SICS Technical Report T2004:03

ISRN: SICS-T—2004/03-SE
ISSN: 1100-3154

A Survey of CVE Technologies and Systems

by Emmanuel Frécon
emmanuel@sics.se

Interactive Collaborative Environments Laboratory
ice@sics.se

Swedish Institute of Computer Science
Box 1263, S-164 29 Kista, Sweden

Abstract

A few years ago, Virtual Reality technologies and Virtual Environments were seen by some as a panacea and the computer interface of the future. VR received a lot of attention in the media and devices such as head mounted displays or data gloves have become widely recognised. Of particular interest was the ability to realise a vision that had been described in a number of science fiction novels: providing a parallel world in which it would be possible to be present, interact and feel as if in the real world. This vision is realised by Collaborative Virtual Environments (CVEs). CVEs are three-dimensional computer-generated environments where users are represented by avatars and can navigate and interact in real-time independently of their physical location. While the technology has not lived up to early expectations, real niched applications and the success of networked games have shown its viability and promises. This report summarises a number of the technologies that are commonly used to interface with virtual environments. Additionally, it presents some of the major CVE systems to date and isolates a number of trends when it comes to network architectures, protocols and techniques and to software choices.

Contents

Chapter 1 Introduction

1.1. The Dawn of Virtual Environments

The term "Virtual Reality" (VR) was coined by Jaron Lanier[1] [1] in 1989. Other related terms include "Artificial Reality" [2] by Myron Krueger in the 1970s, "Cyberspace" by William Gibson in 1984 [3], and, more recently, "Virtual Worlds" and "Virtual Environments" in the 1990s.

In "*Neuromancer*", William Gibson defines Cyberspace as *"A consensual hallucination experienced daily by billions of legitimate operators, in every nation... A graphic representation of data abstracted from the banks of every computer in the human system. Unthinkable complexity. Lines of light ranged in the non-space of the mind, clusters and constellations of data..."*

The ideas of VR have their ground in science fiction books. They shape one or several parallel worlds within which we immerse and feel as if we were in the real world. In the late 1980's and early 1990's, the ideas of VR invaded the public stage through novels and media coverage. VR was to revolutionise the way we interact with computers. While the hype has progressively died out, the numerous research projects that have been conducted along the years have unearthed new domains and new types of applications. For example, evacuation rehearsal is much more effective when users are present within a realistic burning environment, as depicted in Illustration 1 compared to a two dimensional view of the building's floor plan.

Illustration 1: An example scene showing a burning room.

In the media, virtual reality and virtual environments have been used almost interchangeably and without much care. In this document, the term *Virtual Reality* refers to the underlying *technologies*, and the term *Virtual Environment* to the particular *synthetic environment* that the user is interacting with.

1.2. Collaborative Virtual Environments

In shared virtual environments, VR technology is used to immerse multiple individuals in a single shared space. Shared environments have received a lot of consideration in the past decade and have been used to support a range of activities including virtual conferencing [4] and collaborative information visualisation [5]. Commonly, the nature of shared virtual environments is such that the participants are collaborating in some way. Therefore this document refers to them as Collaborative Virtual Environments, or CVEs (see sidebar). In short, CVEs are to virtual environments what CSCW is to HCI.

"A CVE is a computer-based, distributed, virtual space or set of places. In such places, people can meet and interact with others, with agents or with virtual objects. CVEs might vary in their representational richness from 3D graphical spaces, 2.5D or 2D environments, to text-based environments. Access to CVEs is by no means limited to desktop devices, but might well include mobile or wearable devices, public kiosks, etc." (in [6]).

The rapid growth in academic interest has been mirrored by the development of commercial organisations who are offering access to shared communities: ActiveWorlds [7], The Palace [8] and there.com [9] being three of the most well-known. Since the basic standard for distributing models of virtual environments over the Internet, known as the Virtual Reality Modelling Language (VRML [10]) does not provide explicit support for simultaneously shareable worlds, these systems use proprietary extensions. The VRML community that is assembled as the Web3D consortium [11], has started a number of working groups to address and standardise these issues. Lately, the MPEG standardisation effort have added a back channel to complete the SNHC (Synthetic Natural Hybrid Coding), which combines natural video and audio with synthetic graphical objects.

1 Jaron Lanier is the founder of VPL Research, the first company to sell software and hardware VR products.

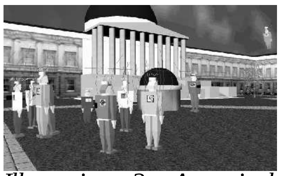


Illustration 2: A typical CVE scene with a number of avatars, each representing a user. In this example, avatars are using colour codes to differentiate their true geographical location. The graphical representation of the avatars in this scene is simplistic, more elaborate graphics can be used if necessary

It is not uncommon for the advocates of virtual environments to argue that they may support social interaction in ways which go beyond what is possible using more familiar CSCW technologies such as video conferences or shared desktop applications. Crucially, virtual environments permit users to become embodied within a shared space by means of an embodiment or *avatar*[2], as exemplified by Illustration 2. It is often claimed that this approach permits a degree of self-expression for users, and many systems support the end-user configuration or design of embodiments. It has also been argued that appropriately designed CVEs enable users to sustain mutual awareness about each other's activities [12].

1.3. Applications

A few years ago, virtual environments were seen by some as the interface that would ultimately replace the current desktop-based interface. Some people predicted that all applications would become three-dimensional in one form or another. However, virtual environments are not a panacea. There are many limitations both at the technological and software levels and this vision has died out. In the mean time, virtual environments have found a number of niched application, driven by real needs. This section summarises some of their most common applications. It points at some representative papers or reviews whenever possible.

Illustration 3: An architectural walk through allows for a better understanding of a building before it is actually built.

Virtual environments provide architects, customers and the public with the ability to experience a new building before it is actually built. Illustration 3 shows an example of an architectural walk-through. Such walk-throughs enable all parties to gain a sense of space in a way which would not be possible without VR technology [13].

Mechanical design enables engineers to test the arrangement of new components and to see and test new designs in operation (see [14] for an example). A number of car manufacturers have started to introduce virtual prototyping in order to cut down the costs of designing a new car and to reduce the number of physical mock-ups.

