peer Resolver Protocol in order to propagate messages.

- The Peer Resolver Protocol (PRP)
  Enables peers to send a generic query to one or more peers and receive a response (or multiple responses) to the query. Unlike PDP and PIP, which are used to query specific predefined information, this protocol allows peer services to define and exchange any arbitrary information they need.

- The Peer Discovery Protocol (PDP)
  Protocol by which a peer publishes its own advertisements, and discovers advertisements from other peers (peer, peergroup, module, pipe and content). PDP uses the Peer Resolver Protocol for sending and propagating discovery advertisement requests.

- The Peer Information Protocol (PIP)
  Protocol by a which a peer may obtain status information about other peers, such as state, uptime, traffic load, capabilities, etc. PIP uses the PRP for sending and propagating peer information requests.

- The Pipe Binding Protocol (PBP)
  Protocol by which a peer can establish a virtual communication channel or pipe between one or more peers. The PBP is used by a peer to bind the two or more pipe ends of the connection (input and output pipe) to a physical peer endpoint. PBP uses the PRP for sending and propagating pipe binding requests.

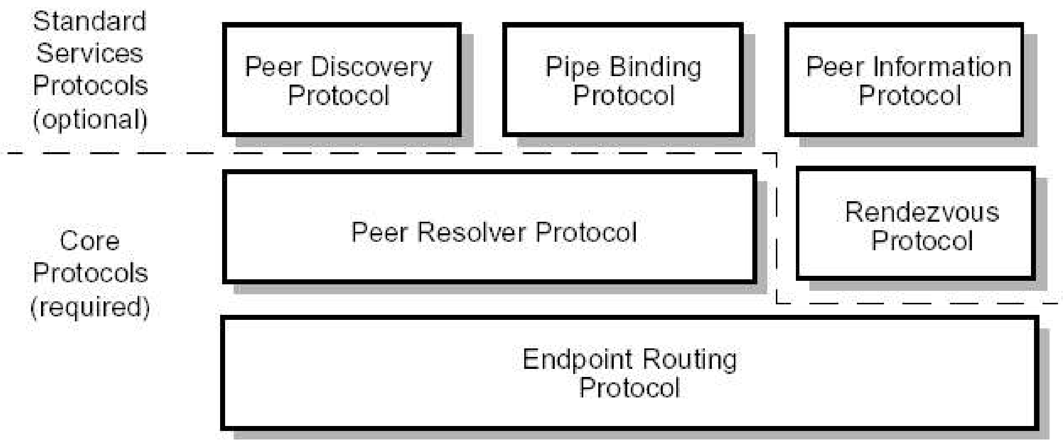

Standard Services Protocols (optional)
Peer Discovery Protocol
Pipe Binding Protocol
Peer Information Protocol
Core Protocols (required)
Peer Resolver Protocol
Rendezvous Protocol
Endpoint Routing Protocol

Illustration 2.22. JXTA Protocols

The JXTA specification is described in the next sections, and further material can be found at [JXTA-START] and [JXTA-P03].

The JXTA framework offers several network communication protocols, and defines specific abstract notions that delimit contextual and functional aspects. The notions 'peergroups', 'advertisements', 'services', 'rendezvous' and 'resources' are a few examples.

2.4.2.3.JXTA Peers and Peergroups

A peer is defined in the JXTA specification as “a device that implements one or more JXTA protocols” [JXTA-P03]. The central idea is that the protocols can be implemented on a device, which is not necessarily a machine. Further, a single machine can host multiple JXTA programs, where each program corresponds to a 'virtual' device.

JXTA peers use their protocols to communicate and exchange data with each other over a network, and the overall architecture of this P2P platform allows a very flexible approach to how the protocols can be used. For example, peers could be tightly grouped together to perform certain tasks, or the network could be build with peers acting as super-nodes. A single peer could even be a distributed application running across multiple machines.

JXTA Peergroups is a core property of the platform, and provides a mechanism for peers to self-organize into groups. These groups have no particular functionality besides being able to provide a closed boundary when certain environments are requested, e.g. security or collaboration. The platform supports the creation of groups and the definition of group membership, but it is up to cooperating peers to define, join, and leave groups.

2.4.2.4.Peer Discovery

In a distributed network, especially a decentralized peer-to-peer based network, finding each other, as well as finding each other's offers and demands is a non-trivial activity. Luckily, the JXTA framework implements this feature.

The process of finding other peers is called discovery, and can be put forth on any JXTA resource. Discovery occurs within the context of a peergroup, that is, a peer can discover resources within a specific peergroup. Resources are made available through a ' pipe advertisement' mechanism.

Peers can discover these resources:

- Other peers
- Peergroups
- Pipes
- Advertisements
- Other resources

Resources can be discovered using two methods:

- Peers discover resources dynamically
- Peers use statically configured peers to discover resources

Dynamic discovery utilizes the JXTA Peer Discovery Protocol (PDP). “The PDP defines the lowest-level technique that is available for peers to discover resources” [JXTA-START].

Hence, PDP over IP would send a multicast message on the local network and use so-called 'rendezvous' peers to discover peers beyond the scope of the local network.

A normal discovery on PDP would look like this:

1. Resources are advertised
2. A peer sends a request message
3. Resources that hear the request respond directly to the peer
4. Resources that are known by the peer's rendezvous peers will be sent directly to the peer
5. The peer will discover all available JXTA resources on the local network

The peer discovery protocol is one of several JXTA protocols.

2.4.2.5.Pipes and Peer Endpoints

The JXTA pipe service is similar to the Unix system pipe, which is used to connect the output from one command to the input of another command. The JXTA notion of a pipe are the endpoints available to a peer. These peer endpoints are a “logical abstraction of an address on a network transport that is capable of sending and receiving network messages” [JXTA-START].

When two peers communicate with each other, one peer's input pipe (receiving endpoint) would be linked to the other peer's output pipe (sending endpoint), and establish a unidirectional, virtual connection. Pipe connections are created independently of the pipe endpoints peer location.

Due to the 'virtual' connection of pipes, it is possible to fully abstract the connection layer of the network (see Illustration 2.23. JXTA Peer Pipe Abstraction), and intermediary routing peers might be used in the communication (i.e. knowledge that end-nodes may disregard).

In summary, pipes virtualize peer connections, 'homogenize' and provide an abstraction of the connection layer, can send messages in different ways, and may provide different quality of service.

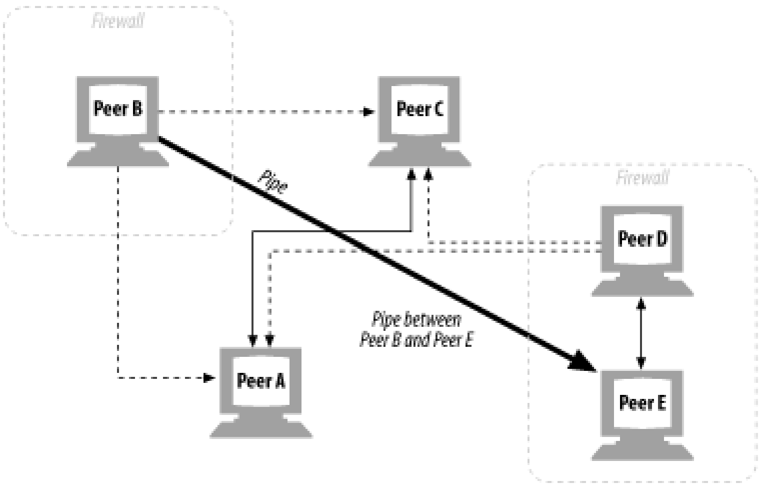


Illustration 2.23. JXTA Peer Pipe Abstraction

And pipe communication can be...

- Unidirectional, asynchronous (uncertain message delivery)
- RPC – Synchronous request/response (messages get acknowledged)
- Publish/subscribe (pipe endpoints subscribe to messages from publishers)
- Bulk data transfer (reliable data transfer of binary data)
- Streaming (efficient data transfer over a flow-controlled channel)

There exist two modes of communication, see Illustration 2.24. JXTA Peer Pipes (modes) [JXTA-START]. A point-to-point pipe connects exactly two pipe endpoints. A propagate pipe connects one output pipe (the propagation source) to multiple input pipes.

Point-to-point

- Input pipe receives messages sent from an output pipe
- No acknowledgment
- Information in the message payload is required to determine the sequence of messages
- Payload may also contain a pipe advertisement that can be used to open a pipe to reply to the sender

Propagate pipe

- Messages flow into the input pipes from the output pipe
- Messages sent over a propagate pipe are sent to all listening input pipes
- On a TCP/IP network, IP multicasting is used as an implementation for propagate pipes when the propagate scope maps to an underlying physical subnet in a one-to-one fashion
- Propagate pipes can also be implemented using point-to-point communication on transports that do not support multicasting

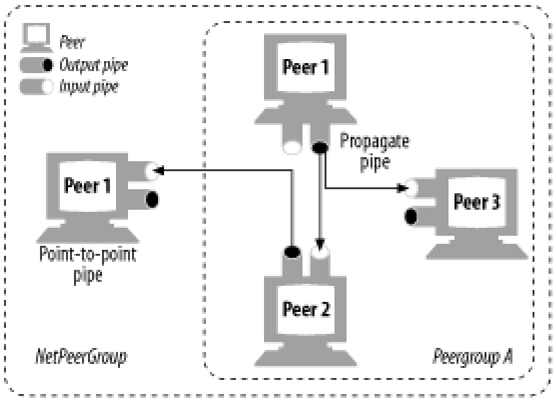


Illustration 2.24. JXTA Peer Pipes (modes)

An example of pipe communication: "...the input pipe endpoint can be located behind a firewall or NAT while the output endpoint can be located on a peer on the Internet. The endpoints may even be on physically different networks: the input pipe endpoint could be on a TCP network while the output pipe endpoint is on a token ring network. As long as there are available JXTA relay peers between the two endpoints, a logical pipe between them may be defined." [JXTA-START].

The following sections shortly describes fault-tolerant constructs, pipe endpoint binding, Pipes and Peergroups, and Messages from the JXTA specification.

2.4.2.6.Fault-Tolerant Constructs

Pipes allow applications to bind to the more appropriate instance of a service, depending on preference. Reliability in JXTA is possible through the proliferation of interchangeable services on peers, which allow adaption to unreliable network environments. The most available or efficient services can be exchanged on the fly and regardless of location.

> Pipes are an essential tool to build such services and applications. JXTA pipes introduce a fundamental network programming shift in which developers should not write applications that connect to a specific peer to access a unique service, but should write applications that discover the closest available service regardless of which peer is running the service. [JXTA-P03]

2.4.2.7.Pipe Endpoint Binding

Pipe endpoints are dynamically bounded to a peer endpoint at runtime via the Pipe Binding Protocol (PBP). A binding is created by searching for and connecting two or more endpoints. Messages are then transferred from the local output pipe to the destination input pipe (the listener). The location of listeners is resolved through the PBP.

Pipe advertisements uniquely identify pipes, contain certain meta-data, and represent a JXTA resource that may be discovered similar to peers and groups. Each advertisement is associated with a unique pipe.

> Applications use a pipe service to create pipe endpoints (both input and output) associated with a particular pipe advertisement. The pipe service uses pipe advertisements to identify the pipe and resolve the input pipe and output pipe endpoints. [JXTA-P03]

2.4.2.8.Pipes and Peergroups

Peergroups are essential boundaries for pipe connectivity. Each group provides their own pipe service. Endpoints have to be located within peergroups in order to be mutually resolved, and hence peers have to have joined the same group to establish a connection.

Since all peers are natural part of the standard NetPeerGroup, though, any pipe connection is possible. The selected group determines which pipe service is used to resolve the pipe, and this context might define certain security properties. Authentication of peers within peergroups can thereby be enabled.

> A peer can maintain different pipe connections to the same peer, holding each of them in a different peergroup context for security reasons. Messages can be sent to the peer with different security levels depending on the pipe used. [JXTA-P03]

2.4.2.9.Messages

All information exchanged between peers are formatted into messages, that represent an envelope for data of any type. A message may contain a variable amount of sections that are uniquely named (namespaces). Using MIME types, Base64 encoding scheme, or CDATA, the body can hold files, binary data, XML documents, and so on.

A peer application can communicate with other peers by constructing and sending messages over their respective input and output pipe endpoints.

> JXTA messages use a binary format to enable the efficient transfer of binary and XML data. A JXTA binary message format is composed of a sequence of elements. Each element has a name and a MIME type and can contain either binary or XML data. In order to send data over a pipe (or to any JXTA peer), the data must be encapsulated in a message. [JXTA-P03]

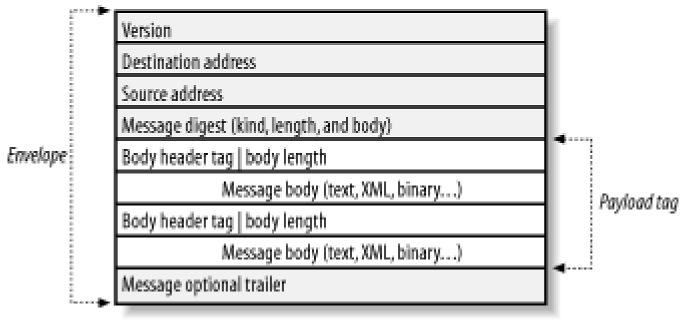

Version
Destination address
Source address
Message digest (kind, length, and body)
Body header tag | body length
Message body (text, XML, binary...)
Body header tag | body length
Message body (text, XML, binary...)
Message optional trailer
Envelope
Payload tag

Illustration 2.25. JXTA Message

Illustration 2.25. JXTA Message depicts the structure of a message. The envelope contains generic field such as source and destination addresses, as well as the payload body of variable length.

2.4.2.10.Advertisements

Advertisements are essential parts of the framework. This notion works in concert with the discovery process, and is used to advertise and locate peers, groups, services, or content. Peers publish advertisements, and consequently respond to requests for it.

JXTA advertisements are represented by 6 specific XML documents, depending on the advertisement context:

Peer advertisement
Peergroup advertisement
Pipe advertisement
Service advertisement
Content advertisement
Endpoint advertisement

Example of a Peergroup Advertisement:

```
<?xml version="1.0"?>
<!DOCTYPE jxta:PGA>
<jxta:PGA xmlns:jxta="http://jxta.org">
<GID> urn:jxta:jxta-NetGroup</GID>
<MSID>urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE000000010206</MSID>
<Name>NetPeerGroup</Name>
<Desc>NetPeerGroup by default</Desc>
</jxta:PGA>
```

Advertisements could be seen as a “platform-independent representation of objects that can be exchanged between different implementations”.

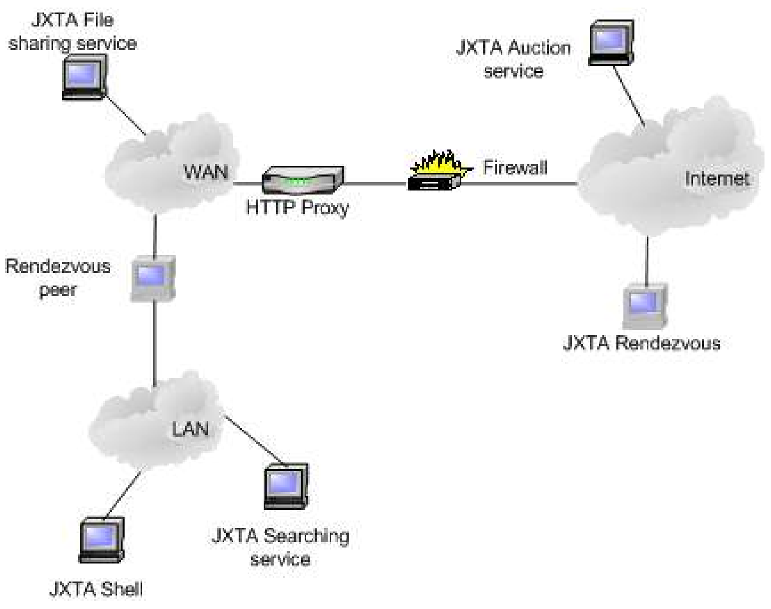


Illustration 2.26. JXTA Example Network

2.5.GUI Design

2.5.1.HCI

A comprehensive resource by Gary Perlman outlines the central issues of Human-Computer Interface (HCI) design [SIGCHI]. Illustration 2.27. Human-Computer Interaction by SIGCHI summarizes the topics quite well.

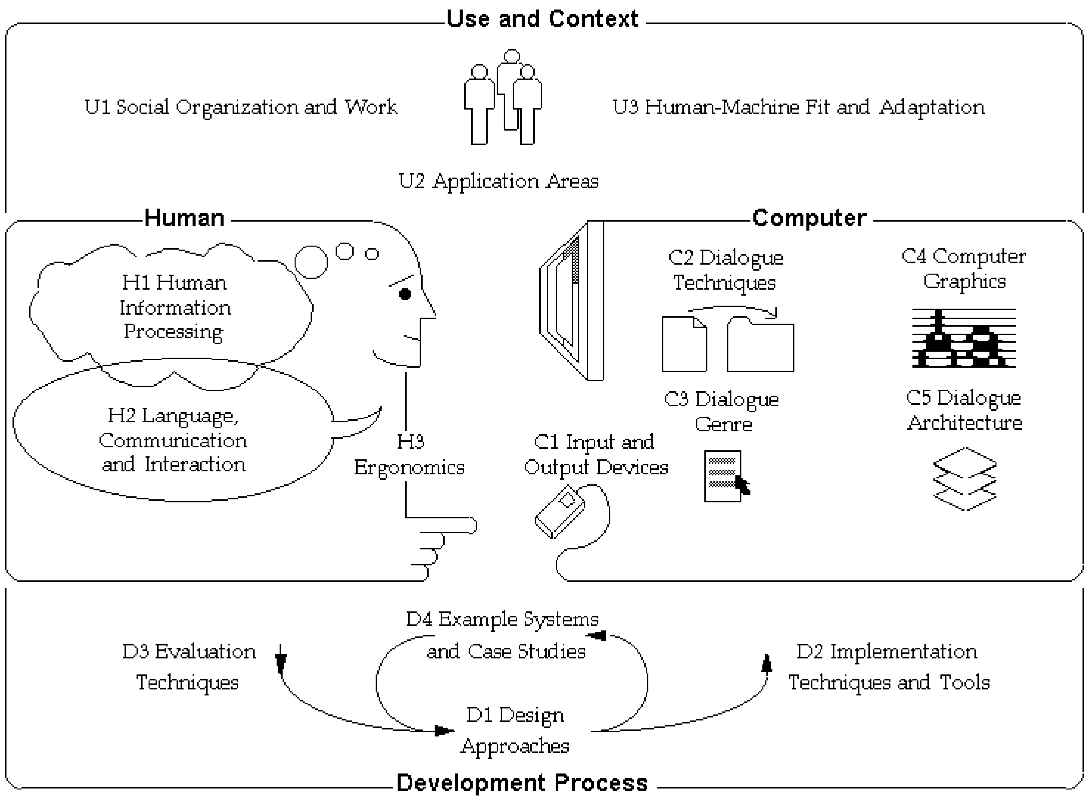

Use and Context
U1 Social Organization and Work
U3 Human-Machine Fit and Adaptation
U2 Application Areas
Human
H1 Human Information Processing
H2 Language, Communication and Interaction
H3 Ergonomics
Computer
C2 Dialogue Techniques
C4 Computer Graphics
C3 Dialogue Genre
C5 Dialogue Architecture
C1 Input and Output Devices
D4 Example Systems and Case Studies
D3 Evaluation Techniques
D2 Implementation Techniques and Tools
D1 Design Approaches
Development Process

Illustration 2.27. Human-Computer Interaction by SIGCHI

Especially the 'Computer System and Interface Architecture' section (C) defines the corner-stones of GUI techniques: Dialogue Techniques (C2), Dialogue Genre (C3), Computer Graphics (C4), and Dialogue Architecture (C5).

The central issues in this section surround dialog input, output and interaction (menus, mouse-picking, gestures, navigation and orientation, etc.), the conceptual uses of an application (illuminated through interaction or content metaphors, point of view, workspace, etc.), graphics primitives, attributes, and representations, as well as layered models, multi-user interfaces, and interoperability [SIGCHI, s.2.3.4].

2.5.2.Application User interface

Most issues are highly relevant to any application GUI design. Dialog interaction, for example, differs substantially in an online web-shop and an office spreadsheet. As user interface design has evolved and become essential in most software, collections of reusable patterns and guidelines have been produced.

The Apple paper on usability is quite a good read [APPLE]. It summarizes these important human interface design principles:

- Metaphors
- See-and-Point
- Direct Manipulation
- User Control
- Feedback and Communication
- Consistency
- WYSIWYG (What You See Is What You Get)
- Forgiveness
- Perceived Stability
- Aesthetic Integrity
- Modelessness

For the TerraPeer application, these guidelines and principles are of importance when VE viewer, P2P network, action controls, space world interaction, information feedback and status all have to be integrated into the GUI layout.

To promote usability, not only does the overall interface design need to adhere to the principles, but each of the sub-systems are required to do as well. It will agitate users, if the initial window screen has a friendly layout, but functionality is complicated and unintuitive. Usability questions will be examined further in the implementation chapter (4.3).

2.5.3.DVE User interface

User interface designers need to convey information and functionality of an application. Metaphors are an integral part of human understanding, and represent a vital approach in the design to provide a clear mental model to the user of what to expect.

Basic GUI principles are metaphors, the ability to directly manipulate objects, “see-and-point”, feedback and dialog, and “modelessness” (not to restrict available operations).

In a section called "The Participant's Model", Bricken [VR-Interface] describes four questions that the user of a DVE might ask:

- "Where am I?" Is the understanding of the VE. Possibility to create own new worlds, or travel through existing ones.

- "Who am I?" - The 'look' of virtual self, such as a floating point of view, or an embodiment. Ability to switch point of view to any object, a process, another person's viewpoint, or simultaneous multiple perspectives.

- "What can I do?" - Behaviors are relocation, manipulation, construction, and navigation. Relocation is the behavior to change position in the VE: walking and turning, reaching, flying, using a gesture, move smoothly with variable speed in any direction, instantly transferring to a new location, etc. "In cyberspace, the concept of "distance" is optional; relocation is independent of time and space" [VR-Interface]. Manipulation involves movement of virtual objects. Construction behavior is the ability to build and alter VE worlds interactively, and the tools needed. Navigation behavior surrounds methods for searching and locating places and objects. It involves way finding and guiding, and UI tools to support these (maps, artificial horizon, etc.). Basic definitions and rules, such as distance and time need to be designed.

- "Who is with me?" - Sharing the DVE requires user representations (appearance), social, cultural, and other common conventions (mutual control, world 'laws' and weighted restrictions).

In TerraPeer these questions are addressed through the design of navigational rules and aids. Typical navigational problems in these virtual spaces are both controlling the view and moving around to reach a particular target. Tunneling through space, when the metaphor is a flat (metric) space, for instance, might be confusing. But the UI has to be as convenient for the user as possible, or the application might quickly become an annoyance.

Walking, a ground-level, directions, and a 'jumping' (tunneling) mechanisms are placed to create a usable VE. Searching and linking inside the space are also important features that have to be included.

VE's imply a directionality that can enhance the user's mental representation of a structure. In many OS desktops it is difficult to get a quick overview of all folders available except on the highest hierarchical level. A lot of navigation is required only to locate folders that contain for instance project names. The virtual zone is a user interface metaphor based on a restricted area. Its aim is to help navigating large collections of information.

The zone metaphor could be extended with a 'city' metaphor as a basis “because people seldom get really lost in a real city even when they are in this city for the first time. Cities provide people with lots of informational and navigational infrastructure. Besides in all cities there are always other people around to help in navigation. It seems reasonable to make use of the city navigation skills most people have to navigate a complex computer generated information city” [CITY94 pp.76].

3.Analysis

3.1.Extracting the theory

Examining the original thoughts, as well as the research and technologies discussed in the previous chapter, an overall strategy of how to implement the DVE system emerges. Further, there are now several choices that need to be addressed.

The basic ingredient of this platform would be the inherent rules, i.e. what properties and restrictions the environment is build upon. This includes features to create, publish, share, and use information and functionality. Then, essential questions about scalability, performance, independence, openness, and usability need to be addressed.

Finally, the technological choices are determined, i.e. which toolkits, frameworks and languages suit the requirements.

3.1.1.Projects and Technologies

The table below (Table 3.1. Networking and DVE) shows a summary of the projects and technologies taken from the theoretical scope of this document.

Network	Projects	Technologies	Single-user	Multi-user
Local Computer	Large amount of 3D Design Applications and Games, Performer	• OpenGL	• 3D Visualization • Virtual Environments (VE) • 3D Games	
Client/Server	Web, Crystal Space, Half-life, Quake, Unreal, CryEngine, Porta Susa, Peloton, OpenWorlds, MUDs, Kali, TEN and Mplayer	• HTTP • VRML, X3D • Java3D • Game Engines	• Virtual Environments (VE) • VR Spaces	• IM • CSCW • Distributed Virtual Environments (DVE) • VR Spaces • 3D Online Games • Avatars
Multi-Server, Distributed or Hybrid	Gnutella, Napster, Kazaa, EverQuest, Aglets, HORB, Electric Sheep, AVOCADO, DIVE, MASSIVE, NPSNET-IV, SPLINE PARADISE, HIVE, SPLINE, MaDViWorld, V-Worlds, Urbi Et Orbi, NOMAD, Voyager	• CORBA • J2EE • JXTA • X3D, VRTP, VSTP • DWTP		

Table 3.1. Networking and DVE

Splitting the table into 'Local' , 'Client/Server' and 'Distributed' networks makes it possible to relate different projects and technologies to each other, depending on where they operate. The technological borders are blurred, since protocols often can be reused. So are single- and multi-user environments, since some applications support both.

The table allows a focus on those projects and technologies, that are relevant. The multi-user, decentralized network requirements lead to the projects Gnutella, HORB, DIVE, AVOCADO, and MaDViWorld, and to the technologies CORBA, J2EE, JXTA, VRTP, and DWTP. Considering the graphical aspect, projects such as CryEngine and OpenWorlds are valuable. The essential technologies to mark are OpenGL, VRML/X3D, Java3D, and Game Engines.

3.1.2. Design Experience

Looking deeper into each of the projects that have been examined, the distinctive approaches and implementations provide valuable information on the aspects of networking architectures, scalability, performance, zoning, visualization, and communication in DVE systems. Table 3.2. DVE Experiences, extracts the experiences that previous research has demonstrated.

Subject	Applications, Projects, Frameworks, and Protocols	Design & Experience
Large-scale DVE	SimNet SPLINE MASSIVE NPSNET HIVE HORB DIVE PARADISE MaDViWorld AVOCADO	Massive distribution of a virtual environment can be implemented using multicasting. Solutions involve multiple users, and can provide 3D graphics and sound, spoken interaction, computer simulations, run-time modifiability, or multicast groups with mobile objects. One project created an unified kernel with a time parameterized environment model. The approach is often to reduce latency by anticipation and advance communication of events. Large-scale worlds with locales, composition of the environment, and services make up this type of DVE. SPLINE splits the world into locales that are processed separately. HIVE implements a model of anticipation and advance communication of events. MASSIVE provides world-composition, locales, and message handling. Each object has an aura which allows state exchange. DIVE partitions the virtual world, and implements multicasting. Scene objects are replicated.
C/S DVE Networks	RING Quake 3 Half-life Unreal CryEngine OpenWorlds	Clients connecting to game servers is a typical setting. Several servers exist, but don't interact, i.e. each game server maintains a unique game state, not shared among the servers. Quake, Half-life, Unreal, and CryEngine are typical 3D games with multiple clients sharing the same environment (map) that is downloaded to each (from one central server). OpenWorlds is a toolkit using X3D to build worlds.
P2P DVE Networks	MASSIVE JXTA Freenet Gnutella	Networks are able to reduced the load of servers, provide content independent of their location, and are not dependent on a single entity. In MASSIVE, each processor runs an agents, communication works as sockets, and environments are replicated. Freenet and Gnutella are information publishing and file-sharing applications that run fully decentralized. JXTA is a pure P2P protocol that provides all essential functionality.
Hybrid DVE Networks	DWTP SETI Electric Sheep	Further information on DWTP could not be made available.SETI and Electric Sheep are P2P applications that have central servers to gather data.
Active DVE Networks	ARA SRM	In the Active Router Architecture, the application selects certain transcoding processes, while the network is able to apply them. Scalable Reliable Multicast implements small amounts of session specific computation at key routing nodes. This reduces congestion. It is fully decentralized and handles arbitrarily large groups of participants. Reliability is supported by multicasting repair requests.
Event Driven DVE	ED	Architecture using event-based notification system implies a DVE being constructed of zones, i.e the DVE represents a network of zones. A zone is here a representation of a collection of information of interest to participants in the DVE. There exist event flows.
Game Engines	EverQuest Quake 3 Half-life Unreal CryEngine DEE	Most game engines use modularized build-up that focuses on mechanisms, such as rendering, user control and interface, and networking. Engines act each as separated client–server systems, and are not real multi-server systems.
Performance	Peloton CyberWalk 'Platform Model' by [DVE-CS03]	One approach is to dynamically adjust the DVE to resources available. Different VE regions match their performance characteristics of underlying computing and communication systems by altering the visual representation depending on the computing and communication resources available.

Subject	Applications, Projects, Frameworks, and Protocols	Design & Experience
Zoning / Locales / Aura	DVE-VRML SPLINE EverQuest	Zones can be defined spatially. The virtual world can be split into locales, which can be processed separately. In the spatial model for interaction among clients, objects can have an aura that determines the space within which interactions are possible. Two participants exchange state updates when their auras collide.
Communication	X3D VRML CORBA JXTA NPSNET-IV NOMAD SRM	Communication can occure via a combination of 3D graphics and real-time packet audio, or multimedia. Message Filtering, Direct Conversation, and Interest Management can be employed. Session Management structures interactions on shared objects, without requiring explicit session preparation. Multicast Framework provides a scalable platform. There exist application level reliability in multicast environments (stable delivery). Can use CORBA middleware as network platform. Main problem is to optimize data load. Methods are required for realizing interaction between a virtual human and its surrounding objects (Sharing attractions).
Scene Graphs	SOFT	Distributed Scene Graphs is a typical setting. An approach for the distribution of graphical data is to apply SG's as a bus.
Building and Visualization	EM SVR	Navigation in VE can be enhanced using metaphors (e.g. a city). Subjects to be considered are Environment Managers, Virtual Bodies, Virtual Object Interaction, and Object Description Languages. Applications can support Urban Planning, Construction and Manufacturing, with communicative tools to improve collaboration and the distributed environment to process information across the networks (CSCW).
Avatars	OAA	Vital are methods for realizing interaction (and awareness) between virtual humans. There is a method to transfer avatar gestures in a direct matter (involving HORB and QoS). The Open Avatar Architecture is composed of heterogeneous multi-servers, and multiview clients. The protocols VSTP, which runs on top of TCP/IP, guarantees the connection, and together with HTTP enables navigation in the VE and the web. An Action Database with techniques for storing acquired motions can be deployed.

Table 3.2. DVE Experiences

Experiences listed in this table supply fundamental knowledge about both design approaches and current state of research. Large-scale DVE's aim to be scalable, provide environment models and composition of worlds, and try to reduce latency.

Projects use multicasting, time parameterizing, locales for zoning and aura for state exchanges, message handling, and SG replication. C/S-, P2P-, Hybrid-, Active-, or Event Driven DVE Networks are all different approaches on how to optimize traffic, and create a feasible multi-user environment.

Game engines lead the way in graphical and network performance on a smaller scale.

Zoning, communication, visualization, scene graphs, and avatars are all specific areas in DVE systems that require a high degree of detail when designed.

3.2.Outline of a solution

3.2.1.Software attributes

A DVE software should aim to support the following standard system attributes:

- Scalability - ability to support many users, nodes, and large-volume content
- Dependability - general availability and reliability of the system

- Interoperability - possibility for heterogeneous DVE implementations to inter-work
- Extensibility - possibility to add or modify an existing system
- Openness - possibility to interface a DVE to other applications
- Content independence - ability to support all forms of data besides graphics and geometry
- Communication - ability for users to communicate with several means, including audio
- Usability - DVE should be easy to learn and use
- Performance - general response time, throughput, and resource utilization

Applying these software principles require considerable efforts, a viable architecture, as well as coherent choices of programming language, tools, libraries, and interfaces, i.e. to achieve these design goals, it is usually necessary to follow extensive development cycles, to identify and take advantage of relevant design patterns, and to run extensive quality assurance throughout the entire process.

The TerraPeer application aims to be scalable, extensible, open, content independent, communication supportive, and usable. Especially the usability factor is a primary focus of this thesis. Several scetches are provided (see Appendix VIII. Drawings) to illustrate how the application design has evolved.

3.2.2.Selection of solutions

The problems associated with DVE are addressed in the table below. The solutions were extracted from the previously examined systems that often focused on certain problems in particular.

Problem	***Description***	***Solution***
Large-scale	Large amount of clients simultaniously connected, and interacting in a shared environment Client/server architecture is not feasible Multiple servers, where each maintains a unique state is not feasible	P2P (distributing the load), multicasting Zones (splitting the world into locales that are processed separately)
Performance	To reduce latency, avoid bottlenecks, and be scalable Pure C/S is not feasible Total replication of world on each peer is not feasible	Multi-server, Hybrid or P2P networks Anticipation and advance communication of events Dynamically adjust the DVE to resources available Zones (locales processed separately)
Independence	Being not dependent on a single central entity Pure C/S is not feasible	True P2P (absolute decentralized)
Shared 3D Environment	To view a common virtual space To update local state changes throughout the world Sharing attractions	Environments or scene objects are replicated Apply SG's as a bus Zones (representation of information of interest to participants)
Building, Information publishing and sharing	Personal composition of the environment To create a common virtual space Interaction between a virtual human and its surrounding objects Visualizing services	Services (virtual representations) Environment Managers Virtual Object Interaction Object Description Languages Enhancement through metaphors

Problem	***Description***	***Solution***
Modification	To access and modify distributed objects	Object has an aura which allows state exchange Event-based notification system (event flows) Session Management (interactions on shared objects) Zones (defined spatially; an aura as spatial model for interaction among clients; determines the space within which interactions are possible)
Communication	Interaction (and awareness) between virtual humans Message handling To provide stable delivery To optimize data load	Message Filtering and Interest Management Multicast Transfer avatar gestures in a direct matter Multiview clients Action Database (storing acquired motions)

Table 3.3. Problem-Solution Listing

Distilling the solutions, the following picture emerges:

- Absolute decentralized P2P can be used to distribute the load, and to support multicasting.
- To increase performance, dynamically adjust the DVE to resources available, anticipate and advance communication of events, and use message filtering.
- Zones can be used to split the world into locales that are processed separately, can represent information of interest to participants, can be an aura as a spatial model for interaction among clients, and can determine the space within which interactions are possible.
- To distribute graphics, environments or scene objects are replicated, and scene-graphs might be applied as a bus.
- An event-based notification system, interest management, and an action database can support distributed interaction.
- Virtual object interaction can be implemented with an aura that allows state exchange, and support through session management.
- Object description languages, graphical metaphors, and abstract services can be used to virtually represent entities.

The experience from previous projects has shown some valid solutions. Most of these assessments will be used in the DVE architecture.

3.3.Approach

3.3.1.Primary implementation goals

Considering the previous thoughts about user perspective, application features, and the virtual space, the design of the TerraPeer application considers certain criteria.

The VE design has to make sure the application is able to dynamically display connected peers, and their resources. It needs to create a user interface which simplifies navigation through the space, logical interaction methods that eases object manipulation, and easy accessible functionality menus.

On the backend side, all virtually represented (and hidden) data has to be stored locally. Any virtual peer homer-zone and attached resources (documents, etc.) is made available through a simple standard format that can be stored and transferred over the net.

As briefly described in the last chapter, there are several DVE-related technologies that have to be integrated in the TerraPeer architecture. The choices of technologies and frameworks that have been applied are shown in a layered structure below (Table 3.4. TerraPeer Component Layers). These layers cover the four basic components required in DVE systems.

Basic Component	***Application Layer***	***Technology***
Graphics engines and displays	3D Data	VRML/X3D
	3D Graphics	Java3D (J3DUI)
Processing systems	Application	J2SE (Java, Swing)
Communication and control devices	Protocol/Repository	XML
A data network	P2P Network	JXTA

Table 3.4. TerraPeer Component Layers

The JXTA platform is able to handle most P2P issues, including unique peer ID, discovery and propagation, ad-hoc connections, peer communication, peer-groups, authentication, integrity, and confidentiality.

The Java3D API can be thought of as a thorough platform for most required 3D functionality. Designing the 3D environment, a coordinate and a spatial system, functionality for object control and building blocks, as well as visualizing peers and objects in 3D is relatively straight forward. There are implications in the design of the VE specification, rather than the VE technicality.

By now, some of the originally raised challenges have been clarified, i.e. network support for mixed traffic, network properties, group participation, persistent state, etc. Each of the applied technologies, and remaining open questions will be addressed in the following sections.

3.3.2.Specification and Technologies

The TerraPeer application is specified in section 3.6. Prior to laying out the detailed structure, though, each of the fundamental technologies will be examined, namely the Java3D and X3D technologies, the J3DUI Framework, the JXTA network technology, the Object-Orientated Programming Method, GUI and Java Swing. Consecutive to that, application-related terminology will be established.

The specification will be an extension of the layered structure introduced in Table 3.4. TerraPeer Component Layers, and describe the responsibility and functionality of each component.

Following the basic composition, the aspects of virtualization capabilities and functionality of the 3D viewer, 3D multi-user environment interface structure, model-view-control modeling, virtual space design, users, content, and the abstraction of network peers will be analyzed.

Disorientation in a P2P environment is among the issues that need to be resolved through certain navigation and interaction functionality. How should the user be able to control the environment? Which movement restrictions should exist? What kind of feedback should the application provide to enhance usability?

Objects created by users have to be constructed adhering to rules, so that the virtual expansion of a large number of participants doesn't become an anarchistic exercise. Virtual building blocks should also enhance the virtualization of the underlaying data or tool.

In one particular project, an interesting idea was to create entities that would inherit attributes from each other, and specifically allot environment entities. These could contain other entities and environments, impose laws and forces upon its contents, and define attributes that are true of its contents (coherently applied laws). This idea provides a flexible structure, and inheritance of properties has been integrated in the TerraPeer service object design.

Other features to be examined closer include the creation and destruction of virtual entities, their methods or behavior, and naming schemes.

Approaching the TerraPeer architecture, a disposition of key attributes of virtual space and distributed platform is shaped:

- Users can start a 3D viewer that displays the VE space with basic properties, navigate through it, and create objects that might publicize information or represent services.
- Restricted areas, 'zones', for enclosed environments that contain virtual building blocks, and might adhere to certain rules are provided for each peer. These zones implicitly represent a peer, and each peer should only be allowed to create one.
- User interaction and collaboration should naturally augment both VE and GUI by avatars, gestures, and 3D connotations, but also by simple (effective) text-based chat or similar.
- Users are able to build virtual representations of services, such as URL links to web sites, that can be invoked or applied either internally or through external applications.
- The peer application is able to reside independently on any machine, and connect to other peers using a discovery mechanism and a messaging protocol.
- The P2P network embraces full decentralization, but also offer a hybrid version by clustering specialized server-peers.

Quite a few of these attributes can be integrated fairly elegantly, as the exposition of Java3D and JXTA will show next.

The next sections address three aspects of the software that should form the basis for the architecture: the user perspective, application features, and the virtual space. Emphasis is put on the user perspective rather than technical cases, such as performance or optimization issues.

3.3.3.User Perspective

From an individual's point of view, the application could be defined as a user interface to a 3D environment, which enables her to intuitively navigate and interact with virtual representations of services and other users. From a group's point of view, the application can organize people, and create a distributed platform for collaboration. By being pluggable or using standard interfaces, the application could be used with other service tools, such as publishing (web), file-sharing, peer trading, and roaming agents.

One of the initial thoughts surrounding this project is depicted in these short paragraphs:

> “A virtual 3D environment, which enables the user to represent his peer with a number of building blocks, and to move between other peers, that are connected through a network.
>
> The building blocks could be a website, a service, or information that are visualized through icons or graphical pieces, or in a final version as a X3D or similar technology based scene (i.e. a 'virtual home').
>
> The VE should be founded on certain basic rules: a coordinate system, a zoning mechanism that enables virtual areas for each peer, access-, ownership- and sharing- rules for objects, and spatial navigation. A user could thus move around in this world, use a service, or build his own.”

The application would represent a dynamic, navigational 3D space that allows free control and interaction of objects while providing a basic network platform that ensures scalability.

The space is created dynamically by it's peers, which does not allow any central entity. Its platform, though, implements certain rules and rights for users, i.e. 'virtual free speech' and 'virtual property' laws. Thus, the space promotes equality and freedom. The objects could be an environment, a resource, a function, bots, a service, or a link to other existing (or legacy) infrastructures, such as the web, X3D or VRML.

The user perspective to TerraPeer is an intuitive 3D VE UI that virtually represents services with interface to external tools, and avatars with support for group activities. Virtual services are created by building blocks. The DVE implements a basic environment setting, zoning, and access control. It also aims to be scalable, and absolute decentralized to promote 'political' equality and freedom.

3.3.4.Application features

By focusing on the balance of keeping the space free, while not anarchistic, the main purpose of an implementation is to create a dynamic distributed infrastructure where user's could roam, build private or public places, provide services, meet people, and search resources.

TerraPeer application features classified by network, space and user:

- **Network**
  1. Ad-hoc connection to a decentralized peer-network
  2. Secure network that support authentication, integrity, and confidentiality
  3. Visualization of peer-network in 3D (virtual sites)
  4. Scalable and extendable (standard plug-ins)
  5. Application support for HTTP services (web browser window)
  6. Search functionality (peers, services)
- **World Space**
  1. Basic 3D environment setting (coordinate system, spatial restrictions, filtering)
  2. Building blocks to create a virtual home-sites that provides resources and services (documents, file-sharing, web shops, data streams etc.)
  3. Storage of peer and service related information (meta-data)
  4. Object control, sharing and interaction mechanisms
  5. Access control for objects, zones, and users
  6. Navigation aids (landmarks, etc.)
- **Users**
  1. Navigation interface (user avatar roaming)
  2. Management of user-groups
  3. User collaboration and messaging mechanisms
  4. Trust management (peer policies, feedback system)
  5. Customizable (personalization)

Features listed above are prioritized.

3.3.5.Space

The space could be build up using a static 3D grid that spawns in a x/y-direction with a specific detail, and maybe several layers in the z-direction (they could provide different levels of virtual transportation). A center “origo” landmark could provide as a starting point and navigational 'well-known-location'. Navigational aids, such as static directions and coordinates could be provided by certain conventions (North/South directions, IPv6 addresses, etc.).

The current view would depend on the peers knowledge of the space, the connectivity with other peers, spatial and visibility limitations, and the current position of the user's avatar.

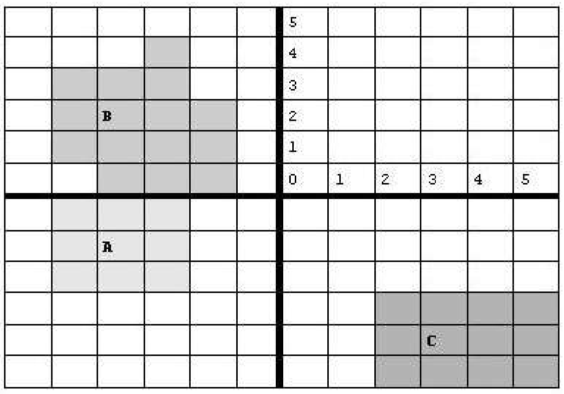


Illustration 3.1. Possible topological grid view of Zones

Visualization aids, such as area coloring, the unlimited origo axis, a horizon, and visual details (or lack thereof) could enhance usability. Space, navigational aids, avatars, bots, sites and objects could thus be displayed dynamically and by choice.

Global space rights and restrictions could enable open creation, secure individual freedom, privacy, and trust, while preventing anarchy. Object control would support this as well. Virtual property could be established, reserved, traded, and connected. Restrictions on each peer's virtual space (borders) could enable full personalization, including access control, interplay with connecting land, 'physical' laws, and functionality inside that space.

Using 'zones' as virtual boundaries between peers could create a dynamic world that is traversable for a user. A grid-based spatial system (see Illustration 3.1. Possible topological grid view of Zones) could be implemented to provide some kind of zone-reservation, and zone 'filtering' might increase performance.

4.Architecture

The approach of the TerraPeer architecture design follows a conceptual thread that begins with an overview of the application specification, and those technologies that have been chosen to play central roles. The thread continues with the actual platform system architecture, and each sub-system of which it is comprised of. The sections throughout this chapter adhere to this flow. Further details are laid out in the next paragraphs.

Most of the application terminology has already been introduced in previous sections, but it would be nice to get an overview. Appendices II and III list abbreviations and definitions of most words. During the remaining sections of this chapter, the structure and functionality of the VE will be explained by accompanying diagrams.

4.1.Architecture Overview

This section outlines the architecture specification of the TerraPeer application, and its layered structure. Integral layers are elaborated; the user interfaces in 2D (GUI Layer) and 3D (VUI Layer) respectively.

4.1.1.Application Features

It is time to answer the questions stated in Chapter 1. Most of the defining implementation guidelines can be accomplished.

Performance and resource utilization is a continuous issue for both 3D world representation and network latency. Optimization of TerraPeer should be fruitful, though, as JXTA is an extremely low-cost protocol. The overhead of the 3D rendering presents more difficulties, as Java3D performance cannot compare to technologies employed by game engines for instance (read the conclusion for more on this).

Scalability is immediately not an issue. The distributed nature of the network removes single-server bottlenecks. But problems might occur in a large-scale multi-user environment when scene updates could suffocate network traffic. This could be avoided by using spatial as well as logical filtering techniques.

Availability is granted similar to the web; the provider of a service can choose to run a persistent site. Also, groups of peers could agree on sharing on offering services. Content or services are available according to their interest.

The ability to support all forms of data - 'content independence' - is an essential feature of TerraPeer. Using both standard and user-defined virtual representations of content or services, the architecture should allow peers to present any kind of resource in the VE.

Communication between avatars can and should be implemented on more than one level. Different formats might be supported: Text-based messaging or chat, a forum or discussion groups, VoIP, Avatar gesturing, etc.

A DVE system user interface should aim to be easy to learn and use. Usability is the last, but not least, important guideline that makes an application successful. This includes designing the system to be cross-platform compatible.

4.1.2.Layered Structure

Application designs today need to adhere to many software principles (see for example previous sections 2.5.2. and 3.2.1.). The object-oriented approach converges with the usually layered network approach. Scalability, extensibility, openness, performance, usability all have to be build into the final system. This requires an abstraction of the application with the architecture on one end, and binary code on the other. Networked applications especially add complexity.

A P2P-based TerraPeer network looks like Illustration 4.1. P2P Communication.

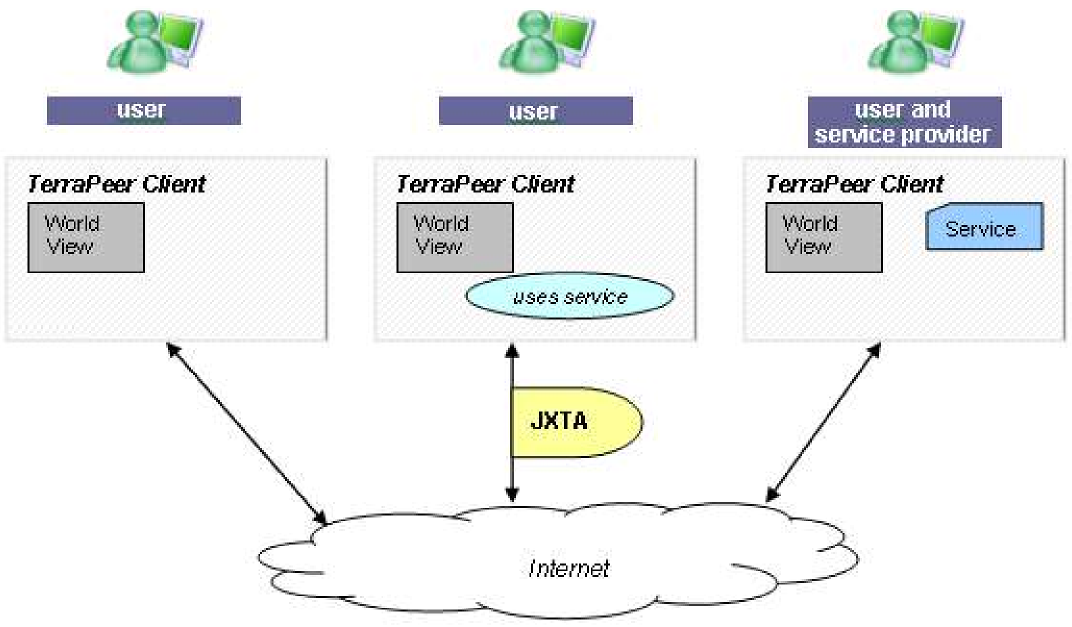


Illustration 4.1. P2P Communication

In this example, a single service provider and two other participants are connected to each other. The P2P protocol enables discovery (initial knowledge of at least one peer in the network is necessary, though), and communication – here shown with the JXTA label. Each user has a world view to visualize the network in a VE.

Shown in Illustration 4.2. P2P Zone Visibility are the optional zones each peer user could have created, and would be visible to the other peers.

The user of peer A 'sees' the virtual peers – Zones – of both B and C. Similar, peer B can see zone A and C. Peer C on the other hand can only see zone A, but not B. The visibility of each peer can depend on whether it has been discovered (is 'known'), or it is filtered out. Hence, peers might have different views on the current VE, especially if located far from each other on the network.

It might seem strange to have a 'dynamic' environment, and object persistence seems to be compromised. The design, though, allows a static environment simply by creating stable peers. Object persistence will always be granted through the availability of peers, their and agreements with others to maintain state, i.e. based on 'trust', influence, or popularity. The environment – in a sense – is 'free'. This design is reflected in the (semantic) web which is constantly changing but has static peers as well.

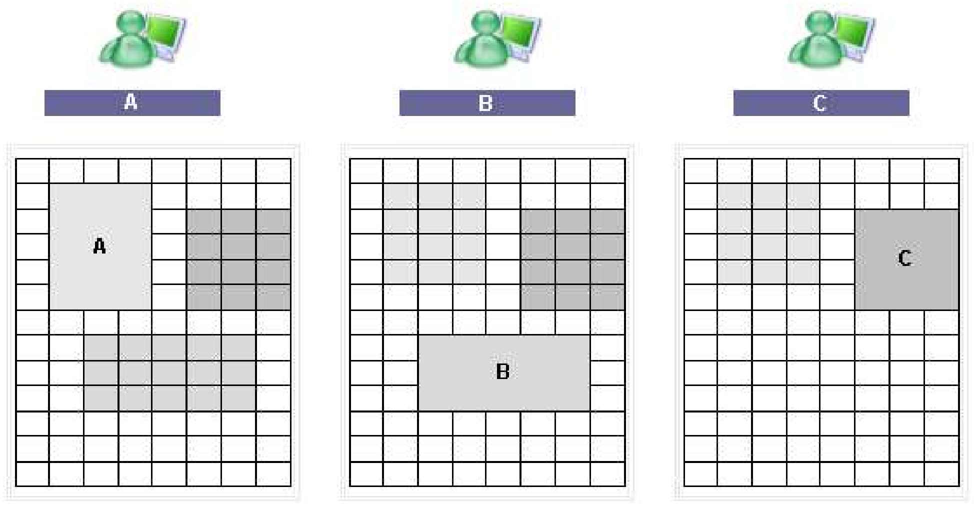


Illustration 4.2. P2P Zone Visibility

Zooming into the application implementation itself, Illustration 4.3. TerraPeer Layered Architecture shows how the user interacts with the GUI layer, that in turn provides a viewer for the VUI construct on one hand and interfaces to the application core on the other. The 'helpers' and 'vars' blocks are supportive. The 'repository' is external storage (a missing arrow should really point to the core on this illustration). At the bottom, the network layer interfaces to the network.