Scientific visualisation is one of the earliest uses of virtual reality. A well-known example is the virtual wind tunnel [15]. While information visualisation is a separate domain, it is a field where collaboration plays a more and more important role and CVE techniques and ideas are slowly migrating into scientic visualisation applications.

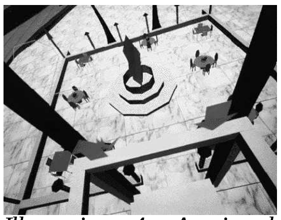


Illustration 4: A virtual hotel lobby as viewed from a simulated glass elevator. This scene is one of several virtual-reality environments used successfully in treating subjects for fear of heights.

In the domain of psychotherapy, virtual environments can alleviate different fears through the provision of a plausible and realistic environment that usually causes the fear in question [16]. Well-known examples are the fear of heights [17] (see Illustration 4), arachnophobia [18] or the fear of public speaking [19]. More generally, in the medical domain, virtual environments can aid surgeons or students to rehearse a particular operation, enabling them to evaluate different approaches. Also, they are used in medical disaster planning and casualty care. A summary of medical applications can be found in [20].

Virtual environments are an interesting way to place students in worlds in ways that were not possible before. Some well-known examples are the virtual gorilla exhibit [21] (see Illustration 5) or virtual gardens and environmental issues such as in the NICE project [22].

A number of art houses have a number of VR-based installations. One example is ZKM in Germany, with applications such as the Web Planetarium [23]. The

2 The naturalness of avatars is the subject of a debate. Virtual humans through a perfect modelling of real humans will typically raise the expectations of users who will assume that these virtual humans actually behave like real humans.

entertainment industry also benefits from virtual environments and computer games are probably the most successful applications of collaborative virtual environments. The quest for visual quality has driven the development of 3D graphics hardware to affordable prices. The game that probably had most impact on this revolution within the game industry was Doom. Additionally, virtual Environments can also be found in theme parks.

Illustration 5: The Virtual Gorilla Exhibit was developed to explore techniques for presenting information that would otherwise be difficult for users to learn and to explore zoo areas that were normally off limits to the casual visitor.

Ranging from training simulators to Augmented Reality devices in the battlefield, Virtual Reality technology and its derivatives can vastly increase the efficiency and accuracy of future military operations (see Illustration 6). Lately, the US army has experienced great success in recruiting staff through the distribution of a free computer game named “America's Army” [24].

1.4. Overview

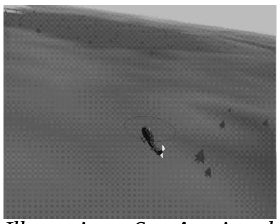


Illustration 6: A virtual helicopter engaged in a battlefield training scenario.

This report has been conducted as part of a PhD thesis and aims at providing insights in the hardware and software technologies that are necessary to the realisation of CVE applications. The report is organised as follows.

Chapter 1 rapidly presents the field of collaborative virtual environments and its applications.

Chapter 2 provides an overview of the technologies necessary to the realisation of the vision of collaborative virtual environments.

Chapter 3 describes some of the major CVE systems to date. The description is based on a loose classification of these systems.

Chapter 4 isolates a number of current trends in CVE systems. These trends span fields as various as communication architectures, communication protocols and major software choices.

Chapter 2 An Overview of VR technologies

2.1. Introduction

Virtual environments are presented to users through the utilisation of as many senses as possible. Users interact with the environment through Virtual Reality technologies. Many of these technologies are more or less familiar to most readers. They have evolved over the last fourty years from a series of novel ideas, inventions and concepts. To realise the vision of a parallel virtual world in which users can be immersed to feel as if this world was real, a number of varying technologies have to exist and be put in place. This chapter describes and categorises these technologies. It is aimed at showing the broad range of issues that exist.

All these technologies seek to integrate the user with the virtual environment so as to give him/her the sense of being immersed in the environment. To achieve this goal, it is necessary that the result of users actions on the input devices are reflected as quickly as possible onto the output devices. For example, when a user wears a Head Mounted Display (HMD), the tracking system should detect head tilting as quickly as possible to send this information to the system that will in return adapt the images shown onto both the user's eyes. Given the amount of information to integrate and process, there are inevitably delays at various stages of this input-output loop. Usually, the human perception system can accommodate minimal delays. However, if these increased, this would have adverse effects on the immersion experience and would reduce the effectiveness of the metaphor.

Illustration 7: The sensorama simulator in use.

2.2. Short Chronology of VR Technologies

This section chronologically outlines the major advances in VR technologies. It attempts to set the scene, but does not aim at being complete in any way. A more complete history of VR technologies can be found in [25]. This sections provides insights into the various and very different technologies that are necessary for the realisation of the vision of VR.

1962 Morton Heilig develops the "Sensorama Simulator" (see Illustration 7). Resembling one of today's arcade machines, the Sensorama combined projected film, audio, vibration, wind, and even pre-packaged odours, all designed to make the users feel as if they were actually in the film rather than simply watching it. The entire experience was pre-recorded, and played back for the user.

Illustration 8: The Head Mounted Display from Ivan Sutherland in use.

1965 Ivan Sutherland, famous for his work with the electronic sketchpad [26], describes what he calls a "kinesthetic display" [27]. It would allow one to use all their senses to interact with and gain knowledge from a computer. Sutherland describes an ideal computer display, which is in fact a room where matter can be completely controlled by a computer. Such a display would allow anyone in the room to have any sensory experience imaginable, hence fulfilling Sutherland's vision of a kinesthetic display.

1968 Ivan Sutherland works on Head Mounted Displays [28] for the first time (see Illustration 8). He presents users with computer generated scenes (in wireframe) and develops a scene generating tool.

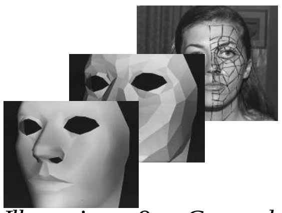


Illustration 9: Gouraud modelled the face of his wife through applying wires on her face and measuring.

1971 Henri Gouraud submits his doctoral thesis “Computer Display of Curved Surfaces” [29]. “Gouraud shading” or “smooth shading” is now a common technique in computer graphics to depict more realistic scenes. It is well suited for hardware acceleration. Gouraud shading (see Illustration 9) approximates the normal to the surface at all vertices of a polygon (using adjacent polygons), calculates intensity at the vertices using illumination equations and interpolates colour within each polygon.