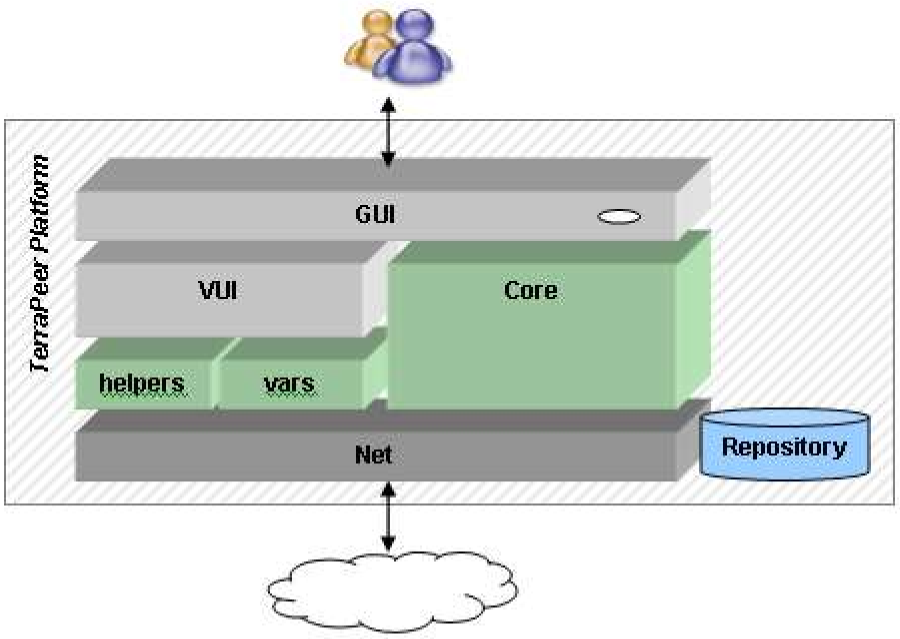


Illustration 4.3. TerraPeer Layered Architecture

4.1.3.VUI Design

The virtual user interface (VUI) sub-system should contain the methods and fields necessary to create and manage the VE space, peer zones, and zone services.

The TerraPeer architecture defines this in the `terrapeer.vui` package, as will be discussed in the next chapter. The overall concept of the VUI, though is shown in Illustration 4.4. TerraPeer Virtual User Interface (VUI) Package.

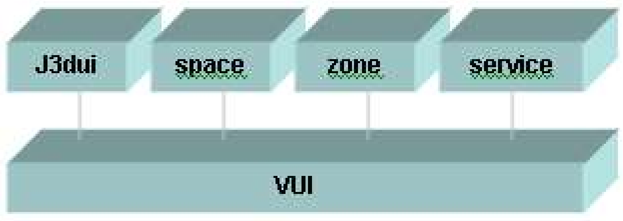


Illustration 4.4. TerraPeer Virtual User Interface (VUI) Package

Each entity handles different aspects of the VE, and is totally abstracted from networking or GUI related issues. The J3dui block mainly is responsible for navigation and object interaction while relying heavily on the Java3D API (not visible here). The space block acts as a common container for environment properties. Similarly the zone block encapsulates all functionality associated to the construction, manipulation, and storing of zones. The service block, finally allows specific definitions of virtual building shapes that can represent certain services.

The packaged structure in the illustration above does not reflect the object-relationship among these blocks, which is better depicted in the UML graphs (see also Appendix V).

4.1.4.User

User representation in the VE is through selectable avatars. The representation can be restricted by some spatial constraints, and is able to 'hide'. An advanced feature might be an algorithm that would automatically allocate space for avatars in confined areas, or rather transform the avatars accordingly. Initially, all avatars are invisible in TerraPeer.

User attention is focused, and should not be distracted too much. If designed well, a user interface can take advantage of user attention, for example by placing information at locations and with certain attributes depending on their content and probable priority to the user. Read also [Attention]. The subject of attention is quite fascinating, but will not be elaborated further here.

Access control, forced visibility, or forced restrictions all are features the designer of a system can decide upon. The weighting is crucial, as it not only controls object sharing or user visibility.

Personal information management and trust and policy management are important to the user, and have direct impact. The distinction between a user's personal information, his access rights, and the information and services available at other sites, becomes blurred in DVE's. Hence, some kind of trust management system should be considered in the future.

The Microsoft 'Passport' for example implements such a system for the web. Many sites and portals today have personal accounts that allow individual content, and convenience i.e. when ordering a service. The last section of this chapter will summarize an idea for how a distributed trust model could look like. This subject is again fascinating, but will not be elaborated either.

4.2.Virtual Space Architecture

4.2.1.A 3D Interface to Cyberspace

There is an aggregation of techniques that have to be implemented to visualize a VE. The virtual world is presented through the view-display-screen chain. Internal and external view geometries, as well as spatial coordinates for the world system on one side and the user interface on the other have to be aligned.

GUI design should attempt to incorporate as much functionality and feedback mechanisms into the 3D environment, as this is the main space the user will move through. There are instances, though, where the traditional 2D interface provides superior capabilities in terms of usability, e.g. navigational maps, setting of attributes, or information display. Especially position-independent information and functionality should be available at all times, as the user roams around the VE.

3D techniques, an overlay display (HUD) for example, can be applied to create a pseudo 2D interface. These require considerations about display facing, constant sizing, overlapping, and visibility characteristics. Constructing the world space itself requires these considerations. The space should accommodate features such as a view layout (occlusion, object arrangement, billboarding, etc.) and restrictions such as visibility radius or filtering.

It is assumed that a user navigates through spaces with two input devices; mouse and keyboard. The 2D mouse movement across the screen in particular has to be translated into spatial 3D vectors. The relationship is further complicated through different usage of the mouse-pointer, such as navigation or object picking.

Hence, the DVE space must consider basic navigation techniques, including first- or second-person viewing, object selection, mouse dragging, data filtering, and data traversal.

3D navigation usually implies a space similar to reality, which is prominent especially in games. The embodiment of human-like avatars and real world objects represents an analogue that is easy to recognize, thus easy to manipulate and navigate.

Alternatively, conceptual navigation, such as browsing through the web, might be appropriate in circumstances where information is associated to each other, or when the context stream has to be traveled forth and back.

Data visualization is the process by which data is presented by the application to the user. From a user interface designer's point of view, this process can be achieved through various means, including icons, graphs, spatial or color alteration. Visualizing peers in a VE is best accomplished by dynamic zoning, where spatial navigation and visibility attributes create the view. Each peer is represented by a zone. Each zone can contain customized objects that visualize (similar to icons) certain resources or services.

Viewers to the VE display a basic virtual world environment on each peer, including grid-lines or a background horizon to easy orientation.

Control
Application support of positioning and navigation in the VE world

- Visual Feedback in 3D
  - A visual static 'origo' landmark, i.e. a vertical z-axis that can be used as a homing-pointer, and other landmarks to distinguish locality
  - A visual static grid, i.e. checkerboard-like lines stretching across the ground x-y plane
  - Visualization of each peer though zoning and meta data on objects and zones
  - A multi-layered space, i.e. static x-y levels in different heights for purpose-specific navigation. This optional feature could offer three levels, 0-ground for 'virtual land', under-ground for virtual bots-only, and sky-level for a navigation space only (providing a free path with no zones that might have obstacles or restrict passage)
- GUI Dashboard
  - Compass rose navigation that associate directions with the North-East-South-West metaphor
  - Virtual x-y-z coordinates (metric) to help orientation
  - Maps with orientation cues, routes and path, etc.
- Control
  - Unrestricted movement for avatars, limited by access control of zones
  - Direct control of view by Keyboard/Mouse (input processing, clamping, etc.)
  - Manual navigation and direct porting to locations
  - Navigation through semantic links (using services)
  - Avatar visibility control

Objects
Application enables peers to create and share instances in the VE world

- Peers can build virtual 'home-zones', i.e. enclosed areas at certain locations with meta-data
- Include standard 3D building blocks (based on VRML/X3D)
- Virtual objects have attached certain meta data (name, description, links, etc.) and functionality
- Peers can publish resources or services at their home in the form of virtual objects
- Enable creation of basic resource/service types, such as a document, multimedia, URL links, a visible (interactive) 3D representation, a web-site, a forum, group messaging, file-sharing, etc.
- Share objects, resources, and services using access control properties (viewable, executable, modifiable) that can be set to certain users or groups

Filtering
Application implements a basic allowance-set to align usage within the DVE, and to optimize rendering, processing and communication load (bandwidth)

- Allow individual filtering of peers, avatars, groups, zones
- Allow setting a visual radius (aura) to limit 3D processing (rendering)
- Visualize peer zones by simplified planar-areas (and colors), instead of showing all associated collections of 3D objects
- Attach meta-information to peer-homes, services, public spaces to ease identification (text-based labels, visual aids, etc.) and use color-coding/attention mechanisms to easy user overview and navigation
- Allow setting of space and object sharing, i.e. space and object permission control to set ownership, read/view, create, modify, and delete permissions.

TerraPeer allows dynamic allocation of peer-space and location by restricting spatial size of home for each peer, and creating a reservation-system. It also allows individual peers to create own rules and restrictions in their space through specific laws that apply only in their home-zone, and restrictions on entries (i.e. by groups).

As soon as objects can be 'shared', or multiple users can interact on the same scene, problems regarding permission as well as 3D techniques have to be accounted for. This is achieved by only allowing a single participant to modify an object. The object modification 'flag' is handed over (relay mechanism). Note that using or executing an object can happen simultaneously without lockups in the semaphore.

4.2.2.Environment and Zones

Three dimensional space for user navigation and interaction require spatial management. With the definition of a position (x-y-z coordinates) an initial rule for the space is set. Positions are constant across the network. Every peer that connects to it would 'see' the same objects within the environment. Objects placed or moved in the space would be 'final'

This is an advantage in terms of navigability and consistency, but of course also a disadvantage especially considering alignment and traffic. To address these problems, the notion of a zone is introduced.

The 'Zone', 'Visibility Radius', 'Aura', or 'Area of Interest' (AOI) are all wordings for a similar appearance. The confusion might occur because of two different problems, namely the problem to optimize network traffic, and the problem to control multi-user access attributes.

Optimizing spatial content in DVE's is a central aspect, as large scale systems have to cope with increased load of traffic. This leads to designs implementing separated areas, or 'aura' in order to reduce the overhead, and let each participant (peer) only send and receive the information within a certain virtual boundary.

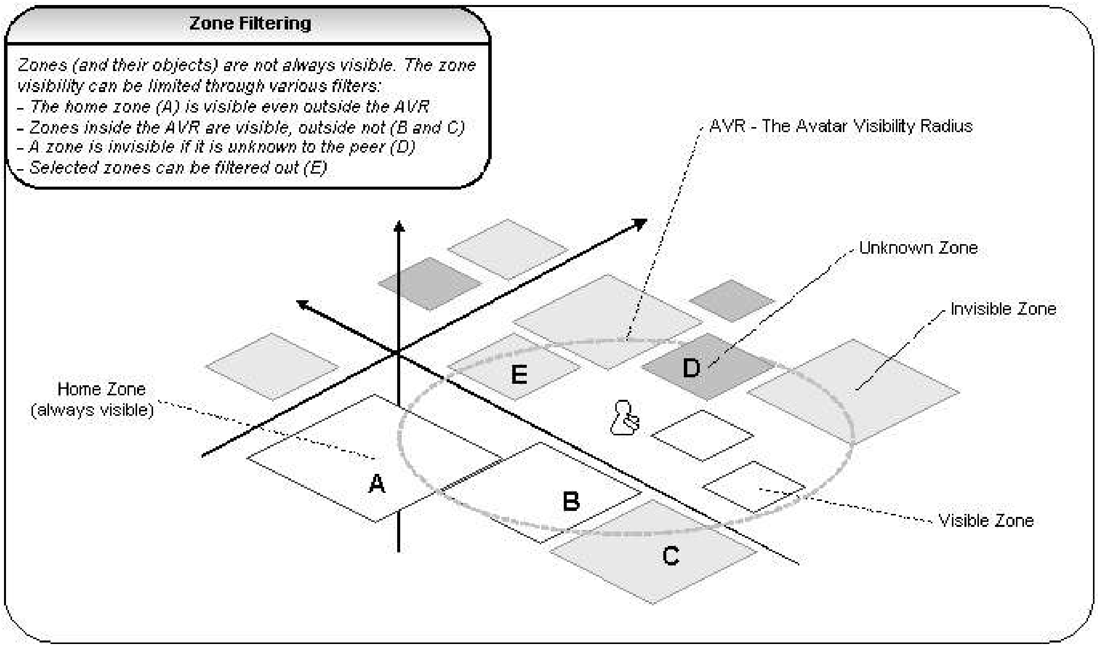

Zone Filtering
Zones (and their objects) are not always visible. The zone visibility can be limited through various filters:
- The home zone (A) is visible even outside the AVR
- Zones inside the AVR are visible, outside not (B and C)
- A zone is invisible if it is unknown to the peer (D)
- Selected zones can be filtered out (E)
AVR - The Avatar Visibility Radius
Unknown Zone
Invisible Zone
Home Zone (always visible)
Visible Zone
A
B
C
D
E

Illustration 4.5. TerraPeer VE Zone Filtering

This technique is similar to filtering. There could theoretically be unlimited ways of filtering out parts of the environment, as it simply means removing some entities. Designers have to decide which information is unnecessary noise, and which the user could decide to be unwanted. Examples in 3D graphics are occlusion and mipmapping techniques.

The other problem – to control access attributes – is rooted further up in the system and could be considered to be integral only to the VE space itself (though this isn't the case in reality, as access settings on virtual entities often would be dispatched to non-virtual parts).

Inside the VE, when more than one user can create or transform an entity, access and sharing attributes suddenly become important. Virtual areas could be used to delimit the access rights of objects (and/or avatars) within their boundaries.

Alas, the 'zones' in TerraPeer context consider both problems. By allowing zone filtering mechanisms it is possible to only render those zones within a certain vicinity or those selected by the user (see Illustration 4.5. TerraPeer VE Zone Filtering). By leaving ownership of the zone to each peer who creates it, access rights of objects that can be build on top of it are implicitly build into the system. Zones will be described in further detail in section 3.9.

Time is of essence when maintaining appropriate frame-rates and reducing input lag. The real-time constraints in DVE's require techniques such as rendering with a level of detail [Wil83], or the use of predictive algorithms to reduce apparent input device lag [FSP93].

There are other issues that must be considered, such as temporal consistency and the ‘local’ time. Causality – cause and effect – affects synchronization (VE events triggered from different sources) and the processing of events [CITY94]. Thus, the time dimension needs to be considered for the processing of behaviors.

This is often realized using distributed event models such as event broadcasting, state change notification handlers, dedicated point-to-point connections between objects, or object-to-object messaging. A specific event mechanism will not be implemented in this architecture, but the basis for synchronization, multicasting, and processing of events and messages will be conceived.

4.2.3.Object Structure

Illustration 4.6. TerraPeer VE Architecture shows how the software blocks responsible for the abstract virtual space look like. The core 'MySpace' object is the central reference point for the other parts of the application.

Below, 'SpaceView' (the 3D viewer), 'Log' (the application logger), 'ZoneTree' (SG and GUI widget), and 'ZoneBuilder' (creating zones, and zone objects, loading, etc.) are interfacing other parts as well.

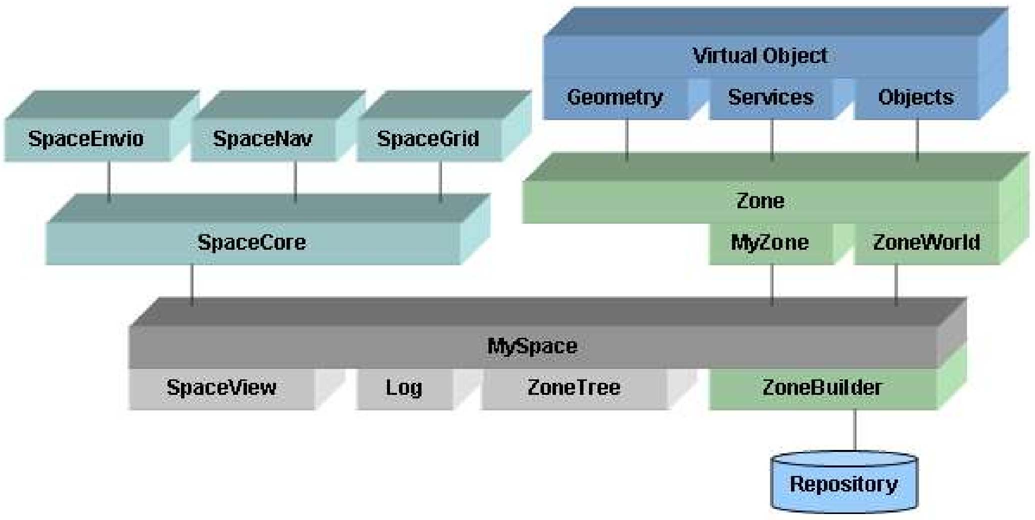


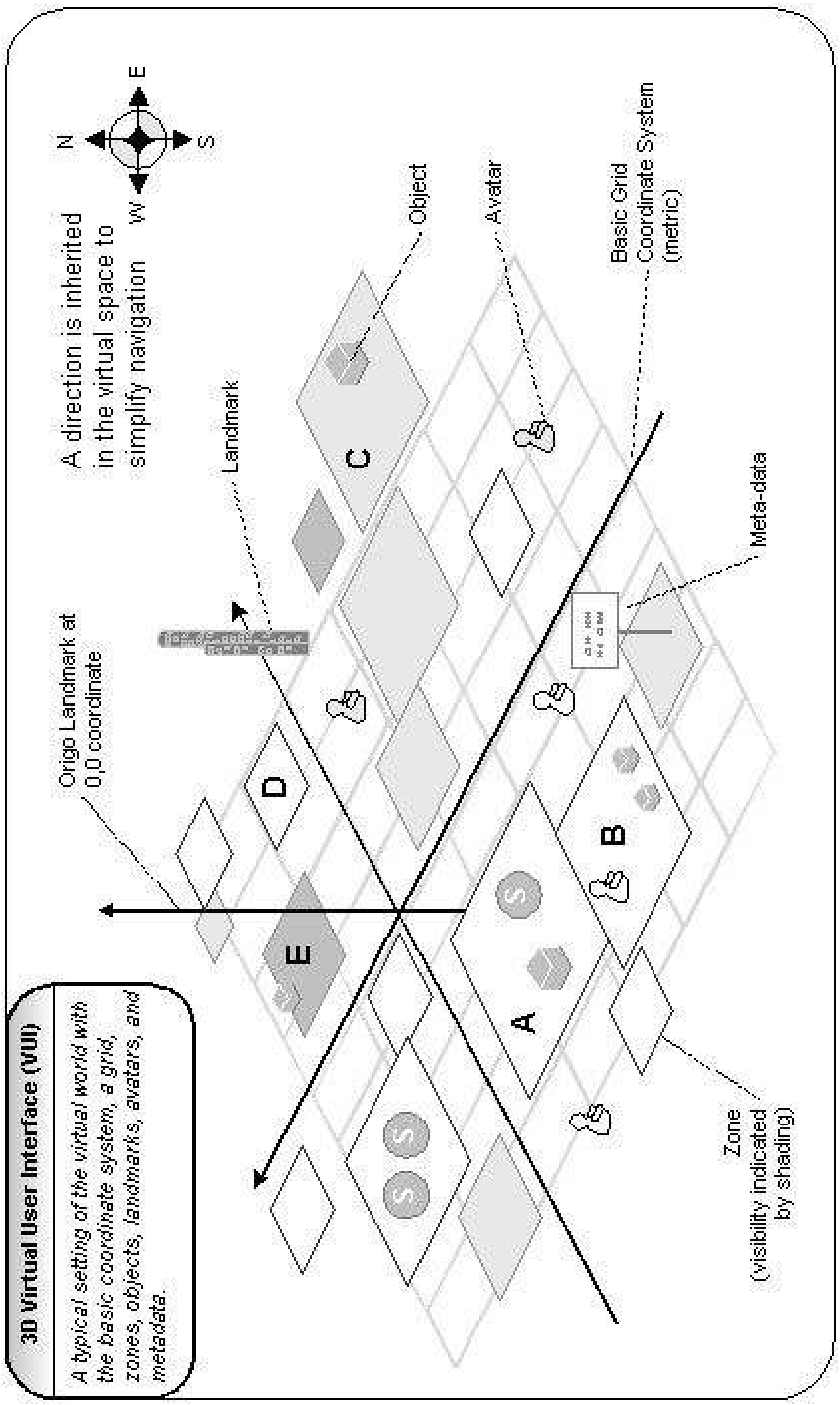
Illustration 4.6. TerraPeer VE Architecture

The 'SpaceCore', 'MyZone', and 'ZoneWorld' objects, though are encapsulated within 'MySpace'. SpaceCore is responsible for the basic VE without any content from peers, and contains environment, navigation, and grid objects. MyZone and ZoneWorld both use the Zone object, which defines a range of properties that are not shown here. The Zone inherits a Geometry (as any virtual entity), and encapsulates Basic Objects and Service Objects.

ZoneWold represents the collection of zones that the particular peer is aware of.

4.2.4.Space

The 3D virtual user interface space is shown in Illustration 4.7. TerraPeer Virtual Space View.


3D Virtual User Interface (VUI)

A typical setting of the virtual world with the basic coordinate system, a grid, zones, objects, landmarks, avatars, and metadata.

A direction is inherited in the virtual space to simplify navigation

N
S
W
E
Object
Avatar
Basic Grid Coordinate System (metric)
Landmark
Meta-data
Origo Landmark at 0,0 coordinate
Zone (visibility indicated by shading)
A
B
C
D
E
S

Illustration 4.7. TerraPeer Virtual Space View

4.2.4.1.Position

A position is based on three spatial coordinates. A static origo landmark is present to represent the center (0,0,0) coordinates (see Illustration 4.8. TerraPeer VE Coordinate System). The 3D world displays an endless line through the z-axis, while the GUI can adjust line-visibility and has a 1-D origo-homing-pointer as navigational aid.

A static grid (matrix or raster) with endless x-y coordinates runs +/- along x-/y-axis from the origo point. The grid is based on the metric system, and the 3D world displays a grid-size depending on the POV (e.g. 10^distance meters). Further, the GUI presents a 2-D compass rose (N-E-S-W) for orientation.

In a future version, the space could be multi-layered, for example with a ground-level for navigation and building virtual land, and a navigation-only level. The GUI should allow instant jumping between levels.

VE background might be adjustable and allow individual settings. It could provide a real-time GMT-based sunrise/sunset circle.

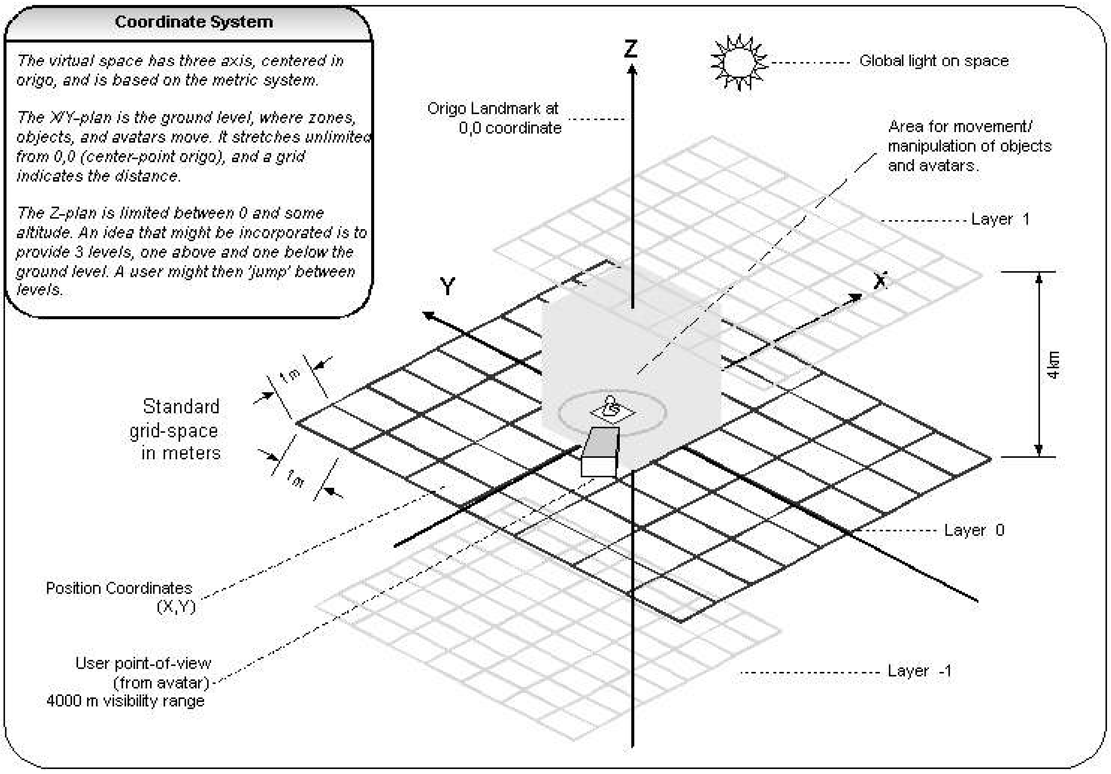

Coordinate System
The virtual space has three axis, centered in origo, and is based on the metric system.
The XY-plan is the ground level, where zones, objects, and avatars move. It stretches unlimited from 0,0 (center-point origo), and a grid indicates the distance.
The Z-plan is limited between 0 and some altitude. An idea that might be incorporated is to provide 3 levels, one above and one below the ground level. A user might then 'jump' between levels.
Z
Global light on space
Origo Landmark at 0,0 coordinate
Area for movement/ manipulation of objects and avatars.
Layer 1
Y
X
4 km
1 m
Standard grid-space in meters
1 m
Layer 0
Position Coordinates (X,Y)
User point-of-view (from avatar) 4000 m visibility range
Layer -1

Illustration 4.8. TerraPeer VE Coordinate System

4.2.4.2.Navigation

All navigation is done through the peer-avatar in real-time. Avatars can be visible or not to certain peers or groups. The 3D world is always seen from the avatar's POV. The VUI can display over-layered grid-coordinates, and landmarks (as orientation cues) can be placed at certain public places to distinguish locality. The GUI could show a 2D map of landmarks, and distance and path to nearest landmark (including origo).

Pathways could be made viewable in 3D world as lines-of-sight to show routes from A to B, to show a "service-trail" (maybe a shopping-passage), or to indicate web-rings. The GUI could show a 2D map of the pathways.

Zoning allows special settings for private and public areas, such as which peer-groups are allowed to enter. Permissions to pass, modify, vote, etc. could be set. The GUI could show a 2D maps of zones. Special entry points could be provided for zones to enhance the experience of instant tunneling through space, i.e. arriving at familiar or decorated pre-defined 'doors' (see Illustration 4.9. Possible topological grid view of Entry-points).

4.2.4.3.Create and Share Instances

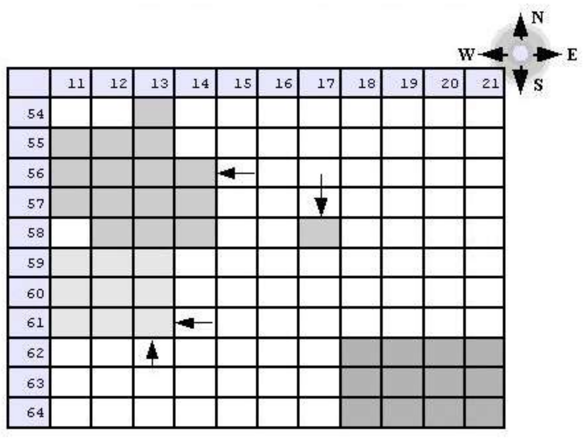


Illustration 4.9. Possible topological grid view of Entry-points

The effect of multiple users sharing an acting in the same scene is shortly examined in a paper “Security in Co-authored Virtual Environments” by Larsen and Christensen. Some considerations have to be made when establishing security based on location. Collision detection, and mechanisms how objects are created and destructed have to be established.

This is addressed in this project through permission-settings. Permissions define which user has access to which resource, and where. A user can only create and modify objects on her own home zone. If a user has permission, he might be able to enter the zone, move objects, or use services of other user's.

4.2.4.4.Basic Rule-set

The common rules let user-avatars move around freely. They let let peers build virtual homes in closed areas at certain locations that cannot overlap. They let user's build the virtual world using standard 3D interfaces (VRML/X3D). A set of rules is required to ease rendering efforts on 3D world visualization of peer avatars, homes, and zones.

The visualization mechanism is based on the rule that while basic space environment is always visible, objects and avatars are not. The application allows filtering of any peers, avatars, groups, and zones, and should provide default-settings to filter everything but the peer's own zone and those of groups it is a member of.

A filtering mechanism to allow the setting of visual radius should by default paint all out-of-radius known zones to simplified colored planar-areas. Avatars could by default be visualized as a standard form.

With regards to creation, each peer is allowed to create only a single avatar and zone. The zone – virtual 'land' – should ideally be of any shape, but has to be enclosed (connected polygon), and should have a maximal border-length. The placement (zone position) is established over time, using a reservation-system. Conflicting areas are resolved by first-come-first-serve.

This reservation-system is structured as:

1. “reserve location” *iff* “no other peer located”
2. “send out reservation to other peers”
3. “wait to see if another peer claims the zone emerges”
4. “settle” *or* ”move” *(go back to 1)*
5. “if another peer that claims the zone emerges” *then* “zone is shared (new group) until one peer moves; a peer moving after settlement is granted higher priority when settlement is in conflict again”

4.2.4.5.Sharing

Sharing rules apply to all instances in the virtual hierarchy:

- ownership to space cannot be taken
- ownership to a peer-home is by default the peer that created it
- ownership to a shared/public zone is shared by all peers that create it
- ownership to an object is by default the peer that created it
- ownership of any object can be transferred
- ownership of zone can only be transferred to a group it is member of
- ownership of avatars cannot be transferred

For the asset control, a peer can set zone and object permissions to other peers that it owns. Permissions can be:

- create (objects in land)
- read (data attached to land/objects, but not meta-data to land)
- view (land or objects)
- pass (land or objects)
- use (service)
- modify (objects in land)
- delete (objects in land)

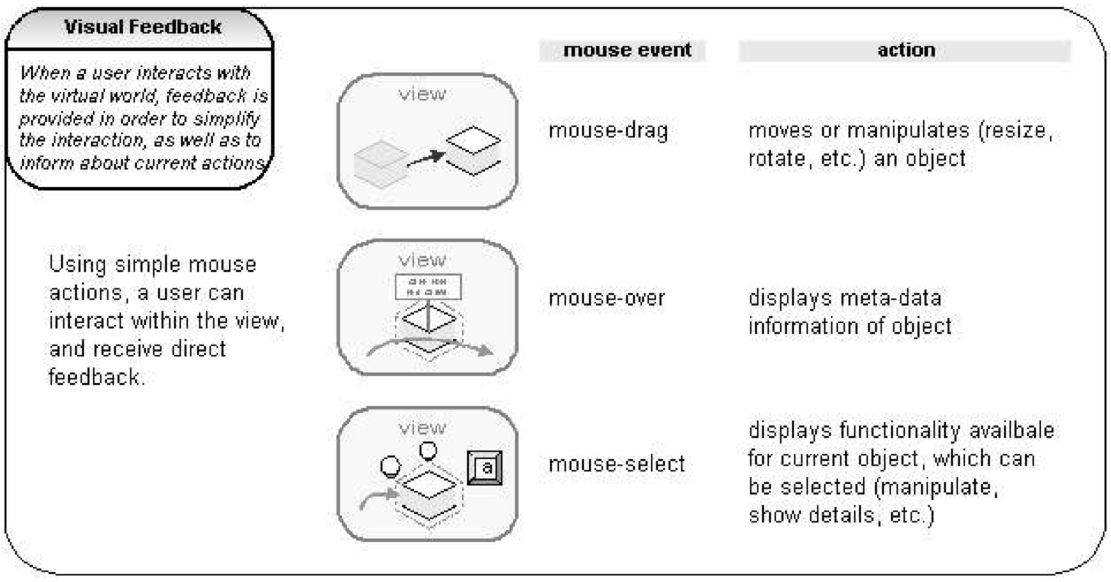

Visual Feedback
When a user interacts with the virtual world, feedback is provided in order to simplify the interaction, as well as to inform about current actions
Using simple mouse actions, a user can interact within the view, and receive direct feedback.
mouse event
action
view
mouse-drag
moves or manipulates (resize, rotate, etc.) an object
view
mouse-over
displays meta-data information of object
view
a
mouse-select
displays functionality availbale for current object, which can be selected (manipulate, show details, etc.)

Illustration 4.10. TerraPeer VE Visual Feedback

4.2.4.6.Interface Feedback

The feedback from the VE to the user should use filtering mechanisms by removing or simplifying entities to ease user orientation. Identifying and attention-related mechanisms, such as color-coding and meta-data could also improve the interface (see Illustration 4.10. TerraPeer VE Visual Feedback). The GUI should be able to adjust the display of meta-data; the types could be zone information, peer-group zones, location or service category related. For instance, labels hovering above zones could be set visible or hidden.

4.2.5.Navigation

The paper “Designing for a clear city image” lists very good design guidelines that have been included here (with references to K.Lynch removed) [CITY94, pp.39]:

10 Guidelines

1. "Singularity or figure-background clarity and sharpness of boundaries". The contrast may be to the immediate surroundings or to the observer's experience. This guideline aims at the qualities of elements which make them identifiable.
2. "Form simplicity". Elements should strive for clarity and simplicity of visible form in the geometrical sense as these forms are more easily incorporated into the overall image.
3. "Continuity" in edges or surfaces, nearness of parts, repetition of rhythmic intervals. These qualities ease the perception of a complex physical reality as one or as interrelated.
4. “Dominance of one element in an ensemble” translates to a cluster of elements grouped around one major element. This quality, like continuity, allows simplification of the image (abstraction).
5. "Clarity of joint" means high visibility of joints and seams, for example at major intersections. Such joints are the strategic moments of structure and should be highly perceptible.
6. "Directional differentiation" means that elements should exhibit directional qualities. These qualities differentiate one end from the other and are very useful in structuring on a larger scale.
7. "Visual scope" are qualities of the environment which influence the range of vision - be it actually or symbolically. Examples for such constellations are transparencies, overlaps, vistas, panoramas and several others. These qualities help grasping of larger complex wholes by increasing the efficiency of vision.
8. "Motion awareness" argues that actual and potential motion shall be made explicit to the observer and that the quality of motion shall be made perceivable to the observer by visual or kinesthetic means.
9. "Time series", for instance sequences of landmarks, are series of elements which are sensed over time. An example is a series of landmarks to be encountered on a route.
10. "Names and meanings" are non-physical characteristics which may enhance the imaginability of elements. They sometimes give clues about the location (like in "North Station") or trigger historical, functional or economical associations. Such non-physical characteristics facilitate structuring of elements in the environment.

The Problem of Poorly Navigable Spaces is also addressed in [NavISpace].

Illustration 4.11. TerraPeer VE Input and Feedback shows how the navigation input and output system of TerraPeer is created.

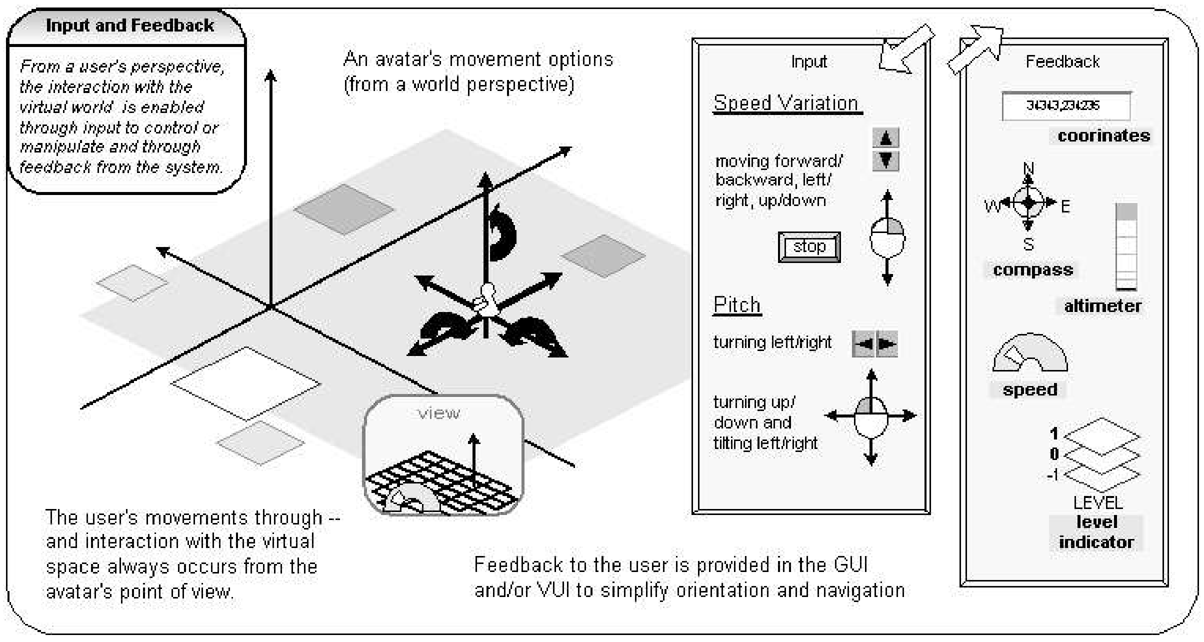

Input and Feedback
From a user's perspective, the interaction with the virtual world is enabled through input to control or manipulate and through feedback from the system.
An avatar's movement options (from a world perspective)
view
The user's movements through -- and interaction with the virtual space always occurs from the avatar's point of view.
Feedback to the user is provided in the GUI and/or VUI to simplify orientation and navigation
Input
Speed Variation
moving forward/ backward, left/ right, up/down
stop
Pitch
turning left/right
turning up/ down and tilting left/right
Feedback
34343,234235
coorinates
N
W
E
S
compass
altimeter
speed
1
0
-1
LEVEL
level indicator

Illustration 4.11. TerraPeer VE Input and Feedback

4.2.6.Object Control and Clamping

Avatars, objects, zones, and services are all virtual representations that can be manipulated. As discussed earlier, it is often necessary to restrict movements or abilities in order to realize a better and easier navigable environment.

Limiting object and visual control by the user in the design is shown in Illustration 4.12. Object control in VE and Illustration 4.13. Visual control in VE:

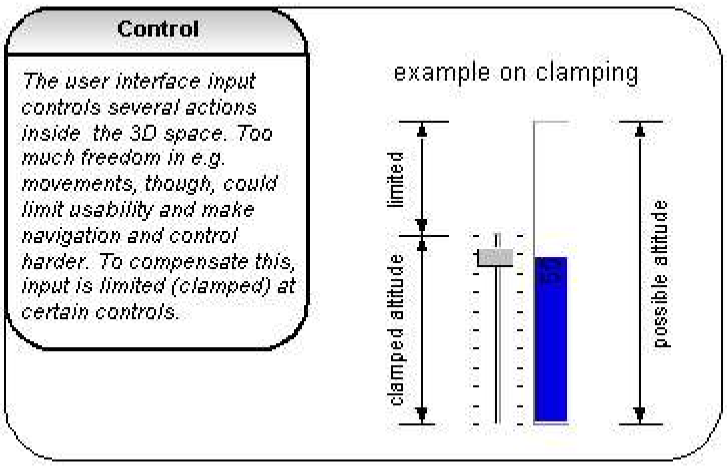

Control
The user interface input controls several actions inside the 3D space. Too much freedom in e.g. movements, though, could limit usability and make navigation and control harder. To compensate this, input is limited (clamped) at certain controls.
example on clamping
limited
clamped altitude
possible altitude

Illustration 4.12. Object control in VE

These illustrations show the method of clamping. The idea is simply to adjust the range of movement according to the current environment. A user-avatar might be able to move rapidly across the world, but not exceed a certain speed. Inside a zone border, movements might be relatively slower but gain more control to create objects, for example.

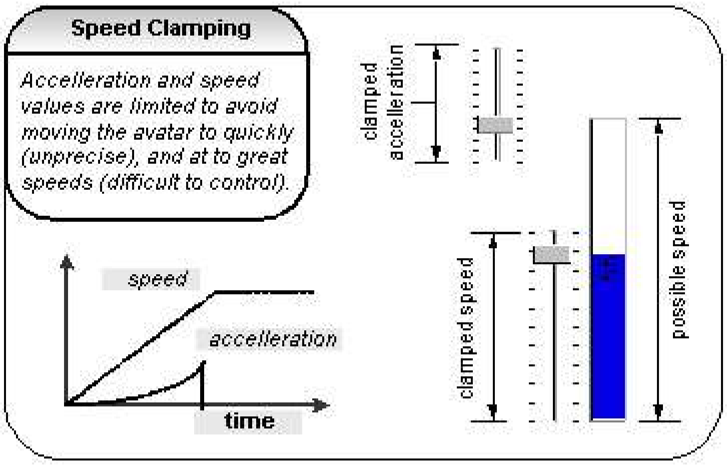

Speed Clamping
Accelleration and speed values are limited to avoid moving the avatar to quickly (unprecise), and at to great speeds (difficult to control).
speed
accelleration
time
clamped acceleration
clamped speed
possible speed

Illustration 4.13. Visual control in VE

4.2.7.Object Persistence

When peers connect and disconnect to each other, objects owned by each peer have to be handled. Object persistence means that an object continues to exist after the original owner peer disconnects (in one particular C/S-based DVE project, their 'zone server' would nominate a new owner).

Rendering techniques, such as vector-based interpolation (as often used in game engines) is often applied in networked VE's to create the illusion of continuity. Usually, in an event-based architecture, the position prediction is calculated by the last known position and velocity of an entity.

This persistence is implicitly handled by the TerraPeer architecture by leaving it up the each participant to arrange persistence. If the peer's information has been distributed to other peers, the essential content remains online. Whether certain services become unavailable depends on where else the resource is located. A service might well point to a very different resource, such as a simple HTTP web-page.

4.2.8.Object Sharing and Access

In TerraPeer, the access level of every virtual object can be adjusted. This mechanism is similar to the Unix file system, where the access attributes of files can be set according to who should be allowed to do what. The idea was to implement attributes for each virtual object (that also could be a service) that the original owner can adjust. Attributes could be set to allow other users to view, access, or modify the object or service. This reflects the 'xwr' (execute-write-read) attribute of a Unix file. Further, the mechanism can be extended to be viable for certain specified groups.

This essential functionality should establish a just way of creating, placing, and sharing virtual objects.

Original owner of an object is the creator. As a user is restricted to create objects within his own zone, he would usually not own any objects outside that virtual boundary. Whether object ownership can be transferred depends on each object's access settings.

4.3.Distributed Platform Architecture

4.3.1.Connecting at the Edge

The application peer-to-peer network runs on the JXTA platform. JXTA itself is composed of several layers and protocols that enable a variety of services (this was described in earlier sections). JXTA provides the necessary functionality to discover other peers, connect to them through pipes, and communicate data between them.

A specific TerraPeer Protocol would have very similar functionality, and though there exist several choices of alternative solutions ranging from open source platforms to peculiar standardization attempts, or even proprietary game engines, the JXTA choice seemed attractive solely by it's pluggable design.

Combining JXTA with X3D as the fundamental layer for a DVE application that could provide services for other protocols, including URL, HTTP, or FTP was a cornerstone of the original architecture.

The JXTA API was created by Sun Microsystems to be flexible to use, and one of its main capabilities is to run on top of various OSI layers, such as TCP or HTTP. Using the pipe mechanism, which enables peers to send and receive data, XML-based TerraPeer messages are exchanged.

Messages basically contain repository data, which in turn is a collection of zone-objects. Each zone object contains all information for a particular peer zone.

The six JXTA protocols ERP, RVP, PRP, PDP, PIP, and PBP are employed through the API. Advertisement-, discovery-, and pipe-service methods are called by the application.

Hence, each distributed peer is calling these JXTA-related operations throughout a life-cycle:

1. Advertises the peer itself on the network
2. Apply discovery service to find other peers in the network
3. Requests virtual space content from other peers, which includes both the zones of connected peers, and zones that these peers have stored locally in their repository
4. Responds to other peer's requests, and sends repository data
5. Update of home zone leads to propagation of this peers own data

4.3.2.Data Exchange

Protocols for the data exchange are working in conjunction with the repository. The JXTA messaging pipe service is a fundamental communication platform. On top of that runs the TerraPeer XML-based protocol, which also embeds X3D data.

The general idea of the peer data exchange is that information is distributed on a if-needed basis. By default the VE space is empty. A peer should be able to fully control the amount of content visible locally. By roaming to other peers, their zones and zone content can become visible. Filtering features, again, are important tools to control the space.

By abstracting objects to a minimal set of properties, it is further possible to optimize traffic. "Because the field values of a compound object completely describes the objects state, the internal representation does not need to be distributed, but can be reconstructed by each process. This can significantly reduce the amount of bandwidth needed to replicate a complex geometric object as only a few field values need to be transmitted instead of a completed geometric description of the object" [AVOCADO].

Spatial filtering can save bandwidth. The method implies that only certain event updates (or any kind of data) is allowed, e.g. traffic is restricted for positions outside some range (a radius). Zones should restrict the event traffic to those peers connected to and within the particular zone.

Filters based on personal preferences or interest could limit traffic further, while not compromising on the service. An anecdote in this context: The P2P network should also be able to provide one-to-one channels (or group-based) to basically circumvent all other environment in certain situations. Dynamic allocation of multicast groups based on location might be useful, too.

An example where data exchange is limited to the transfer of field values could be the URL of a service object. The virtual object representation itself would be reconstructed by the peer locally.

4.3.3.Network Layer

The code package design for the network layer is quite plain (see Illustration 4.14. TerraPeer Network Layers). It references JXTA and XML API's for protocol and repository respectively.

The JXTA block establishes connections, creates advertisements, performs discovery, sends and receives messages, and so on. The XML block parses repository files, and packages X3D into the TerraPeer protocol (see Illustration 4.15. XML Schema for TerraPeer).

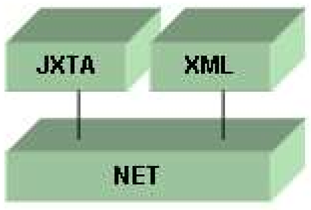


Illustration 4.14. TerraPeer Network Layers

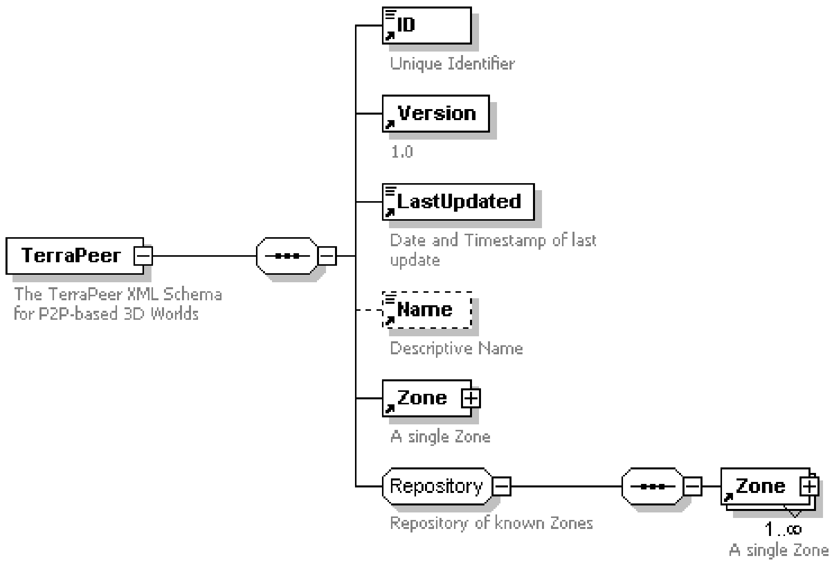


Illustration 4.15. XML Schema for TerraPeer

4.4.Zones

4.4.1.Definition of a Zone

Zones represent peers, and thereby users. The zone metaphor allows users to virtualize their peer. Since each peer is only allowed to create one single zone, the resulting network consists of a maximum number of zones equal to the number of peers part of the network. As a peer not necessarily has to create a zone, there might be less zones than peers.

A specific zone exactly reflects a peer's state, position, and services offered. A user can freely administer his zone, but has to follow the rules of the common space, and is bound to the rules of other zones.

4.4.2.Zone Properties

The XML Schema 'ZoneType' depicted below (Illustration 4.16. Zone Type XML) represents the structure of a Zone, and included properties. In this implementation version, each zone has a UID, a version, name, optional description, a time-stamp, a geometry, and a collection of objects attached.

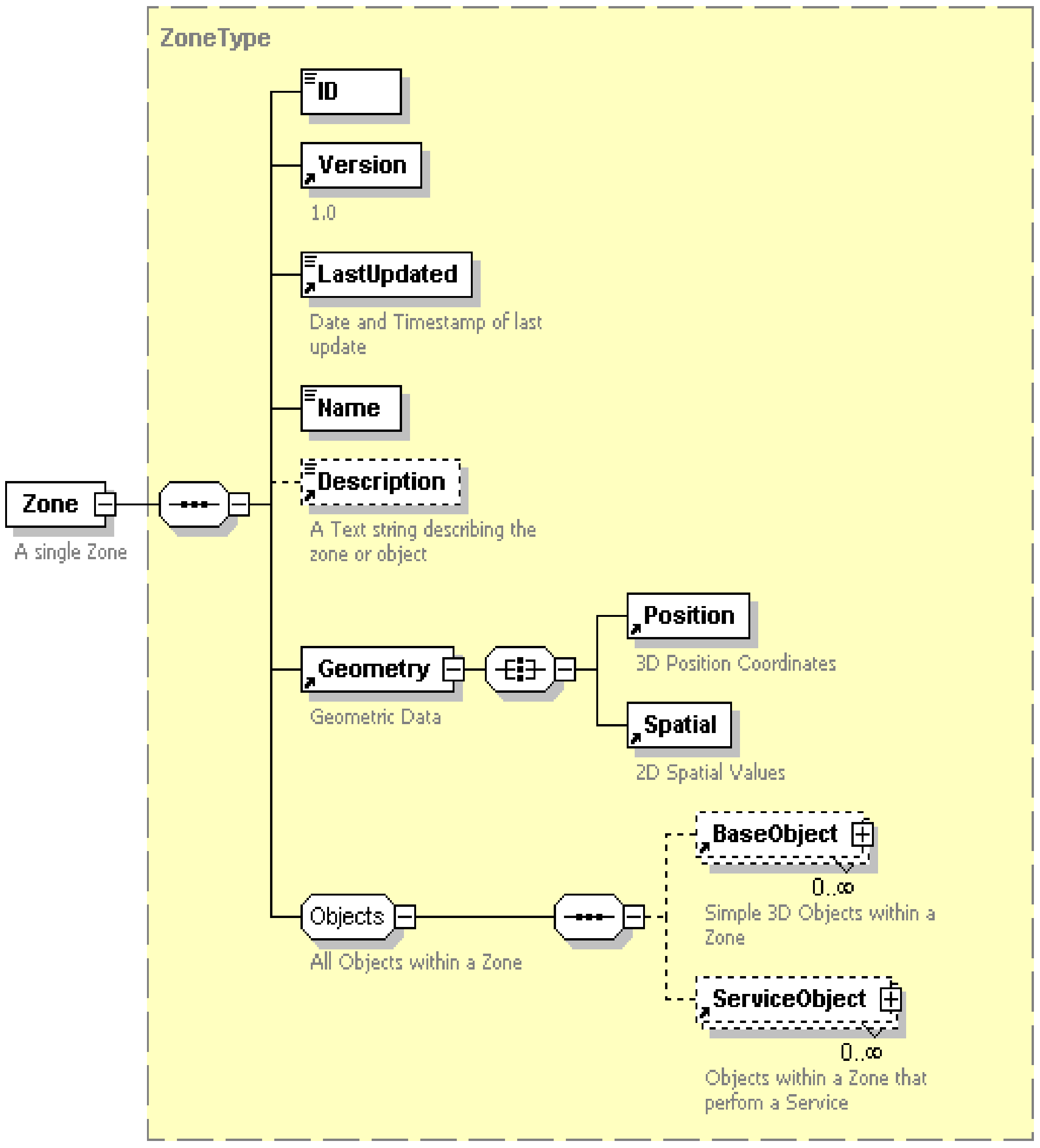
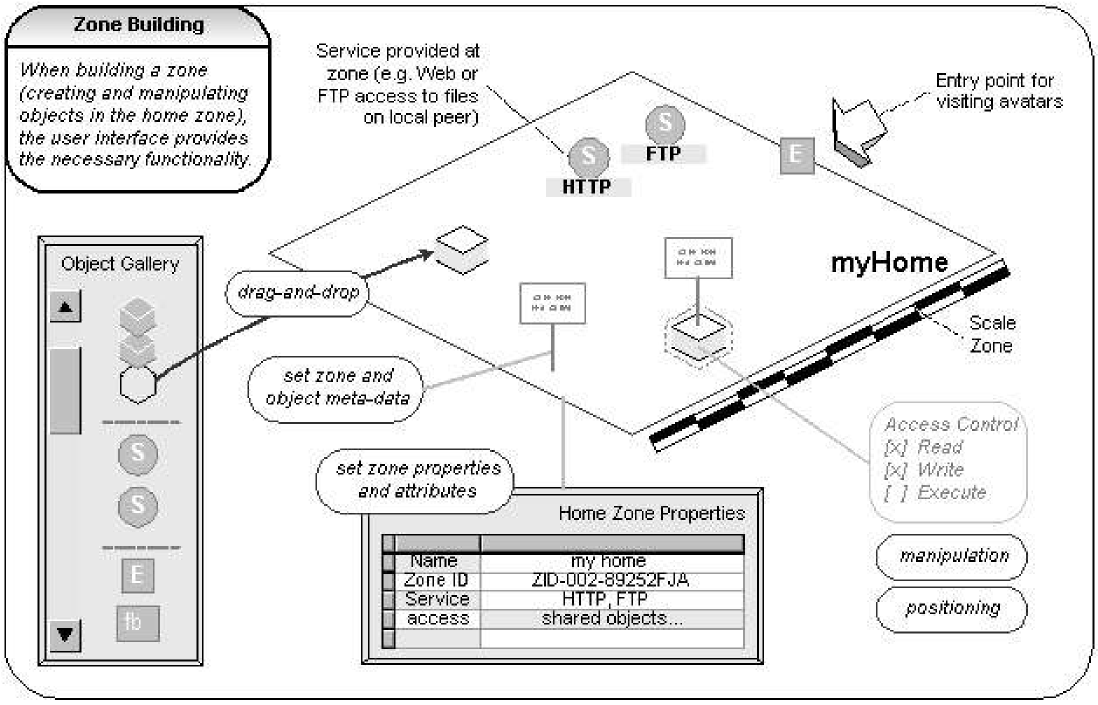


Illustration 4.16. Zone Type XML

4.4.3.Building a Zone

Illustration 4.17. TerraPeer VE Zone Building shows the 3D user interface techniques to build a zone (or rather build on top of it). Through an object palette, similar to an IDE, basic objects and service objects can be selected and placed in the zone. Adjusting attributes for each object allows the user to control permission, transformations, and eventually interaction/response functionalities.