1973 Bui-Tuong Phong submits his doctoral thesis “Illumination for Computer Generated Images” [30]. Phong shading gives better quality shading than Gouraud’s. It includes a detailed specular highlight but is more computationally expensive.

1976 P. Jerome Kilpatrick publishes his doctoral thesis “The Use of Kinaesthetic Supplement in an Interactive Graphics System” [31]. It introduces the basis for force feedback enabled devices.

Illustration 10: The movie map presented an interactive visit of Aspen.

1977 Based on an idea by colleague Rich Sayre, Thomas DeFanti and Dan Sandin develop an inexpensive, lightweight glove to monitor hand movements. The Sayre glove used bend-sensing technique unlike modern gloves which are based on optical sensors.

1978 Andy Lippman produces the “Movie Map” videodisk of Aspen (see Illustration 10). In the movie map, users could travel around the streets of Aspen on the computer, making right or left turns at will at any intersection and have the screen show film sequences of what they would see if actually driving around Aspen.

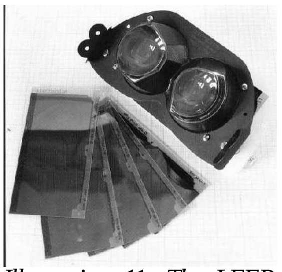


Illustration 11: The LEEP optics are the base component of all modern HMDs.

1979 Eric Howlett (LEEP Systems, Inc.) designs the Large Expanse Enhanced Perspective (LEEP) Optics (see Illustration 11). The LEEP optics provide for a very wide field of view for stereoscopic viewing. These optics are the base of all Head Mounted Displays, even though they introduce deformations at the periphery of the images.

1979 The Polhemus tracking system [32] is released (see Illustration 12). It is a six degrees of freedom tracking system that employs three orthogonal magnetic fields.

1982 Thomas Zimmerman patents a data input glove based upon optical sensors, such that internal refraction could be correlated with finger flexion and extension. This paved the way for a better dataglove [33].

Illustration 12: The Polhemus magnetic tracker.

1983 Gary J. Grimes, assigned to Bell Labs, develops the “Digital Data Entry Glove” [34], with flex sensors, tactile sensors at the fingertips, orientation sensing and wrist-positioning sensor. This is the first widely recognised device for measuring hand positions.

1983 Mark Callahan builds a see-through HMD at MIT. A see-through HMD allows to blend the real scene on the outside with the artificial scene of the virtual environment (see section 2.3.3).

1983 Myron Krueger publishes “Artificial Reality” [2].

1984 William Gibson writes about “Cyberspace” in Neuromancer [3].

1984 Mike McGreevy and Jim Humpries develop VIVED (VIrtual Visual Environment Display), a prototype system for future astronauts at NASA.

1984 Radiosity [35] is born at Cornell University. Radiosity decomposes the graphical scene in small patches and pre-computes the contribution of lights onto those patches. While pre-computation takes time, radiosity integrates well with real-time hardware rendering techniques while providing much more realistic scenes.

Illustration 13: Radiosity enables more realistic scenes that can be rendered in real-time once pre-computed.

1985 Mike McGreevy and Jim Humphries built a Head Mounted Display from monochrome LCD pocket television displays. LCD have a lower resolution but are cheaper and lighter than CRT-based helments.

1985 Tom Furness develops the “super cockpit”, designed to deal with pilot information overload: visual, auditory and tactile (see Illustration 14). The super cockpit was a research project but introduced a number of concepts that are now present in combat fighters. One example is HUD (Heads Up Displays).

Illustration 14: The super cockpit initiated what is now HUDs in combat fighters.

1985 First commercial liquid crystal shutter displays. They provide affordable stereo viewing.

1988 First system [36] capable of synthesizing four virtual 3-D sound sources. The sources were localised even when the head was moved.

1989 Jaron Lanier, CEO of VPL Research (Visual Programming Language), coins the term “Virtual Reality”.

1989 VPL Research and AutoDesk introduce commercial head-mounted displays.

1989 AutoDesk, Inc. demonstrate their PC-based VR CAD system (called Cyberspace) at SIGGRAPH'89.

1989 Robert Stone formed the Virtual Reality & Human Factors Group at the UK's National Advanced Robotics Research Centre.

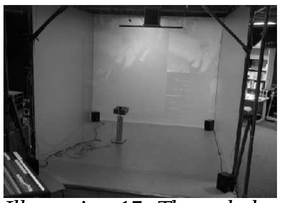


Illustration 15: Through the provision of room-size displays, CAVEs allows for the co-presence of groups, even though all perspective is calculated from the point of view of a single (tracked) user.

1990 J.R. Hennequin and R. Stone, assigned to ARRC, patent a tactile feedback glove.

1991 Division sell their first VR system.

1992 Division demonstrate a commercial *multi-user* VR system.

1992 Thomas DeFanti et al. demonstrate the CAVE [37] system at SIGGRAPH (see Illustration 15). A CAVE form a room in which walls, ceiling and floor are projected surfaces.

1993 SGI announce the RealityEngine, a very powerful 3D image generating engine.

1994 InSys and the Manchester Royal Infirmary launched Europe's first VR R&D Centre for Minimally Invasive Therapy.

1994 Doom hits the game market, it is the first of a new generation of computer games that have in common interaction through a 3D environment and the ability to play with or against other networked participants.

Illustration 16: Doom is the first one-person shooter game that used a 3D environment and co-presence of gamers as crucial aspects of the gaming experience.

2.3. Core VR Technologies

In the previous section, a quick history of VR-related technologies was presented. To realise the vision, a number of input and output aspects have to be covered. Input is essentially concerned with various tracking technologies that attempt to localise body parts in space in more or less unencumbered ways. Tracking is complemented by ways to interact with the environment, possibly receiving feedback. Interaction through data gloves has become famous. Output is mainly concerned with two different senses and channels: vision and hearing. Tactile feedback allows users to “feel” the environment in better ways.

2.3.1. Tracking Technologies

Tracking is one of the most important input channels involved in the field of virtual environments. Tracking devices will attempt to know the orientation and position of one or several body parts. Tracking the hands of the user is necessary to allow interaction with the environment and to show that this interaction takes place to all present users (including remote users in CVEs). A recent review of available tracking technologies can be found in [38]. Tracking is the key technology used when using Head Mounted Displays. In that case, the tracked position and orientation of the head are used to compute and visualise the user's viewpoint within the scene in real-time.

There are a number of available technologies for tracking single points and/or entire bodies:

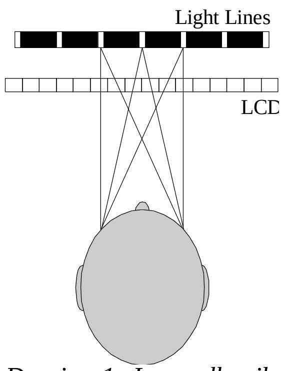
- A mechanical tracker is similar to a robot arm and consists of a jointed structure with rigid links, a supporting base, and an “active end” which is attached to the body part being tracked, often the hand.
- An electromagnetic tracker allows several body parts to be tracked simultaneously and will function correctly if objects come between the source and the detector. This type of tracker uses three magnetic fields and triangulation to compute distance and orientation [39].
- Ultrasonic tracking devices consist of three high frequency sound wave emitters in a rigid formation that form the source for three receivers that are also in a rigid arrangement on the user.
- Infra red (optical) trackers [40] utilise several emitters fixed in a rigid arrangement while cameras or “quad cells” receive the IR light. To fix the position of the tracker, a computer must triangulate a position based on the data from the cameras.
- There are several types of inertial tracking devices that allow the user to move about in a comparatively large working volume because there is no hardware or cabling between a computer and the tracker. Inertial trackers apply the principle of conservation of angular momentum. Miniature gyroscopes can be attached to HMDs, but they tend to drift (up to 10 degrees per minute) and to be sensitive to vibration. Yaw, pitch, and roll are calculated by measuring the resistance of the gyroscope to a change in orientation. If tracking of position is desired, an additional type of tracker must be used. Accelerometers are another option, but they also drift and their output is distorted by the gravitational field.

2.3.2. Presentation and Output Devices

Presentation to the user is made through three major senses: vision, audition and touch. For all of these senses, the major problem is to present the environment accurately as seen, heard or felt from the user's position and orientation within the environment. Additionally, vision is faced with the challenge of presenting a stereoscopic view of the environment. This section focuses on vision and audition. Touch will be covered in the next section, since it is highly connected to input devices.

2.3.2.1. Visual Presentation

LCD Shutter glasses are the most used device for stereoscopic viewing. A signal, sent by a transmitter, tells the glasses to alternatively allow light to pass through the right and left lens. This signal is synchronised with the scene on the screen, so that it is shown from two slightly offset viewpoints corresponding to the right and left eyes. There are also attempts to provide unencumbered stereoscopic viewing. For example, in the DTI display systems [41], both halves of a stereo pair are displayed simultaneously and directed to the corresponding eyes. This is accomplished with a special illumination plate located behind an LCD plate, as depicted in Drawing 1.



Drawing 1: In parallax illumination, the stereo effect is achieved through careful positioning of the light lines behind the LCD panel. To achieve a given horizontal resolution, this type of display requires twice as many pixels for each line of the panel.

HMDs contain two lenses (LEEP optics) through which the user looks at viewing screens. As for shutter glasses above, the computer generates two slightly different images, one for the left eye and one for the right eye. HMDs are often considered to be intrusive, see Illustration 17. To overcome this problem, alternative displays have been developed. One of them is the Binocular Omni Orientation Monitor. The BOOM is similar to a HMD, except that the user does not wear the helmet. The BOOM's viewing box is suspended from a two-part, rotating arm also forming a mechanical tracker.

Rather than presenting the environment to users using small displays close to their eyes, multi-screen displays and the CAVE™ form a more or less closed room in which walls, ceiling and floors are projected surfaces (see Illustration 15). This allows for the co-presence of groups, even though all perspective is calculated from the point of view of a single (tracked) user. Most similar setups use stereo to increase immersion. This is achieved through the use of shutter glasses, as described above.

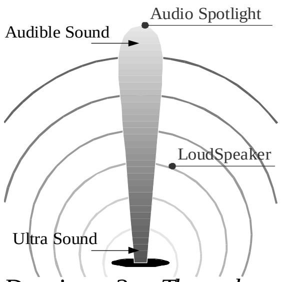
ImmersaDesk overcome the problems of cost and portability of CAVEs by offering one or two projected surfaces in a table-sized display. Again, stereoscopic viewing is based on shutter glasses. The reduction of size comes at the price of worse immersion and the drawback of seeing the remainder of the real surrounding room.

Illustration 17: An example head mounted display, also called a helmet.

2.3.2.2. Auditory Presentation

In addition to visual output, a complete virtual world must incorporate a three dimensional sound field that reflects the conditions modelled in the virtual environment. This sound field has to react to walls, multiple sound sources, and background noise, as well as the absence of them. Three dimensional audio is important since it brings life to environments that would otherwise only be visual. Furthermore, the human perceptual system uses audio cues in combination with vision to detect where objects are, whether they are moving or not, whether they are interesting or not, etc. Also, sounds can help humans detecting artefacts which are located behind other objects.

Surround sound, used in many theatres, uses the idea of stereo but with more speakers. Their delays can be set so that a sound can seem to move from behind the listener to in front of the listener. An example problem with this system is that a plane taking off behind the listener will appear to go by the listener's elbow instead of overhead. To overcome those problems, a little group of researcher have proposed ambisonic surround sound. It is a set of techniques, developed in the 1970s, for the recording, studio processing and reproduction of the complete sound field experienced during the original performance. Ambisonic technology does this by decomposing the directionality of the sound field into spherical harmonic components. The Ambisonic approach is to use all speakers to cooperatively recreate these directional components. That is to say, speakers to the rear of the listener help localise sounds in front of the listener, and vice versa.



Drawing 2: The ultrasound, which contains frequencies far outside our range of hearing, is completely inaudible. But as the ultrasonic beam travels through the air, the inherent properties of the air causes the ultrasound to distort (change shape) in a predictable way. This distortion gives rise to frequency components in the audible bandwidth, which can be accurately predicted, and therefore precisely controlled.