Zone Building
When building a zone (creating and manipulating objects in the home zone), the user interface provides the necessary functionality.
Service provided at zone (e.g. Web or FTP access to files on local peer)
S
HTTP
S
FTP
E
Entry point for visiting avatars
myHome
Scale Zone
Object Gallery
drag-and-drop
set zone and object meta-data
set zone properties and attributes
S
S
E
fb
Access Control
[x] Read
[x] Write
[ ] Execute
manipulation
positioning
Home Zone Properties

Name	my home
Zone ID	ZID-002-89252FJA
Service	HTTP, FTP
access	shared objects...

Illustration 4.17. TerraPeer VE Zone Building

A service provided by a peer can be made publicly available and accessable. For example is peer B linking to a service provided by peer A in Illustration 4.18. Zone Service Linking. The service can either be a 'soft' link that just forwards requests, or be a direct copy (this also depends on permission settings). Replicating resources is one of the strengths in P2P networking.

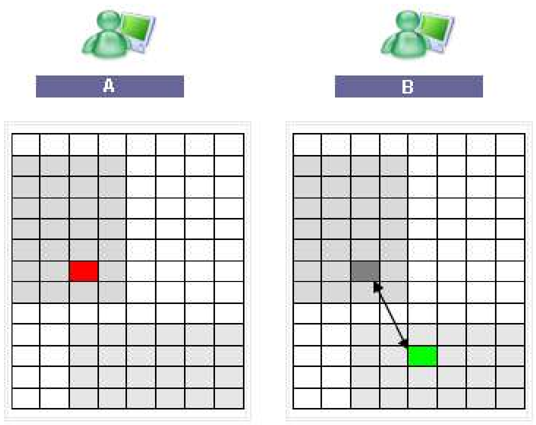
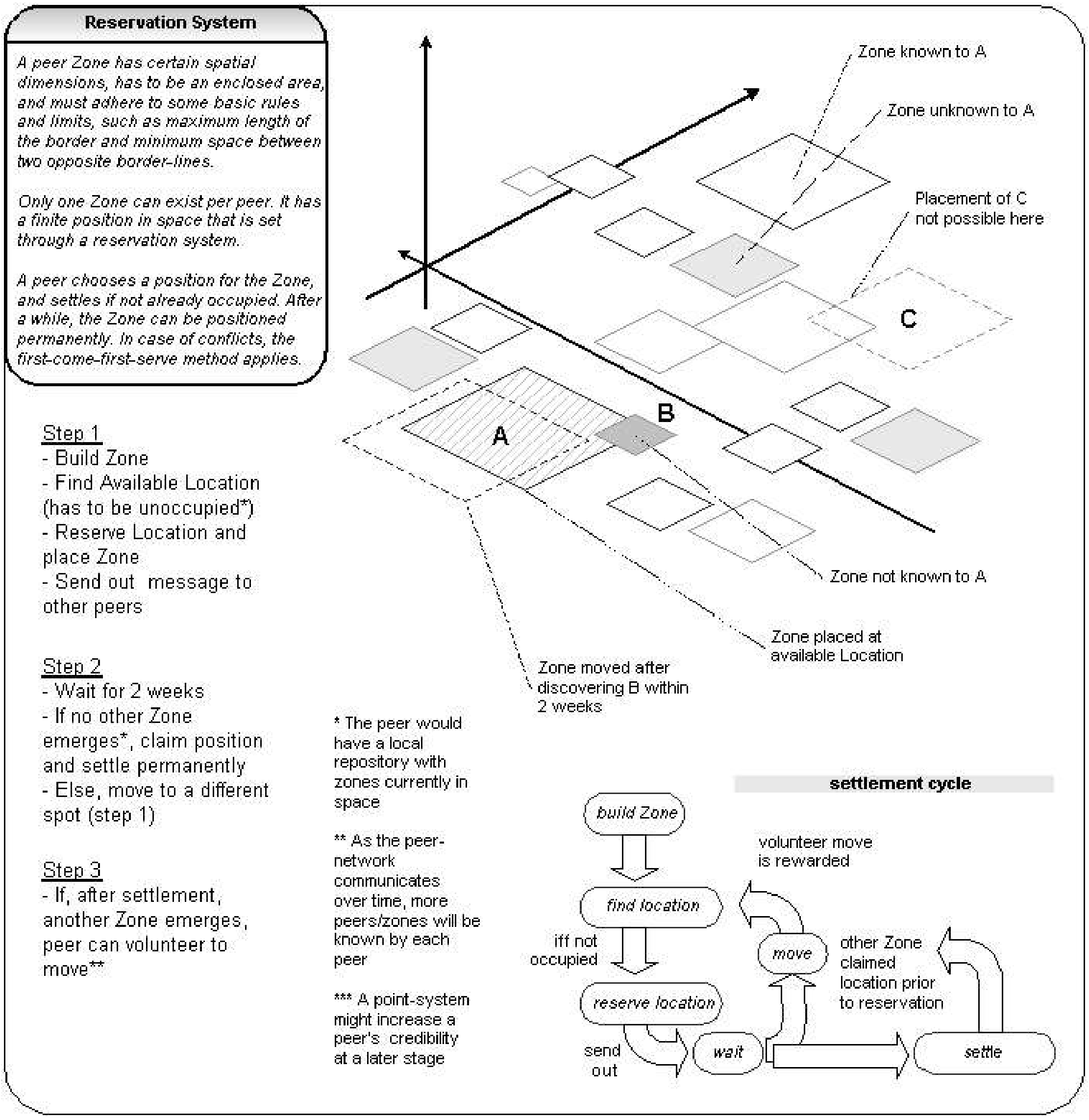


Illustration 4.18. Zone Service Linking

4.4.4.Reservation System

A possible solution for a reservation mechanism to avoid zone overlaps in the VE is shown on Illustration 4.19. Reservation System.



Reservation System

A peer Zone has certain spatial dimensions, has to be an enclosed area, and must adhere to some basic rules and limits, such as maximum length of the border and minimum space between two opposite border-lines.

Only one Zone can exist per peer. It has a finite position in space that is set through a reservation system.

A peer chooses a position for the Zone, and settles if not already occupied. After a while, the Zone can be positioned permanently. In case of conflicts, the first-come-first-serve method applies.

Step 1
- Build Zone
- Find Available Location (has to be unoccupied*)
- Reserve Location and place Zone
- Send out message to other peers

Step 2
- Wait for 2 weeks
- If no other Zone emerges*, claim position and settle permanently
- Else, move to a different spot (step 1)

Step 3
- If, after settlement, another Zone emerges, peer can volunteer to move**

* The peer would have a local repository with zones currently in space

** As the peer-network communicates over time, more peers/zones will be known by each peer

*** A point-system might increase a peer's credibility at a later stage

Zone known to A
Zone unknown to A
Placement of C not possible here
C
A
B
Zone not known to A
Zone placed at available Location
Zone moved after discovering B within 2 weeks
settlement cycle
build Zone
find location
iff not occupied
reserve location
send out
wait
move
volunteer move is rewarded
other Zone claimed location prior to reservation
settle

Illustration 4.19. Reservation System

4.5.Open feedback-based trust model

This section proposes a distributed trust-model service that might be included in future implementations.

4.5.1.Trust Model

This model is network-based, where a peer is an application able to connect to other peers, and let user's create categorized subjects and submit feedback on them. A peer represents that subject specifically or commonly. A feedback is a value in a given range indicating a user's personal view on the particular subject. Comments could be added to feedback-values.

A basic set of rules for this feedback-network could be:

- Each peer is unique (ID)
- Votes and rates are cast using authentication, integrity, and confidentiality
- Authentication mechanisms are person-to-person based
- Integrity and confidentiality is ensured using public key encryption
- One peer can rate and/or vote another peer
- Rates and votes may be authenticated or anonymous
- The authenticity of a peer is public
- A peer's rating/votes may be displayed public, in limited groups, or kept private
- A rate can be a value between 0 and 100 on any rate-subject
- A vote is always the value 1
- Rate-subjects and vote-subjects are categorized
- One peer can only rate another peer once on a rate-subject
- One peer can vote another peer once on a vote-subject
- Rate- and/or vote-groups can be created
- Groups are categorized, and include two or more subjects
- Group ratings and votes are balanced in that the total amount of rates/votes has to be distributed among the group subjects
- Rates and votes may be updated at any time
- Automatic updating mechanisms may be in place (if-then)
- Update feedback-loops may be constructed between peers (if peer A votes X, then vote Y on B)
- The source (peer) of a rate/vote may be tracked
- Auto-updating mechanisms of a rate/vote may be tracked

Examples of Peer-based Feedback follow in the next few paragraphs.

4.5.2.Rating

Illustration 4.20. Trust Feedback System : Peers A, B, and C submit their rating on a Subject S1 on peer D. The average rating of all values (sum divided by count) is displayed at D. Since ratings are between 0 and 100, the average value will be in the same span. Subject S1 could be "Delivery" in the category "Business / Customers / Experience", and it's value described as "% Satisfaction".

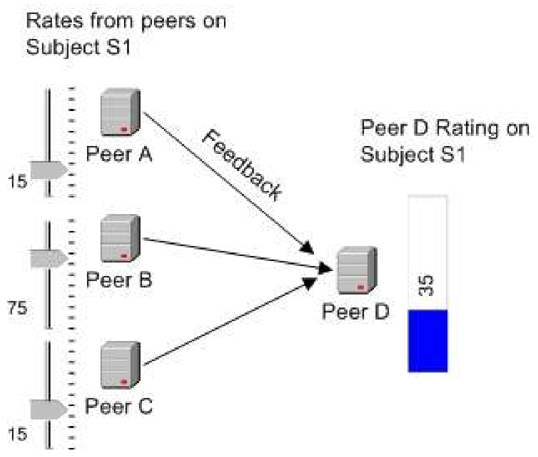


Illustration 4.20. Trust Feedback System

4.5.3.Votes

Similar to rating, votes can be counted: Illustration 4.21. Trusting Votes.

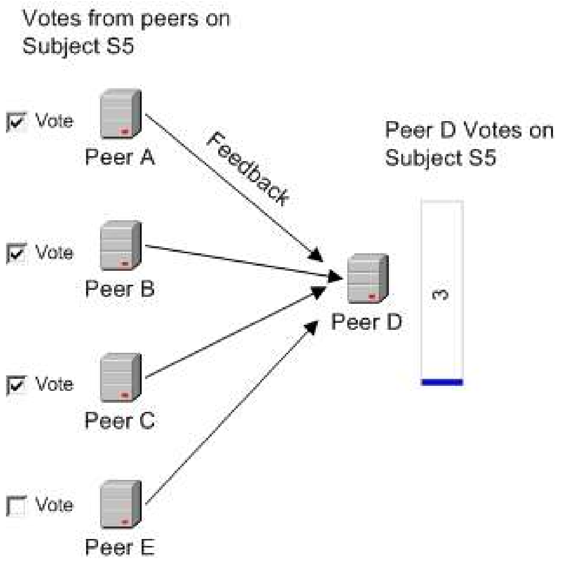


Illustration 4.21. Trusting Votes

4.5.4.Private Feedback

Illustration 4.22. Trust Rating: A scenario where a peer E views at peer D's rating on a particular range of subjects. Since D keeps S1 private, E can view S2, S3, and S4.

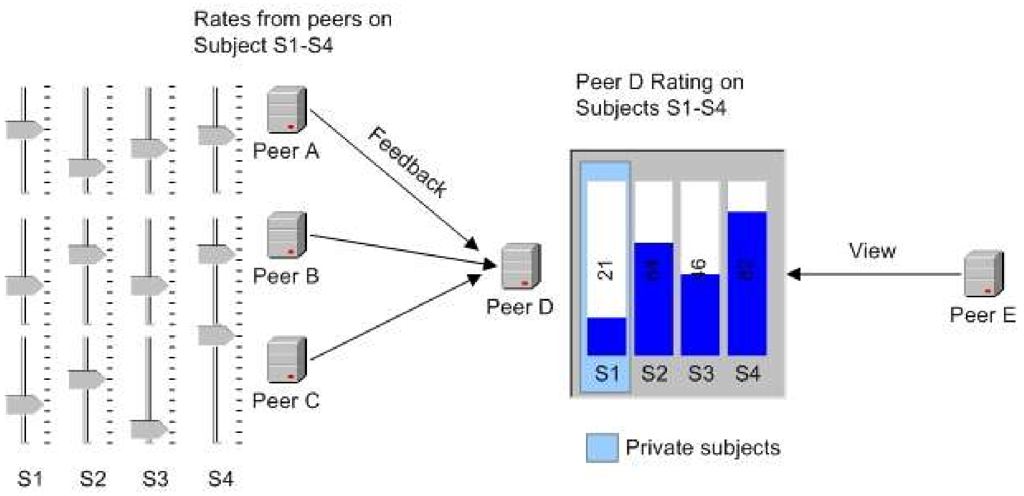


Illustration 4.22. Trust Rating

4.5.5.Automated Feedback

Illustration 4.23. Automated Feedback: This drawing illustrates an optional auto-feedback mechanism, that lets a peer B rate a peer D based on peer A's rating. Before submitting it's rating on subject S4, peer B first inquires A's rating on the same subject. Based on this value and the relationship to A, peer B can adjust it's rating on D.

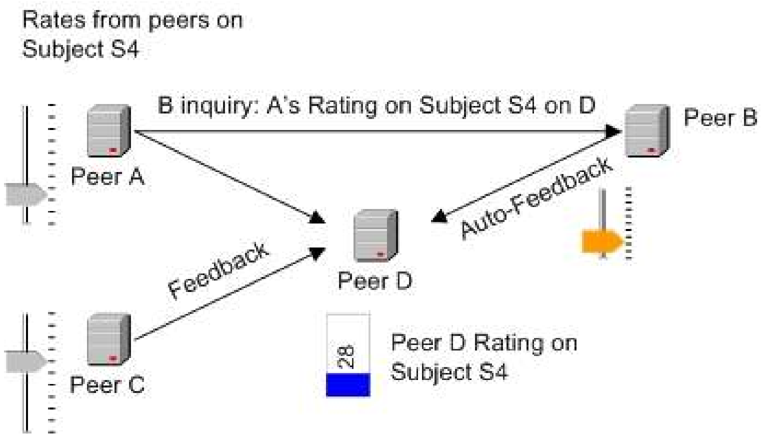

Rates from peers on Subject S4
B inquiry: A's Rating on Subject S4 on D
Peer A
Peer B
Auto-Feedback
Feedback
Peer D
Peer C
28
Peer D Rating on Subject S4

Illustration 4.23. Automated Feedback

Subject S4 could in this situation be “Support of War in Iraq” in the category “Politics / Global Diplomacy / Current Events”, where D could be a public discussion group, and A a political representative (or an index of the Oil-price).

4.5.6.Auto-Feedback Formulas

Illustration 4.24. Trust System Formula: A closer look at the automated feedback-mechanism. Peer D receives feedback from different peers (1) on subject S1. Internally, it can distinguish the feedback (2) by differentiating between anonymous and authenticated feedback, feedback that was submitted during a specific period of time, or feedback that was auto-created.

Peer D could ask other peers, such as peer E, to give their feedback on subject S1 (3). Or peer E itself is interested in submitting feedback to peer D. If subject S1 is of a certain type, it might be useful for E to automate it's feedback on that particular subject, no matter which peer is asking. Peer E thus reads the existing feedback (4) from peer D, and possibly gathers information from interrelated peers (not shown).

Peer E could create an auto-feedback formula (5) that determines it's rating on subject S1 by some factors. In the case below, E would not rate at all on S1 for D if the total feedback D received (yellow) was from less than 30 peers. If the last 10 feedback's were within 4 weeks (red), E would notify it's user, maybe because the high activity requires special attention. Otherwise, E's rate on S1 (orange) would depend on the sum of those peers that rated with authentication (green) and those peers that E in particular trusts (pink). Excluding those feedbacks that themselves are based on formulas (light blue), might provide a better result. In other words, peer E bases it's rate on it's trusted peers and all those not anonymous, but none that use auto-feedback.

4.5.7.Personal Information & Trust System

As part of the application, the Personal Information & Trust System could manage and store user information, terms, policies, certificates, and feedback. This sub-system could create a trusted environment where peer users would be able to send feedback or recommendations to each other.

By applying JXTA's protocol security, a secure channel can easily be established. Peers can allow feedback and recommendation to their services. Service users can send their feedback (ratings). Each peer could then calculate the level of trust and risk based on recommendations of other trusted peers. If this trust system is provided through a friendly user interface, a different kind of culture might emerge.

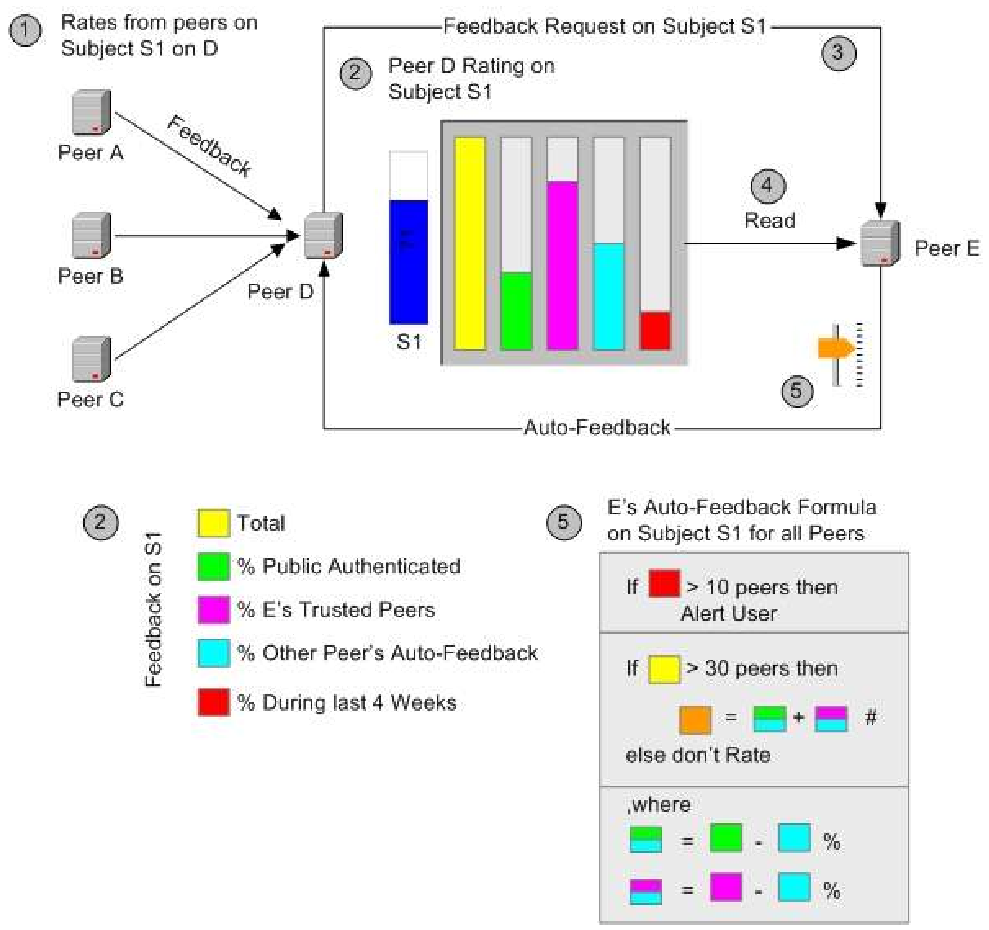

1 Rates from peers on Subject S1 on D
Peer A
Peer B
Peer C
Feedback
Peer D
Feedback Request on Subject S1
2 Peer D Rating on Subject S1
S1
3
4 Read
Peer E
5
Auto-Feedback
2 Feedback on S1
Total
% Public Authenticated
% E's Trusted Peers
% Other Peer's Auto-Feedback
% During last 4 Weeks
5 E's Auto-Feedback Formula on Subject S1 for all Peers
If > 10 peers then Alert User
If > 30 peers then
= + #
else don't Rate
,where
= - %
= - %

Illustration 4.24. Trust System Formula

Subject S1 could in this instance be “Network Uptime” in the category “Sun / Clusters”, i.e. monitor E supports whatever other clusters and external customers think of cluster D, as long as their rating is not automated. Or maybe “Food quality on Campus”, i.e. if you think the food is good and a lot of people put their name under the same statement, I agree, too.

5.Implementation

The TerraPeer architecture dictates a rich pandect of code, but the intention of the application is to demonstrate the viability of certain features, including P2P networking, discovery, peer virtualization, zoning and abstract services, access control, and a 3D interface with navigation controls.

In addition to descriptions for each part of the assembly (and package) of the code-base, there are code-snipplets and UML graphs in this chapter to show the implementation.

Implementation code, class-names, or -packages are marked with `courier font`.

5.1.Use Cases

A few hypothetical use-cases can illustrate how DVE applications could be applied in best-case scenarios.

5.1.1.A Collaboration Space

The establishment of a usable collaborative space is a well-known problem that CSCW in particular focuses on. Collaboration between users is an essential paradigm of networked computer systems, and has many applicable areas: Learning, modeling and engineering of buildings (CAD) or urban planning, project management, software development, art, etc.

The DVE offers a space in three dimensions that often is useful to project structures from reality that otherwise would be too complex to present. VR has been used to model city traffic and chemical molecules.

TerraPeer by it's distributed nature offers the ability to collaborate by sharing objects and services (virtual or representations), and by enabling communication while collectively gather around certain settings using avatars supported by voice- or text-base messenging.

Each desk in Bob's company has it's own virtual zone, and are all connected to the corporate intranet. Bob is looking at a representation of a building he is working on at his office. Through his personal contact list, he contacts Alice, who jumps to his position. Together, they inspect various details of the building by following each others avatars. Using the zone's website service, it's file sharing service, and a simple text-based chat service, they are able to share resources and communicate.

5.1.2.Browsing the Virtual Streets of Cyberspace

A different scenario is social interaction that could occur similar to the virtual worlds, or online games that already exist. A large number of users being 'immersed' in a single world, as in other common spaces, share their environment and communities could emerge. A space that allows building personal sites can foster individual proliferation, but sharing that space with other users and allowing groups also promotes collective entities. Being able to link peers, services, and objects further supports this.

The phenomena of virtual worlds where social interaction is combined with building personalized areas or creating artificial personalities has been widely successful (see Appendices for more resources).

TerraPeer could theoretically support the same kind of environment, as it enables a society based on all connected peers. the creation and sharing of virtual objects, and communication between peer users. The question whether this world could become a vigorous place due to the dynamics of P2P networks remains open, but it should be noted that a peer does not need to act as a hopping client per se – permanent server peers could well be created to establish continuously settled areas.

5.1.3.Enhancing the Interface of Services

Using a virtual service sounds ambitious, but should be seen as a virtualization mechanism to enhance the user experience. In the desktop metaphor of most operating systems today, the broadly used icon that launches an application or connects to a service is nothing more than a technique to ease usability.

In 3D worlds, other metaphors are being constructed. A balance between using familiar real-life objects and not obstructing the users in their tasks has to be found.

TerraPeer might be used to serve as a common space where each user peer can create a virtual platform that serves as a hub for resources and services. Abstracting a service makes sense, as it in terms of networked computer systems really can represent a trumendous range of content and functionality.

A simple use-case would be the user Alice, who would put a virtual service object on her peer zone. The object looks like a phone. It has attributed that allow Alice to determine particular groups of users to activate it. At home at Bob's zone there is a collection of services in a box, which is strictly private and cannot be activated by others. When Bob clicks the box, several virtual representations of files and links pop up above it, and he selects one saying 'Alice Home'.

After instantly jumping to Alice' zone, he finds her phone object and gives her a call by clicking on it, which activates a VoIP service. Later, Bob creates a link to the phone for his own box.

5.1.4.The Ultimate Virtual Reality

Roaming alongside some of the more permanent zones around the center of the space, Alice notices a virtual replicate of a bookstore. Entering the zone provides a collection of stacked books where each stack has a topic. After browsing through several topics, Alice stands in a closed virtual room with selectable bookshelves that enable her to quickly read abstracts, or open a browser to get more information. Next to her is Bob. He's reading a book and Alice asks him what he's looking at by clicking his avatar and speaking into her microphone.

After collecting references about this book from one of her bookmarked opinion-services, Alice decides to purchase it by clicking the price-tag placed on the books cover. Confirming to her virtual credit card that she wants to transfer an amount to the bookstore, she waves goodbye to Bob, and jumps to the Google News zone.

5.2.Class Structure

The structure and packages of TerraPeer is shown in Illustration 5.1. TerraPeer Class Package Overview (UML) that is based on a UML object model.

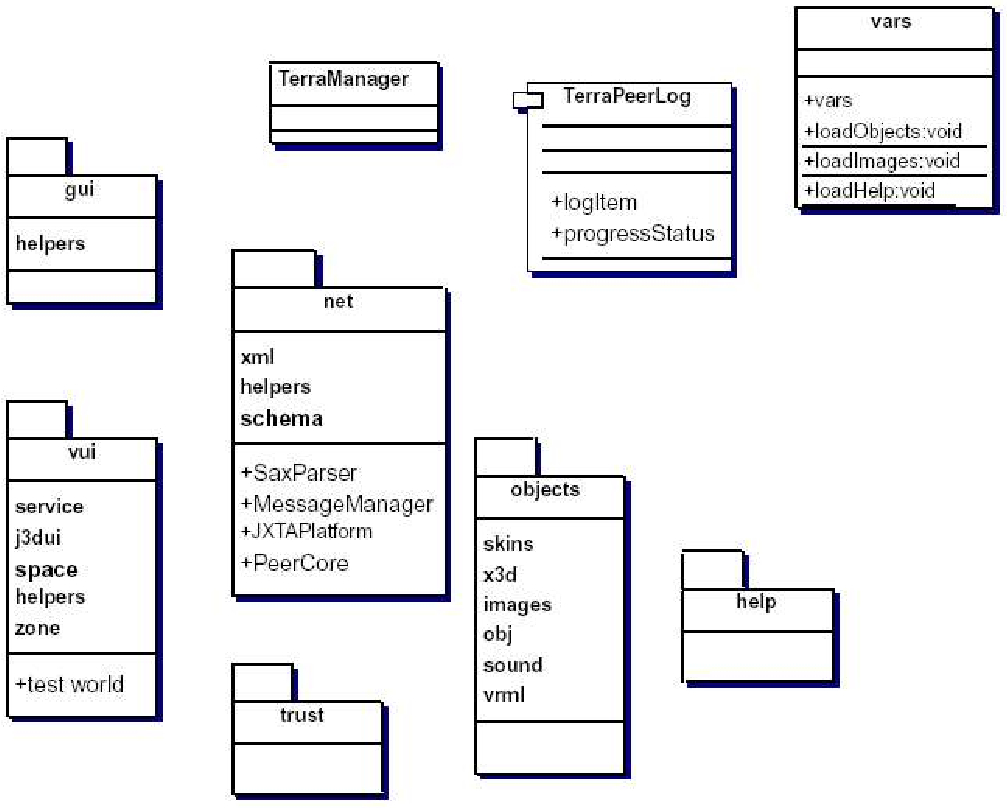


Illustration 5.1. TerraPeer Class Package Overview (UML)

The main functionality and properties of classes in each of these packages have already been anticipated in the previous chapter, and will be explained in more details in the next sections.

5.3.User Interface

5.3.1.Application Functionality and Interaction

The GUI layout tries to accommodate controls, information feedback, and viewers in a comprehensive structure. The application features are presented in a logical, functional, and convenient way that also underlines the flow of events. Knowledge about HCI design guidelines [SIGCHI and HCI], as examined in the previous chapter, dictate several usability techniques.

Due to the broad functionality available to the user, the layout has to be created in a way that allows some kind of 'drilling' technique to avoid showing all controls, views and labels at once and clutter the display.

Controls change depending on the current focus of the user, i.e. user attention and display interface are aligned to optimize usability. That is, if a user is currently navigating through the virtual space, a navigational panel with control buttons and status feedback is made available. And if a user is currently building a virtual service inside his home zone, a panel with dragable building blocks is in the visibility center.

Hence, the GUI layout has been designed to accommodate...

1. Always-available controls, menus, and status
2. Dynamic controls and information feedback, depending on current user focus
3. Ability to show or hide all panels, and resize certain content-containers (personalization)
4. Easy to learn GUI layout structure

The derived GUI layout is depicted below (Illustration 5.2. Original TerraPeer GUI Layout).

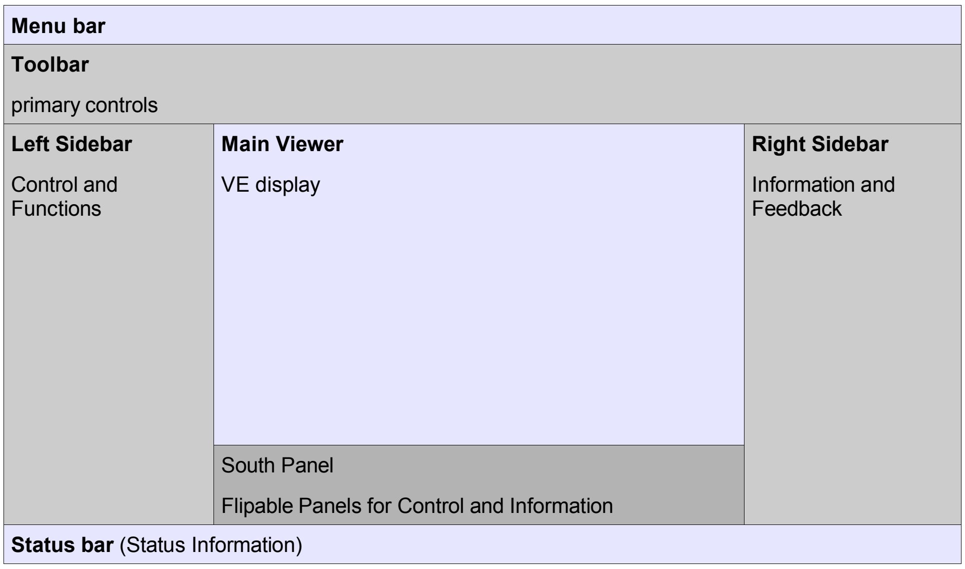


Illustration 5.2. Original TerraPeer GUI Layout

The menu and toolbar on top provide always-available controls to manage network connection or space environment, to edit preferences, or to control the repository (there is room for future features and services, such as security and trust management).

The left and right sidebars respectively are collections of control and feedback panels that can be expanded or collapsed depending on the current activity. The controls, for instance, are divided into headlines (network control, space management, personal information, etc.) that when clicked upon display action-buttons related to the topic.

See also Illustration 5.3. Screenshot - TerraPeer Main Window to see the latest version layout.

Similarly, the so-called 'south panel' that is placed below the main viewer, shows interaction panels depending on activity. Together, these panels provide all functionality and information needed to use all features of the application.

5.3.2.Application GUI

The main GUI class is `TerraPeerGUI`, which extends `JFrame`, and is called originally by the TerraPeer Manager when launched. This frame displays all essential functionality and menus, as well as integrated viewers (see Illustration 5.3. Screenshot - TerraPeer Main Window). The GUI layout was designed to be both flexible and easy to overview. Though, personalization functionality was not implemented in this version, it is possible to blend in or hide many parts of the GUI.

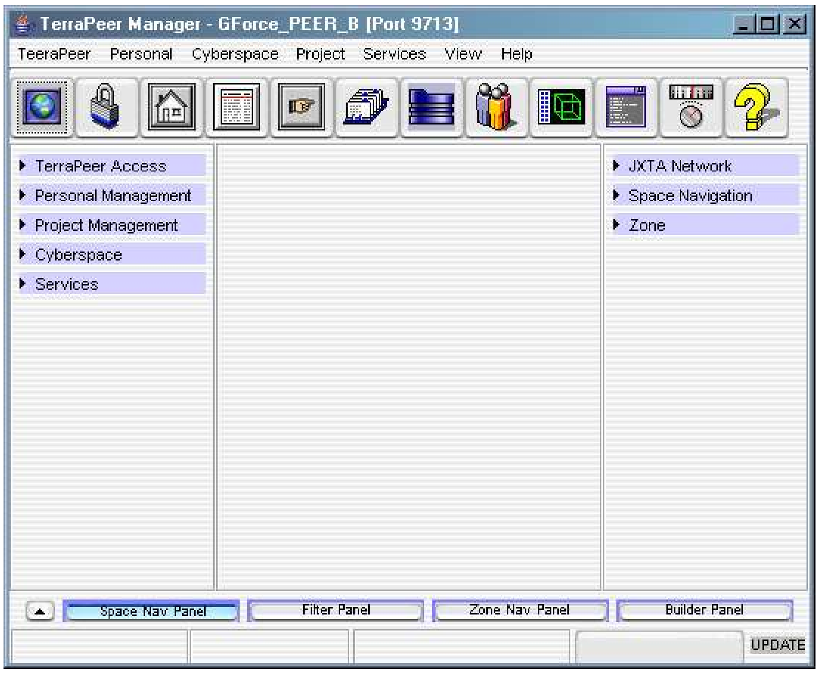


Illustration 5.3. Screenshot - TerraPeer Main Window

Several flags are typically set to indicate certain states of the application, such as whether the peer is connected to network:

```
static public PeerCore TERRA_NetPeer = null;
static boolean peerIsConfigured = false;
static boolean connectionOK = false;
static boolean currentViewPointPersonal = true;
static boolean currentViewPointGrid = true;
static boolean zonegridAutomoveOn = false;
static boolean isCyberspaceRunning = false;
```

The main GUI class encapsulates the `terrapeer.vui.space.MySpace`, and the `jPanelSpaceNavAvatView` objects. The former could be seen as the data model of the VE, while the latter corresponds to the data view.

Secondary initialization procedures of the GUI set space navigation panels:

```
private void initSecondary()
{
  jPanelSpaceNavCompass.setPreferredSize(new Dimension(120, 140));
  jPanelSpaceNavAltitude.initAltitude();
  jPanelSpaceNavHeading.initHeading();
  jPanelSpaceNavCompass.initCompass();
  jPanelSpaceNavCtrl_Dir.initControl();
  initSideThread();

  //...
```

Methods are named according to their function. A `startX` method, where `X` stands for a name, usually starts a particular service, and a `showX` method usually opens a dialog or external frame window. Examples of the latter are `showLog()`, `showLogin()`, `showHelp()`.

Related to the network are `startPeer()`, `connectJXTA()`, `disconnectJXTA()`, and `startDiscovery()` methods. The virtual environment methods include startCyberspace(), `startRepository()`, `zoneAddBuildingBlock()`, `manualSpaceControl()`, and `moveZoneGrid()`. The names should suggest their purpose, but the implementation should describe them in further details (as the produced JavaDoc will show).

A side-thread running parallel to the main application to update various information in real-time:

```
        class TerraSideThread extends Thread
```

,which mainly calls `updateStatus()`. On each update cycle, relevant information is displayed on the status-bar, or the space map is updated, for instance.

Much of the TerraPeerGUI contains interface-related Java Swing code. There are many methods to show and flip panels or components, and widget action event listeners and handlers.

5.3.3.3D Viewer, Widgets, Menus and Toolbars

As other special GUI widgets, `ZoneGrid` extends a Java Swing component and overwrites the `paintComponent()` method in order to customize the visual interface. This class was created to visualize the VE grid, and known peer zones in a 2D panel.

Good data visualization does not necessarily imply more dimensions, or complex graphics. On a contrary, the more simple the data can be represented and conveyed to the user, the better. In case of the grid, presenting a 2D map of the environment is an obvious advantage. The UI should also be able to offer certain functionality, i.e. by clicking on the individual zones to obtain more detailed information.

Other special widgets to enhance navigation are `Compass`, `Altitude`, and `Heading`.

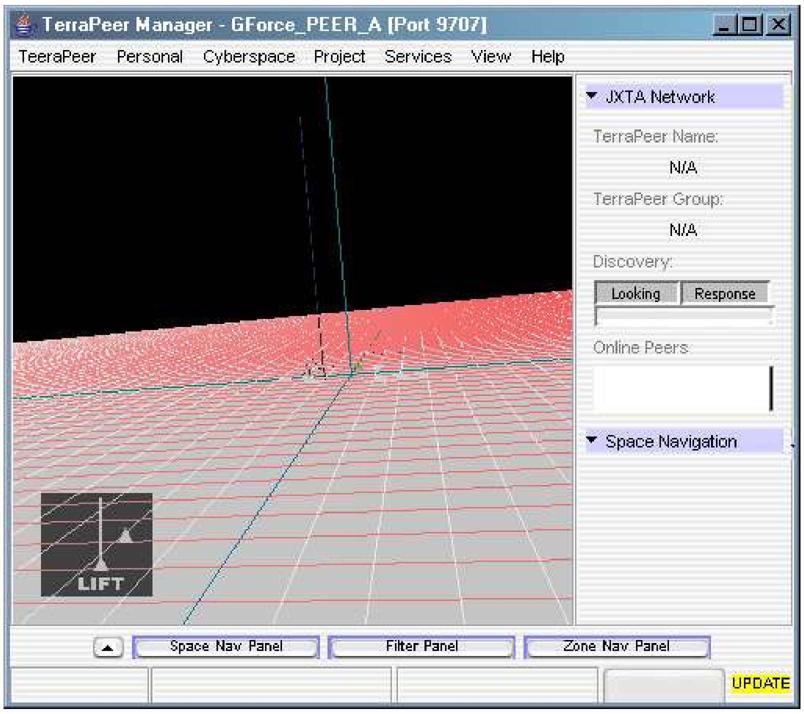


Illustration 5.4. Screenshot - TerraPeer 3D World

The 3D viewer of the VE is the center panel, which is resized alongside the application window. Illustration 5.4. Screenshot - TerraPeer 3D World shows the GUI.

5.3.4.Status, Feedback, and the South Panel

Status messages are feedback from the application to the user that are of shorter length (e.g. notes), and are usually given through the status-bar at the bottom of the application GUI. This layout has become a de-facto standard.

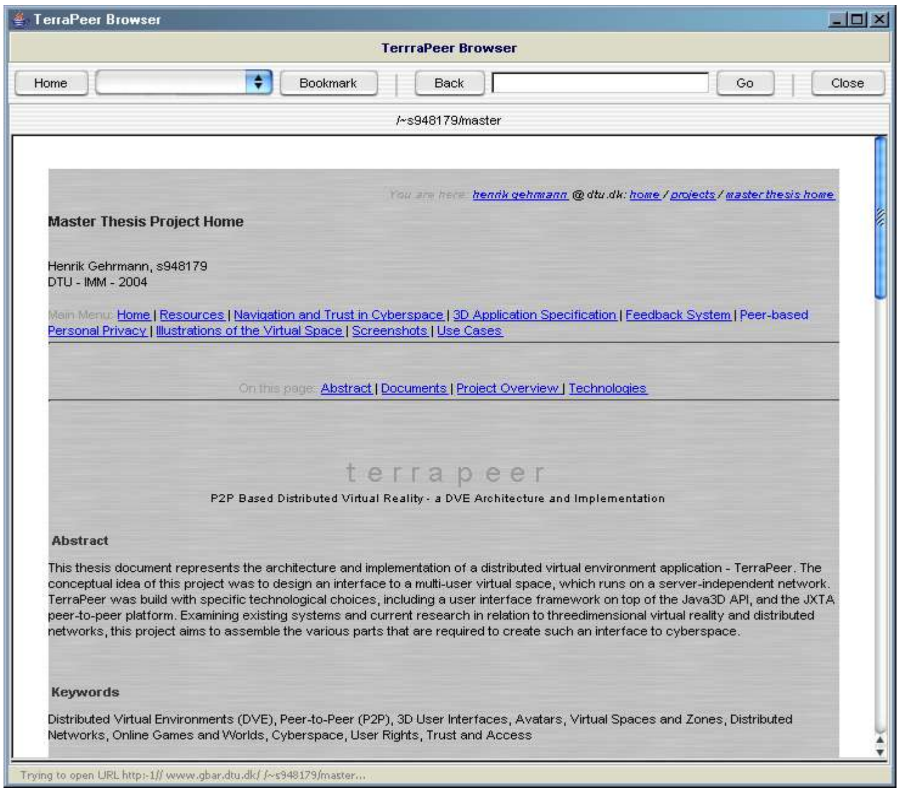
Other feedback, mostly of 'heavier' content, such as zone-related information or peer network status, requires more display real-estate. As feedback changes depending on the current focus of the user, similar to the controls mentioned earlier, the layout has been designed to reflect that of the control panels.

Three major displays are provided: JXTA Network, Space Navigation, and Zones. As their titles indicate, they contain all information related to the specific focus. The network panel would show information such as the currently connected peers, the space panel information about current position coordinates and mapping, and the zone panel information related to currently selected zones.

Navigation, 2D map, and Zone building panels are available on the 'south panel'. Later additions could include panels to manage privacy and feedback, or personal information.

5.3.5.Web Browser Window

The browser window (see Illustration 5.5. TerraPeer Web Browser) implements a simple HTML v.4 compatible web-viewer. The main purpose is to provide an internal client that can be called directly from the VE when an appropriate service is invoked. Additional features include usual URL navigation and bookmarking functionality.



/~s948179/master

You are here: henrik gehmann @ dtu.dk: home / projects / master thesis home

Master Thesis Project Home

Henrik Gehrmann, s948179
DTU - IMM - 2004

Main Menu: Home | Resources | Navigation and Trust in Cyberspace | 3D Application Specification | Feedback System | Peer-based Personal Privacy | Illustrations of the Virtual Space | Screenshots | Use Cases

On this page: Abstract | Documents | Project Overview | Technologies

terrapeer
P2P Based Distributed Virtual Reality - a DVE Architecture and Implementation

Abstract

This thesis document represents the architecture and implementation of a distributed virtual environment application - TerraPeer. The conceptual idea of this project was to design an interface to a multi-user virtual space, which runs on a server-independent network. TerraPeer was build with specific technological choices, including a user interface framework on top of the Java3D API, and the JXTA peer-to-peer platform. Examining existing systems and current research in relation to threedimensional virtual reality and distributed networks, this project aims to assemble the various parts that are required to create such an interface to cyberspace.

Keywords

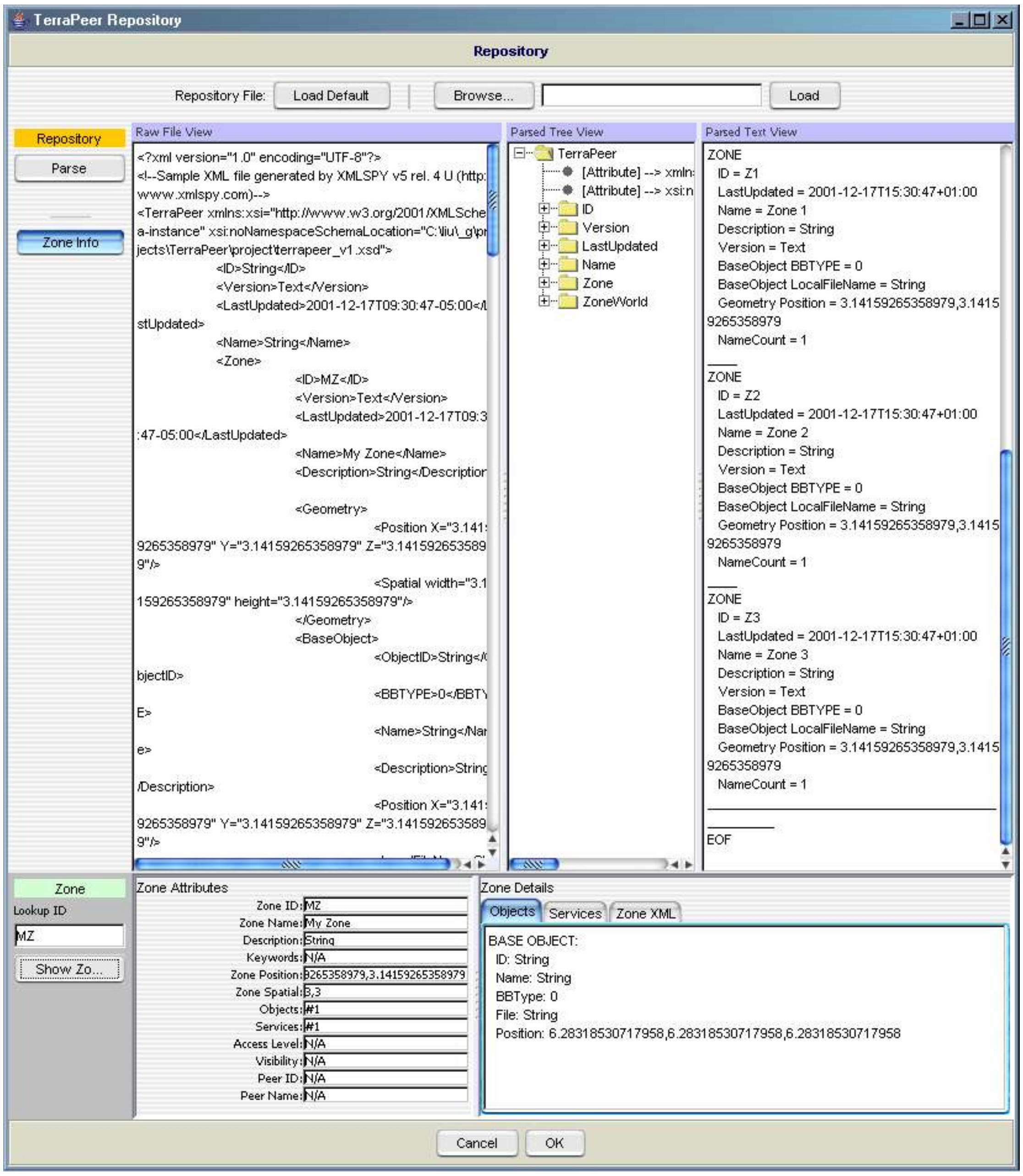
Distributed Virtual Environments (DVE), Peer-to-Peer (P2P), 3D User Interfaces, Avatars, Virtual Spaces and Zones, Distributed Networks, Online Games and Worlds, Cyberspace, User Rights, Trust and Access

Trying to open URL http:-1// www.gbar.dtu.dk/ /~s948179/master...

Illustration 5.5. TerraPeer Web Browser

5.3.6.Repository and Log Viewer

A screenshot of the repository viewer is shown below; Illustration 5.6. Screenshot - TerraPeer Repository Window. This viewer can read the TerraPeer XML protocol. It displays zone-related attributes and properties, and creates a DOM-like tree structure. The purpose of this tool is purely informational, but could be extended to provide other related functionality to manage the repository, edit zones, or services.



```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSPY v5 rel. 4 U (http:
www.xmlspy.com)-->
<TerraPeer xmlns:xsi="http://www.w3.org/2001/XMLSche
a-instance" xsi:noNamespaceSchemaLocation="C:\liu\_g\pr
jects\TerraPeer\project\terrapeer_v1.xsd">
            <ID>String</ID>
            <Version>Text</Version>
            <LastUpdated>2001-12-17T09:30:47-05:00</L
stUpdated>
            <Name>String</Name>
            <Zone>
                        <ID>MZ</ID>
                        <Version>Text</Version>
                        <LastUpdated>2001-12-17T09:3
:47-05:00</LastUpdated>
                        <Name>My Zone</Name>
                        <Description>String</Descriptior

                        <Geometry>
                                    <Position X="3.141
9265358979" Y="3.14159265358979" Z="3.141592653589
9"/>
                                    <Spatial width="3.1
159265358979" height="3.14159265358979"/>
                        </Geometry>
                        <BaseObject>
                                    <ObjectID>String</
bjectID>
                                    <BBTYPE>0</BBTY
E>
                                    <Name>String</Nar
e>
                                    <Description>String
/Description>
                                    <Position X="3.141
9265358979" Y="3.14159265358979" Z="3.141592653589
9"/>
```

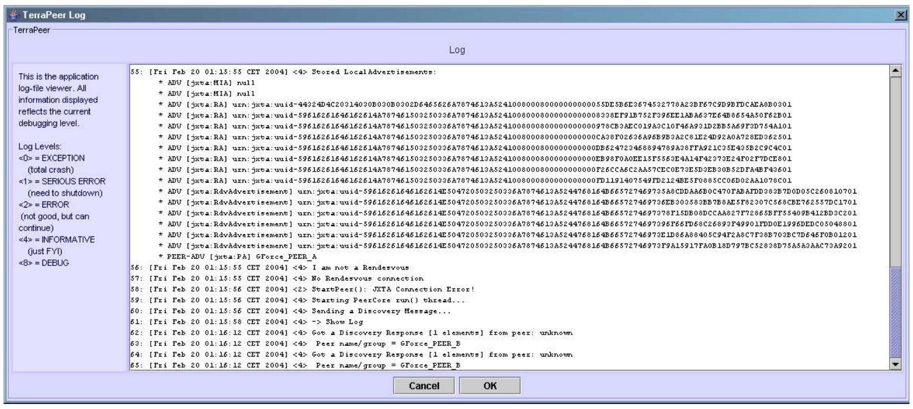
```
ZONE
  ID = Z1
  LastUpdated = 2001-12-17T15:30:47+01:00
  Name = Zone 1
  Description = String
  Version = Text
  BaseObject BBTYPE = 0
  BaseObject LocalFileName = String
  Geometry Position = 3.14159265358979,3.1415
9265358979
  NameCount = 1

____
ZONE
  ID = Z2
  LastUpdated = 2001-12-17T15:30:47+01:00
  Name = Zone 2
  Description = String
  Version = Text
  BaseObject BBTYPE = 0
  BaseObject LocalFileName = String
  Geometry Position = 3.14159265358979,3.1415
9265358979
  NameCount = 1

____
ZONE
  ID = Z3
  LastUpdated = 2001-12-17T15:30:47+01:00
  Name = Zone 3
  Description = String
  Version = Text
  BaseObject BBTYPE = 0
  BaseObject LocalFileName = String
  Geometry Position = 3.14159265358979,3.1415
9265358979
  NameCount = 1

______________________________________________
__________
EOF
```

Lookup ID: MZ

Zone ID:	MZ
Zone Name:	My Zone
Description:	String
Keywords:	N/A
Zone Position:	9265358979,3.14159265358979
Zone Spatial:	3,3
Objects:	#1
Services:	#1
Access Level:	N/A
Visibility:	N/A
Peer ID:	N/A
Peer Name:	N/A

```
BASE OBJECT:
  ID: String
  Name: String
  BBType: 0
  File: String
  Position: 6.28318530717958,6.28318530717958,6.28318530717958
```

Illustration 5.6. Screenshot - TerraPeer Repository Window



This is the application log-file viewer. All information displayed reflects the current debugging level.

Log Levels:
<0> = EXCEPTION (total crash)
<1> = SERIOUS ERROR (need to shutdown)
<2> = ERROR (not good, but can continue)
<4> = INFORMATIVE (just FYI)
<8> = DEBUG

```
55: [Fri Feb 20 01:15:55 CET 2004] <4> Stored LocalAdvertisements:
      * ADV [jxta:MIA] null
      * ADV [jxta:MIA] null
      * ADV [jxta:RA] urn:jxta:uuid-44324D4C20314030B030B0302D6465626A7874613A524100800080000000000000055DE5B6E3674532778A23BF67C9D9BFDCAEA8B0301
      * ADV [jxta:RA] urn:jxta:uuid-59616261646162614A787461503250336A7874613A52410080008000000000000008338EF91B752F396EE1ABA637E64B8654A50F62B01
      * ADV [jxta:RA] urn:jxta:uuid-59616261646162614A787461503250336A7874613A5241008000800000000000000978CB3AEC019A3C10F46A931D2BB5A69F3D754A101
      * ADV [jxta:RA] urn:jxta:uuid-59616261646162614A787461503250336A7874613A5241008000800000000000000CA38F02636A96B9B3A2C81E24D92A0A728ED362501
      * ADV [jxta:RA] urn:jxta:uuid-59616261646162614A787461503250336A7874613A5241008000800000000000000DB62472346889478 9A38FFA921C35E435B2C9C4C01
      * ADV [jxta:RA] urn:jxta:uuid-59616261646162614A787461503250336A7874613A5241008000800000000000000EB98F0A0EE15F5563E4A14F42373E24F02F7DCE801
      * ADV [jxta:RA] urn:jxta:uuid-59616261646162614A787461503250336A7874613A5241008000800000000000000F26CCA6C2AA57CEC0E73E5D3EB30B52DFA4BF43601
      * ADV [jxta:RA] urn:jxta:uuid-59616261646162614A787461503250336A7874613A5241008000800000000000000FD1191407549FD2124BE5F0885CC06D02AA1078C01
      * ADV [jxta:RdvAdvertisement] urn:jxta:uuid-59616261646162614E504720503250336A7874613A5244768164B665727469735A8CDDAA6B0C470FABAFDD383B7D0D05C260810701
      * ADV [jxta:RdvAdvertisement] urn:jxta:uuid-59616261646162614E504720503250336A7874613A5244768164B665727469736EB303583BB7B8AE5F82307C568CBE762557DC1701
      * ADV [jxta:RdvAdvertisement] urn:jxta:uuid-59616261646162614E504720503250336A7874613A5244768164B66572746973 78F15DB08DCCAA827F72865BFF55409B412BD3C201
      * ADV [jxta:RdvAdvertisement] urn:jxta:uuid-59616261646162614E504720503250336A7874613A5244768164B665727469739 6F66FD68C26893F49901FDD0E1996DEDC05048801
      * ADV [jxta:RdvAdvertisement] urn:jxta:uuid-59616261646162614E504720503250336A7874613A5244768164B66572746973E1D86A88405C94F2A8C7F38B703BC7D646F0B01201
      * ADV [jxta:RdvAdvertisement] urn:jxta:uuid-59616261646162614E504720503250336A7874613A5244768164B66572746973F9A15917FA0B18D797BC52838D75A5A3AAC73A9201
      * PEER-ADV [jxta:PA] GForce_PEER_A
56: [Fri Feb 20 01:15:55 CET 2004] <4> I am not a Rendesvous
57: [Fri Feb 20 01:15:55 CET 2004] <4> No Rendesvous connection
58: [Fri Feb 20 01:15:56 CET 2004] <2> StartPeer(): JXTA Connection Error!
59: [Fri Feb 20 01:15:56 CET 2004] <4> Starting PeerCore run() thread...
60: [Fri Feb 20 01:15:56 CET 2004] <4> Sending a Discovery Message...
61: [Fri Feb 20 01:15:58 CET 2004] <4> -> Show Log
62: [Fri Feb 20 01:16:12 CET 2004] <4> Got a Discovery Response [1 elements] from peer: unknown
63: [Fri Feb 20 01:16:12 CET 2004] <4>  Peer name/group = GForce_PEER_B
64: [Fri Feb 20 01:16:12 CET 2004] <4> Got a Discovery Response [1 elements] from peer: unknown
65: [Fri Feb 20 01:16:12 CET 2004] <4>  Peer name/group = GForce_PEER_B
```

Illustration 5.7. TerraPeer Log

The last screenshot represents the log window, which displays all status feedback from the application; Illustration 5.7. TerraPeer Log:

5.4.3D World Code

5.4.1.Basic Settings

Much of the 3D code implementation uses Java3D API classes in the `javax.media.j3d.*` and `javax.vecmath.*` packages.

The following code is extracted from the `MySpace` class, and shows the objects, properties and methods that are encapsulated.

```
//...

public class MySpace //extends JPanel
{
  private boolean spaceInitialized = false;
  private boolean currentVPP = true;
  private static AppWorld spaceWorld = null;
  private static AppView spaceView_avat = null;
  private static UISceneTree zone_tree = null;
  public static SpaceCore spaceCore = null;
  public static ZoneBuilder zoneBuilder = null;
  public static ZoneWorld myZoneWorld = null;
  public static Zone myZone = null;

//...

  public void initSpace(...)
  {
    spaceWorld = new AppWorld();
    spaceView_grid = new AppView();
    spaceView_avat = new AppView();
    spaceView_orbitAvat = new AppView();
    spaceCore = new SpaceCore(...);
    myZoneWorld = new ZoneWorld();
    zoneBuilder = new ZoneBuilder(...);
    spaceInitialized = true;

//...

  public boolean startSpace()
  {
    if(spaceInitialized)
    {
      setupMyZone();
      setupMyZoneWorld();
      zoneBuilder.buildAllZones();
      spaceCore.spaceNav.setupNavigation(...);
```

```
        spaceCore.spaceEnvio.buildEnvironment();\
        finalizeWorld();

//...
```

The `AppWorld` and `AppView` classes represent the “world space” (extends `VirtualUniverse`) and “view space” (extends `BranchGroup`) of the application. The former contains a single scene graph (`BranchGroup`) and controls the scene attributes when modifications are required, and an ambient light. The latter is a viewer object that is added to the SG in order to provide a view of its virtual world. It also encapsulates internal geometry associated with the view.

5.4.2.View and Navigation

Packaged into `terrapeer.vui.space` are the classes `SpaceEnvio`, `SpaceGrid`, and `SpaceNav`.

`SpaceEnvio` is responsible for the basic environment settings, and it's main method is `buildEnvironment()`.Part of this environment is the `SpaceGrid` class (extends the `BranchGroup` object) that creates an axis and grid system for the VE (to enhance navigation).

The coordinates are based on a standardized metric system, but could theoretically be extended to feature more spatial freedom in certain contained sub-spaces (zones).

`SpaceNav` is responsible for user (avatar) navigation, which requires controlling the 'camera' scene-viewer from the instructions received from the input devices. It's method setupAvatarNav(), among other things, limits the movement of the avatar to the volume over the ground-level through clamping:

```
  avatCamera.getThrustActuator().getPlugin().setTargetClamp(...)
```

The class also implements navigating control methods, such as `moveAvatarForward()`, `stopAvatar()`, and `alignAvatar()`, as well as feedback methods, such as getAvatarHeadingX, where `X` could be `Yaw`, `Roll`, `Pitch`, or `Speed`.

Control of avatar movement, for example thrust, is usually triggered by a call similar to:

```
  avatCamera.getThrustActuator().initActuation(...)
```

The user's view is a virtual camera that moves through the VE. Initial implementation does not provide a visible avatar attached to the view node, but this should be fairly simple to add in later versions (especially user-to-user communication requires this feature). The camera is actuated (translated) through methods that convert user input vectors to 3D space vectors.

This view-component and -control is located in the classes `AvatarCamera` (extending `ActuatorGroup`), and `AvatarCameraControl`.

`AvatarCamera` creates an actuator group chain that serves as a camera simulation with thrust translation and spherical attitude (roll, pitch, yaw). The controller creates draggers, filters, and mappers for the camera. Thrust is directly controlled through an acceleration filter, and attitude-roll, -yaw and -pitch are controlled through a rate filter.

As mentioned in the last chapter, the J3DUI framework offers some convenience methods that in this case supports interaction conventions. For example, dragging the mouse across the view after selecting an object for rotation should result in the rotational translation of the object around its own axis. The conversion from the x-y coordinates of the mouse-pointer to the transformation vector is a common catechism to accommodate intuitive user actions.

An example for intuitive mapping conversion is the class `DrmRotationMapper` (implements `InputDragTarget`), which applies an `InputDragTarget` input and an axis-angle Actuator target.

Here, the target position is relative to the position of the drag on the source display. Mouse movement is mapped on the target object, i.e. a drag to the right would rotate the target to the right. The event chain uses an `IntuitiveDragMapper` connected to a source mapper (with a rotation plugin) that again is connected to the target actuator.

5.4.3.User Interaction

5.4.3.1.Input and Triggers

J3DUI input classes are placed in the `terrapeer.vui.j3dui.control.inputs` package, and provide input sensors (keyboard, mouse), filtering, trigger, and clamping chaining mechanisms. As mentioned earlier, the utility classes are usually utilized through source-target chaining. An input source, such as a drag can thus be chained together with a special filter or function to transform the target according to some design.

One example for an input drag filter plugin that limits or clamps the the drag position value is the `ClampInputDragPlugin` class (extends `InputDragFilterPlugin`). An example for input event classes is the `InputButtonTrigger` (extends `EnableTrigger`, and implements `InputButtonTarget`), which is a trigger that monitors button events.

5.4.3.2.Geometry and Transformation

There exist several `TransformGroupPlugin` classes that implement single-user matrix, single-user vector, and multi-user matrix plug-ins. A transform group actuator plugin that assumes exclusive or non-exclusive control of the target node. The actuation state is maintained by a matrix (`Transform3D`) or vector (`Vector4d`).

`TGGeometryPlugin` of the package `terrapeer.vui.j3dui.control.actuators` is an abstract base class for plug-ins used by `TransformGroupPlugins` to define the transform geometry. It also defines the format of the input actuation value.

Geometry plugins that perform geometric translation or rotation are for instance the `TranslationPlugin`, `ScalePlugin`, `RotationPlugin`, and `AxisAnglePlugin` classes (all extending the `TGGeometryPlugin` class). Translation, in this case, occurs in 3D relative to the target transform along its major axis, while axis-angle-rotation occurs in 3D about an arbitrary axis defined relative to the target transform. `ScalePlugin` and `RotationPlugin` perform geometric scaling and rotation.

5.4.3.3.Feedback

Trigger-based feedback interaction for targets (that are attached) are managed by descendants of the abstract `FeedbackManager` class that is located inside the `terrapeer.vui.j3dui.feedback` package. Feedback types for the managers are 'status', 'select', or 'action' that for instance can be triggered when the mouse hovers, clicks, or drags an object.

5.4.4.Visualization and Visibility

Visualization and visibility are very interesting subjects as they define many attributes of what the user actually will be able to see and perceive.

5.4.4.1.Principles

It is at the designers discretion to implement a view that has certain visual properties enabled, and certain others limited or hidden. As Apple's GUI design principles (section 3.4.4.) implied, the user experience is critical. What information and controls should be visible, where, when, and how?

As much possible, the TerraPeer application was committed to adhere to user interface design principles.

Metaphors have been used to create the space environment, navigation controls, visualize service objects, and icons.

See-and-point functionality, as well as direct manipulation has been integrated in the 3D space, where objects can be picked up and dragged, the zone map, and navigation controls.

User control is enhanced through the GUI layout, which allows focusing on specific tools, and 'drilling' of controls and information. Similarly the feedback and communication flow has been implemented to correspond to the current focus of the user rather than that of the application.

Consistency is preserved throughout the design, which can be observed in the windows and dialogs, font types, icons, frame and panel sizes and alignments, topic selections in menus and toolbars, visibility options and personalization.

The WYSIWYG (What You See Is What You Get) principle has been applied to some extend (i.e. 3D zone building and navigation), but could be developed further (i.e. a 'pure' 3D interface for the zone builder).

Forgiveness and perceived stability principles have not been implemented demonstrably (unless exception catching and open modification on zone attributes can be considered as such).

Aesthetic integrity and modelessness are present to some extend as well. The former is debatable, the latter true in the sense that the user is not 'locked up' at any time (inside a 'mode') throughout the application life.

5.4.4.2.Classes

Several parts of the program code include visibility and virtualization decisions that effect the GUI as well as the VUI. For example, the J3DUI framework has been helpful in this regard by offering certain visualization classes. These utilities mostly control the behavior of 3D objects.

Examples of the utility classes residing in the `terrapeer.vui.j3dui.visualize` package are:

`DisplayFaceGroup` class - An actuator group that supports "display facing" visualization behavior, where this group is oriented to always face towards the view display (not the view eyepoint).

`ConstantSizeGroup` class - An actuator group that supports "constant size" visualization behavior, where this group is scaled to always appear at the same size on the display screen (not display window).

`WorldOverlayGroup` class - An actuator group that supports "world overlay" visualization behavior, where this group will appear to overlay or underlay other objects in the world (no parallel projection).

`DisplayOverlayGroup` class - This class is an actuator group that helps supports "display overlay" visualization behavior. Together with a DisplayOverlayRoot, it defines a 2D space parallel to the view display plane where objects are positioned in X-Y in units of pixels relative to this groups origin. Objects are placed in separate "overlay planes" that define their Z overlay order, with "0" as the base plane, +Z in front towards the eye, and -Z in back away from the eye.

`PerfectOverlayGroup` class - An actuator group that supports "perfect overlay" visualization behavior by selectively combining the other world visualization groups. The order in which the groups are combined, from head to tail, are: world overlay, constant size, display facing.

`DisplayFaceGroup`, `ConstantSizeGroup`, and `WorldOverlayGroup` all extend the `ActuatorTransformGroup` class. `DisplayOverlayGroup` and `PerfectOverlayGroup` extend the `ActuatorGroup` class.

Code in the `terrapeer.gui` package contains other examples on visualization, though in a 2D environment.

5.4.5.Zones

5.4.5.1.Zone and Zone World

The DVE contains a zone for each peer. Not connected, A peer initially is not being able to see other zones than those already stored from prior sessions. When the peer connects to the network, it discovers other peers and their zones. These will be gathered, displayed in the particular peer's VE viewer, and stored locally in the repository. Every visible zone, besides the peer's own, is managed by the `ZoneWorld` class. Typical methods are `addZone()`, `getZone()`, `removeZone()`, and `findZone()`.

The `Zone` class contains all relevant information, state, and functionality for a virtual zone object. A Zone is a virtual area placed on the ground level, which can have different geometrical 2D shapes. It contains attributes, such as `Zone_ID`, `Zone_Name`, `Zone_Description`, and `GlobalState`, and other objects, such as `AccessRights`, `ZoneGeometry`, `Settlement`, `ZoneServices`, and `ZoneObjects`.

5.4.5.2.Zone Builder

`ZoneBuilder` class is responsible for keeping and building the peer's own zone ('MyZone'). It provides the methods `createNewZone()`, `generateZoneID()`, and `getMyZone()` to manage 'MyZone'. ZoneBuilder also provides the methods `buildAllZones()`, `initZoneObjects()`, and `addTarget()` to build all known zones as 3D entities, and initialize and add virtual objects that are attached to a given zone.

The generated ID for a zone, currently has the following format:

```
"TerraPeer.ZID" + ":T-"
      + YEAR + "." + MONTH + "." + DAY_OF_MONTH + "."
      + HOUR_OF_DAY + "." + MINUTE + "." + SECOND
      + ":PID-" + JXTA PEER ID + "." + JXTA PEER NAME
```

The generated ID for a virtual object has the following format:

```
"TERRA-Z" + Zone ID + "-”
      + Service Type + ”-"
      + Object Type + "-"
      + Object Counter
```

This method creates a visual base plane 'room' that represents the virtual Zone:

```
private BasePlane buildZone(...)
```

This method adds a Service of the URL type to the Zone:

```
public String addURLServiceToZone(..., String urlStr)
```

5.4.5.3.Repository

All Zone storage I/O with the local file-system is handled by the `ZoneRepository` class. Some of the important methods are:

```
createXMLRepository()
saveZone()
addZoneToRepository()
loadMyZoneFromFile()
convertZoneObj2XML()
convertZoneXML2Obj()
loadObject()
```

The class `ZoneServices` contains the service objects that a zone can have attached. These are plain virtual objects of some form that have extended attributes depending on their particular type, and represent a certain service that can be called by users. The service would invoke internal or external functionality or applications. Services types available include URL (any links), HTTP (a website), FTP (a file server), etc.

5.5.P2P Network Code

The most important class in the `terrapeer.net` package is `PeerCore`. It includes methods to configure, start, and run the JXTA protocol. Core methods handle discovery, peergroups, advertisements, and message communication:

```
startJxta()
configureJxta()
runDiscovery()
runBidirectionalAcceptPipe()
createPeerGroup()
createPeerAd()
```

The class itself contains several JXTA-related properties, which can be seen in the code snipplet below.

```
public class PeerCore implements Runnable, DiscoveryListener
{
  private static TerraPeerLog myLog = TerraPeerLog.getLogger();

  private static final Logger LOG = Logger.getLogger(PeerCore.class.getName());

  private static PeerGroup peerGroup = null;
  private static PeerGroupAdvertisement groupAd = null;
  private static DiscoveryService discoveryService = null;
  private static PipeService pipeService = null;
  private static RendezVousService rendezvousService = null;
  private static InputPipe inputPipe = null; // input pipe for the service
  private static OutputPipe outputPipe = null; // output pipe for the service
  private static Message msg = null;
  private static ID gid = null; // group id
  private static BidirectionalPipeService bps = null;
  public boolean JXTASTATUS_IS_ON = true;

  //...
```

The `terrapeer.net path` handles the XML repository classes and methods that load, save, and modify file information. It is often necessary to convert between flat XML style content and Java objects. TerraPeer uses objects internally to quickly access and run different data-types. The XML format is used for storing and transmitting data.

Conversion, parsing and object creation methods are located in the schema and xml sub-packages. `XType` classes reside inside the `terrapeer.net.schema` package, where `X` stands for a name. For example, the class `ZoneType` (extends `terrapeer.net.xml.Node`), and implements `get`, `set`, and `add` procedures to create or modify the XML representation of the Zone object type.

5.6.Utility Code

Libraries included in the project:

```
C:\JBuilder9\lib\jbcl.jar;
C:\JBuilder9\lib\dx.jar;
C:\JBuilder9\lib\beandt.jar;
C:\_dev\jxta\jxta2.1.1\jxta.jar;
C:\_dev\jxta\jxta2.1.1\jxtacms.jar;
C:\_dev\jxta\jxta2.1.1\jxtasecurity.jar;
C:\_dev\jxta\jxta2.1.1\pjxta.jar;
C:\JBuilder9\jdk1.4\jre\lib\ext\j3dcore.jar;
C:\JBuilder9\jdk1.4\jre\lib\ext\j3daudio.jar;
C:\JBuilder9\jdk1.4\jre\lib\ext\j3dutils.jar;
C:\_dev\xj3d\jars\gnu-regexp-1.0.8.jar;
C:\_dev\xj3d\jars\httpclient.jar;
C:\_dev\xj3d\jars\j3d-org.jar;
C:\_dev\xj3d\jars\j3d-org-images.jar;
C:\_dev\xj3d\jars\js.jar;
C:\_dev\xj3d\jars\uri.jar;
C:\_dev\xj3d\jars\vlc_uri.jar;
C:\_dev\_libs\vrml97.jar;
C:\JBuilder9\jdk1.4\jre\lib\ext\vecmath.jar;
C:\JBuilder9\jdk1.4\jre\lib\ext\mail.jar;
C:\JBuilder9\jdk1.4\jre\lib\ext\mysql-connector-java-3.0.8-bin.jar;
C:\Java\lookandfeel\skinlf-1.2.3-20020729\skinlf-1.2.3\lib\skinlf.jar;
C:\JBuilder9\jdk1.4\jre\lib\charsets.jar;
C:\JBuilder9\jdk1.4\jre\lib\ext\3DLF.jar;
C:\JBuilder9\jdk1.4\jre\lib\ext\dnsns.jar;
C:\JBuilder9\jdk1.4\jre\lib\ext\ldapsec.jar;
C:\JBuilder9\jdk1.4\jre\lib\ext\localedata.jar;
C:\JBuilder9\jdk1.4\jre\lib\ext\sunjce_provider.jar;
C:\JBuilder9\jdk1.4\jre\lib\im\indicim.jar;
C:\JBuilder9\jdk1.4\jre\lib\jaws.jar;
C:\JBuilder9\jdk1.4\jre\lib\jce.jar;
C:\JBuilder9\jdk1.4\jre\lib\jsse.jar;
C:\JBuilder9\jdk1.4\jre\lib\rt.jar;
C:\JBuilder9\jdk1.4\jre\lib\sunrsasign.jar;
C:\JBuilder9\jdk1.4\lib\dt.jar;
C:\JBuilder9\jdk1.4\lib\htmlconverter.jar;
C:\JBuilder9\jdk1.4\lib\tools.jar
```

An entire collection of global variables and properties was gathered in the public class vars. It contains a long list of static objects that are used throughout the application code. Among these are:

- GUI properties
- JXTA properties and Pipe names
- Timing properties
- Application States
- Generic Colors
- Zone States
- VUI, Space, and Grid properties
- All icon images and static objects

Another class implemented in most of the code is the log (`TerraPeerLog` class), which only purpose is to aggregate the messages, errors, and exceptions that occur. The log is easily included with on line:

```
private static TerraPeerLog myLog = TerraPeerLog.getLogger();
```

There exist several helping classes that usually perform mundane jobs or store some sort of collected data.

A simple helper class for instance is `terrapeer.vui.helpers.VGeometry`, which has these properties:

```
private Transform3D transform3D;
private Vector3d position;
private Vector3d translation;
private double scale;
private Vector3d rotation;
private long alphaCoords;
```

5.7.Data Repositories

Any application storing some kind of user information or data to be processed needs some form of data storage repository. In many instances, this is simply solved by using flat files. In other cases, light- or heavy-weight database servers might be chosen. In a networked system, storage placement becomes ambiguous.

TerraPeer uses an XML-based format to store data as files, and provide easy conversation to for example tree-structured or binary format.

Traditional three-tier client/server settings most often employ a backend database that is connected to a business middleware tier, which handles all interaction with the database storage on one hand, and solely communicates to the client user on the other. This model allows a layer of abstraction, similar to the object-oriented encapsulation design pattern, i.e. make access to data only available through certain predefined methods so the user cannot foobar the system.

In a DVE context, the C/S setting allows data to be placed on a central shared database. As was the case with the original MUD system, this procedure simplifies the client (see Illustration 5.8. C/S Centralized Repository).

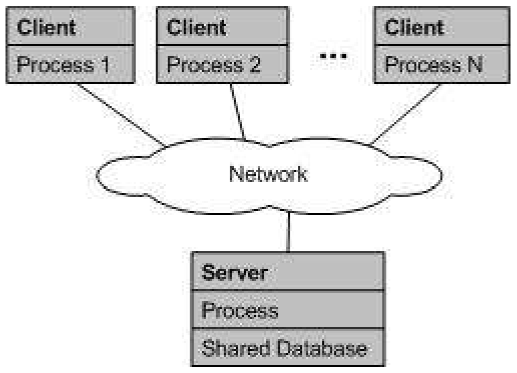


Illustration 5.8. C/S Centralized Repository

Using a server-less network, the DVE data is distributed across each peer client (see Illustration 5.9. P2P Repository Distribution). Replication and synchronization have to be considered.

This mechanism is build into TerraPeer using advertisements to present data to peers in the vicinity. Peers can control the amount of data they are willing to receive (using filtering), and the frequency of updates. As each peer corresponds to a zone, the essential idea is to let each client manage its own local environment, as well as store other peer's local environments if it chooses to.

By creating a flexible, open XML protocol that can include X3D scene-data, it is possible to optimize the distributed peer-network. The XML data contains the appropriate format depending on the usage, i.e. whether only zone meta-data or entire sets of object attributes are transmitted.

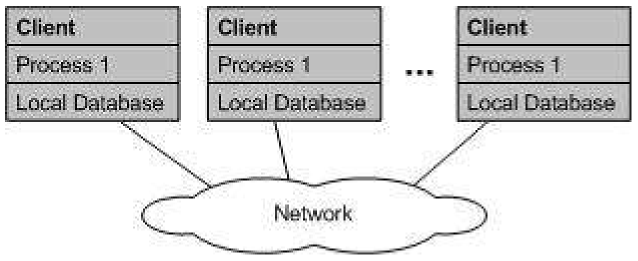


Illustration 5.9. P2P Repository Distribution

5.7.1.Information Storage

TerraPeer is a P2P-based DVE and as such requires all implicit and explicit data – core application libraries, user data, and common environment data – to be located locally. The distributed-peer model obviously does not rely on any central server to handle all common or individual data, and hence has to place the data on each participating peer.

While this location issue is easily resolved by for example creating a file on each peer, it is much more difficult to manage the distribution of data among multiple peers.

In this implementation, a simple data model was chosen to represent the accumulating data, including virtual space objects, user and identification information. Based on the XML standard, the essential idea was to integrate the X3D schema into the repository. X3D is the successor of VRML, and specifically tuned for multi-user networks.

The TerraPeer storage repository is represented by a XML Schema that was created to dynamically capture peer- and worl-related information, while being able to be easily transmitted over the network.

Each part of the virtual environment is not represented by a scene-tree structure and attached objects, but rather by carefully picked meta-data that can be used to recreate the VE. In other words, a given 3D scene would not be saved as a VRML-style file with detailed node-listings and transformation data, but instead be abstracted to its functionality.

An example of how this works: Peer A creates a service object that represents a link to his homepage. The object is placed on his zone, and saved in the repository. When another Peer B visits A's zone, she would have downloaded the corresponding data, and recreated the scene on her local interface. Instead of saving and transferring all specific object-data, such as form, transformation or coordinates, the only information stored is that the service exists and what it does (i.e. it contains a link).

Limiting the workload in exchange for virtual extravaganza might not seem attractive; actually counter-productive considering the original purpose of VE's to enhance the user experience. The immediate limits of this meta-data approach can be overcome, though, as will be shown below.

5.7.2.Distribution and Integrity

Since communication between peers is an essential operation, and repository data has to be shipped forth and back between entities, it is necessary to choose a dynamic protocol. As XML has gained popularity due to it's easy parsing capabilities and readable structure, a distribution format based on XML also makes sense from a network perspective.

On a side note, this decision reveals bandwidth problems. XML is a pretty heavy load to send through the ether. Transforming the XML to binary format will decrease network traffic substantially.

One of the essential problems in distributed spaces is integrity. Integrity has to be preserved across a dispersed network of peers, each randomly logging on and off each other. This architectural hurdle is usually overcome in server-centric systems by exactly letting only a single entity handle all information.

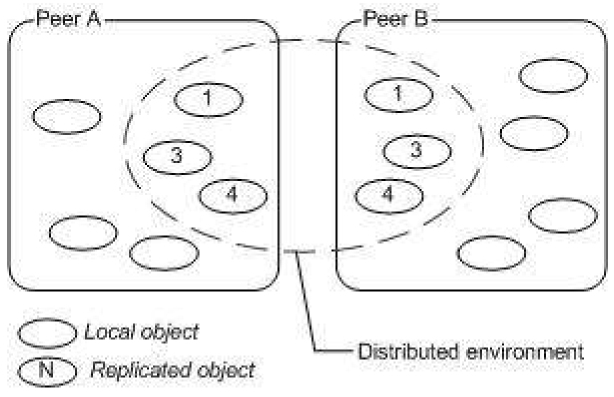


Illustration 5.10. Object Replication

Each peer can choose to replicate certain data, such as entire zones and objects, or selected attributes (see Illustration 5.10. Object Replication). Replicated data have timestamps, and are thus always dependent on the actual peer that 'owns' the data, i.e. using the access/ownership settings all zones and objects 'belong' to a certain peer. Other peers might re-distribute data and thus maintain the VE even though the original peer is not present or nearby, but the timestamp remains. If a peer should disappear from the network for a longer period of time, the other peers simply can let the data expire.

The earlier mentioned meta-data is not constrained. It is possible using X3D, VRML, or other object formats, or extending the information of particular objects within the application.

5.7.3.Data Formats

As most of the visible 3D zone- and object-structure is assembled on-the-fly by instantiation, the data load can be minimized. In many instances it is not necessary to transfer 3D-specific information.

Transferring a virtual representation of a service for example would not actually transfer data about a sphere-object, position, and transformation, but merely service attributes such as an URL. The local peer would be able to reconstruct the virtual object by assuming standard settings.

Depicting the entire specification of the data formats applied would exceed this scope. To retain the overview, it should only be noted that the structure is hierachical, and allows easy extension.

The TerraPeer data format is written in XML Schema, and the base structure looks like depicted below (Illustration 5.11. TerraPeer Repository XML Schema).

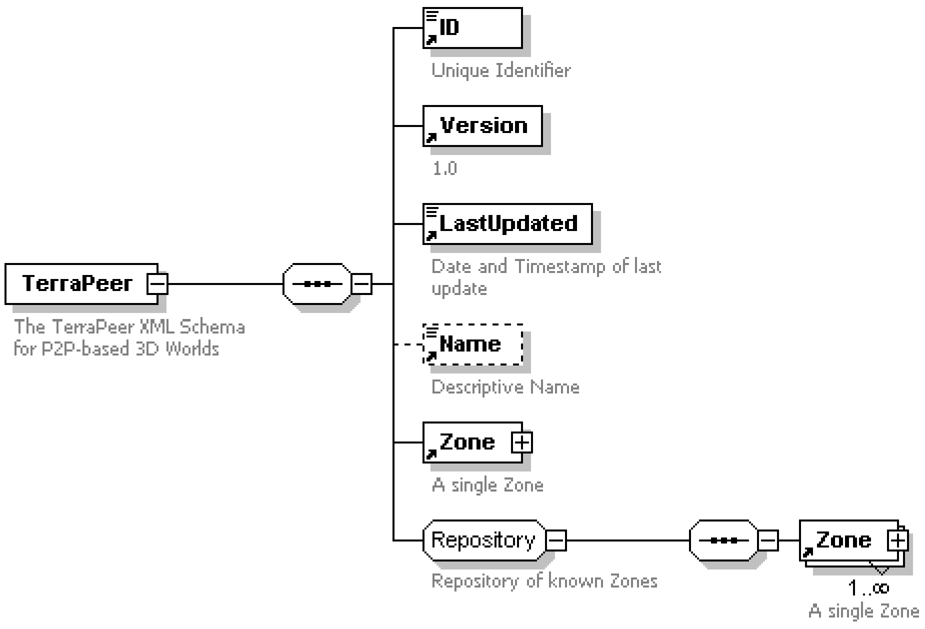


Illustration 5.11. TerraPeer Repository XML Schema

Appendix VII shows more illustrations of the XML Repository Schema design.

5.8.Unresolved Issues

There are dozens of unresolved issues that unfortunately are only partially documented. As of this writing, there exists a number of class methods in the code that have not been implemented (fully or partly). Depending on the continuous development (debugging) of the application, as well as tracking missing bits (by indicative '//todo' markings), the project website might display a more extensive listing of issues in the future.

Nevertheless, there are a few issues listed here:

- User Interface
  - implement and simplify visibility filtering

    - enable better map navigation
- Navigation
    - implement zone-teleporting (linking) functionality
    - implement additional navigational aids (landmark mapping, etc.)
    - simplify (restrict) available user movements for accessing service, modeling (building), and interaction with other avatars, i.e. 'guide' the user to the visible objects by leading the viewer (alignment, zoom translation, etc.)
    - implement 3D levels (an upper navigation-only layer) and 'jump' functionality
- Services
    - basic URL service link should open an integrated web-browser window
    - implementation of chat to send messages between 2 peers
    - implementation of access control of virtual models
- Scene data repository
    - import entire scene for a zone directly from X3D
- Performance
    - real-time/threading of multiple zones in aura
    - optimize scene request for speed
    - optimize scene propagating
    - create XML/X3D protocol buffer

5.9.Installation and Documentation

System requirements:

- Java SDK v.1.4

All other required libraries are included in the software package. The Application Tutorial is attached in Appendix IV. A generated JavaDoc can be found on the software CD.

All resources are available online as well (see on this project's website for more information).

5.10.Testing

There are many testing scenarios that could be envisioned, such as testing the scalability of the distributed network peers by deploying hundreds or more instances, or testing the user interaction when sharing or transferring objects in the virtual space. It would require a thorough test-plan and some QA routines to be able to debug the code.

While implementing TerraPeer, debugging and testing were natural parts of the development effort. Resolving coding issues was usually accomplished by using techniques such as simple code dumps, more advanced logging or writing test classes.

TerraPeer's Log window currently shows all major events of the application process, and should also display relevant information in the event of an exception. Some helper classes might have survived several code cleanups, and still show a few test functions.

The scope of this project cannot provide the execution of an entire test phase, and would suggest that the current implementation rather might be viewed as an on-going development cycle, where testing is an important part of the course.

A useful distinction for test-cases, based on classifying different scenarios:

- Connection based
    - Single User
    - Establish connection of >3 peers
    - Large-scale Testbed (>100 Peers)
    - Test system on different operating systems
- Virtual Environment
    - Zones - Creation (building) and destruction; Test zone reservation system cycle

  - Objects and Services - Creation, destruction, and transformation; Access control and sharing; Service search and usage; Visibility and filtering
- Avatars and User interaction
  - Navigation issues
  - Test object sharing and control for multiple users
  - Avatar communication, interaction and collaboration

On this page, Illustration 5.12. Testing Scenarios, shows a variety of situations that the application tester should use to approach the test systematically.

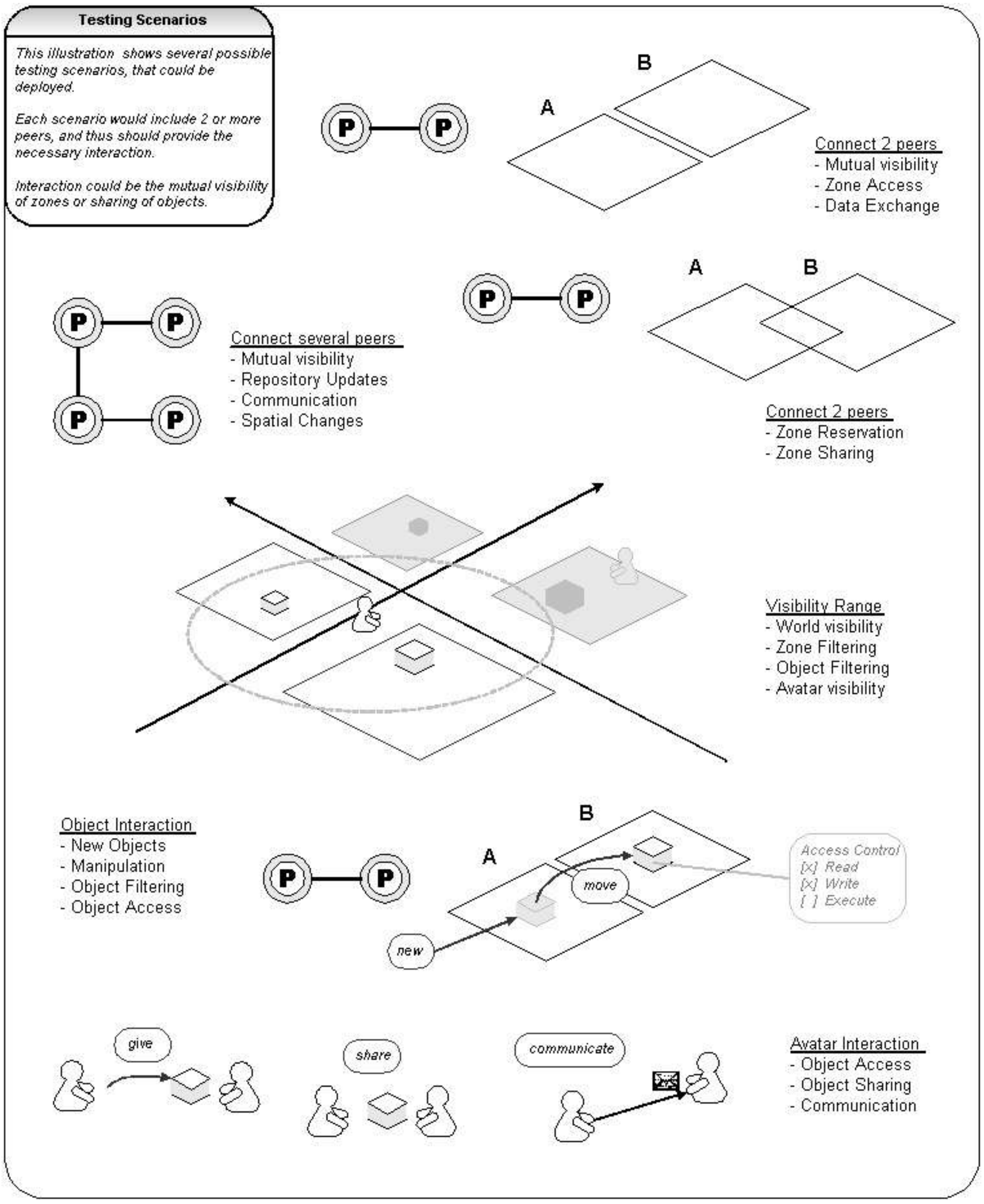


Illustration 5.12. Testing Scenarios

6.Discussion

This chapter examines the results and discusses their implications. Further, it illuminates some of the difficulties that where encountered throughout the project.

6.1.Results

The result of this thesis is represented by a prototype application that is able to establish a distributed virtual environment.

6.1.1.Architecture and Implementation

Following the approach outlined in the beginning of this document, the project was created by means of analyzing and applying tools to accomplish it's goals.

The objective of this project was "to construct an application for a distributed virtual environment that is based on a server-independent network". This goal has more or less been reached. Contemplating about an accumulation of aspects that need to be considered, it could be said that a 'basic' architecture and a running application are the result of this project.

The initial question of "how a peer-to-peer based multi-user 3D environment can be build" has been answered by example. The question of how to visualize the distributed activity in 3D space has been shown by the definition of peer-zones and services.

All together the concept of the distributed application was demonstrated, and works as expected. An intuitive user interface, 3D building and the zoning technique of multiple virtual areas, ad-hoc connectivity of peers has been realized.

As stated, this projects did not aim to study particular properties of 3D techniques, VE's or distributed peer networking technologies.

6.1.2.Results Overview

This project suggests a DVE architecture that focuses mainly on creating a fully decentralized multi-user environment.

The design and prototype implementation have the following features:

- An underlying, fully decentralized P2P network based on JXTA technology, that can be used for any data and communication format, supporting multicasting.
- Performance through load distribution, zoning, and the possibility for filtering at several application layers (messaging, graphical and functional aura)
- Zones to split the world into locales that are stored separately, processed by selection as a spatial model for interaction
- Scene objects are replicated
- An event and state exchange notification system is possible through ad-hoc subscription between peers (event caster and listeners model)
- Virtual object interaction, including access control based on Zones, with the possibility to implement automatic state exchange directly between interacting peers
- Abstract services are used to virtually represent information, functionality, or other published service types
- Viewers to the VE display a basic virtual world environment on each peer, including grid-lines or a background horizon to easy orientation.
- TerraPeer supports positioning and navigation in the VE world

- TerraPeer enables peers to create and share instances
- Sharing rules apply to all instances in the virtual hierarchy
- Data visualization through abstract services (information publishing)
- Zone and object permissions to other peers can be set
- Object Persistence and Sharing
- The zone metaphor allows users to virtualize their peer, and reflects a peer's state, position, and services offered
- Spatial reservation-system based on zones

6.1.3.Results Discussed

In retro perspective, what are the results of this thesis, and which issues were valuable in this experience?

The result is a survey of existing DVE projects, and an architecture and implementation of a totally distributed VE with a user interface and some functionality.

While not a complete solution nor a framework, the prototype application is displaying the overall 'case' of this thesis: How to implement DVE operations to support a decentralized network, visualize peers, provide a platform to build and publish services, and to package everything in a GUI with navigation and feedback components.

The working features that have been implemented allow the following usage:

- Functionality for ad-hoc peer discovery and connection, requiring only some other initial peer address.
- Automatic VE persistence through zone replication across peers.
- Performance advantage by sending meta-data and recreating 3D representations on-the-fly, avoiding transferring more heavy 3D data.
- User navigation in the 3D world.
- Building virtual objects that can represent a service on virtual zones that represent a peer.
- Viewing and filtering peer zones in the world.
- Using a basic service.

The discovery and virtualization of peers and the interface to create and publish services are primary advantages the TerraPeer application has above other DVE systems.

With the choices of network and graphical technologies, both advantages and disadvantages have been observed. Clearly, the JXTA API provides a rich peer-to-peer framework that integrates essential routines, including peer discovery and message piping.

It is questionable how effective the framework is compared to other projects, such as Gnutella, but considering that protocols are written with performance in mind, as well as their flexibility, there might not be better alternatives.

The graphical framework on the other hand - Java3D - shows considerable performance issues, though it is based on OpenGL. This is partly due to the JVM, as well as general tuning for 3D environments. Further explanations of the performance limitation are given in a subsequent section.

A better graphical engine would enhance the user experience, and allow more complex creations in the virtual world. The survey provides some helpful hints of how to address this problem. Game engines have proved to be highly effective for multi-user environments. Though they lack the ability for massive distribution, the engines are tuned for performance and graphical content. Other projects and studies suggest approaches to circumvent some of the obstacles associated with handling events and communication in massive distribution.

If the core of game engines would be integrated as a graphical layer on top of a P2P network, combined with specific mechanisms to filter or multicast events, the result could enhance the 3D display significantly.

Performance has also been addressed through different solutions, including multi-server networks, multicasting and zoning. This application's architecture theoretically allows any kind of messaging and event model. And multi-server networking can easily be emulated, as a peer can act as an independent server with as much power as required. Inherently, the processing of 3D data and services is distributed throughout the network, both avoiding bottlenecks and server-dependence.

Compared to other DVE systems, this project examined specifically the feasibility of creating a user-centric application that in essence can be installed anywhere, and connect ad-hoc to the virtual world, which in turn is persistent 'within the network'. This means that the world only 'exists' to the extend of how many peers are connected.

As has been discussed earlier, this might occur as a weakness, since a few peers connecting on-and-off would produce a very 'fluid' environment. But the dynamic is actually quite similar to the web or a file-sharing network, which has obvious advantages. Again, nothing hinders peers to be permanently running, thereby acting as more stable entities.

The outcome of this thesis is not a framework, and might prove of less value for further development unless the graphical performance is enhanced. The benefits of the result, though, are useful for further research and development of DVE systems. It is the experience of designing a P2P-based DVE combined with the abstract definition of virtual services on a user-centric interface, which can provide helpful insights. The initial survey also asserts an overview of most of the systems related to DVE research.

Looking at the usage of this application as a user, the following cases are possible:

A Java-based application that is build to run arbitrary services on any JVM system with a flexible GUI.
The user can connect to the distributed virtual world, and automatically discover peers and their services.
Zones can be created to virtually represent a peer in the world. They are distributed ambiguously to the user, depending on personal filtering settings.
Users can create a few 3D objects, as well as a basic web 'service', i.e. a URL link.
These can be made public, i.e. allow access to other peer users.
Users can roam the virtual space with the help of navigational aids.
A web service object that is placed on a zone can be selected and viewed in a browser window.

At this stage, several scenarios with possible impact on the overall design have not been tested:

Running more than 10 peers simultaneously to test zone discovery
Connecting multiple peers and test service publishing and usage, i.e. event and message latency
Zone filtering mechanism to limit replication and visibility radius
Object creation, destruction, and transformation, access control and sharing, search and usage

A comprehensive study of these scenarios would produce valuable information on how scalability, performance, and general feasibility of massive distribution affect this solution. Section 5.10 lists a few test-cases. Ideally, an entire test plan should be devised.

In conclusion, this thesis could demonstrate:

- Total distribution of a 3D environment using P2P
- Virtual representation of peers in a 3D environment using Zoning
- Visualization of services as 3D objects
- Openness to any service, including a simple website

Unresolved problems and absent investigations of this thesis:

- Extensive testing of performance and functionality
- Lack of performance for 3D graphics
- Zone visibility filtering
- Simplified navigation through zone tele-porting, aids, movements for accessing services, etc.
- Broader range of services
- Better event casting and listening management
- Avatar visibility
- User-to-user grouping and communication, such as chat

There are good possibilities to continue the work from here. Especially by testing the different performance aspects, and by addressing the 3D graphics that should be based on a tuned game engine, this DVE system could prove beneficial for both future P2P-DVE studies, and actual communities.

A good extension would be to build advanced services, such as a chat function or business-to-business (B2B) communications. This could for example be a XML-based service that aggregates a news-feed, or maybe streams warehouse data to shops. In VE, the service can be represented accordingly, using metaphors.

An interesting thought could be the visualization of connecting lines between zone-services. This could depict how peers communicate, or use (subscribe and push) each other's services. Lines could be public or only visible to certain groups. The 3D environment would allow members to follow the routes, and gain insight into how the chain is set up. Ultimately the activity might be monitored and adjusted directly within the VE, i.e. by virtually connecting pipes of data.

6.2.Experience and Difficulties

6.2.1.3D and Java Performance Limitations

One of the fundamental problems encountered in the implementation of the virtual environment, was the observed effectiveness of 3D graphics, or rather it's lack of. While testing VRML based virtual worlds did not seem to behave particularly faster, other 3D spaces, namely most current online game engines could far outrun this application.

As previously described, online game engines provide a large array of specific 3D functionality, and are able to highly optimize network traffic as well as scene rendering. Many engines are also exercising underlaying hardware to a full extend, which tunes performance quite effectively.

A research paper by Jacob Marner examined the Java performance, specifically in relation to game design. Though his results should be taken cautiously, as numbers depend on many assumptions, the usual assumption of Java performance compared to those of C++ seems true: Benchmarks show that even 'tweaked' Java code is typically 20-50% slower than C++, while 'untweaked' code is much slower, often by a factor of three or four [JAVAGAME02 pp.87].

This finding should be noted when implementing 3D applications, even if performance generally depends on 3D hardware and not on the programming language used. On the other hand, the paper asserts that Java enhances overall productivity in software development by 30% or more. In its conclusion, Marner writes that Java provides a good platform for low-profile games due to lower cost, or in combination with C++ code for low level 3D engine functionality.

This project aimed to implement several design goals, including performance, but must concede in its conclusion that the Java3D API would not pose a valid platform for DVE applications.

Hence, though one of the initial goals of creating a platform-independent, object-based, scalable application was realized, performance is a major drawback.

6.2.2.J3DUI Framework Problems

Though initial assertion of the J3DUI framework seemed promising, the actual usage throughout the project development was tedious. Not using true object-oriented design, the assembled packages provided a VRML-like source-connects-to-target methodology, which makes sense in 3D context, but is rather difficult in programming terms. The filter-like pattern required certain modifications to existing objects and sometimes lengthy re-routes to solve a particular task.

In a future implementation, it is recommended to directly use the Java API, or even submerge to the OpenGL level. Whether this results in more efficient processing of 3D scenes is questionable, but not unlikely.

7.Conclusion

7.1.Reaching the Objectives

By researching the existing DVE systems to learn about the implication of design, technical obstacles, as well as existing systems and frameworks, it was possible to gain an overview of how to approach a design. Issues that are integrate in multi-user environments, such as network topologies and performance, graphical engines, event mechanisms, zoning, object sharing and access control, as well as 3D user interfaces have been examined.

An analysis of the various studies, revealed a couple of interesting points. It revealed the decentralized role of P2P to distribute the load when replicating scene objects, that performance can be enhanced by dynamically adjusting the DVE to resources and by applying zoning and message filtering.

Beyond the underlaying mechanisms, it also revealed that zones can be used as a spatial model for interaction among clients, and can determine the space within which interactions are possible, that an event-based notification system, as well as aura are vital for state exchange and interaction, and that graphical metaphors and abstract services can be used to virtually represent entities.

The result of this thesis project is the creation of an architecture that ensembles several technologies and design principles, and can be used to produce a DVE software that is based on a P2P network. The design reflects a peer GUI that displays a 3D world interface in combination with navigational and building components. The object-oriented, layered structure allows open and extensible code.

Further, through a detailed specification and an actual prototype implementation of a peer application - TerraPeer - the different aspects of the architecture were highlighted.

Topics related to DVE design that have been modeled are creation, virtualization, and usage of services as virtual objects, the virtualization of distributed peer-network, total decentralization and discovery, usability, 3D world interaction, and feedback and navigation.

The application has a technological consistent foundation by having applied the API's Java3D and JXTA, and the X3D standard.

In particular, it was possible to demonstrate the use case, where the application would be able to display the 3D space, discover other peers, and connect to a shared virtual environment. The user could create a zone representing his peer, build a simple web service on top with a link to his homepage, and publish it. Another peer could then navigate through space, find the service, and run it, which would open a web browser and display the page.

The complexity of DVE systems was thus ambiguous to the user. The possibility to connect ad-hoc to a space that has no central entity awards him with control. The openness of abstract services, and the XML-based protocol allow any kind of extensions.

There are numerous subjects that, though mentioned in the context, have not been addressed to a greater extend in the implementation. Performance optimization, shared space access control, and multi-server (super server) network models are among them.

7.2.The Future

The extensive amount of components that comprise a DVE system are currently being studied, and several viable solutions have been suggested.

A range of problems remain, though. Among other issues, primarily the 3D rendering process and networking communication are expensive tasks that have to be optimized in an implementation. The distribution of data requires network support for mixed traffic types. Management is necessary for user collaboration, state updates, and access control.

However, DVE research is at an stage where the accelerating performance of graphical and networking hardware combined with advances in optimization will develop this field rapidly. As a platform for commercial applications, corporations such as Fujitsu, IBM, Intel, Mitsubishi, Softbank, and Sony, as well as ventures such as Black Sun Interactive, Chaco Communications, OnLive!, ParaGraph International, OZ Virtual, and Worlds are already developing distributed virtual environments for the mass market.

How would it be possible to enhance a DVE system? Using primarily game engines in combination with hybrid P2P networks and open standards, an interface to the virtual world could be build that would be able to reach proportions of that of a world-wide-web browser today.