A solution to the problem of creating a three dimensional sound field comes from production of sound which is tuned to an individual's head. When sound reaches the outer ear, the outer ear bends the sound wave front and channels it down the ear canal. The sound that actually reaches the eardrum is different for each person. To resolve this problem, the computer must create a sound that is custom designed for a particular user. This is done by placing small microphones inside the ear canal, then creating reference sounds from various locations around the listener. Then the computer solves a set of mathematical relationships that describe how the sound is changed from being produced to being received inside the ear canal. These mathematical relationships are called Head Related Transfer Functions (HRTFs).

Finally, another solution for producing 3D sound is through audio spotlights. An audio spotlight produces an audio beam, similar to a flash light. The technology makes use of interference from ultrasonic waves, as described in Drawing 2. An audio spotlight can be used in two different ways: as directed audio -sound is directed at a specific listener or area, to provide a private or area specific listening space; as projected audio -sound is projected against a distant object, creating an audio image. In VR settings, it is possible to track the user's ears and aim the spotlight at the head. Augmented Reality (see section 2.3.3) can make use of projected audio to project onto a wall or object so that the sound will be heard as coming right from the projection point.

2.3.2.3. Input Devices

2.3.2.3.1. Input Devices

Illustration 18: The 5DT data glove measures finger flexure (one sensor per finger) and the orientation (pitch and roll) of the user's hand. It can emulate a mouse as well as a baseless joystick

Wands are the simplest of the interface devices and come in all shapes and variations. Most incorporate on-off buttons to control variables in the Virtual Environment. Others have knobs, dials, or joysticks. Their design and manner of response are tailored to the application. Most wands operate with six degrees of freedom. This versatility coupled with simplicity are the reasons for the wand's popularity.

Data gloves such as the one shown in Illustration 18 offer a simple means of gesturing commands to the computer. They use the combination of a 6DOF tracker to determine the position and orientation of the hand and of bending sensors to control an accurate virtual model of a hand in space. Data suits are an elaboration on the data glove concept by creating an entire body suit.

More generally, almost anything can be converted into a sensing device. For example, locomotion interfaces are energy-extractive devices that, in a confined space, simulate unrestrained human mobility such as walking and running. Locomotion interfaces overcome limitations of using joysticks for manoeuvring or whole-body motion platforms, in which the user is seated and does not expend energy, and of room environments, where only short distances can be traversed.

2.3.2.3.2. Tactile and Force Feedback

One of the biggest complaints about virtual environment applications is often the “lack of tangibility”. Although the area of tactile feedback is only a few years old, it has produced some impressive results. However, there is no interface currently built that will simulate the interactions of shape, texture, temperature, firmness, and force.

The area of touch has been broken down into two different areas. Force feedback deals with how the virtual environment affects a user. For example, walls should stop someone instead of letting him/her pass through, and pipes should knock a user in the shin to let him/her know that they are there. Tactile feedback deals with how a virtual object feels. Temperature, size, shape, firmness, and texture are some of the bits of information gained through the sense of touch.

Motion platforms were originally designed for use in flight simulators. A platform is bolted to a set of hydraulic lift arms. As the motion from a visual display changes, the platform tilts and moves in a synchronous path to give the user a “feeling” that they are actually flying. For interaction with small objects in a virtual world, the user can use one of several gloves designed to give feedback on the characteristics of the object. This can be done with pneumatic pistons, which are mounted on the palm of the glove as in the Rutgers Master II [42] (see Illustration 19) or through a lightweight, unencumbered force-reflecting exoskeleton that fits over a data glove and adds resistive force feedback to each finger, as shown in Illustration 20. In addition to providing haptic feedback to gloves it is possible to add it to a range of other objects. A common method for achieving this is via a flexibly movable arm, which resists the user’s movements according to the Virtual Environment. Many of these devices, such as the Workspace PHANToM system employ a stylus onto which the user’s hand or any other object can be attached. Such device also implement a mechanical tracker.

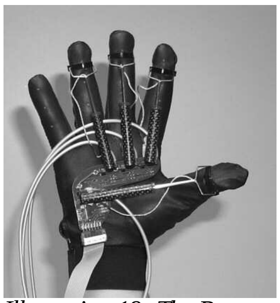


Illustration 19: The Rutgers Master uses pneumatic pistons to provide haptic feedback.

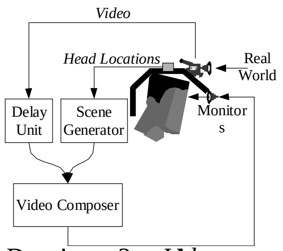
Illustration 20: The cybergrasp uses an exoskeleton to provide haptic feedback.

Any attempt to model the texture of a surface faces tremendous challenges because of the way the human haptic system functions. There are several types of nerves which serve different functions, including: temperature sensors, pressure sensors, rapid-varying pressure sensors, sensors to detect force exerted by muscles, and sensors to detect hair movements on the skin. All of these human factors must be taken into consideration when attempting to develop a tactile human-machine interface. For example, the Teletact Commander [43], use either air filled bladders sown into a glove, or piezo-electric transducers to provide the sensation of pressure or vibrations.

2.3.3. Augmented Reality

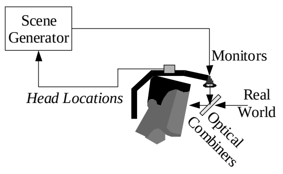
The real world environment provides a lot of information that is difficult to duplicate inside a Virtual Environment. An augmented reality system (see [44] for a recent survey) generates a composite view for the user. It is a combination of the real scene viewed by the user and a virtual scene generated by the computer that augments the scene with additional information. There are three components that are needed to make such an augmented reality system to work: a head-mounted display, a tracking system (or combination of such) and mobile computing power.

In most applications (see [45] for a number of application domains) the augmented reality presented to the user enhances that person's performance in, and perception of, the real world. The ultimate goal is to create a system such that the user cannot tell the difference between the real world and the virtual augmentation of it. To the user of this ultimate system it would appear that they are looking at a single real scene.



Drawing 3: Video see through HMDs use a closed view HMD. Those closed view HMDs are well known from virtual reality. Two cameras are mounted on the head, and the virtual image from the scene generator is combined with the image delivered by the cameras.

The computer generated virtual objects must be accurately registered with the real world in all dimensions. Errors in this registration will prevent the user from seeing the real and virtual images as fused. The correct registration must also be maintained while the user moves about within the real environment. Discrepancies or changes in the apparent registration will range from distracting which makes working with the augmented view more difficult, to physically disturbing for the user, making the system completely unusable. Errors of mis-registration in an augmented reality system are between two visual stimuli which we are trying to fuse to see as one scene [46].