Game engines have specialized in enhancing the rendering process. Zoning, scene graph manipulation techniques, algorithms to optimize rendering, and filtering techniques can produce similar performance for large-scale environments. Hybrids of distributed network models can avoid the communication bottlenecks that accumulate in multi-user environments. GUI design principles can create intuitive controls to navigate and interact inside the VE.

Certainly, we are at the beginning of a genuinely intriguing and interesting future of distributed virtual networks.

Appendix I.Resources

References

[3DDMF01] "A Dynamic Message Filerting Technique for 3D cyberspaces", S.J. Yu and Y.c. Choy, Yonsei University, South Korea, April 2001

dve_msg01.pdf

[ADHA99] M. Endo, T. Yasuda, and S. Yokoi, “A Method for Constructing Action Database for VRML Humanoid Animation,” 4th Annual Conf. Proc., The Virtual Reality Society of Japan, 1999, pp.109-112.

[AGLETS02] IBM. "IBM Aglets Software Development Kit".[4 February 2002].

http://www.trl.ibm.com/aglets/

[AMSSS03] "Automatic Management of Sessions in Shared Spaces", Géraldine Texier geraldine.texier@rennes.enst-bretagne.fr École Nationale Supérieure des Télécommunications de Bretagne, BP 78, 2 rue de la Châtaigneraie, France; Noël Plouzeau noel.plouzeau@irisa.fr, Irisa, Campus de Beaulieu, France, 2003

Automatic Management of Sessions in Shared Spaces.pdf

[AN-Survey97] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall and G. Minden, "A Survey of Active Network Research", IEEE Communications Magazine, 35(1), pp. 80-86, 1997.

ieeecomms97.ps

[AOHA00] M. Endo, T. Yasuda, and S. Yokoi, “An Application Oriented Humanoid Animation System Based on VRML,” Proc. 7th Int’l Conf. Parallel and Distributed Systems: Workshops (MMNS2000), IEEE CS Press, 2000, pp. 213-218.

[APPLE] Apple Human Interface Guidelines > Human Interface Design; Human Interface Design Principles

http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/

[ARA00] R. Keller, S.Choi, M.Dasen, D.Decasper, G.Fankhauser, B.Plattner, "An Active Router Architecture for Multicast Video Distribution", Infocom 2000, March 2000.

[Attention] Mark A Folz, Randall Davis, "Query by Attention: Visually Searchable Information Maps". MIT AI Lab.

[AVAT02] "Realizing external avatars Action Control on WWW-based pseudo 3D space", Taro Nakao and Takerumi Ogawa, Osaka Univerity, Japan; 2001 IEEE

Realizing external avatar.pdf

[AVOCADO] "Avocado: A Distributed Virtual Environment Framework", 2003, Henrik Tramberend

tramberend03avocado.pdf

[Capin99] T.K. Capin, I.S. Pandzic, and N. Magnenat-Thalmann, Avatars in Networked Virtual Environments, John Wiley and Sons, 1999.

[CITY94] "Navigation in Textual Virtual Environments using a City Metaphor", by Andreas Dieberger, Vienna University of Technology, Faculty of Technology and Sciences, November, 1994

dieberger00_Navigation in Textual Virtual Environments using a City Metaphor.pdf

[CN96] A.S. Tanenbaum, "Computer Networks", Prentice-Hall, 1996.

[COLVS01] "Collaborative Distributed Virtual Sculptin" by F.W.B.Li, R.W.H.Lau, and F.F.C.Ng, Dept. of Computer Science, City University of Hong Kong, 2001 IEEE.

Collab_distr_v_sculpting01.pdf

[CRYSTAL] Crystal Space

http://www.cs.lth.se/Education/Courses/EDA045/assignments/assignment1/assignment1.html

http://crystal.sourceforge.net

[CVE95] ”Interacting in Distributed Collaborative Virtual Environments”, W.Broll, IEEE VRAIS’95, pp148~155, 1995

[DEE98] "DEE An architecture for distributed VE gaming", Powers et.al, 1998

dee_game98.pdf

[DISTR-SG99] "A Network Design Architecture For Distribution Of Generic Scene Graphs", Panagiotis Fiambolis, Master’s Thesis, September 1999; Naval Postgraduate School, Monterey, CA, USA.

[DIVE96] O. Hagsand, “Interactive multiuser VE's in the DIVE system,” IEEE Multimedia, Vol. 3, No.1, pp. 30–39, 1996.

[DIVE98] E. Frecon and M. Stenius, "DIVE: A Scalable Network Architecture for Distributed Virtual Environments", Distributed Systems Engineering Journal, Special Issue on Distributed Virtual Environments, 5(3), 1998. Emmanuel Frconyand Marten Steniusz, Swedish Institute of Computer Science, Box 1263, SE-164 28 Kista, Sweden, Received 6 March 1998

scaleable network architecture for distributed VE dive98.pdf

[DJV] Web3D Distributed Interactive Simulation DIS-Java-VRML Working Group;

http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml

[DVE97] "Distributed Virtual Reality for Everyone - a Framework for Networked VR on the Internet", W.Broll, Proc. IEEE VRAIS’97, pp121, 1997

[DVE-AVAT] "The Avatar Navigation of Distributed Virtual Environment By Using Multiview Client" by Man Kyu Sung, Chan Jong Park; VR Lab., Human Computer Interaction Department, Systems Engineering Research Institute (305-666 Eoueun-dong Yusung-ku Taejun Korea, {mksung, cjpark} @seri.re.kr.

avatar navigation.pdf

[DVE-CS03] "Evaluating the Impact of the Communication System on Distributed Virtual Environments". Renata Cruz Teixeira, teixeira@cs.ucsd.edu; http://www-cse.ucsd.edu/~teixeira; Computer Science and Engineering Department, University of California, San Diego, CA, USA. and Otto Carlos M.B. Duarte, Otto@gta.ufrj.br; http://www.gta.ufrj.br/~otto; Grupo de Teleinform´atica e Automac¸ ˜ao (GTA), COPPE/EE, Programa de Engenharia El´etrica, Universidade Federal do Rio de Janeiro, Brazil. Multimedia Tools and Applications, 19, 259–278, 2003.

dve_com03.pdf

[DVE-E00] "Enhancing Engineering Education through Distributed Virtual Environments" by T.Sulbaran and N.C.Baker, Georgia Institute of Technology, Atlanta, USA, IEEE 2000.

edu_distr_vr.pdf

[DVE-Fid00] "Fidelity Optimization in Distributed Virtual Environments", Michael V. Capps, Naval Postgraduate School, Monterey, California, June 2000

Fidelity Optimization In Distributed Virtual Environments - Michael.Capps.pdf

[DVE-Filter02] "A Generalized Perception FIlter for DVE's", Jiang Du et.al., Singapore, IEEE 2002

perc_filt02.pdf

[DVE-Frame02] "A Framework for Multiuser Distributed Virtual Environments", Maja Matijasevic, Denis Gracanin, Kimon P. Valavanis, and Ignac Lovrek, 2002

frame_distr_vr02.pdf

[DVE-Future02] "Distributed Virtual Environments - An Active Future?", Tatiana Balikhina, Frank Ball, David Duce; School of Computing and Mathematical Sciences, Oxford Brookes University ; {tbalikhina, fball,daduce}@brookes.ac.uk ; Proceedings of the 20th Eurographics UK Conference (EGUK.02)

dve_eu_futu02.pdf

[DVE-MM99] "Study on DVE with multimedia communication: Introduction for flexible method for DVE expansion", D.Iwata et.al., Kyoto University, IEEE 1999

exdve_mm99.pdf

[DVE-MNG00] Fabre Y, Pitel G, Soubrevilla L, Marchand E, G´eraud T, Demaille A. "A framework to dynamically manage distributed virtual environments". Virtual Worlds, Proceedings of the Second International Conference, VW 2000, Paris, France, 5–7, July. Springer: Berlin, 2000; 54–64.

[DVE-P98] "A VEplatform system: A system for distributed virtual reality", Demuynck et.al., University Antwerp, Belgium, 1998

VEplatform.pdf

[DVE-PERF02] "A performance study on multi-server DVE systems" by Beatrice Ng, Frederick W.B. Li, Rynson W.H. Lau, Antonio Si, Angus Siu, Department of Computer Science, City University of Hong Kong, and Oracle Corporation, 500 Oracle Parkway, Redwood Shores, CA 94065, USA;

accepted 30 December 2002. As part of their study, a DVE system called 'CyberWalk' was used; a web-based distributed virtual walkthrough environment.

dve_perf03.pdf

[DVE-PLIM] "Accommodating performance limitations in distributed virtual reality systems" by J.R. Ensor, G.U. Carraro, J.T. Edmark. Bell Laboratories, 101 Crawfords Corner Road #4F607 Holmdel, NJ 07733, USA. 2000 Published by Elsevier Science B.V.

distr_vr_res.pdf

[DVE-TAX97] M.R. Macedonia and M.J. Zyda, “A taxonomy for networked virtual environment,” IEEE Multimedia, Vol. 4, No. 1, 1997. And Macedonia MR, Zyda MJ. "A taxonomy for networked virtual environments." IEEE Multimedia 1997; 4(1):48–56.

[DVE-SA95] M.R. Macedonia, “A network software architecture for large scale virtual environments,” Ph.D. Thesis, Naval Postgraduate School, Monterey, California, June 1995.

[DVE-VJC01] Diehl S. "Distributed Virtual Worlds: Foundations and Implementation Techniques Using VRML, Java and CORBA". Springer: New York, 2001.

[DVE-VRML] "Distributed Virtual Environments and VRML: an Event-based Architecture"; Mike Wray (mjw@hplb.hpl.hp.com) & Rycharde Hawkes (rjh@hplb.hpl.hp.com); HP Labs (Bristol), Filton Road, Bristol, BS12 6QZ, UK, 1998

http://keryxsoft.hpl.hp.com/documents/dve/vrml.htm

dve_vrml.htm and vrml_event98.pdf

[DWTP98] W. Broll, "DWTP - An Internet Protocol for Shared Virtual Environments". Proceedings of the Virtual Reality Modeling Language Symposium 1998 (VRML'98), ACM pp. 49-56, 1997.

[EIM-SDVE99] H.H. Abrams, "Extensible Interest Management for Scalable Persistent Distributed Virtual Environments", PhD Thesis, Naval Postgraduate School, Monterey CA USA, 1999.

[EM95] ”EM-An Environment Manager For Building Networked Virtual Environments ”, Q. Wang, M. Green, C. Shaw, IEEE VRAIS’95, pp11, 1995

[FARCRY] CryENGINE

http://www.farcry-thegame.com

[FLEXI99] Herbert AJ, Hayton RJ, Bursell M. "Mobile Java objects". BT Technology Journal 1999; 17 (2). Networked distributed systems

http://www.bt.com/bttj/

[Game98] "Games on the ‘Net!", Michael Zyda, Naval Postgraduate School, 1998

GamesNet_ZydaMunich98.pdf

[GE02] "Game Engine Anatomy 101", 2002, Jake Simpson

http://www.extremetech.com/print_article/0,3998,a=29517,00.asp

[HCI] HCI Bibliography : Human-Computer Interaction Resources; Gary Perlman

http://www.hcibib.org/

[HET03] "Handling Heterogeneity in Networked Virtual Environments", Helmuth Trefftz, Ivan Marsic, Michael Zyda, 2003 by the Massachusetts Institute of Technology

Handling Heterogeneity in Networked Virtual Environments Presence12.1-2003.pdf

[HIVE] Large Scale Real Time Multi-User Virtual Reality Research (HIVE); Dave Snowdon (dns@cs.nott.ac.uk); Department of Computer Science. MASSIVE-3 represents the ongoing development of the HIVE Project Kernel (HIVEK).

http://www.crg.cs.nott.ac.uk/research/projects/HIVE/

[Honda95] Y. Honda et al., “Virtual Society: Extending the WWW to Support a Multiuser Interactive Shared 3D Environment,” Proc. 1st Symposium Virtual Reality Modeling Language, ACM Press, 1995, pp.109-116.

[HORB] is a lightweight Java Object Request Broker (ORB) developed by H.Satoshi.

http://www.horb.org

[IEEE1278] Standard Commitee on Interactive Simulation, IEEE Computer Society. IEEE Standard for Distributed Interactive Simulation. IEEE Std 1278.1-1995, 1995.

[J3DUI00] "3D User Interfaces with Java 3D", Jon Barrilleaux, August 2000, Softbound, 520 pages, ISBN 1884777902. From the book description: “3D User Interfaces with Java 3D is a practical guide for providing next-generation applications with 3D user interfaces for manipulation of in-scene objects. Emphasis is on standalone and web-based business applications, such as for online sales and mass customization, but much of what this book offers has broad applicability to 3D user interfaces in other pursuits such as scientific visualization and gaming.”

[Java3D] The Java 3D API

http://java.sun.com/products/java-media/3D/

http://java.sun.com/products/java-media/3D/collateral/j3d_api/j3d_api_3.html

[JAVAGAME02] "Evaluating Java for Game Development" By Jacob Marner, B.Sc., Department of Computer Science, University of Copenhagen, Denmark (jacob@marner.dk), March 4th, 2002

Evaluating Java for Game Development.pdf

[Joslin01] C. Joslin et al., “Sharing Attractions on the Net with VPARK,” IEEE Computer Graphics and Applications, vol. 21, no. 1, Jan./Feb. 2001, pp. 61-71.

[JXTA-HELLO] "Hello JXTA!" by Raffi Krikorian, 04/25/2001

http://www.onjava.com/pub/a/onjava/2001/04/25/jxta.html

[JXTA-START] "Getting Started with JXTA, Part 1-5", O'Reilly Book Excerpts: JXTA in a Nutshell. Scott Oaks is a Java Technologist at Sun Microsystems. Bernard Traversat is a well-known developer in the Java Community and an active member of the Project JXTA. Bernard is the Engineering

Manager for the JXTA CORE. Li Gong is a well-known developer in the Java Community and an active member of the Project JXTA. Li is the JXTA Engineering Director for the JXTA CORE.

http://www.onjava.com/pub/a/onjava/excerpt/jxtaian_2/index4.html

[JXTA-P02] "Project JXTA Virtual Network", Bernard Traversat, Mohamed Abdelaziz, Mike Duigou, Jean-Christophe Hugly, Eric Pouyoul and Bill Yeager, Sun Microsystems, Inc., October 28, 2002

JXTAprotocols_01nov02.pdf

[JXTA-P03] "JXTA v2.0 Protocols Specification", Project JXTA http://www.jxta.org, Published 2003, Copyright © 2001, 2002 Sun Microsystems Inc.

http://spec.jxta.org/v1.0/docbook/JXTAProtocols.pdf

jxtaprogguide_final.pdf

[Kalra98] P. Kalra et al., "Real-Time Animation of Realistic Virtual Humans," IEEE Computer Graphics and Applications, vol. 18, no. 5, Sept./Oct. 1998, pp.42-55.

[Kawakami98] Y. Kawakami, T. Yasuda, and S. Yokoi, "A Study on Multiuser 3D Virtual Space Based on VRML,", Technical Report of the Institute of Electronics Information and Information Engineers, Oct. 1998, pp.7-14.

[Lea97] R. Lea et al., "Community Place: Architecture and Performance," Proc. VRML97, ACM Press, 1997, p.41-49.

[Learn99] "Learning and Building Together in an Immersive Virtual World"; Maria Roussos, Andrew Johnson, Thomas Moher, Jason Leigh, Christina Vasilakis, Craig Barnes; University of Illinois at Chicago; 1999 by the Massachusetts Institute of Technology

roussos99_Learning and Building ImmersiveVE.pdf

[LSVR99] "An Operating Environment For Large Scale Virtual Reality"; A Thesis Submitted To The University Of Manchester For The Degree Of Doctor Of Philosophy In The Faculty Of Science And Engineering April 1999; By Stephen Robert Pettifer, Department of Computer Science

An Operating Environment For Large Scale Virtual Reality srp-phd.pdf

[MaDVi02] "MaDViWorld: a software framework for massively distributed virtual worlds"; Patrik Fuhrer*,†, Ghita Kouadri Most´efaoui and Jacques Pasquier-Rocha; Softw. Pract. Exper. 2002; 32:645–668 (DOI: 10.1002/spe.453)

[MaDViSPE] "Massively Distributed Virtual Worlds a Framework Approach MaDViWorld: a Java Software Framework for Massively Distributed Virtual Worlds"; Patrik Fuhrer and Jacques Pasquier-Rocha, University of Fribourg, Department of Informatics, Rue P.-A. de Faucigny 2, CH-1700 Fribourg, Switzerland, patrik.fuhrer@unifr.ch

http://diuf.unifr.ch/~fuhrer/

[MASSIVE-3] An ongoing development at the University of Nottingham of the HIVE project distributed VR Kernel, HIVEK. MASSIVE-3 is a distributed multi-user virtual reality system, current features of which include multiple users communicating via a combination of 3D graphics and real-time packet audio.

http://www.crg.cs.nott.ac.uk/research/systems/MASSIVE-3/

[MASSIVE95] C. Greenhalgh and S. Benford, “MASSIVE: A collaborative virtual environment for teleconferencing,” ACM Transactions on Computer-Human Interaction, Vol. 2, No. 3, pp. 239–261, 1995. And “MASSIVE: A distributed virtual reality system incorporating spatial trading,” in Proceedings IEEE DCS’95, Vancouver, Canada, 1995.

[MASSIVE99] C. Greenhalgh, "Large Scale Collaborative Virtual Environments", Springer 1999.

[MCG95] M.R. Macedonia, M.J. Zyda, D.R. Pratt, D.P. Brutzman, and P.T. Barham, “Exploiting reality with multicast groups,” IEEE Computer Graphics and Applications, pp. 38–50, 1995. and ”Exploiting Reality with Multicast Groups : A Network Architecture for Largescale Virtual Environments”, M.Macedonia, M.zyda, D. Pratt et al, IEEE VRAIS’95, pp2~10, 1995

[MOBJ95] "Architectural Support for Mobile Objects in Large Scale Systems", Caughey et.al, 1995

arch_agent_distr_vr.pdf

[MOSAIC] Mosaic was the name of the first web browser capable of understanding the HTTP protocol. 1994.

[Moser99] H. Moser and D. Thalmann, “A Rule-Based Interactive Behavioral Animation System for Humanoids,” IEEE Trans. Visualization and Computer Graphics, IEEE CS Press, vol. 5, no. 4, 1999, pp. 281-307.

[MSVW02] Microsoft Corporation. Virtual Worlds Group. [4 February 2002].

http://www.vworlds.org

[NavlSpace] Mark A. Foltz, "Designing Navigable Information Spaces". Washington University in St. Louis. 1998.

Designing Navigable Information Spaces mfoltz-thesis.pdf

[NOMAD01] Wilson S, Sayers H, McNeill MDJ. "Using CORBA middleware to support the development of distributed virtual environment applications". Proceedings of the 9th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2001 (WSCG’2001), Plzen, Czech Republic, February 5–9, 2001.

[NPSNET94] M.R. Macedonia, M.J. Zyda, D.R. Pratt, P.T. Barham, and S. Zeswitz, “NPSNET: A network software architecture for large scale virtual environments,” Presence: Teleoperators and Virtual Environments, Vol. 3, No. 4, pp. 265–287. Naval Postgraduate School, Department of Computer Science, Monterey, 1994.

Network Software Architecture For Large Scale Virtual Environments Presence.3.4.pdf

[NPSOFF94] "NPSOFF: An Object Description Language for Supporting Virtual World Construction", Michael J. Zyda*, Kalin P. Wilson, David R. Pratt,
James G. Monahan and John S. Falby, 1994

NPSOFF Language Virtual World Construction CG.17.4.pdf

[NVE-D&I99] S. Singhal and M. Zyda, "Networked Virtual Environments. Design and Implementation", Addison-Wesley, New York, 1999.

Mamoru Endo, Takami Yasuda, and Shigeki Yokoi Nagoya University, Japan, "A Distributed Multiuser Virtual Space System", 2003, Published by the IEEE Computer Society

distr_vspace03.pdf

[OpenGL-Perf] "OpenGL Performer Getting Started Guide". SGI. Tutorial

[OW] OpenWorlds

http://www.openworlds.com/

[P2P-ACAD] "Peer-to-Peer for Academia", Andy Oram is an editor at O'Reilly & Associates; 10/29/2001;

http://www.openp2p.com/pub/a/p2p/2001/10/29/oram_speech.html

[P2PS03] “Introduction to the Peer-to-Peer Sockets Project” by Brad Neuberg, 12/03/2003

http://www.onjava.com/pub/a/onjava/2003/12/03/p2psockets.html

[P2P-Shirky] “What is P2P... and What Isn't?” Clay Shirky, 2000

http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html?page=2

[PARADISE] Project at the Stanford Distributed Systems Group led by Professor David Cheriton.

http://www.dsg.stanford.edu/paradise.html

[PORTA01] "Shared virtual reality for design and management: the Porta Susa project", Luca Caneparo, Design Network Lab, Dipartimento di Progettazione architettonica, Politecnico di Torino, Õ.le Mattioli 39, I 10125 Turin, Italy; 2001 Elsevier Science B.V.
Porta Susa project.pdf

[VRBC] R.A. Issa Ed., State of the Art Report: "Virtual Reality in Construction, International Council for Research and Innovation in Building and Construction" CIB : http://www.bcn.ufl.edurtg24.

[Quake-BSP] Ben Humphrey, "Unofficial Quake 3 BSP Format".

www.gametutorials.com

[RING95] The Distributed Systems Research Group has T.A. Funkhouser, "RING:A client-server system for multi-user virtual environments". In Proceedings of the 1995 Symposium on Interactive 3D Graphics, 85-92. ACM SIGGRAPH, March 1995. And ”RING:A Client-Server System for Multiuser Virtual Environment, Symposium on Interactive 3D Graphics”, pp85~92, T. Funkhouser, 1995

[Rodden96] Tom Rodden. Populating the application: a model of awareness for cooperative applications. In Proceedings of the Conference on Computer Supported Cooperative Work, pp. 87–96, Cambridge, MA, USA, 1996.

[SDL-MM] Marco Kraus, "Multimedia Entwicklung mit SDL".

www.pl-berichte.de/work/sdl

[Sheep03] “The Interpretation of Dreams: An Explanation of the Electric Sheep Distributed Screen Saver” by Scott Draves, 12/22/2003

http://www.openp2p.com/pub/a/p2p/2003/12/22/sheep.html

[SIGCHI] ACM SIGCHI; Curricula for Human-Computer Interaction; Copyright © 1996 by the Association for Computing Machinery, Inc.

http://sigchi.org/cdg/

[Singal99] S.Singhal and M.Zyda, "Networked Virtual Environments Design and Implementation", ACM Press, Siggraph Series, vol23, 1999.

[SPLINE] Scalable Platform for Large Interactive Networked Environments (SPLINE) research project at Mitsubishi Electric Research Laboratories (MERL) led by Richard Waters and David Anderson.

http://www.merl.com/projects/spline/

[SPLINE96] J.W. Barrus, R.C. Waters, and D.B. Anderson, “Locales: Supporting large multiuser virtual environments,” IEEE Computer Graphics and Applications, Vol. 16, No. 6, pp. 50–57, 1996.

[SRM97] S. Floyd, V. Jacobson, Ching-gung Lin, S.McCanne and Lixia Zhang. "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing", IEEE/ACM Transactions on Networking, December 1997.

[SRM00] S. Kasera, et al, "Scalable Fair Reliable Multicast Using Active Services", IEEE Network Magazine, 14(1), 2000.

[Toshiya97] N. Toshiya and M. Yoshiyuki, “Wonder Space: Interactive 3D Animation Browser,” Proc. Siggraph 97, ACM Press, 1997, p. 111.

[VBVS01] "Virtual bodies and virtual spaces", J.M. Bishop, Department of Cybernetics, University of Reading, UK, 2001

Virtual bodies and virtual spaces.pdf

[VOY02] Recursion Software. "Recursion Software: Products – Voyager 4.5.",[4 February 2002].

http://www.objectspace.com/products/voyager/

[VR-Interface] Meredith Bricken, "Virtual Worlds: No Interface to Design". Human Interface Technology Laboratory (HITL), Washington Technology CenterUniversity of Washington. Tech Report HITL

http://www.hitl.washington.edu/publications/papers/interface.html

[VRML97] VRML Consortium, "VRML 97 - The Virtual Reality Modeling Language", ISO/IEC, 1997.

[VRTP] Web3D Consortium Web site, VRTP Working Group

http://www.web3d.org/WorkingGroups/vrtp

[VSCollab96] "Virtual Society: Collaboration in 3D Spaces on the Internet", RODGER LEA, Yasuaki Honda And Kouichi Matsuda, Sony Computer Science Laboratory Tokyo, Japan and Sony Architecture Labs, Tokyo, Japan, in final form 29 November 1996

v_soc97.pdf

[VSVM01] "Virtual Spaces and Virtual Manufacturing", Kraiem, Tunisia, IEEE 2001

Virtual spaces and virtual manufacturing.pdf

[X3D] Extensible 3D (X3D)

http://www.web3d.org/x3d/spec/ISO-IEC-19775/index.html

Bibliography and other Resources

Security in Co-authored Virtual Environments
Larsen, Christensen, DTU

Octree Tutorial; Ben Humphrey

www.gametutorials.com

Multimedia Entwicklung mit SDL; Marco Kraus

www.pl-berichte.de/work/sdl

Getting Started with the Java3D API
Sun Microsystems Tutorial

Real Time 3D Graphics with OpenGL
Chris Halsall, O'Reilly Network, 2000

Extending a Collaborative Architecture to Support Emotional Awareness
Garcia, Favela, Licea, Machorro, CICESE, Mexico

W3C Platform for Privacy Preferences Project (P3P)
Introduction Paper and v1.0 Deployment Guide

www.w3c.org/p3p

JXTA in a Nutshell
O'Reilly

Project JXTA: Java Programmer's Guide
Sun Microsystems, Inc., 2001

3D User Interfaces with Java3D
Jon Barrilleaux, Manning Publications, 2001

Java Network Programming
O'Reilly, 2nd Edition, Elliotte Rusty Harold, 2000

Database Nation - The death of privacy in the 21st century
O'Reilly, Simson Garfinkel, 2001

CODE - and other laws of cyberspace
Lawrence Lessig, Basic Books, 1999

Trust Economies in the Free Haven Project
Brian T. Sniffen, MIT, June 2000

PKI Security for JXTA Overlay Networks
Jeffrey E. Altman, IAM Consulting, February 2003

Peer-to-Peer Distributed Business Process
Atui Saini, Fiorano Software Inc, 2001

Peer-to-Peer Sharing of Web Applications
Robert J. Bayardo, Adina Costea, Rakesh Agrawal, IBM Research Report, Nov 2002

Escrow Services and Incentives in Peer-to-Peer Networks
Bill Horne, Benny Pinkas, Tomas Sander, Intertrust Tech, 2000

Reputation Systems
Communications of the ACM, Dec 2000

The value of reputation on eBay: A Controlled Experiment
Paul Resnick, Richard Zeckhauser, John Swanson, and Kate Lcokwood, June 2002

Poblano - A distributed trust model for peer-to-peer networks
Rita Chen and William Yeager, Sun Microsystems

A Distributed Trust Model
Alfarez Abdul-Rahman, Stephen Hailes, University College London

Supporting Structured Credentials and Sensitive Policies through Interoperable Strategies for Automated Trust Negotiation
Ting Yu, Marianne Winslett, University of Illinois, Kent E. Seamons, Brigham Young University, Feb 2003

A Trust Model for Peer-to-Peer Content Distribution Networks
Guillaume Pierre, Maarten van Steen, Vrije Universiteit, Amsterdam

Building Trust in Decentralized Peer-to-Peer Electronic Communities
Li Xiong, Ling Liu, Georgia Inst. of Tech., ICECR-5

Privacy and Security in Location-based Systems with Spatial Models
Christian Hauser, University of Stuttgart, 2001

Supporting Trust in Virtual Communitites
Alfarez Abdul-Rahman, Stephen Hailes, University College London

Dynamic Trust Models for Ubiquitous Computing Environments
Coling English, Paddy Nixon, Sotirios Terzis, University of Strathclyde, Glasgow

A survey of Trust in Internet Applications
Tyrone Grandison, Morris Sloman, IEEE, 2000

Merging and Extending the PGP and PEM Trust Models - The ICE-TEL Trust Model
Chadwick, Young, Cicovic

Trust Model - Defining and Applying Generic Trust Relationship in a Networked Computing Environment
Dr. Jack Stinson, Stephen V. Pellissier, Archie D. Andrews, ATI IPT, May 2000

Autonomous Cooperating Web Crawlers
Gregory Louis McLearn, Thesis, University of Waterloo, 2002

Booch, G. and Rumbaugh, J. and Jacobson, I. 1999. The Unified Modeling Language User Guide. Addison Wesley.

Flanagan, D. 1997. JAVA in a Nutshell, Second Edition. O'Reilly.

Larman, G. 1998. Applying UML and Patterns. Prentice Hall.

Lea, R. and Matsuda, K. and Miyashita, K. 1996. Java for 3D and VRML Worlds. New Riders.

Oaks, S. and Wong, H. 1999. JAVA Threads, Second Edition. O'Reilly.

Rumbaugh, J. and Jacobson, I. and Booch, G. 1999. The Unified Modeling Language Reference Manual. Addison Wesley.

Schach, S. 1999. Classical and Object-Oriented Software Engineering with UML and JAVA, Fourth Edition. McGraw-Hill.

Stroustrup, B. 1997. The C++ Programming Language, Third Edition. Addison Wesley.

DIVE - Distributed Interaction Virtual Environment

http://www.sics.se/dive/

Distributed Virtual Environments Bibliography

http://www.hitl.washington.edu/kb/distvr/

"Project JXTA: Technical Shell Overview", Sun Microsystems, Inc., April 25, 2001

TechShellOverview.pdf

PowerPoint presentation about Networked Virtual Environments

Networked Virtual Environments presentat.pdf

"Multicast Grouping For Data Distribution Management", Katherine L. Morse, Michael Zyda, 2001

Multicast Grouping For Data Distribution Management MorseSIMPRA2001.pdf

O'Reilly P2P Directory What's New – The O'Reilly P2P directory lists companies, projects and initiatives related to peer-to-peer technologies.

http://www.openp2p.com/pub/q/p2p_category

"The Graphical User Interface. Time for a Paradigm Shift?", Christine Zmoelnig, MA Hypermedia Studies.

http://www.sensomatic.com/chz/gui/index.html

"Hot Virtual Reality Sites"

http://www.itl.nist.gov/iaui/ovrt/hotvr.html#Academia

"Graphical User Interface Timeline" by Nathan Lineback

http://toastytech.com/guis/guitimeline.html

Google directory Computers > Virtual Reality > Multi-User Systems

http://directory.google.com/Top/Computers/Virtual_Reality/Multi-User_Systems/?il=1

Jakob Nielsen's useit.com on usability

Web Resources for Java

http://www.javootoo.com

http://backend.userland.com/directory/167/howtosarticles

Web Resources for XML, SAX Parser, and XML Schema

http://www.xmlhack.com/read.php?item=2023

http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JAXPSAX3.html

http://java.sun.com/xml/jaxp/index.html

http://www.saxproject.org

Web Resources for Java Swing

http://www.jguru.com/faq/Swing

http://java.sun.com/products/jfc/tsc/

http://java.sun.com/products/jfc/tsc/articles/mixing/index.html

http://java.sun.com/products/jfc/tsc/articles/threads/threads1.htm

http://www.oyoaha.com/lookandfeel/help.html

Computer Graphics Group, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology

http://graphics.lcs.mit.edu/publications.html

"Optimization of Motion Transmission for Virtual Actors", Bernhard Spanlang 9356122; GUP University Linz, Austria, Last update: 07.05.1999; This work focuses on optimization methods for information transmission of animated virtual objects.

1997, Robert Rockwell, Black Sun Interactive

http://www11.informatik.tu-muenchen.de/lehre/seminare/seminarWS9798/rockwell/spectrum/

"A Peer-To-Peer Message Exchange Scheme For Large Scale Networked Virtual Environments", 2002, Yoshihiro Kawahara, Hiroyuki Morikawa, Tomonori Aoyama

kawahara02peertopeer.pdf

"Distributed Applications for Collaborative Augmented Reality", Dieter Schmalstieg, Gerd Hesina; This paper focuses on the distributed architecture of the collaborative augmented reality system Studierstube.

distributed-applications-for-collaborative.pdf

Internet2
A consortium being led by 205 universities working in partnership with industry and government to develop and deploy network applications and technologies.

http://www.internet2.edu

ExtremeTech 3D Pipeline Tutorial, June 13, 2001

http://www.extremetech.com/article2/0,1558,9722,00.asp

Cisco technology highlights

http://www.cisco.com/en/US/tech/

Game Server Statistics

http://www.serverspy.net/site/

"Entertainment R&D for Defense"; Michael Zyda, John Hiles, Alex Mayberry, Casey Wardynski, Michael Capps, Brian Osborn, Russell, Shilling, Martin Robaszewski, and Margaret Davis, The Moves Institute, IEEE February 2003

Entertainment R&D for Defense MOVES-IEEE-CGA-2003.pdf

Appendix II.Glossary

Abbreviations and Acronyms

ACK	Acknowledgment
ADU	Abstract Data Unit
AI	Artificial Intelligence
AO	Application Object
AOI	Area of Interest; see Zones
API	Application Programming Interface
AR	Augmented Reality
Blog	Weblog - a web-based diary generally focused on specific subject area
BSP	Binary Space Partitioning
CAD	Computer Aided Design
CORBA	Common Object Request Broker
CP	Community Place
C/S	Client/Server model
CSCW	Computer Supported Cooperative Work (sometimes Collaborative)
DaBP	Dial-a-Behavior Protocol
DCOM	Distributed Component Object Model
DIS	Distributed Interactive Simulation
DIVE	Distributed Interaction Virtual Environment
DNS	Domain Name System
DOM	Document Object Model - platform and language neutral interface that allows programs to dynamically access and modify content, structure and style of documents
DNS	Domain Name Service; translates internet site names to their numeric addresses
DR	Designated Receiver
DRM	Display Relative Mapping
DS	Designated Server
DSM	Distributed Shared Memory
DVE	Distributed Virtual Environment
DWTP	Distributed World's Transfer and Communication Protocol
FOV	Field of View
GPU	Graphical Processing Unit - similar to a CPU, but with specific support for graphical algorithms
GUI	Graphical User Interface
HCI	Human Computer Interaction
HIVE	Large Scale Real Time Multi-User Virtual Reality Research
HORB	Lightweight Java Object Request Broker
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
HUD	Heads-Up Displays are usually placed in front of the 3D scene (used for feedback and control in VE's and games)
ICQ	'I seek you' application for internet chat based on messaging (others include Yahoo and MSN Instant Messengers, and Trillian)
IDE	Integrated Development Environment
IIOP	Internet Inter-ORB Protocol - communication for distributed objects based around the CORBA architecture; language/platform independent
IM	Instant Messaging
IK	Inverse Kinematics
IP	Internet Protocol (Layer 3)
IPv6	Internet Protocol version 6 - reserves larger addressing space and includes new options; successor of current IPv4
IRC	Internet Relay Chat

J2EE	Java 2 Enterprise Edition
J3DUI	Java3D user interface framework that provides convenience classes
Java3D	The Java 3D API
JDBC	Java Data-Base Connectivity
JVM	Java Virtual Machine
LAD	Look-at Direction
LOA	Level-of-Articulation
LOD	Level of Detail
MaDViWorld	Massively Distributed Virtual World
MASSIVE	Model, Architecture and System for Spatial Interaction in Virtual Environments system
ML	Mapping Layer
ms	milliseconds
MUD	Multi-user dungeon; text-based virtual adventure games played on the internet, mostly based on the Telnet protocol
MVC	Model-View-Control
NACK	Negative acknowledgment or "repair request", i.e. when a packet was received in a corrupted state
Net-VE	Networked Virtual Environments
NFS	Network File System
NPC	Non-player characters (in games)
NSG	Network Scene Graph
OAA	Open Avatar Architecture
OMG	Object Management Group
ORB	Object Request Broker
OSI	Open System Interconnection - communications framework model that includes 7 layers of communication organized according to events and occurrences
P2P	Peer-to-Peer
PARADISE	Performance Architecture for Advanced Distributed Interactive Simulation Environments
PARIS	Personal Augmented Reality Immersive System
POV	Point of view
PVS	Potentially Visible Sets (similar to BSP; method for determining surfaces of objects actually in view at any given time and location)
RMTP	Reliable Multicast Transport Protocol
SETI	Search for Extraterrestrial Intelligence; global scientific project aimed at discovering life elsewhere in the universe through the detection of signals; through a screen saver application, remote processing power is used to distribute the workload
SG	Scene Graph
SGML	Standard Generalized Markup Language
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol; enables object communication over the internet, relies on XML and runs on top of HTTP to avoid firewalls
SOFT	Software Framework for Tele-immersion
SPLINE	Scalable Platform for Large Interactive Networked Environments
SRM	Scalable Reliable Multicast
SSG	Standard Scene Graph
SSS	Simple Shared Script
SVR	Shared Virtual Reality
TAWS	Totally Active Workspace
TCP	Transmission Control Protocol
UDP	User Datagram Protocol; lost packets and out of order packets are not handled
UI	User Interface
UID	Unique Identifier
VE	Virtual Environment
VoIP	Voice of IP

VRML	Virtual Reality Modeling Language [VRML97]
VSCP	Virtual Space Client Protocol or Virtual Society Client Protocol
VSTP	Application protocol for shared VE's based on top of standard Internet protocols such as TCP/IP
VUI	Virtual User Interface
X3D	Extensible 3D; International Draft Standard using XML file format
XML	Extensible Markup Language
XML-RPC	A very simple remote procedure call protocol encoded in XML.

Definitions and Connotations

Illustration 7.1. TerraPeer Terminology and Conventions on page 152 is a collection of conventions used, and acts as a reference.

3D Space or **3D World** are often used interchangeable (also with VE), but the 'space' focuses more on the VE as a whole (including the graphics), while the 'world' is more a perspective from the user's (as a tool; in social context) point-of-view.

Advertisements are JXTA's language-neutral meta data structures that describe peer resources such as peers, peer groups, pipes, and services. Advertisements are represented as XML documents.

Avatar - In Hinduism an avatar is the Terrestrial incarnation of a god or goddess. In SVR an avatar is the user ‘‘incarnation’’ in cyberspace. Avatars can be defined as the "virtual representations of the users" [DVE-VJC01].

Aura - The aura is the pre-definable radius of visibility and interaction among avatars.

Cyberspace - According to the definition of William Gibson, cyberspace is the total digital network, a place of meeting and communication.

Binding is an implementation of the Project JXTA protocols for a particular environment (e.g., the J2SE platform binding).

Codat – Containers objects that are used to hold any kinds of objects or data content (commonly a document or file).

DVE – The Distributed Virtual Environment is a multi-user VE extended with a common network infrastructure.

The **environment** has some basic settings that are equal for all peers, but otherwise consists of zones and objects. A view of the environment is presented to the user. The **Virtual Environment** (VE) represents the pure (single-user) 3D space, and usually all technical aspects required to run it as well. The **Distributed Virtual Environment** (DVE) is a multi-user VE extended with a common network infrastructure.

JXTA ID – Credential or a token used to uniquely identify the sender of a message; can be used to provide message authorization.

JXTA - JXTA is not an acronym, and in particular the “J” does not refer to Java. JXTA is a made up word coined by the project's original sponsor, Bill Joy. JXTA is derived from the word Juxtapose, as in side by side. It is a recognition that peer-to-peer is juxtaposed to client/server or Web based computing -- what is considered today's traditional computing model.

jux·ta·pose tr. v. To place side by side, especially for comparison or contrast.

JXTA Core Specification - The JXTA Core Specification consists of the required components and behaviors which are present in all conforming JXTA implementations. This includes the Peer Endpoint Protocol (PEP) and the Peer Resolver Protocol (PRP).

JXTA ID Format - A JXTA ID Format is a scheme for representing it IDs of JXTA entities. Each ID Format is identified by as sub-namespace of the URN namespace "jxta".

JXTA ID Type - A JXTA ID Type is describes the characteristics of JXTA IDs which refer to a particular sort of JXTA entity. Currently this includes peer groups, peers, codats, pipes, module classes, module specifications and module implementations, but may be extended to refer to other types of entities in the future or in specific implementations.

JXTA Standard Services - The JXTA Standard Services are optional JXTA components and behaviors. Implementations are not required to provide these services, but are strongly RECOMMENDED to do so. This includes Peer Discovery Protocol (PDP), Peer Information Protocol (PIP), Pipe Binding Protocol (PBP) and Rendezvous Protocol (RVP).

Mosaic was the name of the first web browser capable of understanding the HTTP protocol.

Multi-user worlds are "several users working on different machines can move through the world and interact with one another or with the objects at the same time" [DVE-VJC01]

Module - An abstraction used to represent any piece of "code" used to implement a behavior in the JXTA world. Network services are the mode common example of behavior that can be instantiated on a peer.

Module Class - Represents an expected behavior and an expected binding to support the module; is used primarily to advertise the existence of a behavior.

Module Implementation - The implementation of a given module specification; there may be multiple module implementations for a given module specification.

Module Specification - Describes a specification of a given module class; it is one approach to providing the functionality that a module class implies. There can be multiple module specifications for a given module class. The module specification is primarily used to access a module.

Objects populate the environment. The state of an object is completely described by its attributes. A **Virtual Object** is sometimes called an 'entity', and could be any geometric shape placed within the VE. An object can represent a **Service**, and as such has additional properties. **Basic Objects** are simply graphical nodes (or collections of) in a **Scene Graph** (SG). **Service Objects** extend the basic ones, and add properties, such as a URL.

P2P or Peer-to-Peer - A decentralized networking paradigm in which distributed nodes, or peers, communicate and work collaboratively to provide services.

Peers are applications that communicate to each other on a distributed (P2P) network using some protocol. A peer can also be used as a notion for 'nodes' in a network model, or for 'homes' in a virtual world context (i.e. a “peer zone” could be equivalent to a “homepage”). A peer process does not need a server. All peers are created equal and communicate directly with other peer processes.

Peer Group - A collection of peers that have a common set of interests and have agreed upon a common set of services.

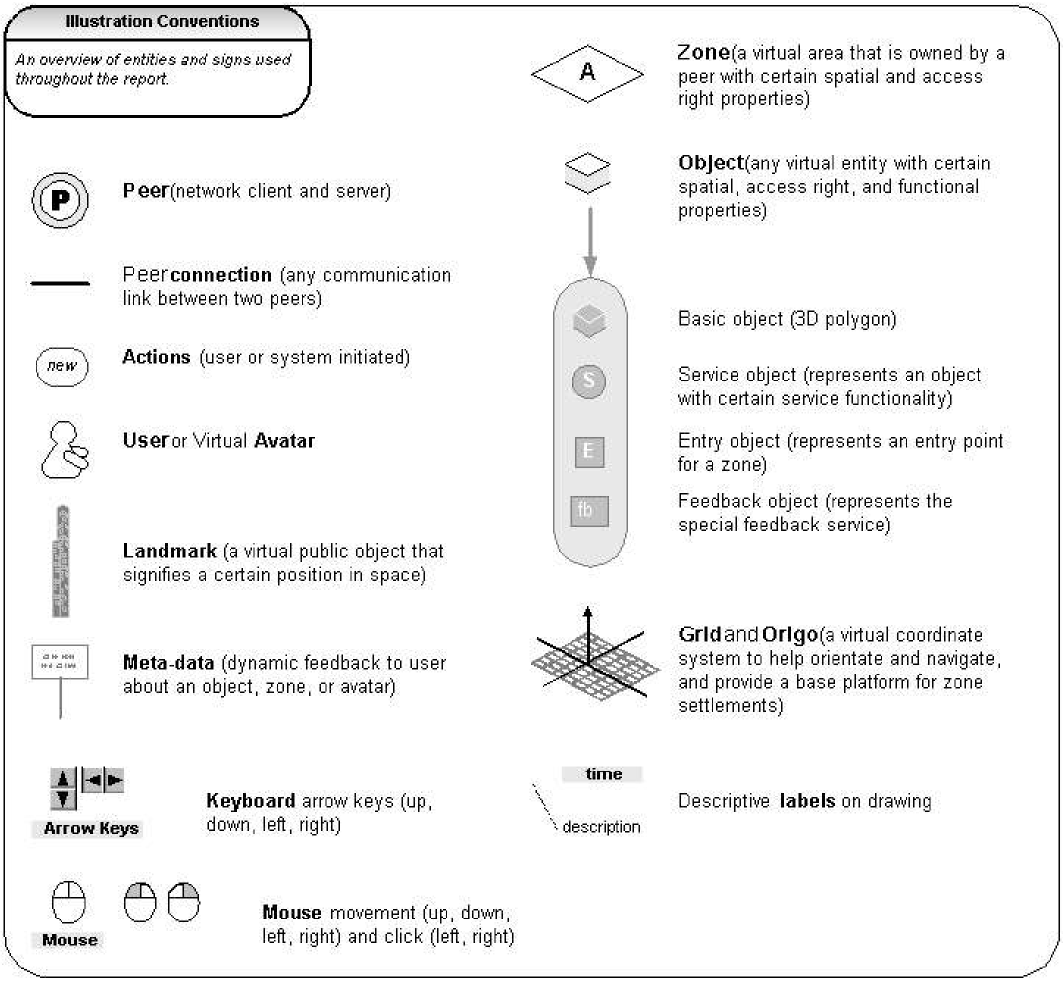


Illustration Conventions
An overview of entities and signs used throughout the report.
Zone(a virtual area that is owned by a peer with certain spatial and access right properties)
Object(any virtual entity with certain spatial, access right, and functional properties)
Peer(network client and server)
Peerconnection (any communication link between two peers)
Basic object (3D polygon)
Actions (user or system initiated)
new
Service object (represents an object with certain service functionality)
User or Virtual Avatar
Entry object (represents an entry point for a zone)
Feedback object (represents the special feedback service)
Landmark (a virtual public object that signifies a certain position in space)
Grid and Origo(a virtual coordinate system to help orientate and navigate, and provide a base platform for zone settlements)
Meta-data (dynamic feedback to user about an object, zone, or avatar)
Arrow Keys
Keyboard arrow keys (up, down, left, right)
time
description
Descriptive labels on drawing
Mouse
Mouse movement (up, down, left, right) and click (left, right)

Illustration 7.1. TerraPeer Terminology and Conventions

A **process** runs on one machine and handles the computational resources used to generate a view for one user. A process corresponds to a peer.

Service – can be any kind of resource, for example a link, a chat-room, a file-sharing container, a photo-album, or e-mail. The **Virtual Service** (a.k.a. Service Object) does not necessarily be functional in the VE, but might launch external applications.

Space or **World** are often used interchangable, but the 'space' focuses more on the VE as a whole (including the graphics), while the 'world' is more a perspective from the user's (as a tool; in social context) point-of-view. **Virtual worlds** in general are defined as "computer-based models of three-dimensional spaces of objects with restricted interaction" [DVE-VJC01].

System, as in VE or DVE system, means the entire software setup, and might include the hardware as well.

SVR or Shared Virtual Reality - "SVR differs from Virtual Reality VR in that the experience of virtual spaces is no longer individual, but rather shared across the Internet with other users simultaneously connected. SVR offers an effective approach to Construction Data Model and Computer Supported Collaborative Work, because it integrates both the communicative tools to improve collaboration and the distributed environment to process information across the networks" [PORTA01].

User – an entity (not necessarily human) whom a view of the environment is presented to.

View – a view of the environment is generated under the control of one process on one machine.

Visibility Radius – see Zone and 'Aura'.

Virtual Environment - **VE's** are distributed "if active parts of it are spread throughout different computers in a network" [DVE-VJC01]. The Virtual Environment represents the pure (single-user) 3D space, and usually all technical aspects required to run it as well.

A **Virtual Object** is sometimes called an '**entity**', and could be any geometric shape placed within the VE.

Virtual Worlds are defined as "computer-based models of three-dimensional spaces of objects with restricted interaction" [DVE-VJC01].

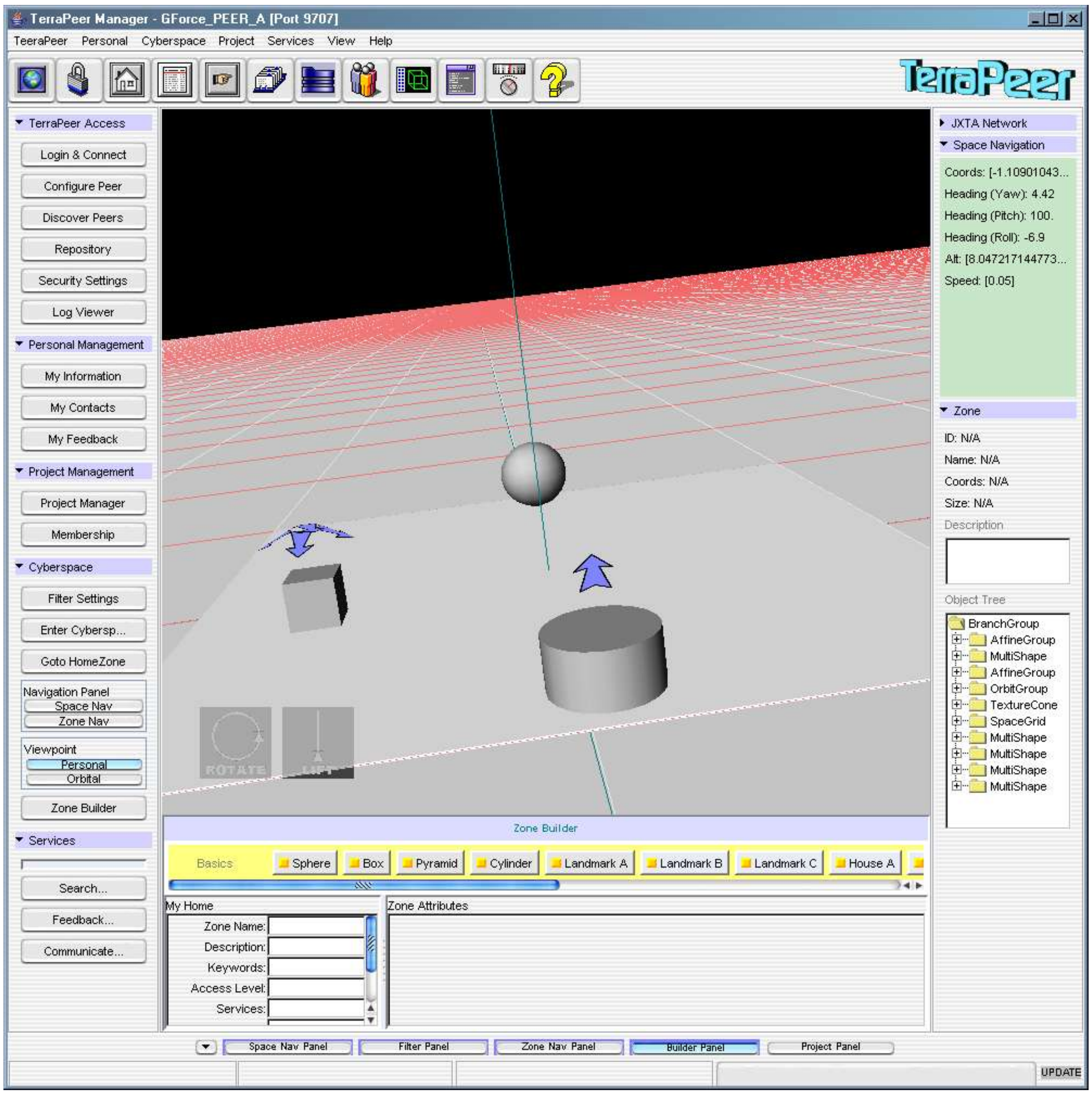
VRML is the Virtual Reality Markup or Modeling Language. At the end of 1997 a revised version VRML97 became an official ISO-standard for web-based 3D graphics (ISO/IEC 14772). [VRML97]

X3D is the successor to the VRML. X3D improves upon VRML with new features, advanced application programmer interfaces, additional data encoding formats, stricter conformance, and a 'componentized' architecture that allows for a modular approach to supporting the standard.