Drawing 4: See through HMDs use optical combiners to mix the real world's image, and the virtual image from monitors. The opaque displays reduce the amount of light from the real world by about 30%.

The combination of real and virtual images into a single image presents new technical challenges for designers of augmented reality systems. There are basically two types of HMD that can be used: video see-through and optical see-through. An HMD gives the user complete visual isolation from the surrounding environment. Since the display is visually isolating, the system must use video cameras that are aligned with the display to obtain the view of the real world, as shown in Drawing 3. On the contrary, the optical see-through HMD eliminates the video channel that is looking at the real scene. Instead, the merging of real world and virtual augmentation is done optically in front of the user, as shown in Drawing 4. This technology is similar to heads up displays (HUD) that commonly appear in military air plane cockpits and recently some experimental automobiles.

The biggest challenge facing developers of augmented reality is the need to know where the user is located in reference to his or her surroundings. There's also the additional problem of tracking the movement of users' eyes and heads. A tracking system has to recognize these movements and project the graphics related to the real-world environment the user is seeing at any given moment. As suggested by [47], today's systems combine standard position tracking for gross registration and image based methods for the final fine tuning

Chapter 3 CVE Systems Survey

3.1. Introduction

In [48], a number of challenges are presented. The first challenge is the various kinds of distributed architectures used within systems, together with possible combinations of client-server and peer-to-peer at various levels of these systems. The second challenge is the scalability of the number of participants, active entities and the behavioural complexity of the virtual worlds and how interest management has been proposed as a solution to achieve this scalability. The third challenge is the migration of a number of relevant findings from 2D interfaces into 3D environments. Finally, the last challenge concerns human factors and how this new metaphor can change our use of computers.

This chapter summarises a number of past and present systems for shared multi-user virtual environments. This chapter is placed at the system level, looking at the differences and similarities between the number of existing CVE-oriented platforms.

Depending on the application domain targeted by the systems, different architectural solutions are used. For example, current multi-player games are making use of a client-server infrastructure. This choice is not only driven by technical reasons, it is also based on commercial reasons. Through a centralised architecture, game manufacturers gain control over the distribution and life of the virtual worlds. Servers are able to restrict the number of users, to implement some billing system if necessary (through the introduction of a single entry point), to control what users are able to do and not do within the game environment, etc.

Based on these considerations, and the number of applications that were highlighted in section 1.3, this chapter divides systems dependent on their application domains and capability in regard to application development. Some systems are tuned for a wide variety of applications. Therefore, the association of systems is sometimes a bit arbitrary. However, this classification provides with a more structured view of the current state of the art in the field of CVE. There are, broadly speaking four different categories of systems for the implementation and deployment of CVE applications.

- On-line systems are for the most part aimed at the entertainment market. Such systems have a number of requirements, the major one being their ability to run on the Internet, sometimes using low-quality connections such as old-fashioned modems. Companies seeking to establish such systems have to keep in mind the scalability of the solution in order to host a number of customers. Furthermore, commercial models have to be put in place behind the establishment of these systems and applications for the survival of these companies.
- Active systems are “closed” systems that aim at a broader range of applications. Such systems provide a number of facilities to develop applications within some sort of framework imposed by the system. Typically, they will impose a view on how data and applications should be organised and written and will make use of this view at a number of levels to optimise resources such as CPU or network utilisation.

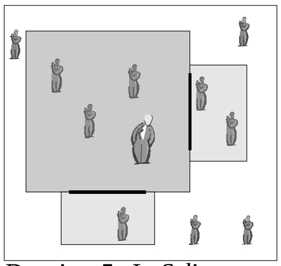
- Active toolkits and kernels will typically provide an application programming interface (API) on top of which designers and programmers will be able to build and develop applications. The level of this API will vary, but it will typically offer a number of facilities that are commonplace in CVE applications so as to relieve the burden of application development.
- Inactive systems are closed systems that offer little space for interactive applications. Such systems are tuned for the very restricted number of activities that they support.

Given the broad range of applications and systems, a number of standards and efforts towards standards have emerged. They hope to unify efforts worldwide towards common goals and to provide inter-connection of systems at a number of levels. Standards such as VRML or, to a lesser extent MPEG, are aimed at unifying data exchange between applications. Standards such as DIS or HLA are aimed at unifying network exchange between applications.

3.2. On-Line Systems

3.2.1. Spline - 1997

Spline [49] provides an architecture for implementing large-scale CVE based on a shared world model. The major contribution of Spline is the introduction of a novel division of space called “locales”. The world model is an object database containing all information about the content of a virtual environment. Applications interact with one another through making changes to the world model and observing changes made by other applications to the world model. The database is partially replicated to allow for rapid interaction. Copies are maintained approximately, but not exactly, consistent.


Drawing 5: In Spline, processes subscribe to the locale containing a participant and their immediate neighbours. Neighbouring relations are expressed through explicit boundaries. In this example, the locales subscribed to by the taller participant are grey.

Communication in Spline is mostly through multicast. However, to support users with low speed links, a special Spline server can intercept all communication to and from the user. The message traffic to the user is compressed to take maximum advantage of the bandwidth available. As part of this, audio streams are combined and localised before sending them to the user. Spline servers are replicated as needed so that no one server has to support more users than it can handle.

The key to scalability in Spline is its division of the virtual worlds into “locales” [50]. A locale has arbitrary geometry and this division of the world is purely an implementation issue. At a given time, a user only sees a subset of all the locales that compose a world. These are generally the locale containing the user's point of view and those neighbouring it. Each locale is associated to a separate set of multicast addresses. Using different addresses accommodates the communication of different kinds of data, for example, audio data, visual data and motion data. To help maintaining floating point precision over long distance, each locale has its own coordinate system.

Conceptually, locales failed to model a number of real world occurrences. For example, while it is natural for designers to associate locales to rooms in an environment that contains buildings and rooms, seeing through windows cannot always be possible in all configurations. While this could be argued to be a careless design by the environment developer, one has to remember that locales seek to solve the problem of scalability and that combining a number of locales into a bigger one can impair scalability by increasing the number of potential participants. Another example of conceptual deficiency is the inability for locales to model the differing permeability of media. Locales are associated to a number of multicast addresses for the different media that are supported by the system and a participant subscribes to the multicast groups of a locale and its immediate neighbours. This does not accommodate for the fact that sound can travel differently than light, in other words that it is possible to hear without seeing what is happening behind a wall. Again, such problems can be alleviated through the merging of locales, but this is at the price of scalability.