Zones – a bound virtual space representing a peer (sometimes a group of peers); zones in TerraPeer represent the metaphor of a 'home' and are leveled planes restricted in size and located at specific coordinates in the VE. In a different context, a zone can be a network architecture that defines a mechanism to reduce network traffic among DVE users by dividing the VE space into areas, or zones

Appendix III.Screenshots

Main application screenshot with all available controls and feedback, and the zone builder south panel; objects in the VE are selected and display feedback on current operation (lifting, sliding)



Coords: [-1.10901043...
Heading (Yaw): 4.42
Heading (Pitch): 100.
Heading (Roll): -6.9
Alt: [8.047217144773...
Speed: [0.05]

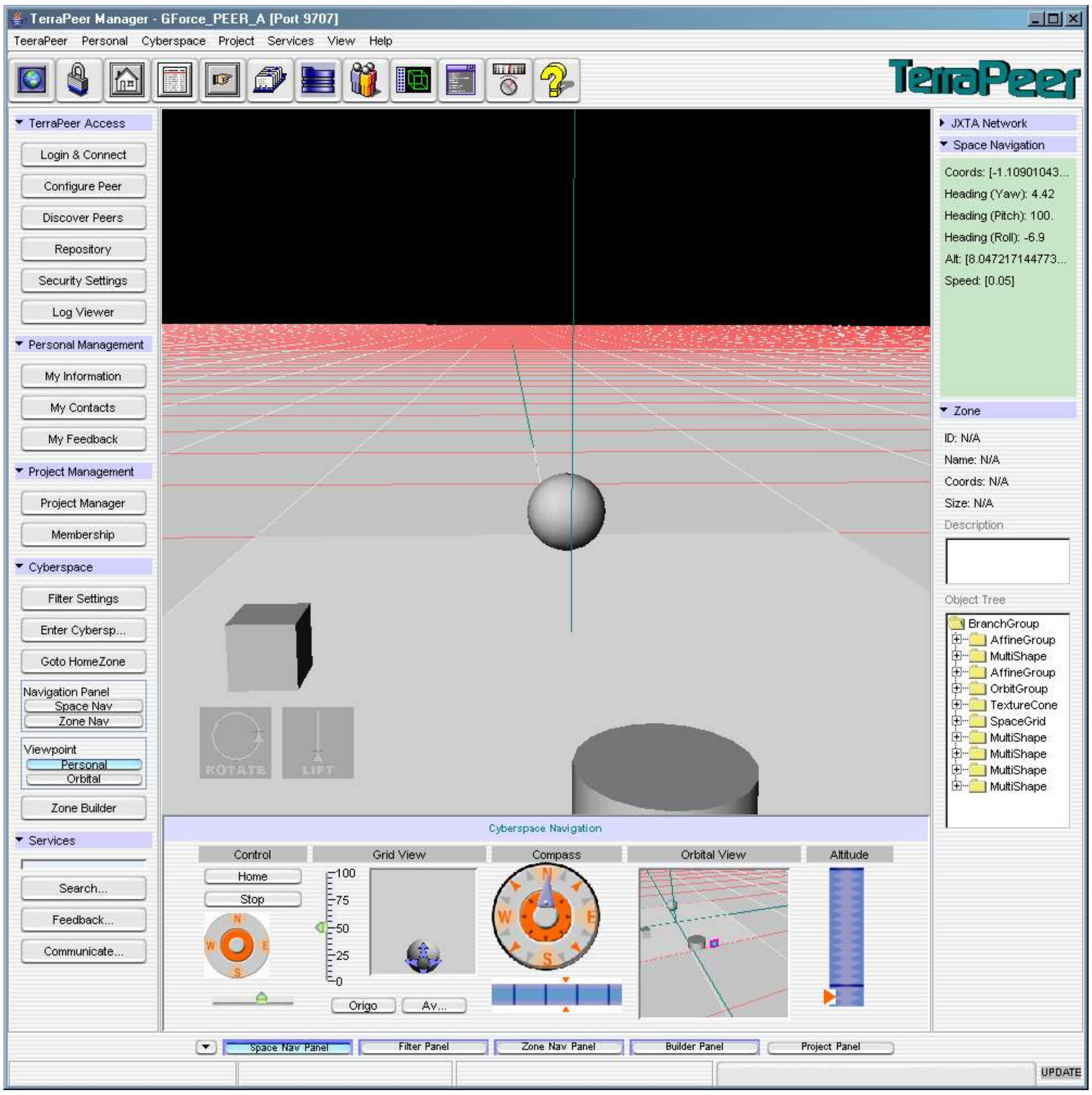
ID: N/A
Name: N/A
Coords: N/A
Size: N/A

BranchGroup
AffineGroup
MultiShape
AffineGroup
OrbitGroup
TextureCone
SpaceGrid
MultiShape
MultiShape
MultiShape
MultiShape

Zone Name:
Description:
Keywords:
Access Level:
Services:

Illustration 7.2. Application screenshot 0

Main application screenshot with space navigation south panel



Coords: [-1.10901043...
Heading (Yaw): 4.42
Heading (Pitch): 100.
Heading (Roll): -6.9
Alt: [8.047217144773...
Speed: [0.05]

ID: N/A
Name: N/A
Coords: N/A
Size: N/A

Illustration 7.3. Application screenshot 1

Main application screenshot with hidden side panels

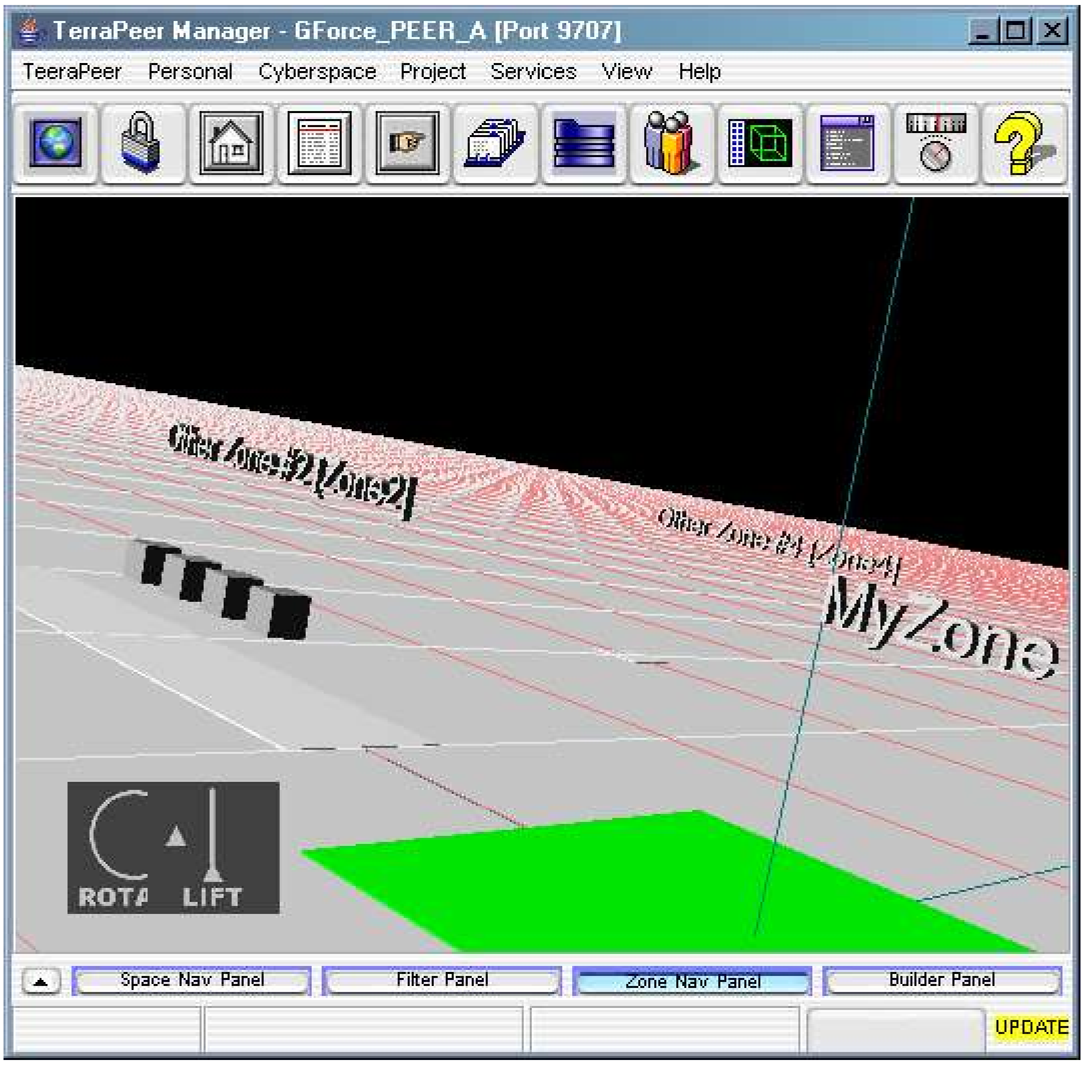


Illustration 7.4. Application screenshot 2

Main application screenshot with 'MyZone' in VE display foreground

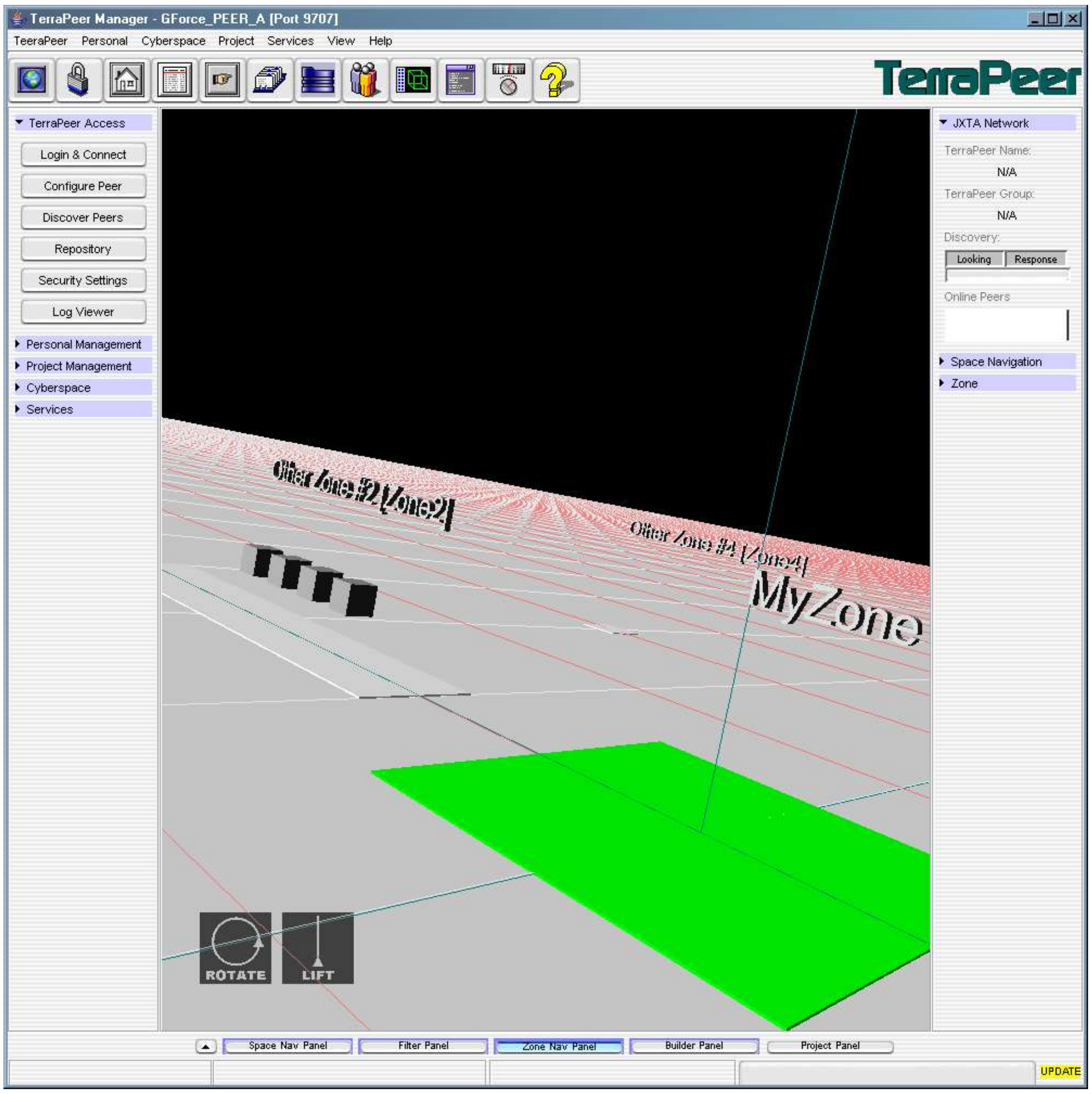


Illustration 7.5. Application screenshot 3

Main application screenshot

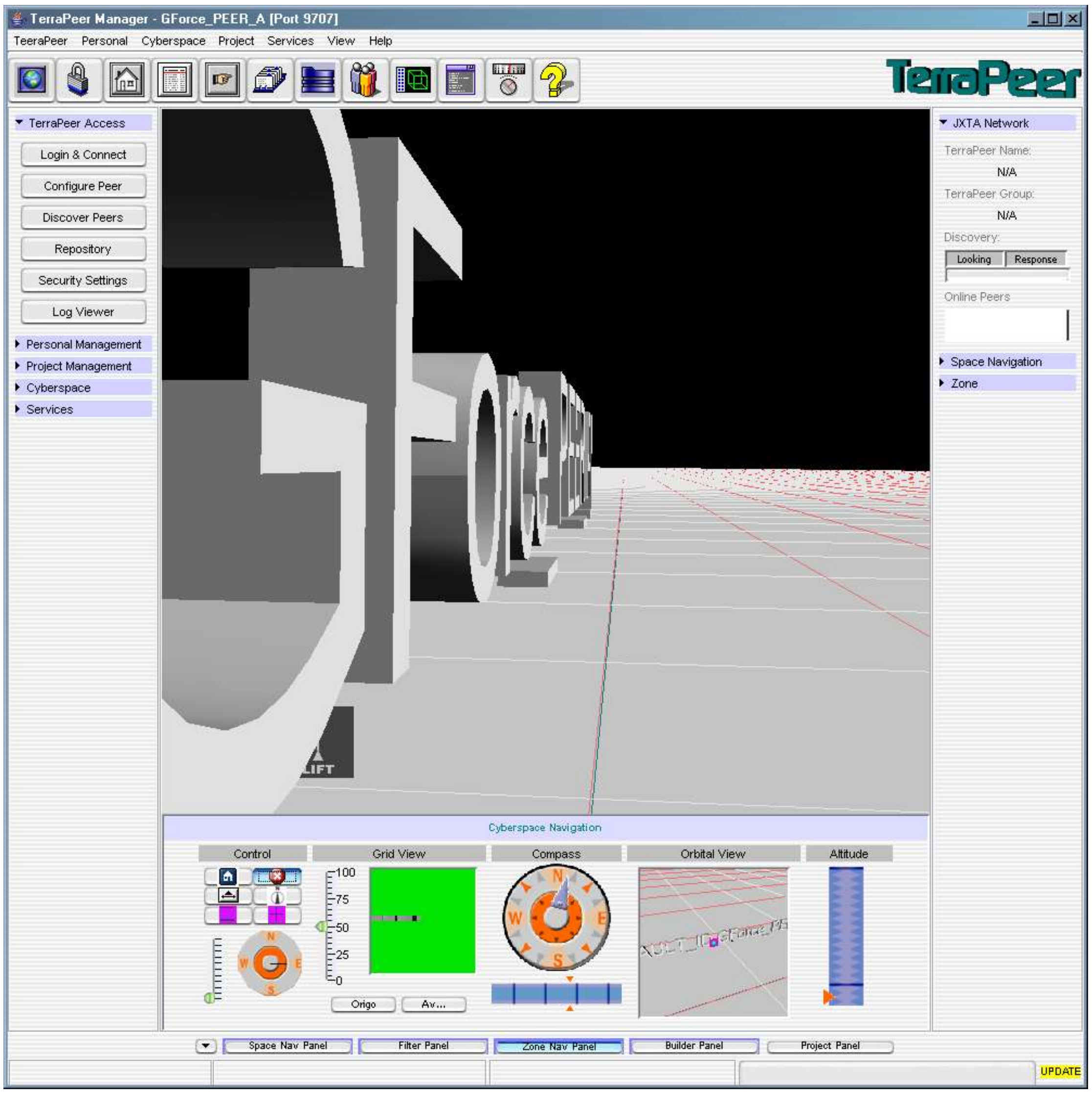


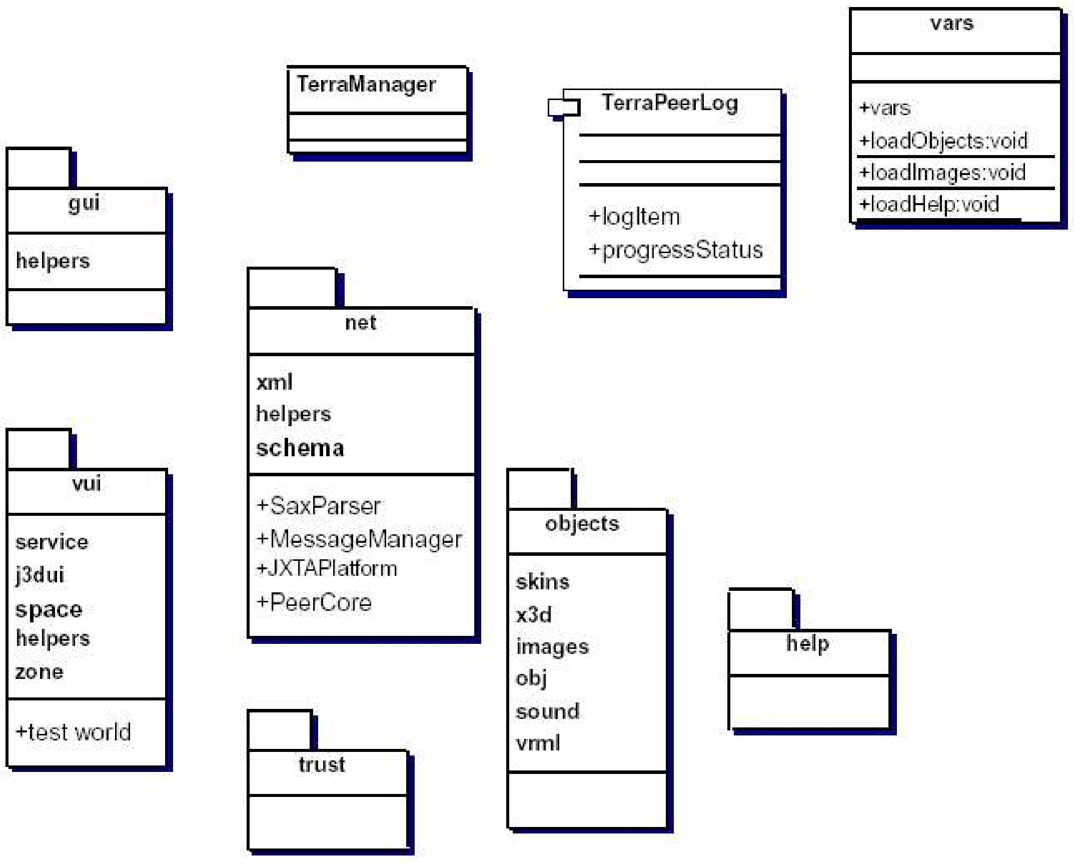
Illustration 7.6. Application screenshot 4

Appendix IV.Application Tutorial

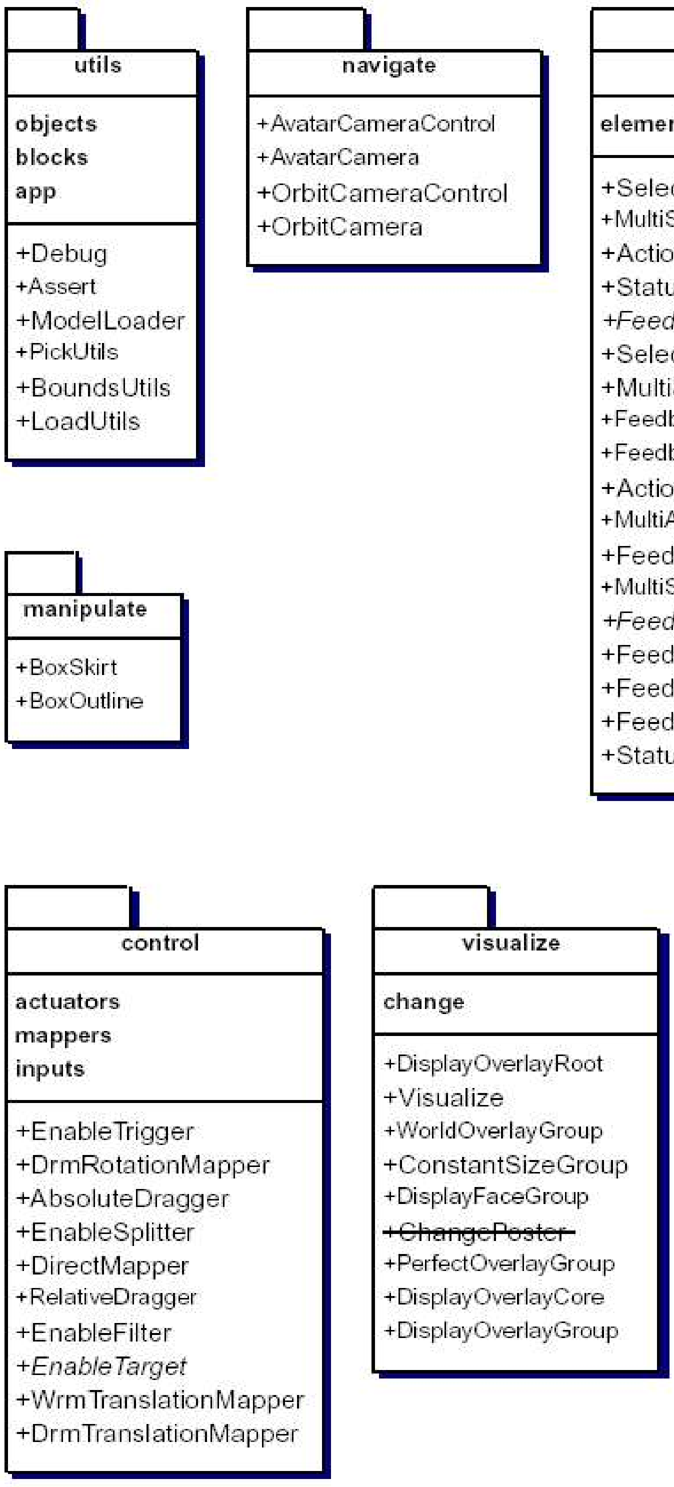
Please go to the project website for a tutorial.

Appendix V.UML Design

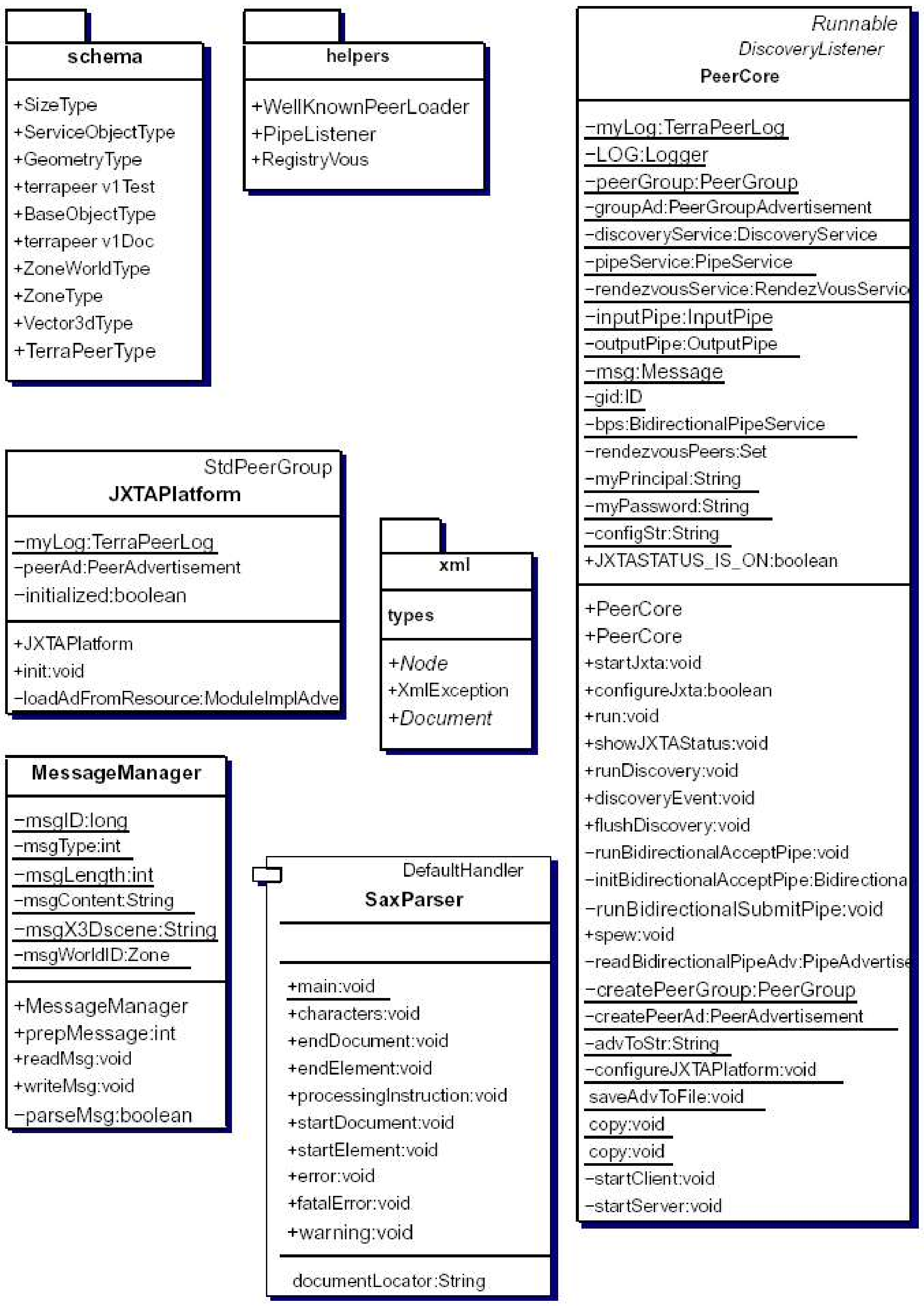
Package overview



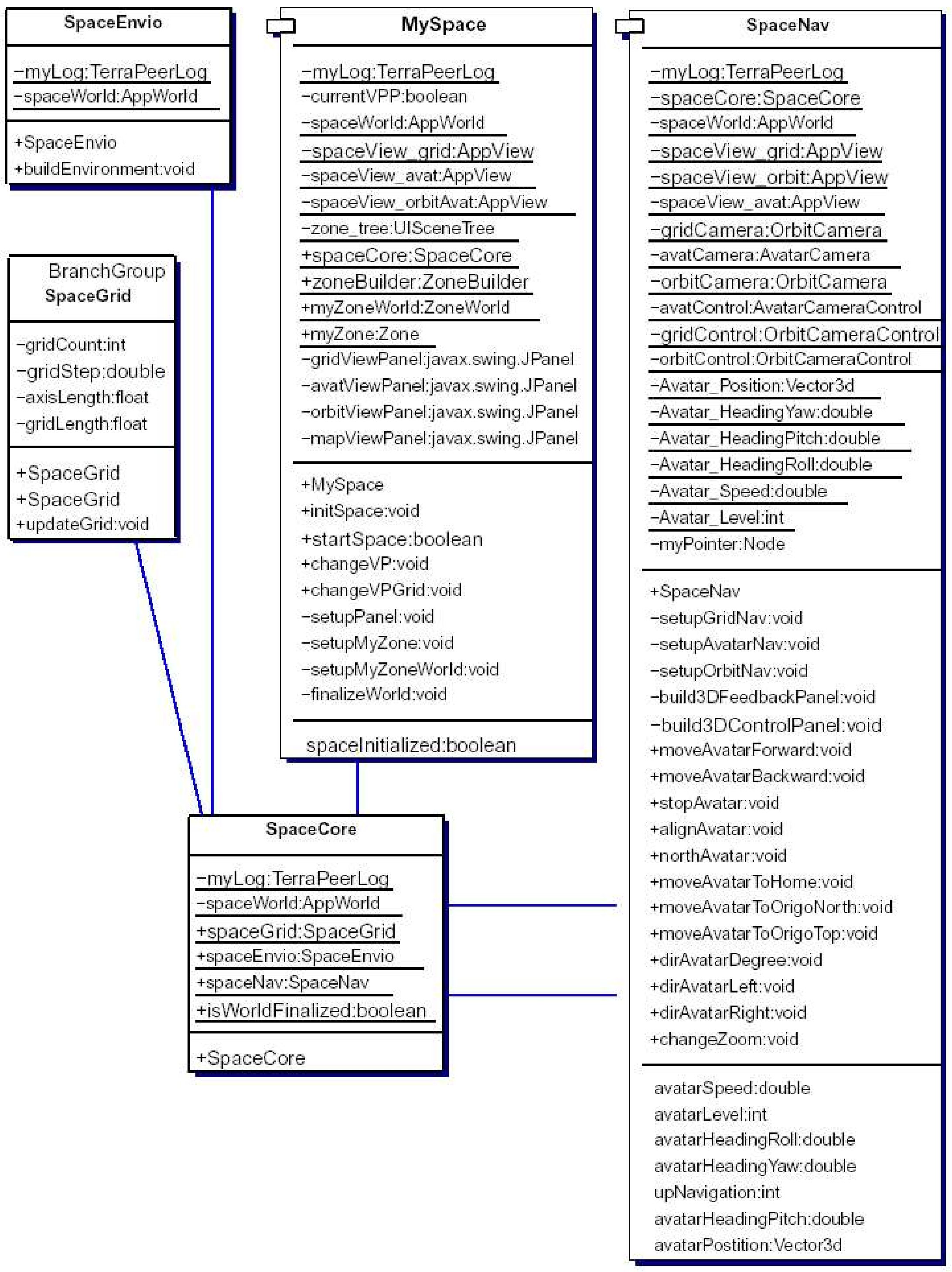
3D UI – package: terrapeer.vui.j3dui



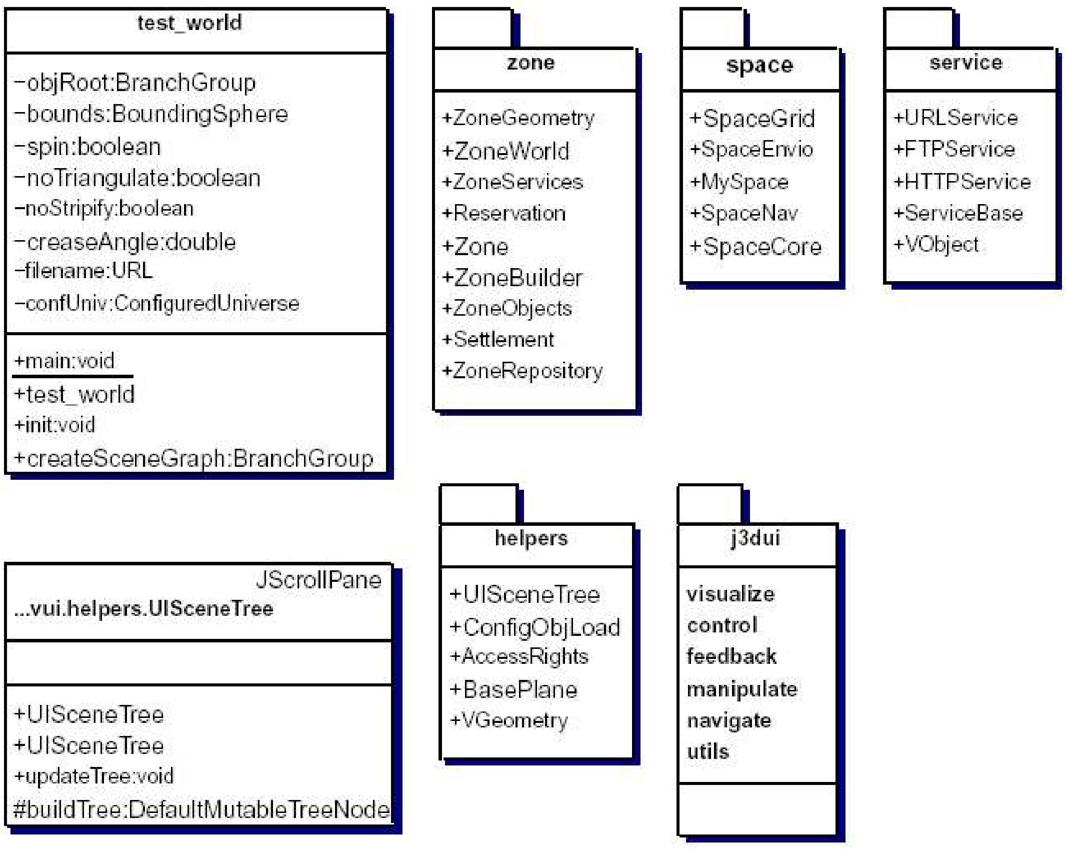
Network – package: terrapeer.net



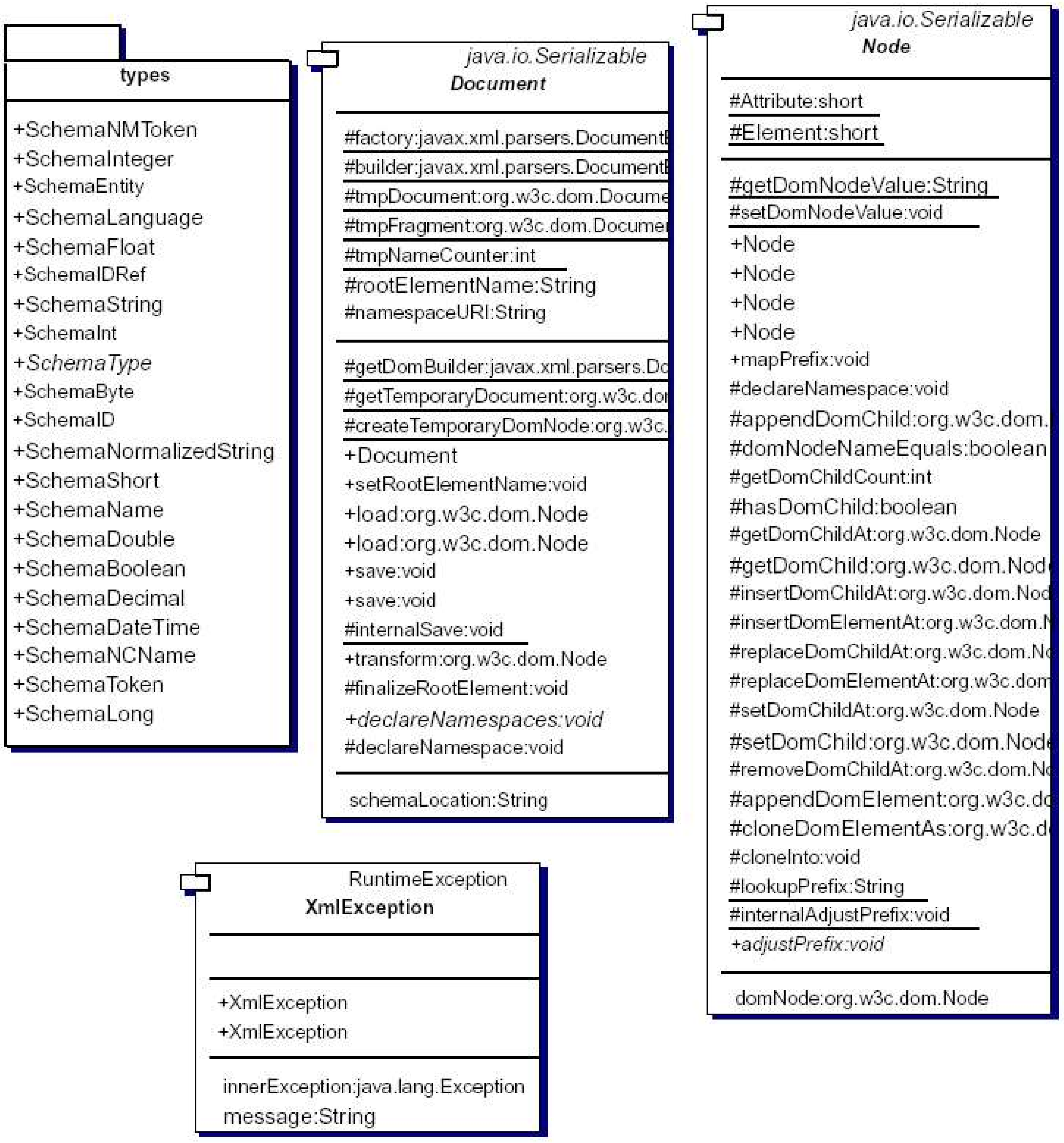
Space – package: terrapeer.vui.space



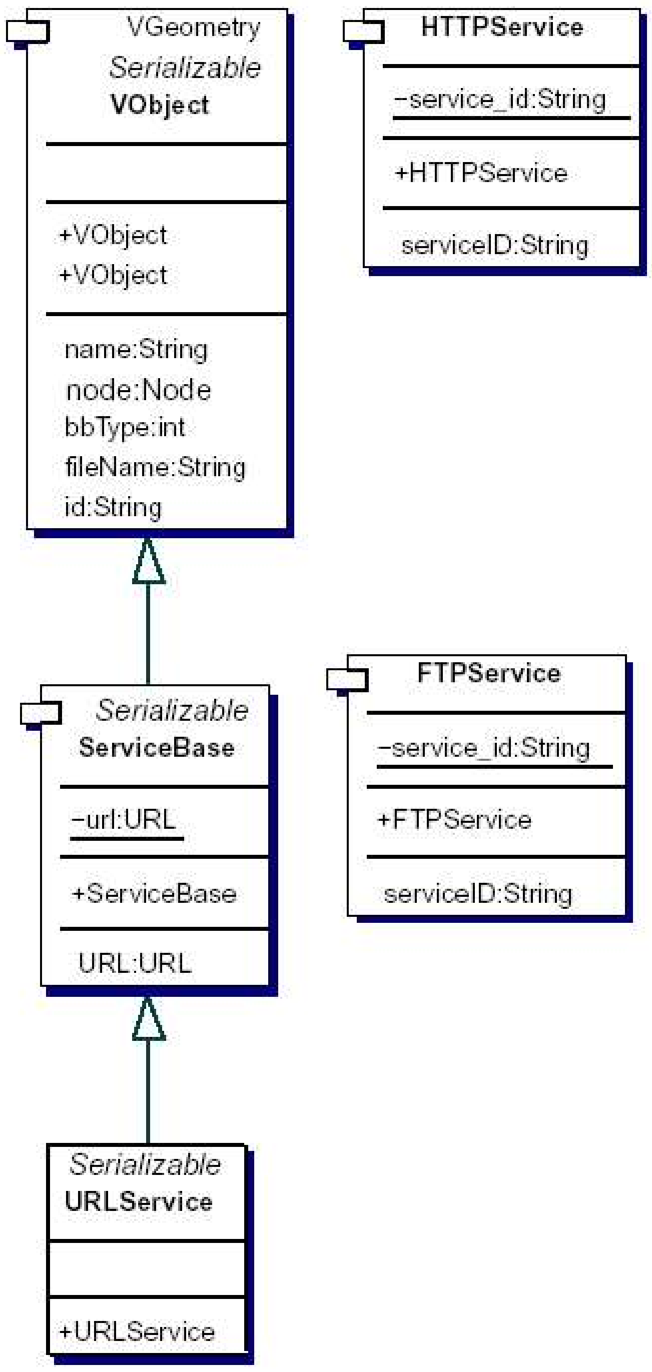
VUI – package: terrapeer.vui



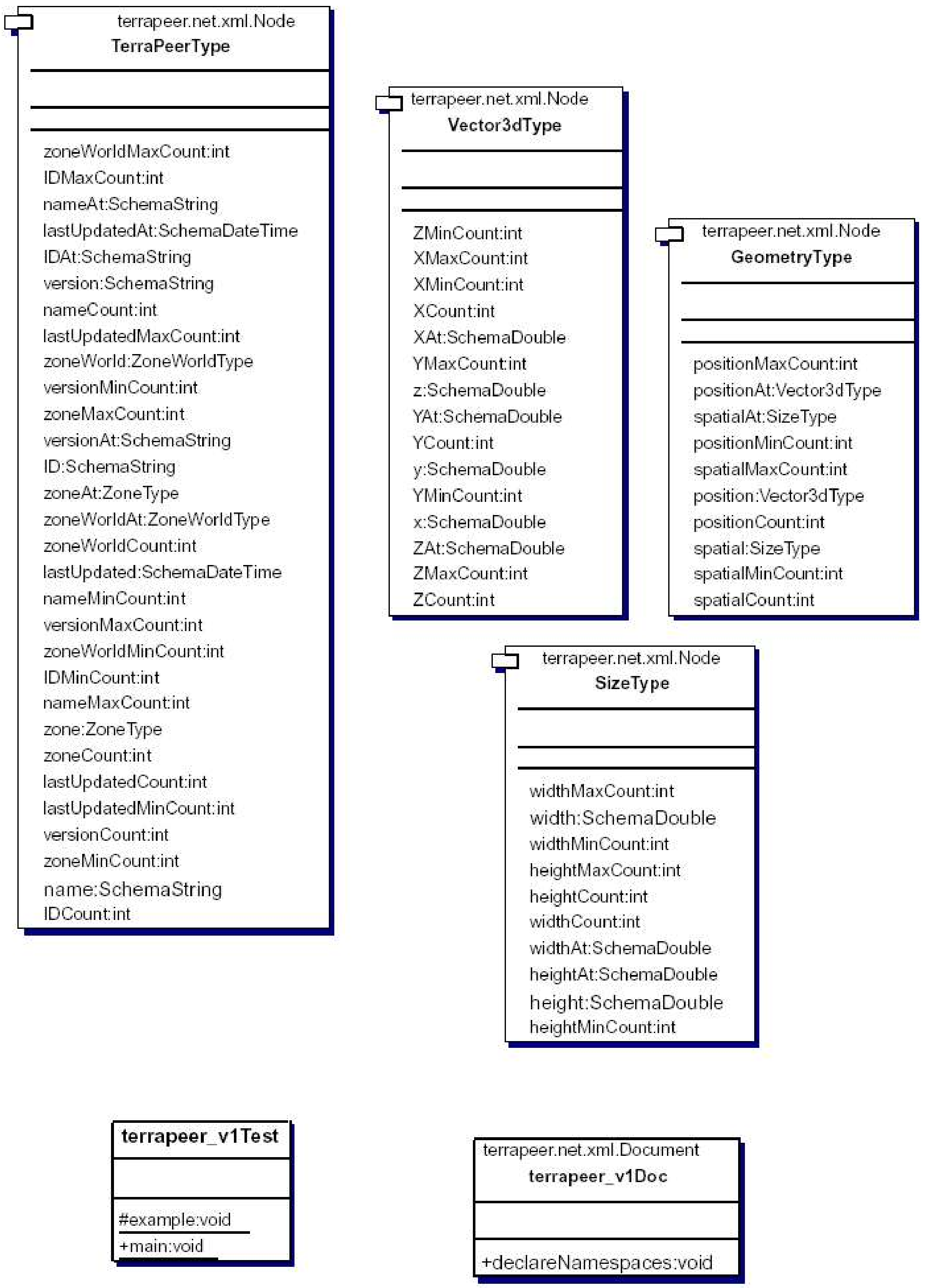
XML – package: terrapeer.net.xml



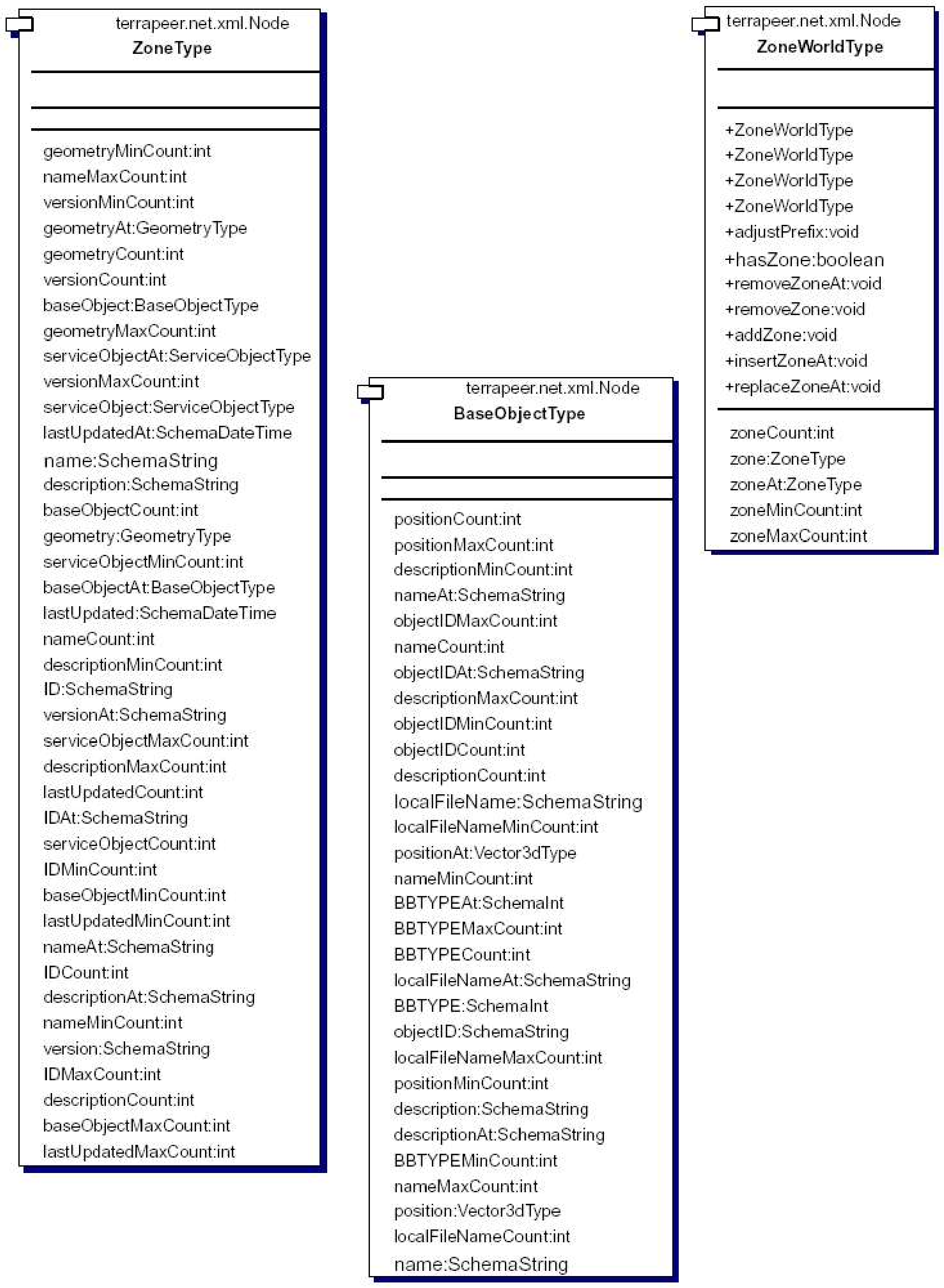
Service – package: terrapeer.vui.service



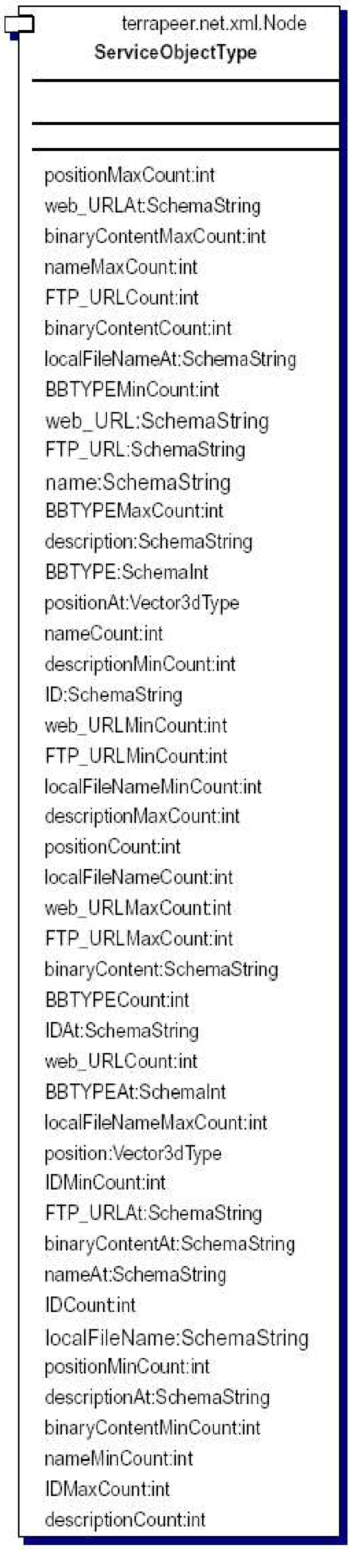
Schema 1 – package: terrapeer.net.schema



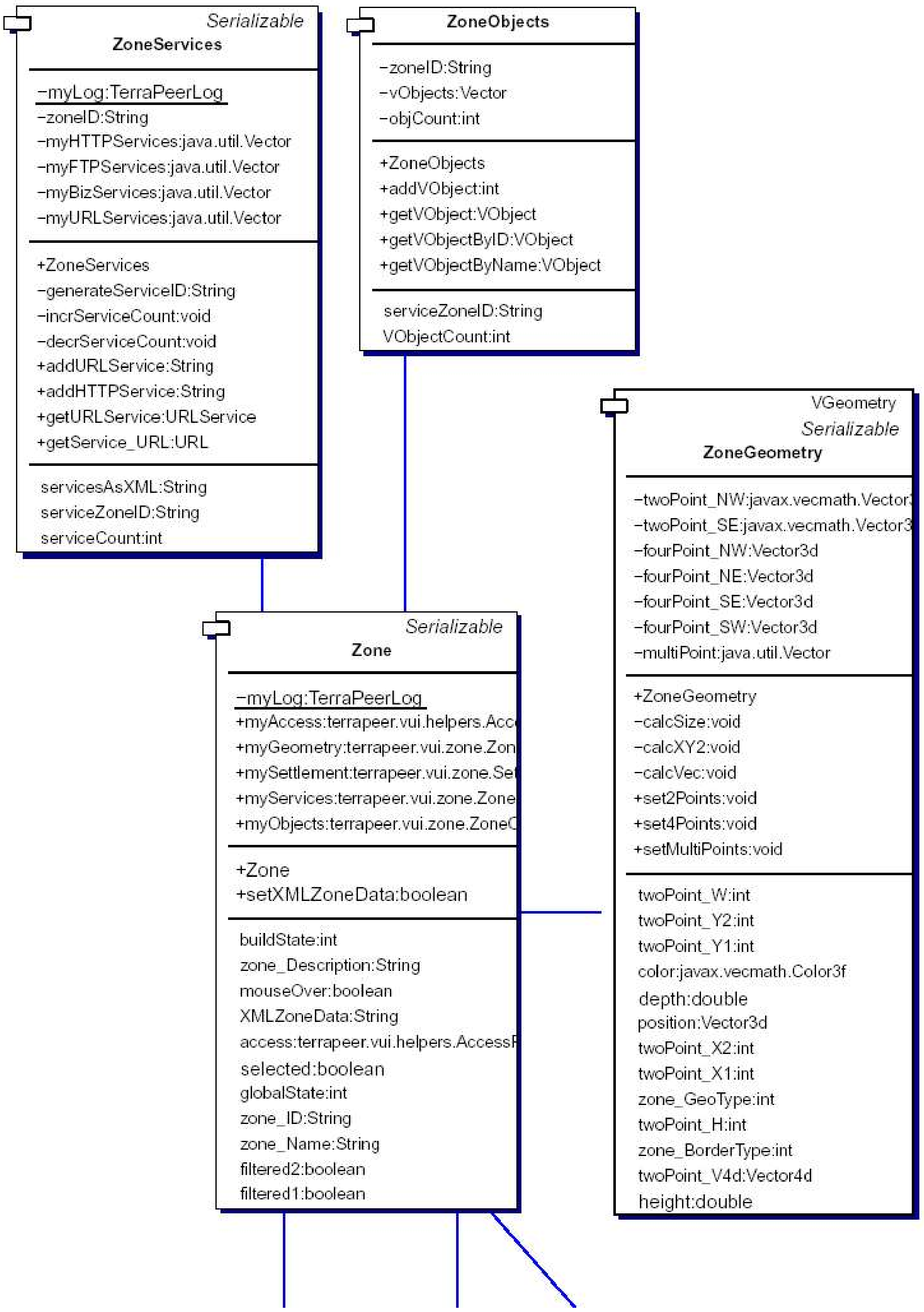
Schema 2 – package: terrapeer.net.schema



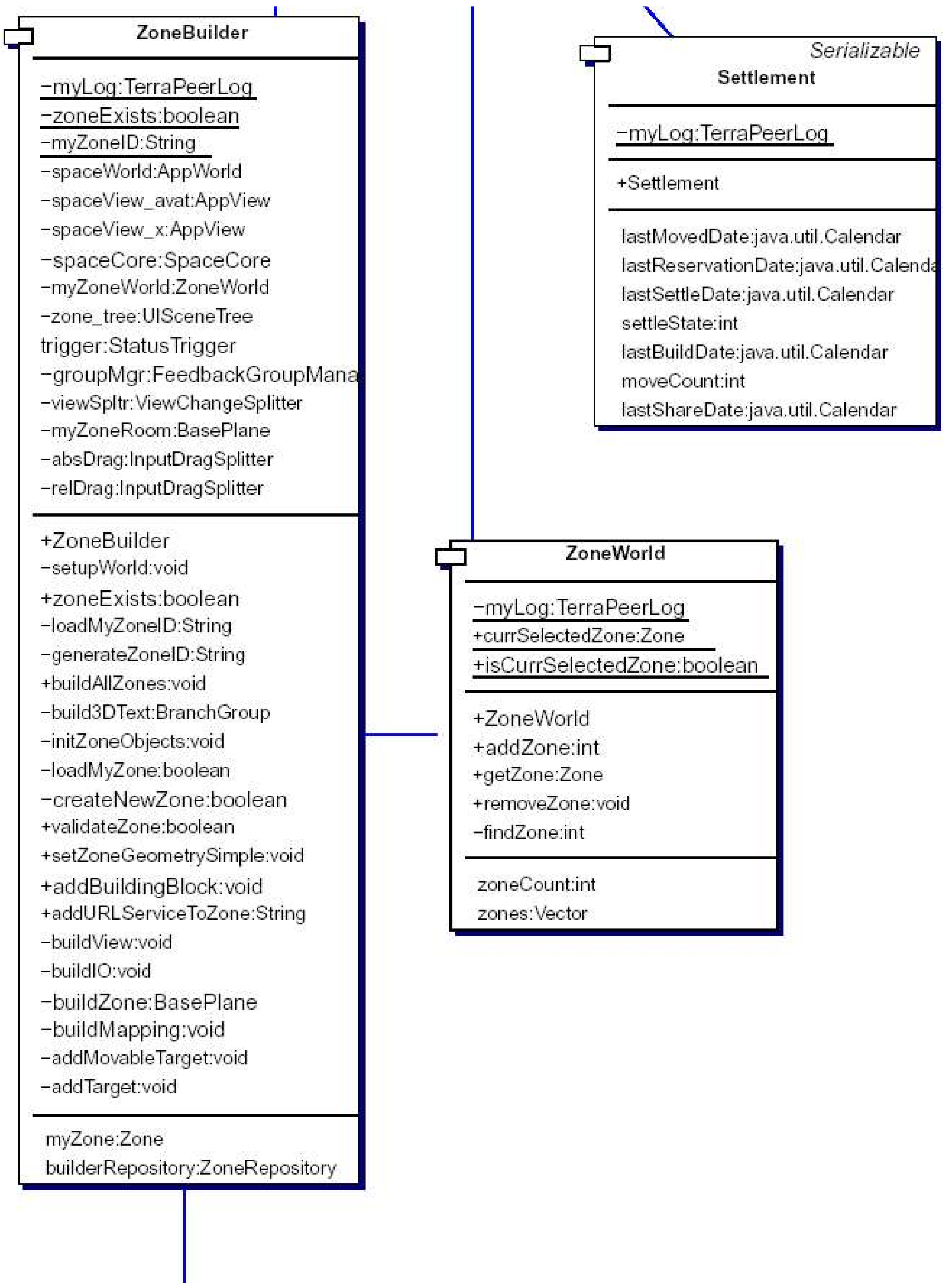
Schema 3 – package: terrapeer.net.schema



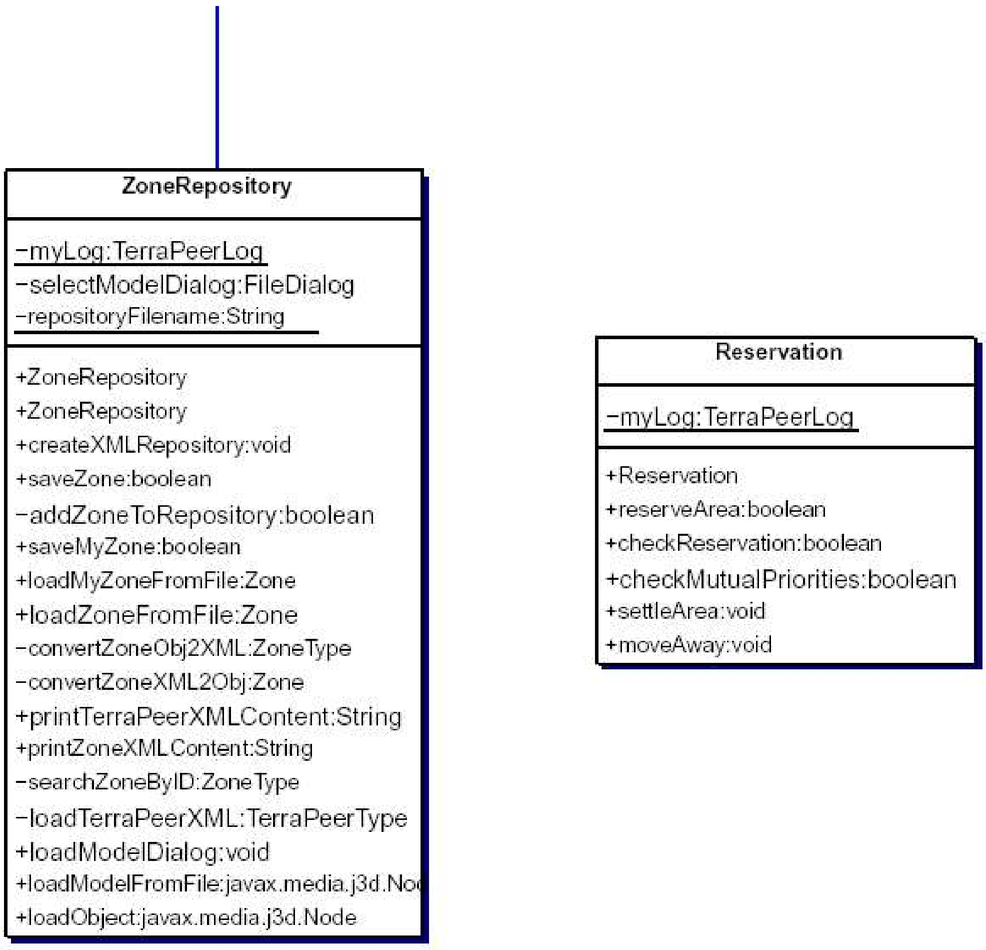
Zone 1 – package: terrapeer.vui.zone



Zone 2 – package: terrapeer.vui.zone



Zone 3 – package: terrapeer.vui.zone


ZoneRepository
-myLog:TerraPeerLog
-selectModelDialog:FileDialog
-repositoryFilename:String
+ZoneRepository
+ZoneRepository
+createXMLRepository:void
+saveZone:boolean
-addZoneToRepository:boolean
+saveMyZone:boolean
+loadMyZoneFromFile:Zone
+loadZoneFromFile:Zone
-convertZoneObj2XML:ZoneType
-convertZoneXML2Obj:Zone
+printTerraPeerXMLContent:String
+printZoneXMLContent:String
-searchZoneByID:ZoneType
-loadTerraPeerXML:TerraPeerType
+loadModelDialog:void
+loadModelFromFile:javax.media.j3d.No
+loadObject:javax.media.j3d.Node
Reservation
-myLog:TerraPeerLog
+Reservation
+reserveArea:boolean
+checkReservation:boolean
+checkMutualPriorities:boolean
+settleArea:void
+moveAway:void

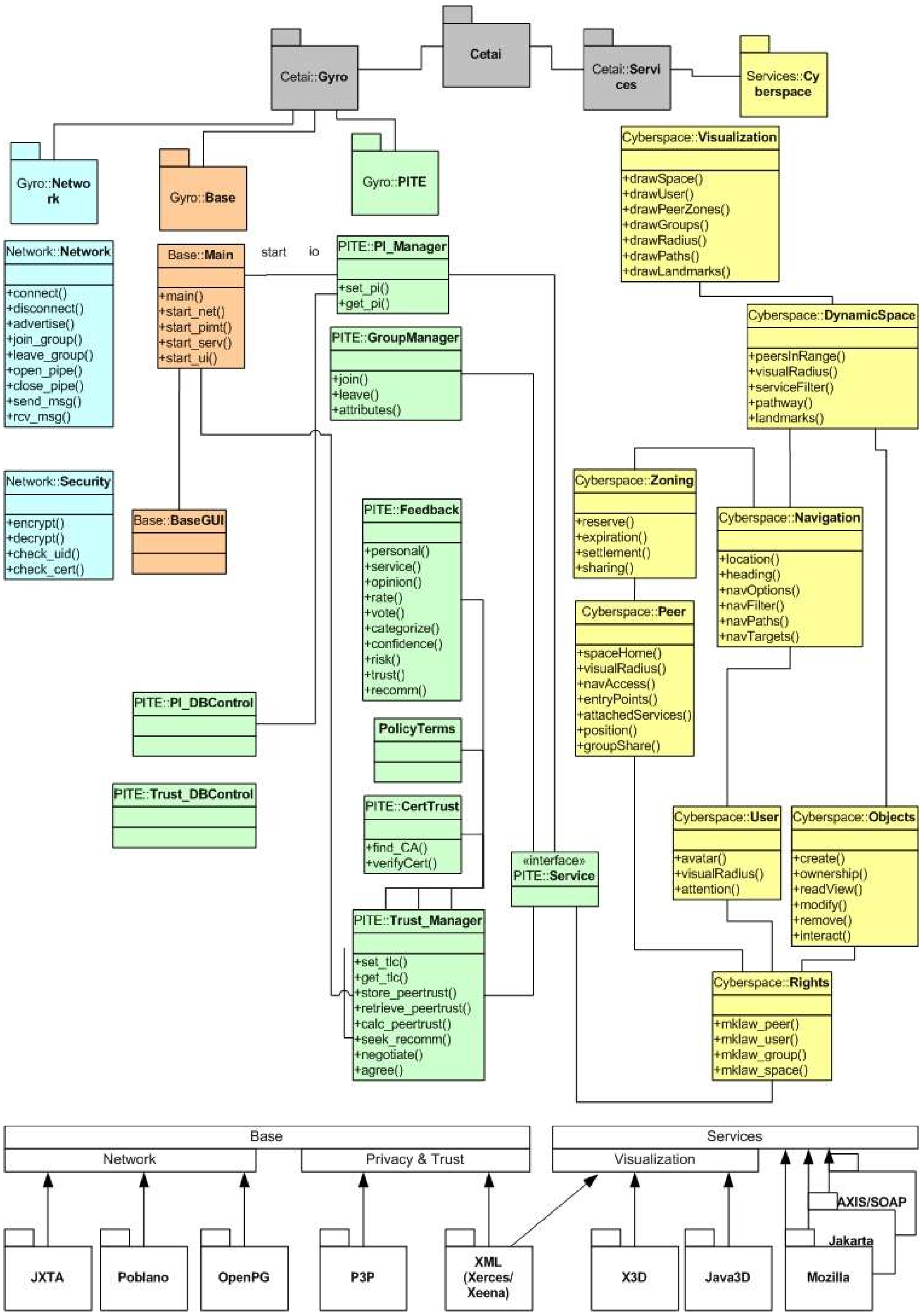


Illustration 7.7. UML Layout (draft version - depricated)

Appendix VI.Use Cases

The three diagrams illustrate different aspects of the application: The P2P network, space navigation, and the trust sub-system.

The last one was included merely for convenience as it is not implemented in the current version, but was one of the consideration of this author during the early stage of this project. This stage emphasized primarily the trust aspect of a P2P system. Similarly, the other drawings might not reflect the true state of the application due to many changes in the specification since they were envisioned. The abbreviations also reflect different wording: The 'Gyro' is now 'TerraPeer', PI stands for Personal Information, and TLC for 'Trust Level Control'.

Thus, the reader is encouraged to view the attached use-cases purely with the purpose of abstracting the essential concepts.

P2P Network

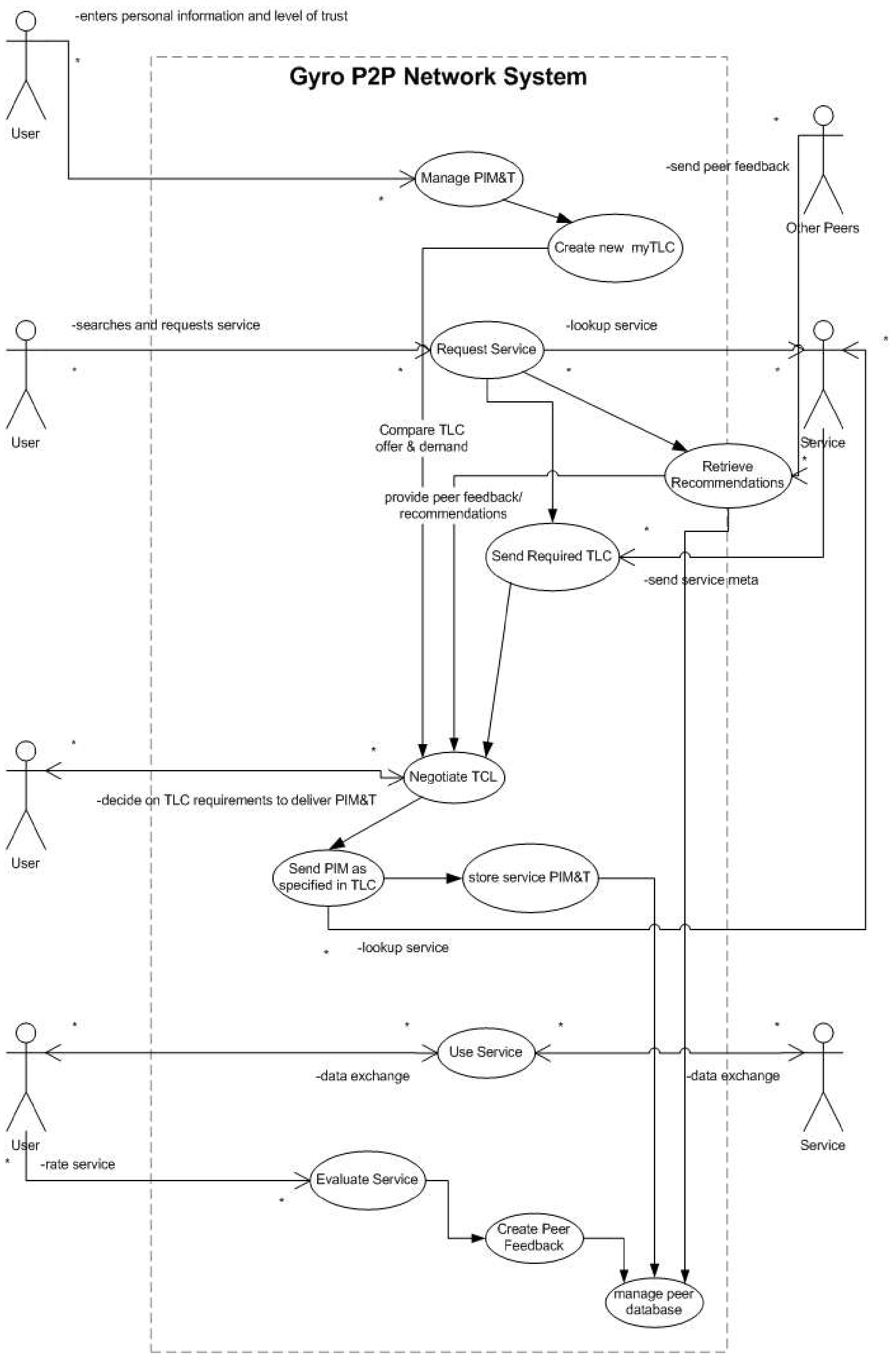


Illustration 7.8. Use Case - Network

Space Navigation

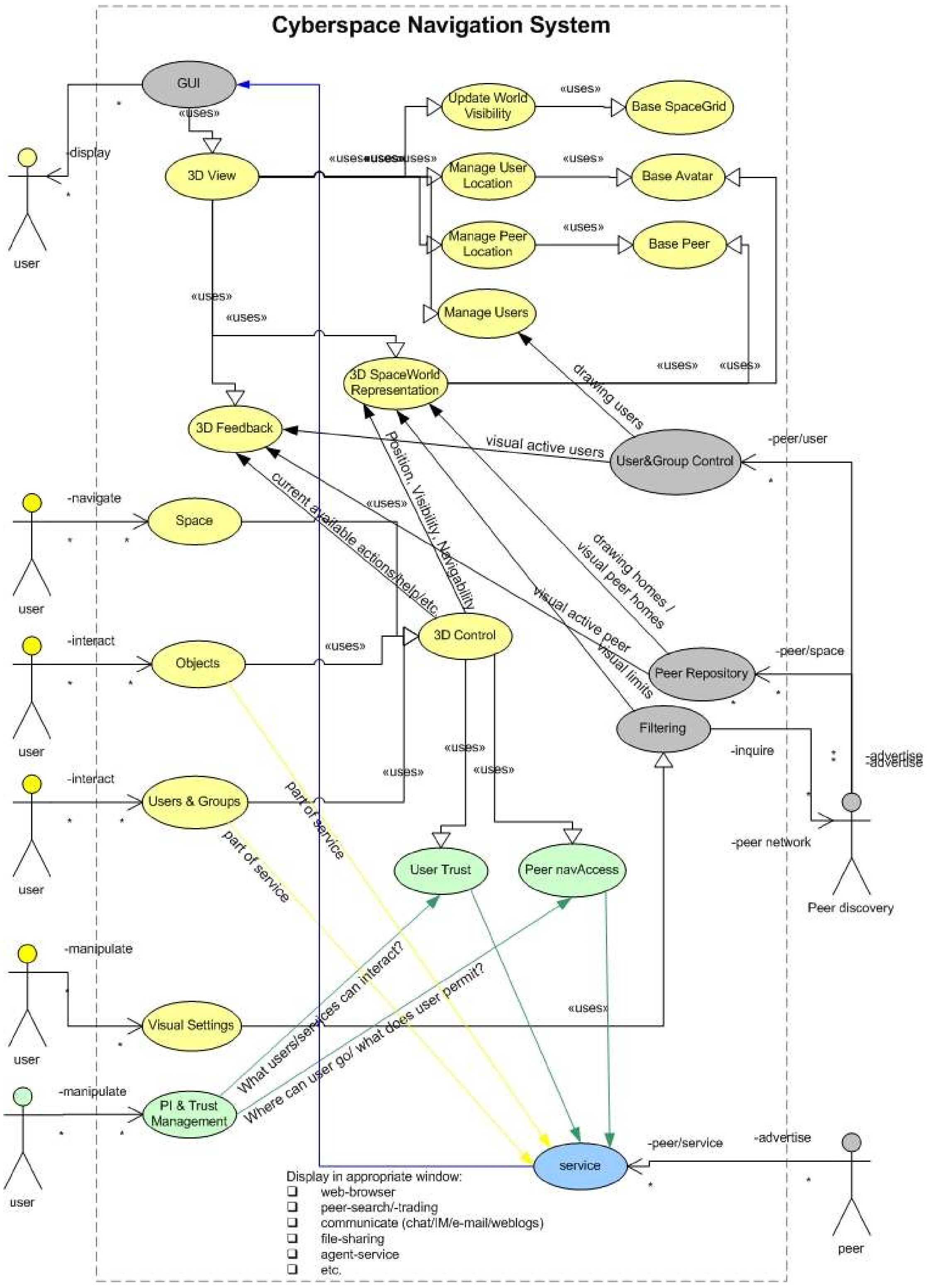


Illustration 7.9. Use Case - Space and Navigation

Trust System

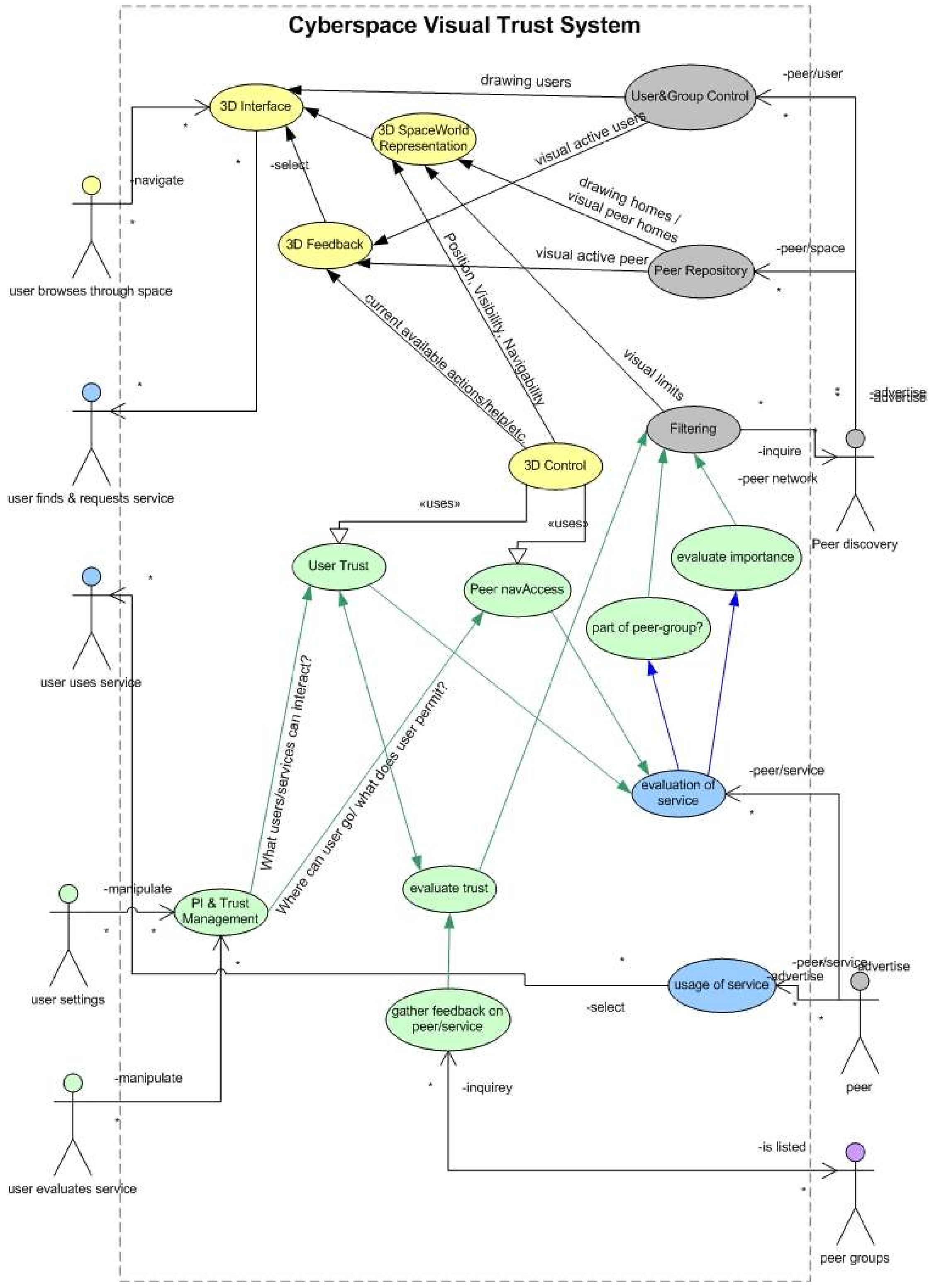


Illustration 7.10. Use Case - Trust System

Appendix VII.TerraPeer Repository

Repository XML Schema Design

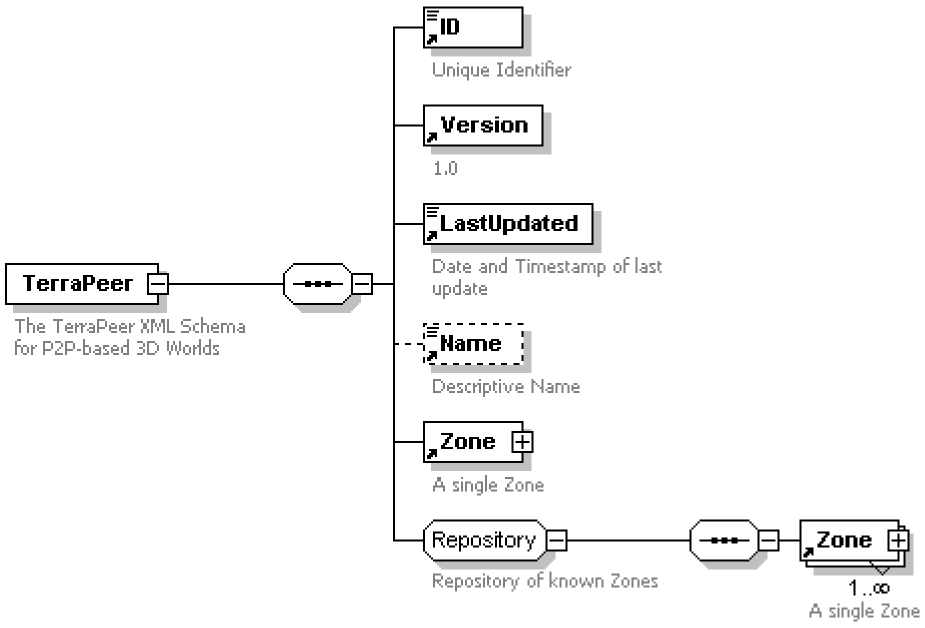


Illustration 7.11. Structure of TerraPeer Repository with collection of zones

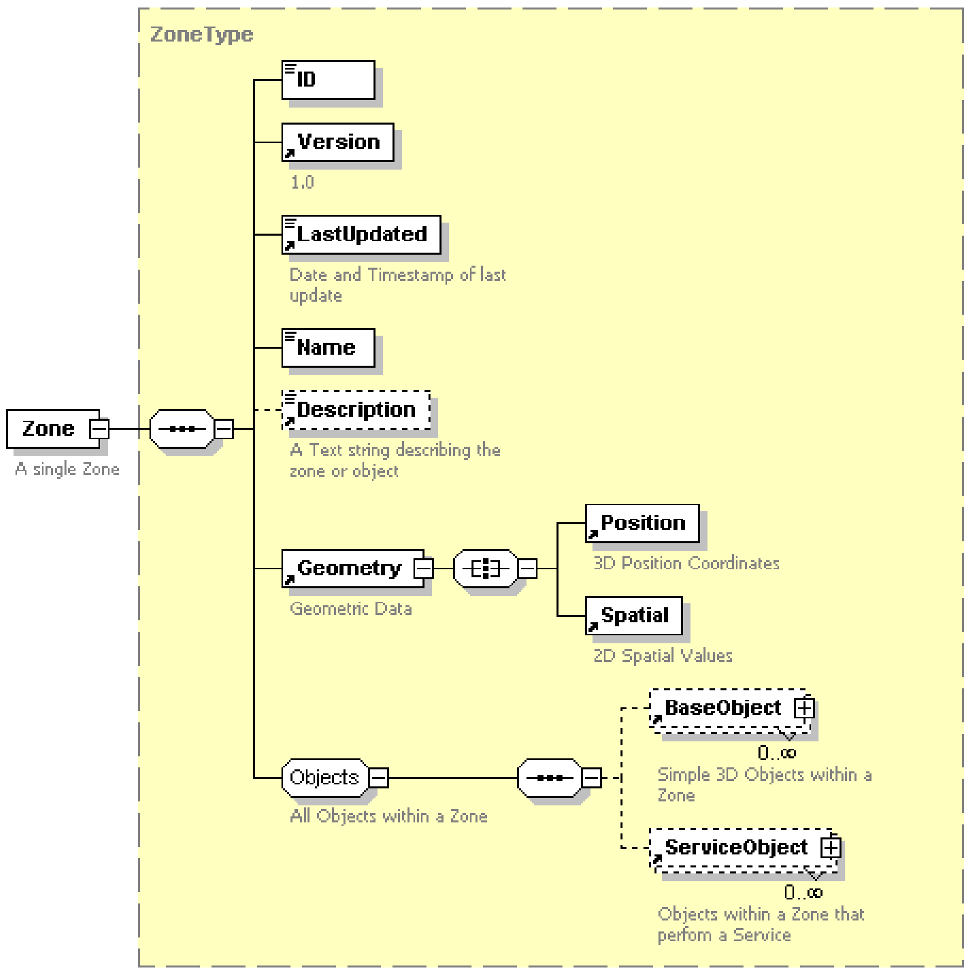


Illustration 7.12. Structure of ZoneType

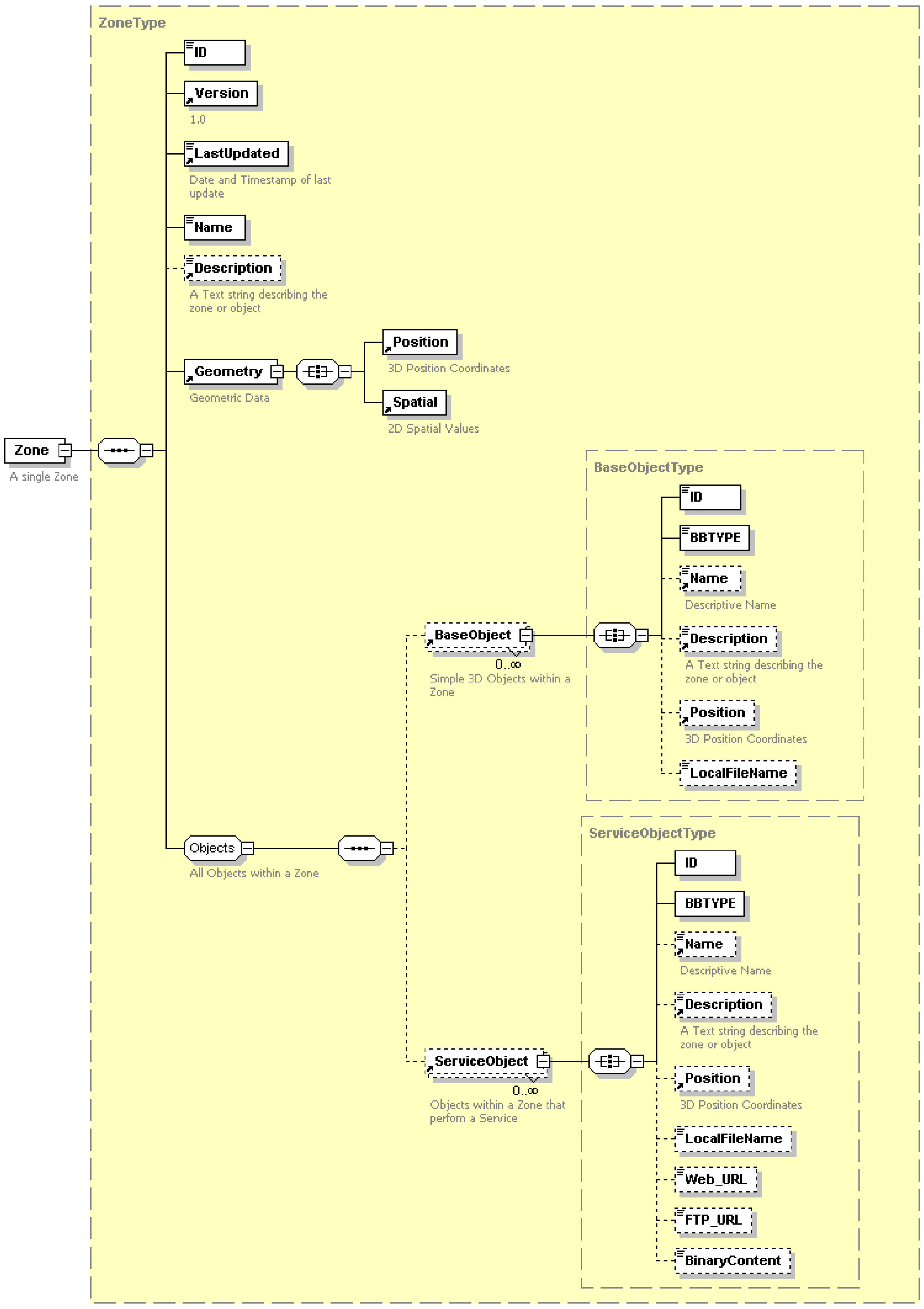


Illustration 7.13. Structure of a Zone

Repository XML Schema Code

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by TEAM (RENEGADE)
-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:element name="TerraPeer">
		<xs:annotation>
			<xs:documentation>The TerraPeer XML Schema for P2P-based 3D
Worlds</xs:documentation>
		</xs:annotation>
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="ID"/>
				<xs:element ref="Version"/>
				<xs:element ref="LastUpdated"/>
				<xs:element ref="Name" minOccurs="0"/>
				<xs:element ref="Zone"/>
				<xs:element name="ZoneWorld" minOccurs="0">
					<xs:complexType>
						<xs:group ref="Repository"/>
					</xs:complexType>
				</xs:element>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="Version">
		<xs:annotation>
			<xs:documentation>1.0</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:element name="LastUpdated" type="xs:dateTime">
		<xs:annotation>
			<xs:documentation>Date and Timestamp of last
update</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:complexType name="ZoneType">
		<xs:annotation>
			<xs:documentation>Basic Zone Type</xs:documentation>
		</xs:annotation>
		<xs:sequence>
			<xs:element name="ID" type="xs:string"/>
			<xs:element ref="Version"/>
			<xs:element ref="LastUpdated"/>
			<xs:element name="Name" type="xs:string"/>
			<xs:element ref="Description" minOccurs="0"/>
			<xs:element ref="Geometry"/>
			<xs:group ref="Objects"/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="Vector3dType">
		<xs:annotation>
			<xs:documentation>3D Vector Type</xs:documentation>
		</xs:annotation>
		<xs:attribute name="X" type="xs:double"/>
		<xs:attribute name="Y" type="xs:double"/>
		<xs:attribute name="Z" type="xs:double"/>
	</xs:complexType>
	<xs:complexType name="SizeType">
		<xs:annotation>
			<xs:documentation>Size Type</xs:documentation>
		</xs:annotation>
		<xs:attribute name="width" type="xs:double"/>
		<xs:attribute name="height" type="xs:double"/>
	</xs:complexType>
	<xs:complexType name="BaseObjectType">
		<xs:annotation>
			<xs:documentation>Basic Object Type</xs:documentation>
		</xs:annotation>
		<xs:all>
			<xs:element ref="ObjectID"/>
```

```
                        <xs:element ref="BBTYPE"/>
                        <xs:element ref="Name" minOccurs="0"/>
                        <xs:element ref="Description" minOccurs="0"/>
                        <xs:element ref="Position" minOccurs="0"/>
                        <xs:element name="LocalFileName" type="xs:string"
minOccurs="0"/>
                </xs:all>
        </xs:complexType>
        <xs:complexType name="ServiceObjectType">
                <xs:annotation>
                        <xs:documentation>Service Object Type</xs:documentation>
                </xs:annotation>
                <xs:all>
                        <xs:element ref="ID"/>
                        <xs:element ref="BBTYPE"/>
                        <xs:element ref="Name" minOccurs="0"/>
                        <xs:element ref="Description" minOccurs="0"/>
                        <xs:element ref="Position" minOccurs="0"/>
                        <xs:element name="LocalFileName" type="xs:string"
minOccurs="0"/>
                        <xs:element name="Web_URL" type="xs:string" minOccurs="0"/>
                        <xs:element name="FTP_URL" type="xs:string" minOccurs="0"/>
                        <xs:element name="BinaryContent" type="xs:base64Binary"
minOccurs="0"/>
                </xs:all>
        </xs:complexType>
        <xs:group name="Repository">
                <xs:annotation>
                        <xs:documentation>Repository of known
Zones</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element ref="Zone" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:group>
        <xs:group name="Objects">
                <xs:annotation>
                        <xs:documentation>All Objects within a
Zone</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element ref="BaseObject" minOccurs="0"
maxOccurs="unbounded"/>
                        <xs:element ref="ServiceObject" minOccurs="0"
maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:group>
        <xs:element name="Position" type="Vector3dType">
                <xs:annotation>
                        <xs:documentation>3D Position
Coordinates</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="Spatial" type="SizeType">
                <xs:annotation>
                        <xs:documentation>2D Spatial Values</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="Zone" type="ZoneType">
                <xs:annotation>
                        <xs:documentation>A single Zone</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="ID" type="xs:string">
                <xs:annotation>
                        <xs:documentation>Unique Identifier</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="Name" type="xs:string">
                <xs:annotation>
                        <xs:documentation>Descriptive Name</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="Description" type="xs:string">
                <xs:annotation>
```

```
                        <xs:documentation>A Text string describing the zone or
object</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="Geometry">
                <xs:annotation>
                        <xs:documentation>Geometric Data</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                        <xs:all>
                                <xs:element ref="Position"/>
                                <xs:element ref="Spatial"/>
                        </xs:all>
                </xs:complexType>
        </xs:element>
        <xs:element name="ServiceObject" type="ServiceObjectType">
                <xs:annotation>
                        <xs:documentation>Objects within a Zone that perfom a
Service</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="BaseObject" type="BaseObjectType">
                <xs:annotation>
                        <xs:documentation>Simple 3D Objects within a
Zone</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="BBTYPE" type="xs:int">
                <xs:annotation>
                        <xs:documentation>Building Block Type recognized by
TerraPeer</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="ObjectID" type="xs:string">
                <xs:annotation>
                        <xs:documentation>Unique Object ID</xs:documentation>
                </xs:annotation>
        </xs:element>
</xs:schema>
```

Example XML Repository Document

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSPY v5 rel. 4 U (http://www.xmlspy.com)-->
<TerraPeer xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\liu\_g\projects\TerraPeer\project\terrapeer_v1
.xsd">
	<ID>String</ID>
	<Version>Text</Version>
	<LastUpdated>2001-12-17T09:30:47-05:00</LastUpdated>
	<Name>String</Name>
	<Zone>
		<ID>MZ</ID>
		<Version>Text</Version>
		<LastUpdated>2001-12-17T09:30:47-05:00</LastUpdated>
		<Name>My Zone</Name>
		<Description>String</Description>
		<Geometry>
			<Position X="3.1415" Y="3.1415" Z="3.1415"/>
			<Spatial width="3.1415" height="3.1415"/>
		</Geometry>
		<BaseObject>
			<ObjectID>String</ObjectID>
			<BBTYPE>0</BBTYPE>
			<Name>String</Name>
			<Description>String</Description>
			<Position X="3.1415" Y="3.1415" Z="3.1415"/>
			<LocalFileName>String</LocalFileName>
		</BaseObject>
		<ServiceObject>
			<ID>String</ID>
			<BBTYPE>0</BBTYPE>
			<Name>String</Name>
			<Description>String</Description>
			<Position X="3.1415" Y="3.1415" Z="3.1415"/>
			<LocalFileName>String</LocalFileName>
			<Web_URL>String</Web_URL>
			<FTP_URL>String</FTP_URL>

<BinaryContent>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</BinaryContent>
		</ServiceObject>
	</Zone>
	<ZoneWorld>
		<Zone>
			<ID>Z1</ID>
			<Version>Text</Version>
			<LastUpdated>2001-12-17T09:30:47-05:00</LastUpdated>
			<Name>Zone 1</Name>
			<Description>String</Description>
			<Geometry>
				<Position X="3.1415" Y="3.1415" Z="2"/>
				<Spatial width="3.1415" height="3.1415"/>
			</Geometry>
			<BaseObject>
				<ObjectID>String</ObjectID>
				<BBTYPE>0</BBTYPE>
				<Name>String</Name>
				<Description>String</Description>
				<Position X="3.1415" Y="3.1415" Z="1"/>
				<LocalFileName>String</LocalFileName>
			</BaseObject>
			<ServiceObject>
				<ID>String</ID>
				<BBTYPE>0</BBTYPE>
				<Name>String</Name>
				<Description>String</Description>
				<Position X="3.1415" Y="3.1415" Z="3"/>
				<LocalFileName>String</LocalFileName>
				<Web_URL>String</Web_URL>
				<FTP_URL>String</FTP_URL>

<BinaryContent>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</BinaryContent>
			</ServiceObject>
```

```
                </Zone>
                <Zone>
                        <ID>Z2</ID>
                        <Version>Text</Version>
                        <LastUpdated>2001-12-17T09:30:47-05:00</LastUpdated>
                        <Name>Zone 2</Name>
                        <Description>String</Description>
                        <Geometry>
                                <Position X="3.1415" Y="3.1415" Z="6"/>
                                <Spatial width="3.1415" height="3.1415"/>
                        </Geometry>
                        <BaseObject>
                                <ObjectID>String</ObjectID>
                                <BBTYPE>0</BBTYPE>
                                <Name>String</Name>
                                <Description>String</Description>
                                <Position X="3.1415" Y="3.1415" Z="5"/>
                                <LocalFileName>String</LocalFileName>
                        </BaseObject>
                        <ServiceObject>
                                <ID>String</ID>
                                <BBTYPE>0</BBTYPE>
                                <Name>String</Name>
                                <Description>String</Description>
                                <Position X="3.1415" Y="3.1415" Z="4"/>
                                <LocalFileName>String</LocalFileName>
                                <Web_URL>String</Web_URL>
                                <FTP_URL>String</FTP_URL>

<BinaryContent>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</BinaryContent>
                        </ServiceObject>
                </Zone>
                <Zone>
                        <ID>Z3</ID>
                        <Version>Text</Version>
                        <LastUpdated>2001-12-17T09:30:47-05:00</LastUpdated>
                        <Name>Zone 3</Name>
                        <Description>String</Description>
                        <Geometry>
                                <Position X="3.1415" Y="3.1415" Z="7"/>
                                <Spatial width="3.1415" height="3.1415"/>
                        </Geometry>
                        <BaseObject>
                                <ObjectID>String</ObjectID>
                                <BBTYPE>0</BBTYPE>
                                <Name>String</Name>
                                <Description>String</Description>
                                <Position X="3.1415" Y="3.1415" Z="4"/>
                                <LocalFileName>String</LocalFileName>
                        </BaseObject>
                        <ServiceObject>
                                <ID>String</ID>
                                <BBTYPE>0</BBTYPE>
                                <Name>String</Name>
                                <Description>String</Description>
                                <Position X="3.1415" Y="3.1415" Z="5"/>
                                <LocalFileName>String</LocalFileName>
                                <Web_URL>String</Web_URL>
                                <FTP_URL>String</FTP_URL>

<BinaryContent>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</BinaryContent>
                        </ServiceObject>
                </Zone>
        </ZoneWorld>
</TerraPeer>
```