The protocol used for Spline (ISTP, the Interactive Sharing Transfer Protocol [51]) uses a hybrid UDP and TCP approach for the transmission of object updates within a locale. It uses a best-effort approach through the transmission of updates via UDP (multicast or unicast) in order to ensure the best possible interactivity. To detect packet loss ISTP uses sequence numbers that are incremented each time an object is updated. ISTP guarantees reliability through the resending of the full state via a TCP connection when gaps in sequence numbers have been detected or when state changes arrive too late. This technique impairs interactivity since it does not support causal ordering between related objects and since it relies on a constant delay for the discovery of packet loss.

3.2.2. GreenSpace - 1995

GreenSpace [52] is based on a peer-based distributed database that represents the virtual world. Every client is presented with an individual world view of the shared global GreenSpace world. The world data structure is based on so-called groups, which are collections of chunks. Chunks are objects of specialised types that define the data and methods for a world. Chunks are organised into groups and clients subscribe to groups. A client process, which has a world view of the collection of groups that it is interested in, manipulates the chunks of that group. The client's action are reflected to all other remote clients that have the same interest in that group.

GreenSpace uses several communication mechanisms for the transmission of information between remote clients. Multicasting is predominantly used to pass transitional messages. An example of a transitional message is a position update. The transmission frequency of these messages itself ensures their reliable transmission. Messages that involve a change of state are more critical and a reliable multicasting protocol (RMP [53]) is used for that purpose. Finally, peer-to-peer TCP/IP communication is used in particular cases such as when sending the initialisation data on world entrance.

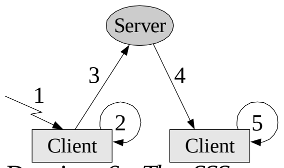
The network architecture of clients [54] is based on two different modules that can either run on the same machine or on two separate machines. This has a number of advantages. The communication layer can be moved to another machine that has full multicast access (for example, outside a firewall). Furthermore, this arrangement allows changes to the communication protocol between clients without modifying the way applications actually communicate with one another.

A central, lightweight server is assigned with two simple tasks: assigning multicast channels and allowing each host to discover who else is in the world or universe.

Greenspace relies solely on the existence of a multinational multicast architecture for communication between remote peers. However, the establishment of such an architecture (called the MBone) is impaired by precautions at the corporate level and its non-availability for home users. Finally, there are not enough details in the various papers describing the system to judge whether RMP is an appropriate protocol for the transmission of critical state data. RMP has four levels of quality of service: unreliable, reliable, source ordered, and total ordered. The adequate selection of these levels has a number of implications on the interactivity of the system and none of the standard levels seems to provide support for causal ordering.

3.2.3. Community Place - 1997

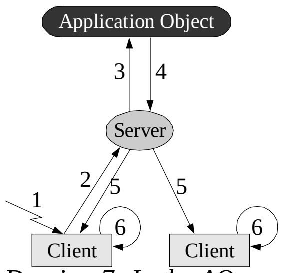
Community Place [55] is based on a shared-world abstraction where the common world is composed of a database of objects. Community Place has a particular emphasis on the Internet and its technologies. As a result, it uses VRML as the description language for the content of the world and attempts to scale in the number of users and active objects while trying to address the capabilities of low-end consumer client PCs (slow modem connection, no graphics acceleration).



Drawing 6: The SSS approach is suitable for a number of simple shared scene updates. It propagates script messages created at the interacting client to all other clients via the server.

Community Place is based on a hybrid client-server and peer-to-peer architecture. 3D browsers perform communication solely through servers, which are responsible for the dispatching of communication messages between relevant browsers. At initialisation time, a 3D browser reads the initial description of the scene in the form of a VRML file with associated behaviours. It then contacts the server which will inform the client of any other users in the scene and any other objects not contained in the original description scene, and their respective location.

The communication between the browser and the server is optimised in two ways. First, it is based on a very efficient representation of 3D scene transformations. Second, it has an open-ended support for script specific messages. This mechanism enables Community Place to send and receive script-level messages that allow the browser to share events and so support interaction with the 3D scene. There are two possibilities for this cross-browser communication. In the simpler model ("Simple Shared Scripts"), scripts communicate directly with one another through the server, as depicted in Drawing 6. The drawbacks of this model are ownership and persistence: Issues such as object ownership and locking have to be resolved at the script level and for each script; furthermore, all modifications applied to an object will be lost once all browsers have left. As a solution to those problems, Community Place introduces the concept of "Application Objects" (AO). These reside off the browsers, on the network, and communicate with the server, as depicted in Drawing 7. They are composed of three parts: the 3D representation, the associated scripts that accept user input and communicate back to the AO, and the AO side code that implements the application logic. AO define a controller for the application and let it live even when all browsers have left the virtual world.



Drawing 7: In the AO approach, messages generated at the interacting clients are forwarded to a specific process where all code resides via the server. Results are propagated back to all clients, including the origin of the interaction.

Community Place's approach to scalability is two-fold. It reduces communication between the clients and the servers as much as possible as explained above. Furthermore, it uses spatial areas of interest to select which clients should receive information sent by a given client or application object. This is based on auras that surround participants and a distributed aura manager that automatically creates groups of clients based on the intersections of their respective auras. The groups are associated to multicast groups, which allows a hierarchy of servers to calculate aura collisions between objects in a distributed manner.

The major drawback of the AOs used in community place is the introduction of a number of message exchanges before the result of an interaction can be transmitted to all interested parties. Indeed, interaction will be discovered at one client, will have to transit through the server to the host in charge of the AO before being processed and sent back to all interested clients via the server(s). While this technique alleviates the problems of synchronisation, it impairs interaction through the introduction of a number of delays, even for the interacting client.

3.2.4. AGORA - 1998

AGORA [56] is a system for the realisation of virtual communities based on the VRML standard. AGORA has a shared and centralised database and is based on the client-server approach. AGORA divides the space into regions of static sizes and client browsers are associated to one and only one region at a time. The central server filters information so that clients will only receive information about other clients that are part of the same region. To minimise the delay for an incoming client to receive the initial state of the virtual world, AGORA introduces the concept of an interactive VRML server (I-VRML). The principle consists of storing information sent by all remote clients in a single server so as to be able to reproduce a complete VRML snapshot of the environment upon new connection of a client. As a result, the incremental addition of objects that have been modified or of remote avatars is reduced to a minimum.