Appendix VIII.Drawings

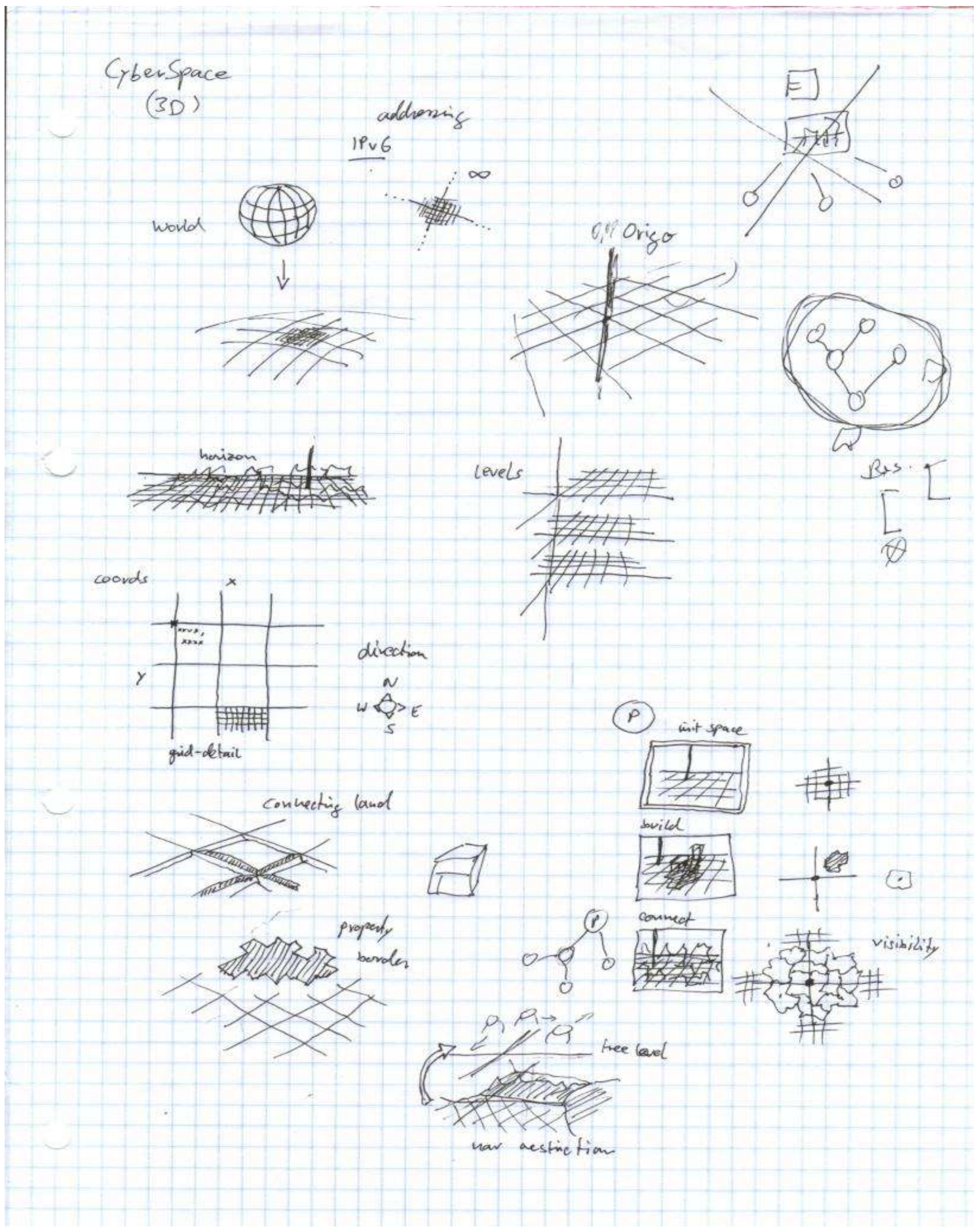


Illustration 7.14. Notes on virtual space representations and zoning

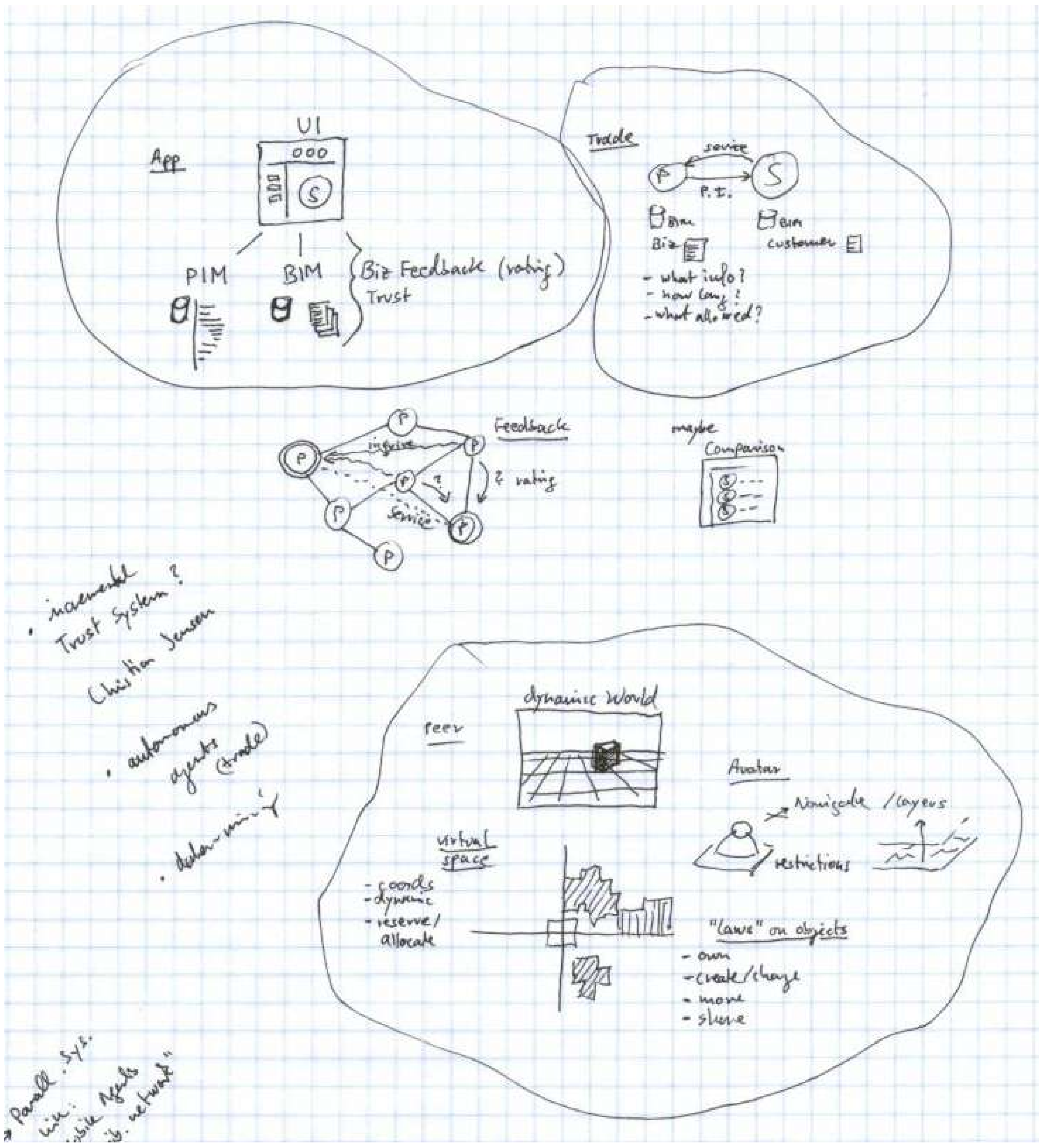


Illustration 7.15. Notes on the application and world design

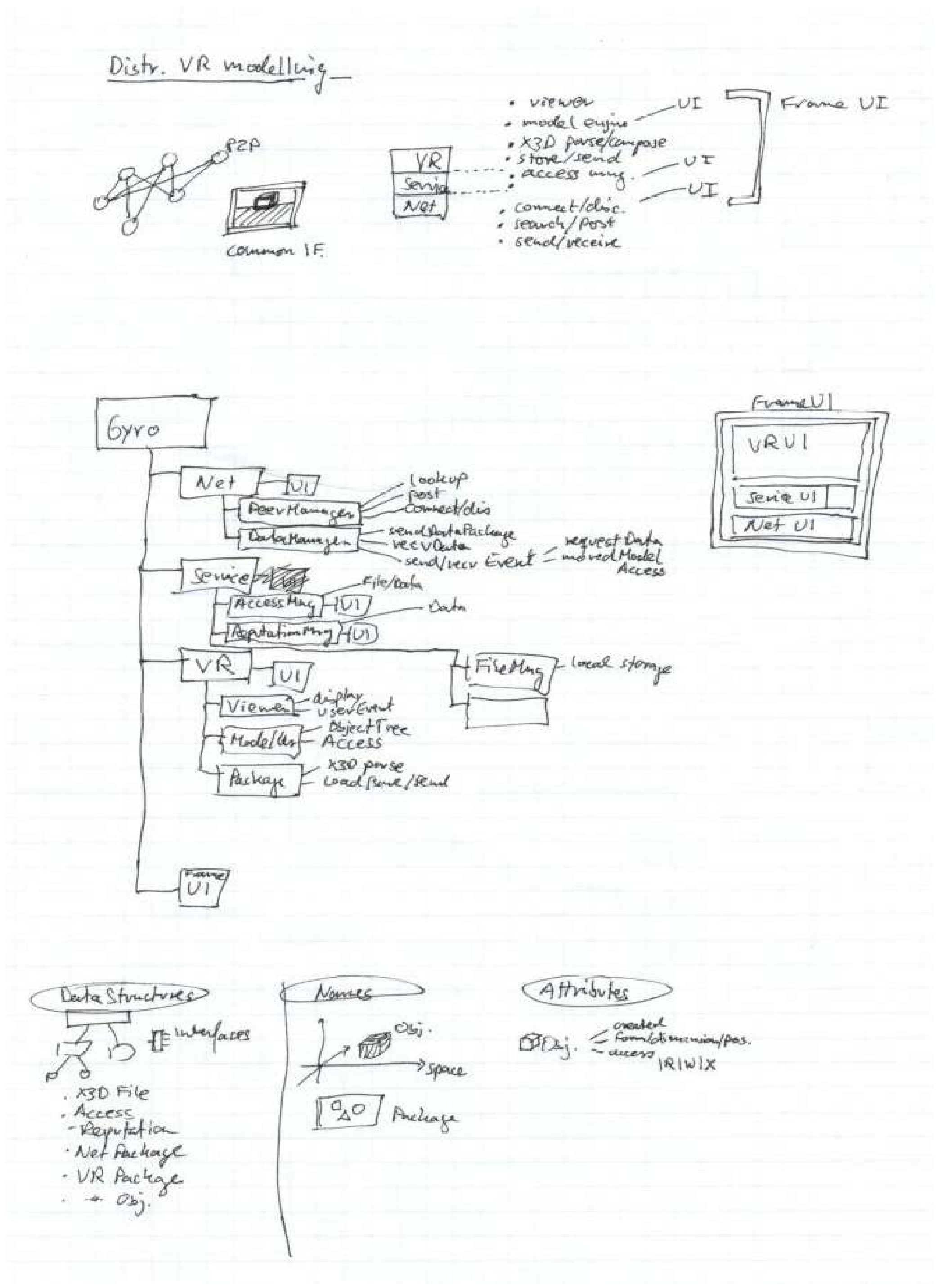


Illustration 7.16. Hand drawing of class and object structures

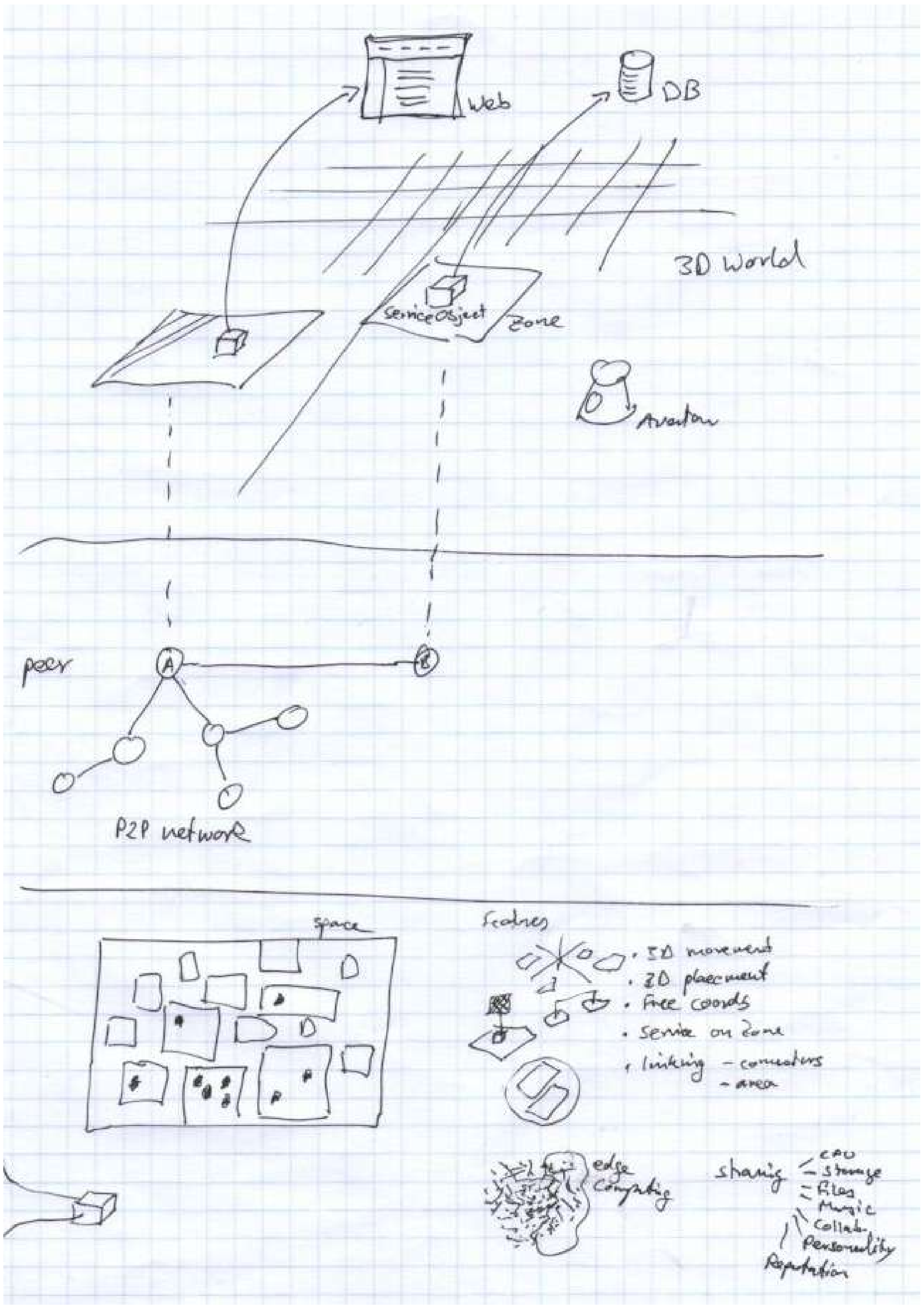


Illustration 7.17. Hand drawing of zone-peer-service relationships

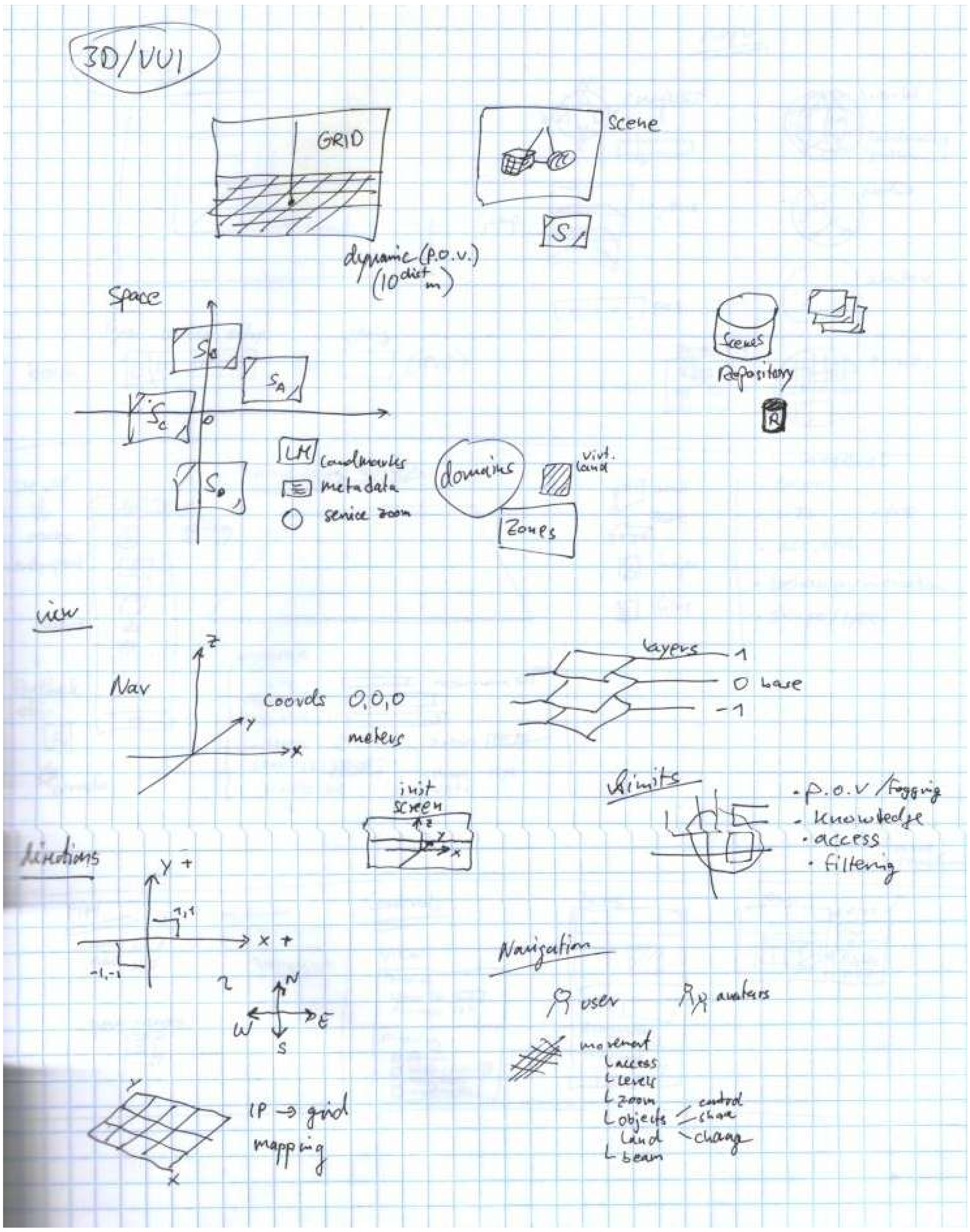


Illustration 7.18. Hand drawing of 3D environment specs

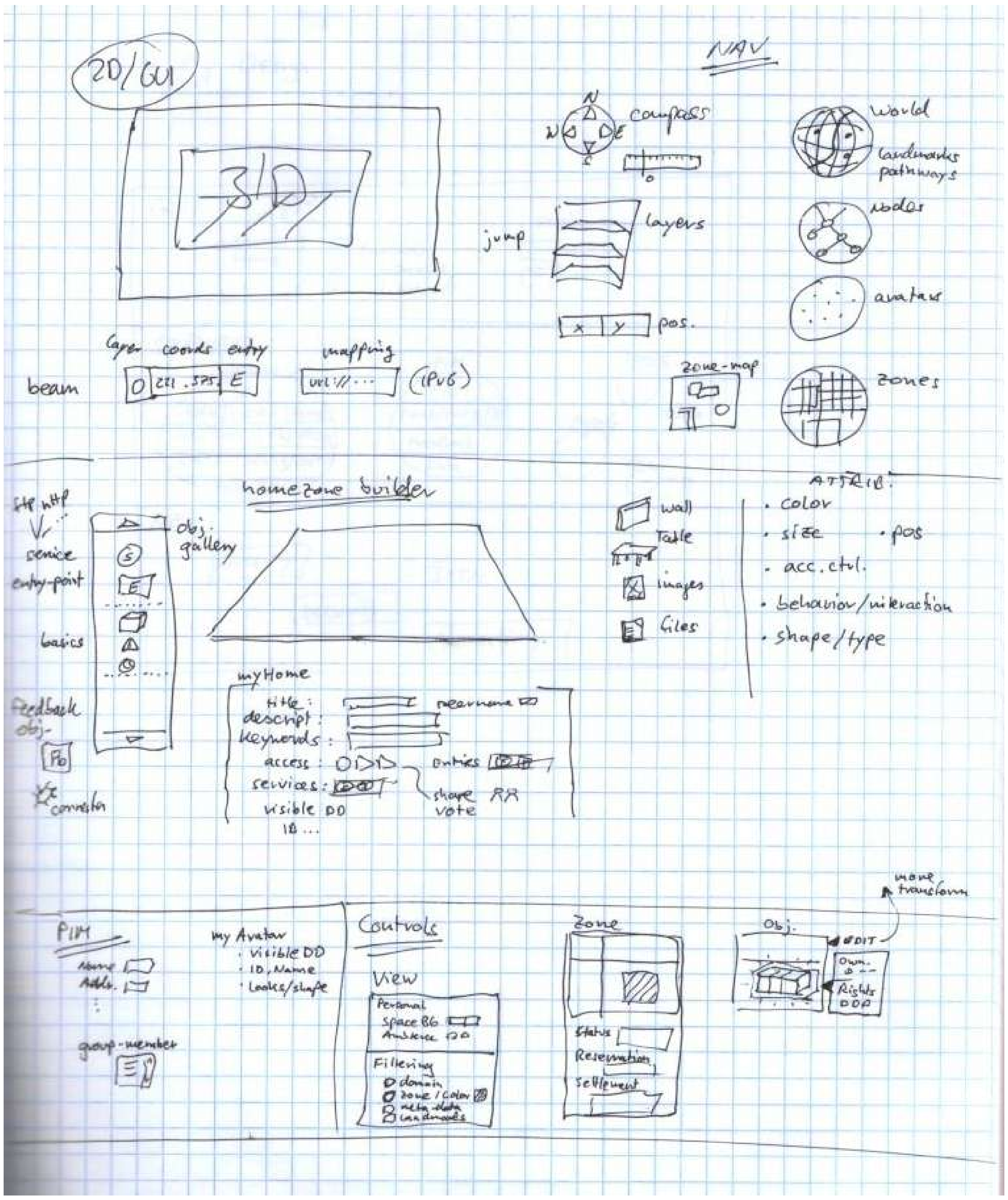


Illustration 7.19. Hand drawing of navigation interface, builder interface, and controls

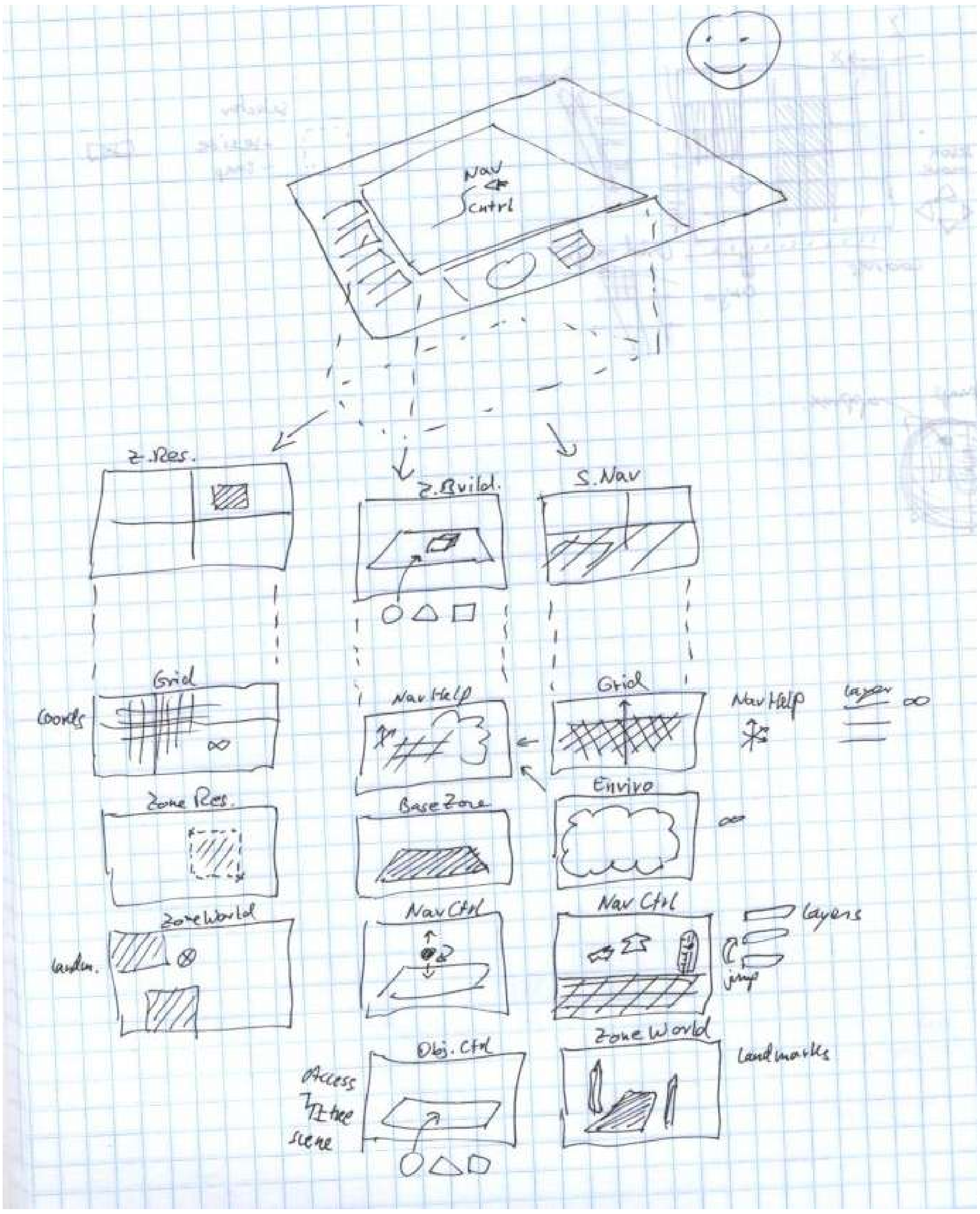


Illustration 7.20. Hand drawing of user interface layers

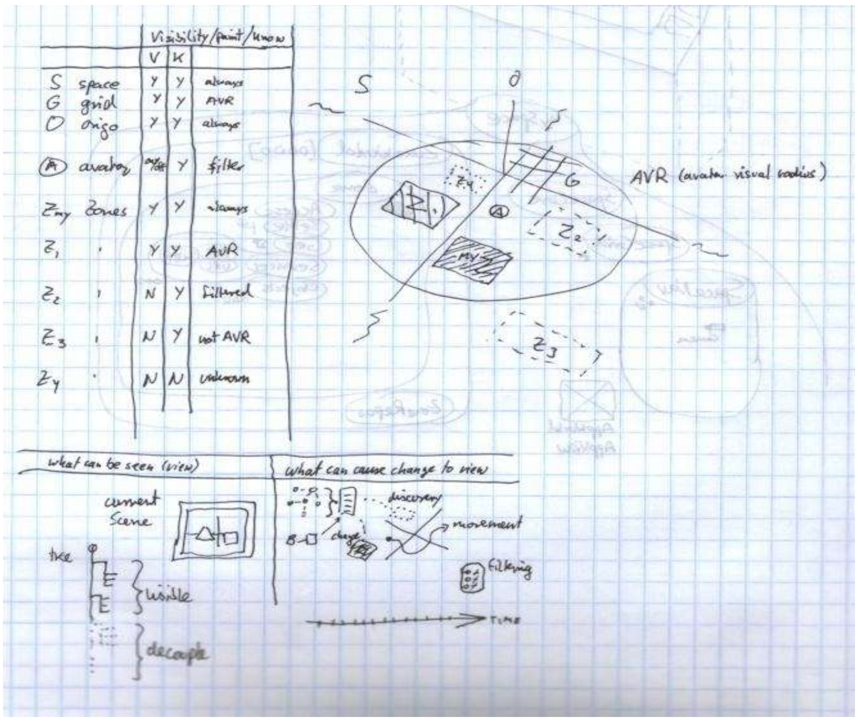


Illustration 7.21. Hand drawing of visibility properties

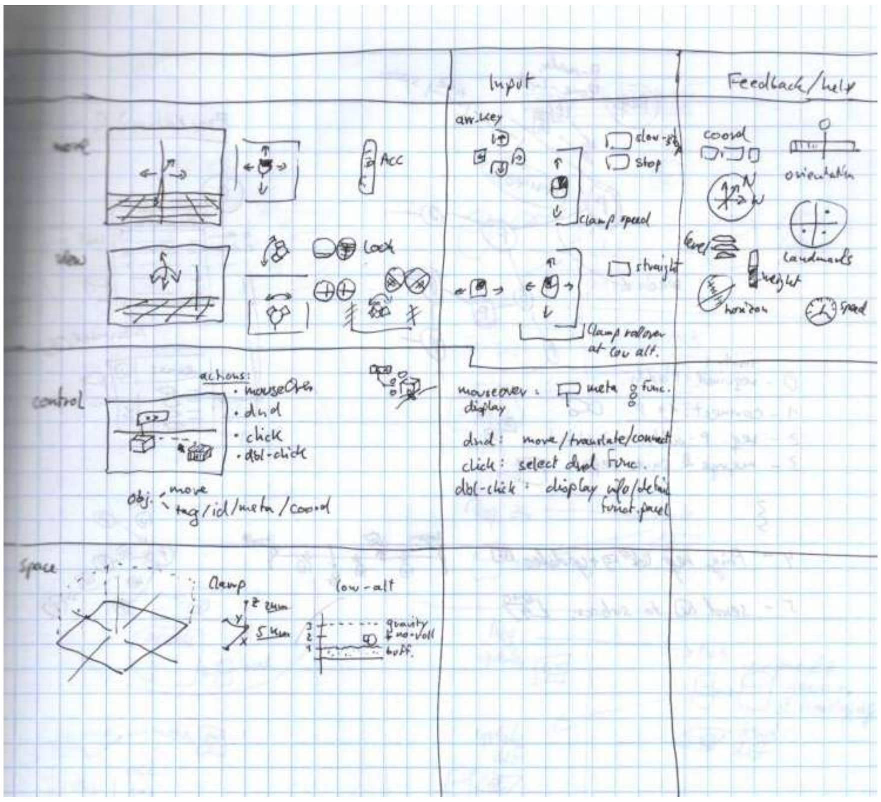


Illustration 7.22. Hand drawing of user input and feedback

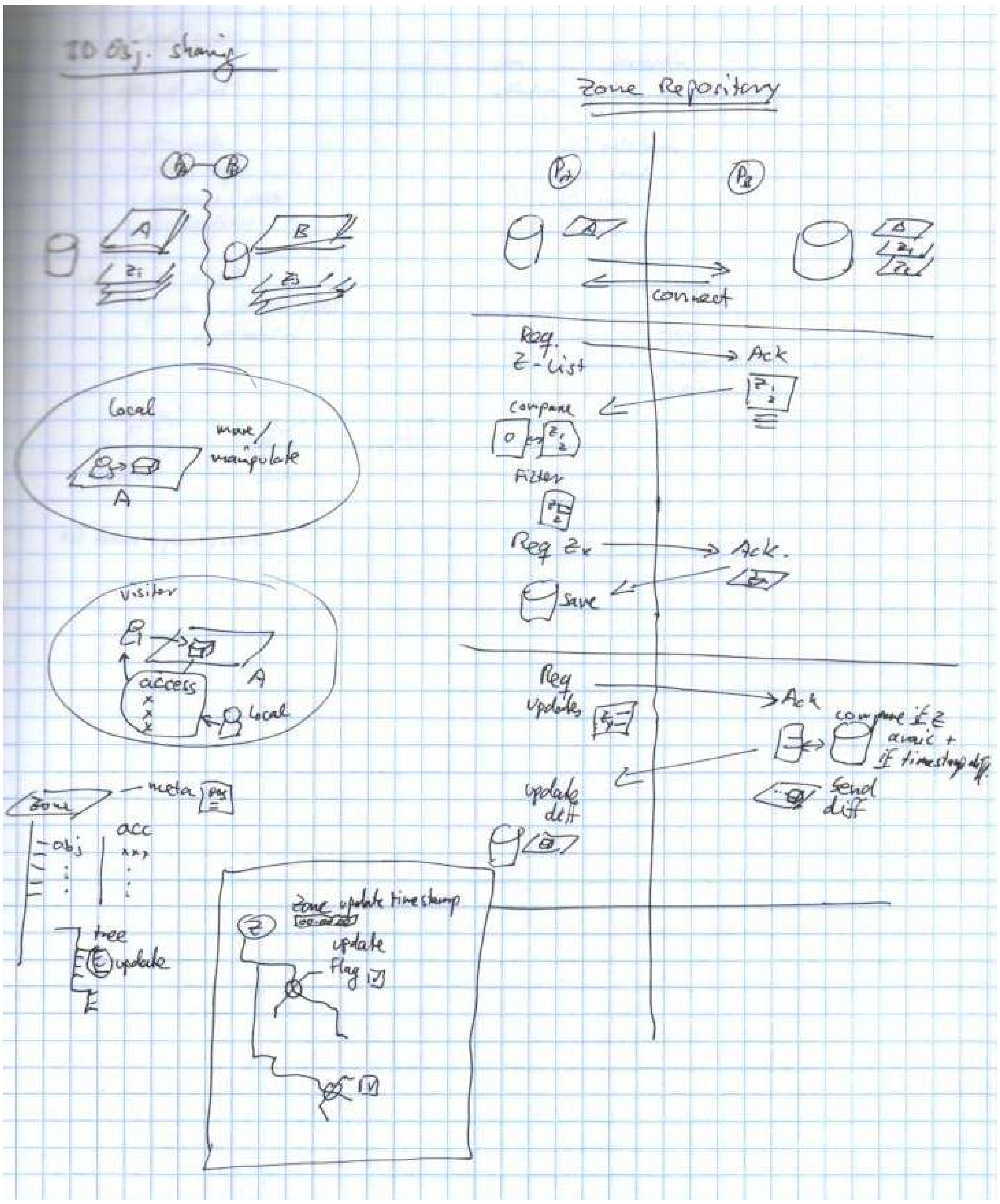
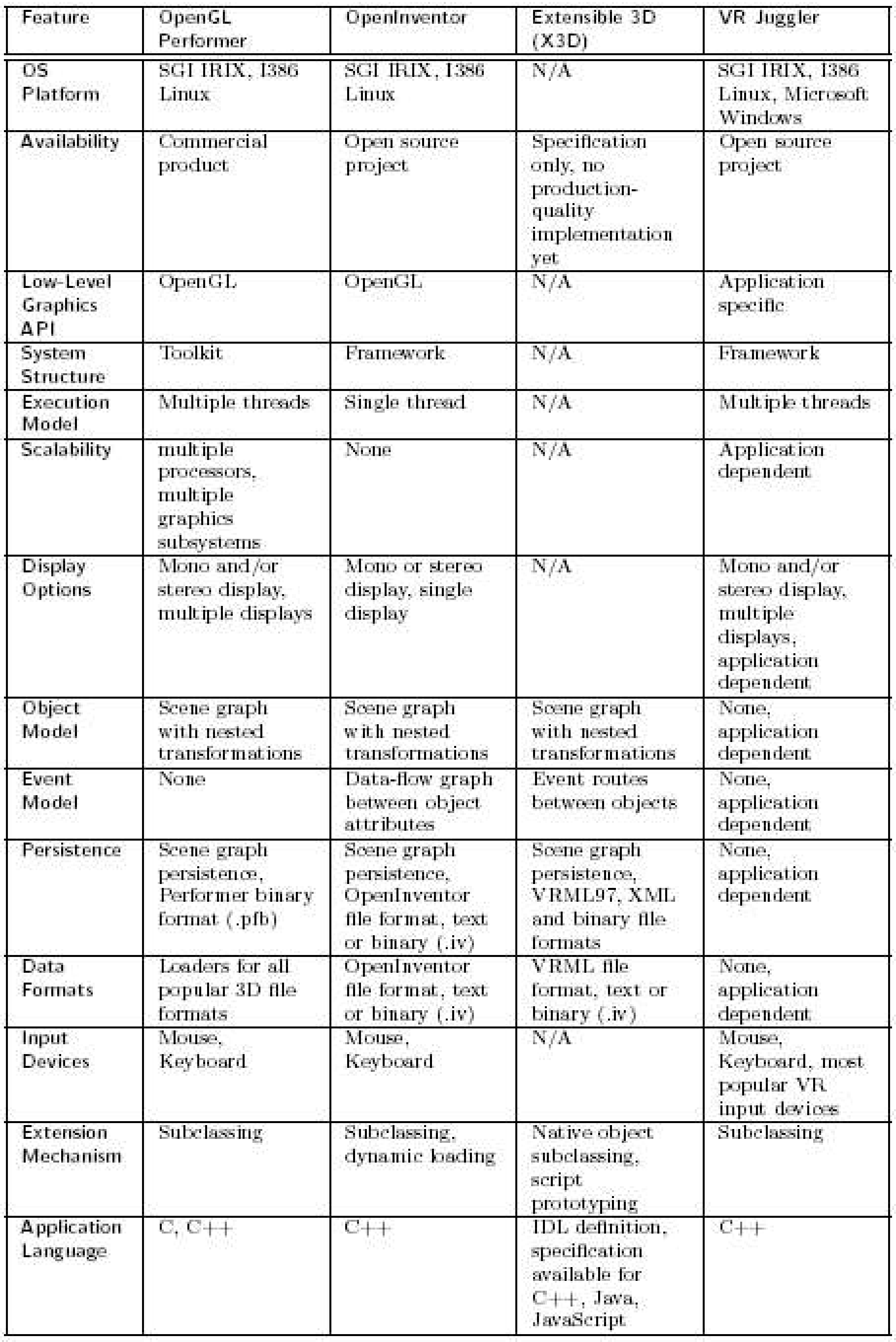


Illustration 7.23. Hand drawing of object sharing and zone repository updates

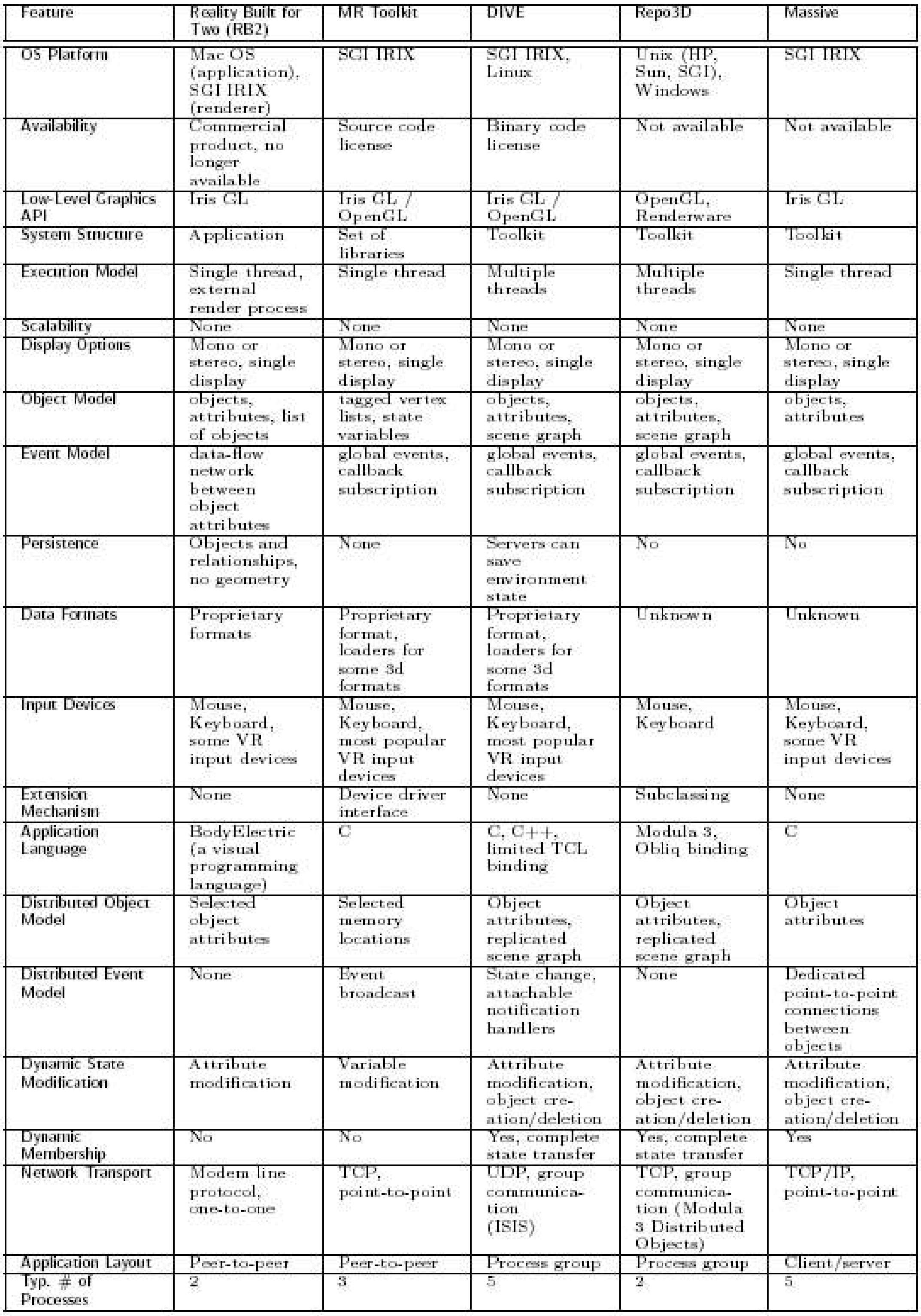
Appendix IX.VE and DVE Comparison Tables



Feature	OpenGL Performer	OpenInventor	Extensible 3D (X3D)	VR Juggler
OS Platform	SGI IRIX, I386 Linux	SGI IRIX, I386 Linux	N/A	SGI IRIX, I386 Linux, Microsoft Windows
Availability	Commercial product	Open source project	Specification only, no production-quality implementation yet	Open source project
Low-Level Graphics API	OpenGL	OpenGL	N/A	Application specific
System Structure	Toolkit	Framework	N/A	Framework
Execution Model	Multiple threads	Single thread	N/A	Multiple threads
Scalability	multiple processors, multiple graphics subsystems	None	N/A	Application dependent
Display Options	Mono and/or stereo display, multiple displays	Mono or stereo display, single display	N/A	Mono and/or stereo display, multiple displays, application dependent
Object Model	Scene graph with nested transformations	Scene graph with nested transformations	Scene graph with nested transformations	None, application dependent
Event Model	None	Data-flow graph between object attributes	Event routes between objects	None, application dependent
Persistence	Scene graph persistence, Performer binary format (.pfb)	Scene graph persistence, OpenInventor file format, text or binary (.iv)	Scene graph persistence, VRML97, XML and binary file formats	None, application dependent
Data Formats	Loaders for all popular 3D file formats	OpenInventor file format, text or binary (.iv)	VRML file format, text or binary (.iv)	None, application dependent
Input Devices	Mouse, Keyboard	Mouse, Keyboard	N/A	Mouse, Keyboard, most popular VR input devices
Extension Mechanism	Subclassing	Subclassing, dynamic loading	Native object subclassing, script prototyping	Subclassing
Application Language	C, C++	C++	IDL definition, specification available for C++, Java, JavaScript	C++

Illustration 7.24. DVE Comparison Table [AVOCADO, pp.31]

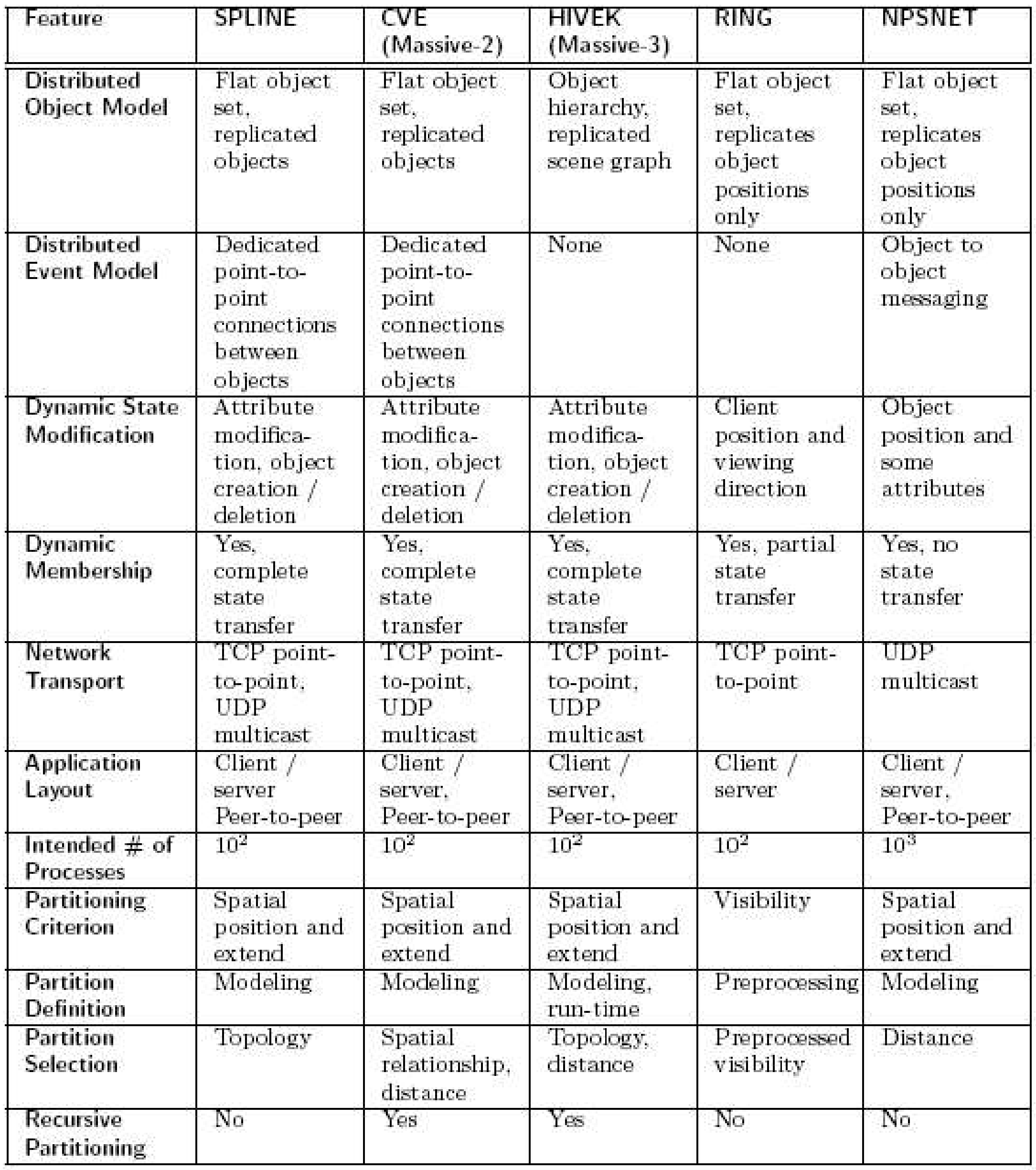
A comparative feature summary of non-distributed VE frameworks, toolkits and technologies.



Feature	Reality Built for Two (RB2)	MR Toolkit	DIVE	Repo3D	Massive
OS Platform	Mac OS (application), SGI IRIX (renderer)	SGI IRIX	SGI IRIX, Linux	Unix (HP, Sun, SGI), Windows	SGI IRIX
Availability	Commercial product, no longer available	Source code license	Binary code license	Not available	Not available
Low-Level Graphics API	Iris GL	Iris GL / OpenGL	Iris GL / OpenGL	OpenGL, Renderware	Iris GL
System Structure	Application	Set of libraries	Toolkit	Toolkit	Toolkit
Execution Model	Single thread, external render process	Single thread	Multiple threads	Multiple threads	Single thread
Scalability	None	None	None	None	None
Display Options	Mono or stereo, single display	Mono or stereo, single display	Mono or stereo, single display	Mono or stereo, single display	Mono or stereo, single display
Object Model	objects, attributes, list of objects	tagged vertex lists, state variables	objects, attributes, scene graph	objects, attributes, scene graph	objects, attributes
Event Model	data-flow network between object attributes	global events, callback subscription	global events, callback subscription	global events, callback subscription	global events, callback subscription
Persistence	Objects and relationships, no geometry	None	Servers can save environment state	No	No
Data Formats	Proprietary formats	Proprietary format, loaders for some 3d formats	Proprietary format, loaders for some 3d formats	Unknown	Unknown
Input Devices	Mouse, Keyboard, some VR input devices	Mouse, Keyboard, most popular VR input devices	Mouse, Keyboard, most popular VR input devices	Mouse, Keyboard	Mouse, Keyboard, some VR input devices
Extension Mechanism	None	Device driver interface	None	Subclassing	None
Application Language	BodyElectric (a visual programming language)	C	C, C++, limited TCL binding	Modula 3, Obliq binding	C
Distributed Object Model	Selected object attributes	Selected memory locations	Object attributes, replicated scene graph	Object attributes, replicated scene graph	Object attributes
Distributed Event Model	None	Event broadcast	State change, attachable notification handlers	None	Dedicated point-to-point connections between objects
Dynamic State Modification	Attribute modification	Variable modification	Attribute modification, object creation/deletion	Attribute modification, object creation/deletion	Attribute modification, object creation/deletion
Dynamic Membership	No	No	Yes, complete state transfer	Yes, complete state transfer	Yes
Network Transport	Modem line protocol, one-to-one	TCP, point-to-point	UDP, group communication (ISIS)	TCP, group communication (Modula 3 Distributed Objects)	TCP/IP, point-to-point
Application Layout	Peer-to-peer	Peer-to-peer	Process group	Process group	Client/server
Typ. # of Processes	2	3	5	2	5

Illustration 7.25. DVE Comparison Table [AVOCADO, pp.32]

A comparative summary of DVE frameworks and toolkits.



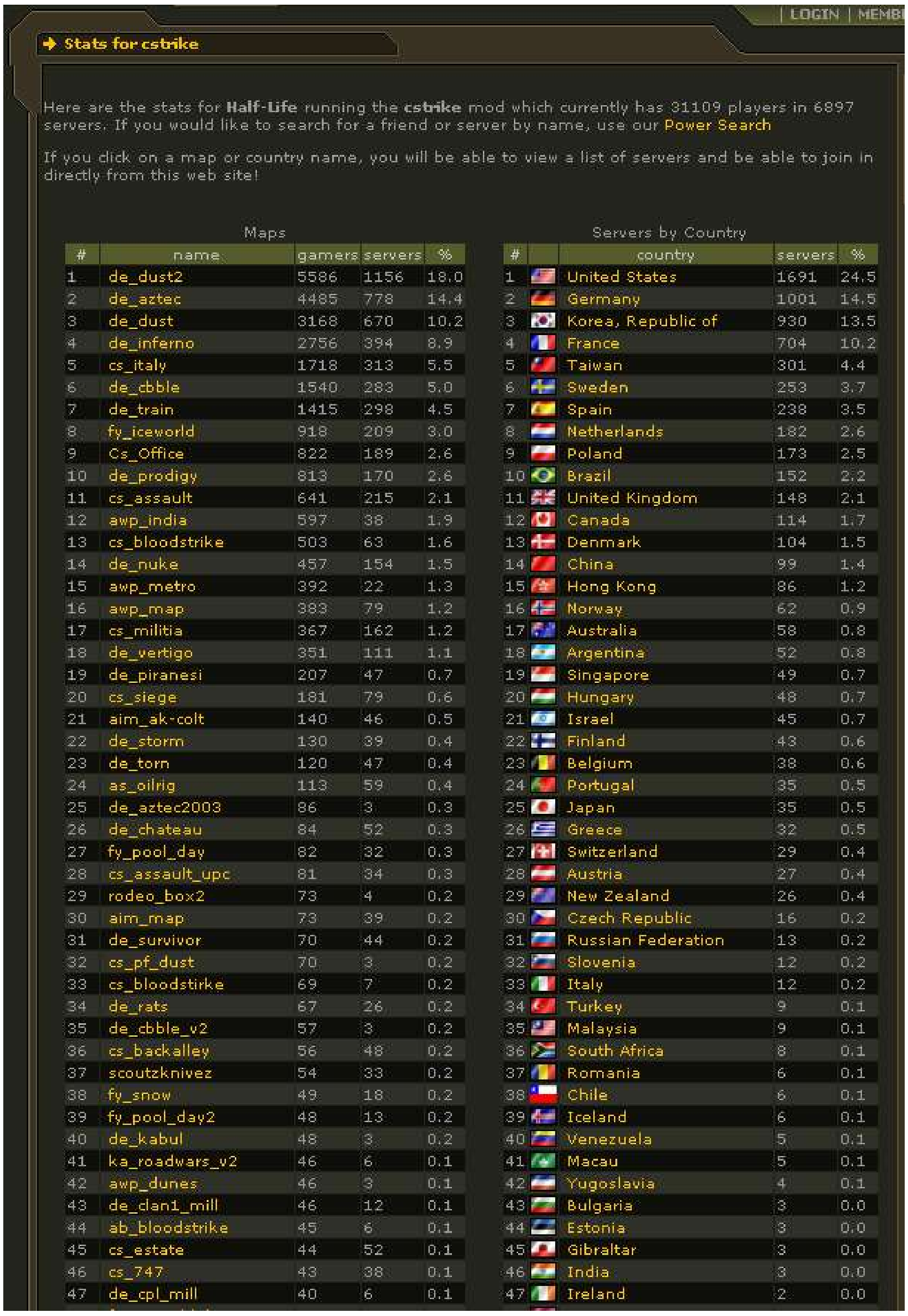
Feature	SPLINE	CVE (Massive-2)	HIVEK (Massive-3)	RING	NPSNET
Distributed Object Model	Flat object set, replicated objects	Flat object set, replicated objects	Object hierarchy, replicated scene graph	Flat object set, replicates object positions only	Flat object set, replicates object positions only
Distributed Event Model	Dedicated point-to-point connections between objects	Dedicated point-to-point connections between objects	None	None	Object to object messaging
Dynamic State Modification	Attribute modifica-tion, object creation / deletion	Attribute modifica-tion, object creation / deletion	Attribute modifica-tion, object creation / deletion	Client position and viewing direction	Object position and some attributes
Dynamic Membership	Yes, complete state transfer	Yes, complete state transfer	Yes, complete state transfer	Yes, partial state transfer	Yes, no state transfer
Network Transport	TCP point-to-point, UDP multicast	TCP point-to-point, UDP multicast	TCP point-to-point, UDP multicast	TCP point-to-point	UDP multicast
Application Layout	Client / server Peer-to-peer	Client / server, Peer-to-peer	Client / server, Peer-to-peer	Client / server	Client / server, Peer-to-peer
Intended # of Processes	10^2	10^2	10^2	10^2	10^3
Partitioning Criterion	Spatial position and extend	Spatial position and extend	Spatial position and extend	Visibility	Spatial position and extend
Partition Definition	Modeling	Modeling	Modeling, run-time	Preprocessing	Modeling
Partition Selection	Topology	Spatial relationship, distance	Topology, distance	Preprocessed visibility	Distance
Recursive Partitioning	No	Yes	Yes	No	No

Illustration 7.26. DVE Comparison Table [AVOCADO, pp.33]

A comparative summary of large-scale DVE systems. Because the presented systems are primarily research prototypes, little is known about platforms, object model, data formats, supported input devices and such. Therefore, this table only lists aspects relevant to scalability.

Appendix X.Game Servers

The screenshot below shows a listing with running game severs on an arbitrary day. The game selected in this statistic is Half-life with the 'Counter Strike' module.



Stats for cstrike

Here are the stats for **Half-Life** running the **cstrike** mod which currently has 31109 players in 6897 servers. If you would like to search for a friend or server by name, use our Power Search

If you click on a map or country name, you will be able to view a list of servers and be able to join in directly from this web site!

Maps

#	name	gamers	servers	%
1	de_dust2	5586	1156	18.0
2	de_aztec	4485	778	14.4
3	de_dust	3168	670	10.2
4	de_inferno	2756	394	8.9
5	cs_italy	1718	313	5.5
6	de_cbble	1540	283	5.0
7	de_train	1415	298	4.5
8	fy_iceworld	918	209	3.0
9	Cs_Office	822	189	2.6
10	de_prodigy	813	170	2.6
11	cs_assault	641	215	2.1
12	awp_india	597	38	1.9
13	cs_bloodstrike	503	63	1.6
14	de_nuke	457	154	1.5
15	awp_metro	392	22	1.3
16	awp_map	383	79	1.2
17	cs_militia	367	162	1.2
18	de_vertigo	351	111	1.1
19	de_piranesi	207	47	0.7
20	cs_siege	181	79	0.6
21	aim_ak-colt	140	46	0.5
22	de_storm	130	39	0.4
23	de_torn	120	47	0.4
24	as_oilrig	113	59	0.4
25	de_aztec2003	86	3	0.3
26	de_chateau	84	52	0.3
27	fy_pool_day	82	32	0.3
28	cs_assault_upc	81	34	0.3
29	rodeo_box2	73	4	0.2
30	aim_map	73	39	0.2
31	de_survivor	70	44	0.2
32	cs_pf_dust	70	3	0.2
33	cs_bloodstirke	69	7	0.2
34	de_rats	67	26	0.2
35	de_cbble_v2	57	3	0.2
36	cs_backalley	56	48	0.2
37	scoutzknivez	54	33	0.2
38	fy_snow	49	18	0.2
39	fy_pool_day2	48	13	0.2
40	de_kabul	48	3	0.2
41	ka_roadwars_v2	46	6	0.1
42	awp_dunes	46	3	0.1
43	de_clan1_mill	46	12	0.1
44	ab_bloodstrike	45	6	0.1
45	cs_estate	44	52	0.1
46	cs_747	43	38	0.1
47	de_cpl_mill	40	6	0.1

Servers by Country

#		country	servers	%
1		United States	1691	24.5
2		Germany	1001	14.5
3		Korea, Republic of	930	13.5
4		France	704	10.2
5		Taiwan	301	4.4
6		Sweden	253	3.7
7		Spain	238	3.5
8		Netherlands	182	2.6
9		Poland	173	2.5
10		Brazil	152	2.2
11		United Kingdom	148	2.1
12		Canada	114	1.7
13		Denmark	104	1.5
14		China	99	1.4
15		Hong Kong	86	1.2
16		Norway	62	0.9
17		Australia	58	0.8
18		Argentina	52	0.8
19		Singapore	49	0.7
20		Hungary	48	0.7
21		Israel	45	0.7
22		Finland	43	0.6
23		Belgium	38	0.6
24		Portugal	35	0.5
25		Japan	35	0.5
26		Greece	32	0.5
27		Switzerland	29	0.4
28		Austria	27	0.4
29		New Zealand	26	0.4
30		Czech Republic	16	0.2
31		Russian Federation	13	0.2
32		Slovenia	12	0.2
33		Italy	12	0.2
34		Turkey	9	0.1
35		Malaysia	9	0.1
36		South Africa	8	0.1
37		Romania	6	0.1
38		Chile	6	0.1
39		Iceland	6	0.1
40		Venezuela	5	0.1
41		Macau	5	0.1
42		Yugoslavia	4	0.1
43		Bulgaria	3	0.0
44		Estonia	3	0.0
45		Gibraltar	3	0.0
46		India	3	0.0
47		Ireland	2	0.0

Illustration 7.27. Counter Strike Game Servers