To minimise traffic between the clients and the server, the server uses a special packet-delivery technique that consists of grouping avatars and object updates into so-called notice vectors. For this grouping to happen, the server delays update delivery and the clients rely on dead-reckoning techniques for the interpolation of the positions of objects and avatars. This is similar to the techniques employed in NetEffect (see section 3.2.7).

AGORA solely relies on a single server and while special attention is paid to reduce the traffic from the server to the clients, AGORA suffers from the problem of scale at the server side. As the environment grows and the number of participant grows, the amount of networking and computing resources will grow and the server will stop being able to cope with this flow of data. This is especially true since the server is also in charge of the dynamic construction of the initial VRML scene sent to new connecting clients. Furthermore, the concept of notice vectors, while minimising the number of messages sent to clients and thus saving precious bandwidth impairs interaction by introducing arbitrary delays.

3.2.5. Living Worlds - 1998

In [57], an implementation of the former Living Worlds [58] proposal is presented. Living Worlds was an effort to standardise multi-user extensions to VRML97. The implementation is based on three layers. The lowest level is a generic notification system called Keryx. Above this notification system, generic support for state sharing is provided by an event interface. Finally, the top layer consists of the support for zones, a region of the space according to the Living Worlds proposal.

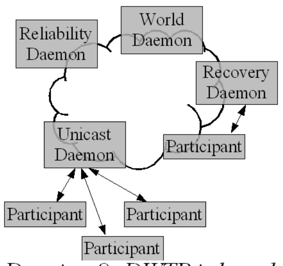
Keryx supports anonymous interaction between loosely coupled parties. It implements a notification server as a cloud of events. Sources inject events into the cloud, and the notification service takes care of delivering them to clients. Clients of the notification system are decoupled by an Event Distributor (ED). Sources send events to an ED which delivers them to clients. An ED also implements a number of services through the interception of certain types of events. Clients are able to send subscriptions in the form of filters to an ED. They will in return receive all events that match the subscription filter. To provide more scalability, event distributors can be connected together. The preferred transport mechanism is TCP, but a reliable protocol on top of UDP is also provided.

In the paper, the authors divide the virtual environment into zones. Each zone is associated to an extended Event Distributor: the zone server. Zones do not need to be defined spatially, but they represent collections of information of interest to participants. The zone server implements a generic state-sharing protocol using events such as object creation, differential state update, complete state update or object deletion. Clients use subscription filters to restrict their event flows to the zone that they are interested in. To solve the problem of concurrent access to objects, a pilot (a specific client) is associated to shared objects. Only the pilot is able to modify an object. Ownership migrates from client to client and finally to the zone server if no client is interested in an object any more. This ensures persistence. The implementation also has some support for prediction (dead-reckoning techniques) and for spatial filtering.

This implementation of the Living Worlds proposal suffers from the introduction of a relaying server that delays the arrival of packets at all interested clients. This is especially true since the implementation only shares a single environment and does not provide for any partitioning of the worlds. Furthermore, TCP, as the preferred choice for communication, has a number of disadvantages. It introduces unnecessary queues and enforces reliable transmission of all packets, while virtual environments should accommodate for the non-arrival of less important packets to achieve a high degree of interaction and minimise delays, e.g. position updates.

3.2.6. SmallTool - 1997

SmallTool [59] is a VRML-based browser and architecture for the realisation of large-scale multi-user virtual environments. SmallTool uses a specifically tailored protocol called DWTP – The Distributed World Transfer and communication Protocol [60]. DWTP is based on a hybrid approach. It relies both on TCP and UDP for the realisation of its goals. Typically, TCP is used for the transmission of information chunks of large sizes or during the initialisation phase. Unicast of multicast UDP is used for the remaining of communication.



Drawing 8: DWTP is based on a number of services implemented as daemons. Communication is essentially multicast based. However, recovery of lost packets is based on unicast communication between participants and a specific daemons. Multicast-unaware participants can connect through unicast daemons.

DWTP enforces an infrastructure in the form of a number of types of daemons that can be replicated over the Internet to achieve greater scalability. These are depicted in Drawing 8. Reliability daemons detect UDP packet losses using a protocol based on positive acknowledgement. Recovery daemons allow connected applications to recover lost packets. World daemons transmit the initial content of virtual worlds and their populating avatars and embodied applications. Unicast daemons allow multicast unaware clients to join and participate through the implementation of multicast-to-unicast bridges.

SmallTool introduces a number of VRML extensions to divide virtual environments into hierarchical regions, to allow transportation from one world to the other through portals, to represent users as logical entities and to represent shared applications embodied within the environment by a number of geometrical entities.

SmallTool introduces several classes of behaviours to reason about shared behaviours and interactions. Autonomous behaviours are either completely deterministic or independent of the state of their shared copies. However, such behaviours might be influenced by user interactions and might then need resynchronisation. Synchronised behaviours are not completely deterministic, but can be treated as such for time periods until the next resynchronisation. Independent interactions are interactions that do not depend on any other interaction performed concurrently. The effects of such an interaction require immediate synchronisation of all the copies of the concerned objects. Finally, shared interactions occur when several users have the possibility to perform a certain behaviour and might experience concurrent access to objects.

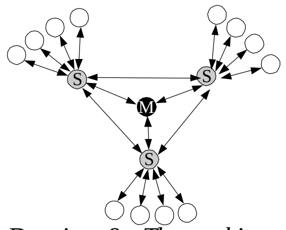
To accommodate this classification and provide for shared behaviours, SmallTool distributes events generated at user interactions. The resulting cascade of VRML events is not distributed until an explicit synchronisation. Synchronisation is usually initiated by the process that generated the interaction, but this cannot always be the case, and the world daemon might take over responsibility. Shared interactions use a hierarchical object locking mechanism. Locks are only satisfied by the world daemon and a time-out mechanism avoids starvation.

In SmallTool, the world daemon is in charge of two major tasks: ensuring persistence of the worlds (and transmitting their contents to newcomers) and synchronising interaction. World daemons are associated with an environment and there is no partitioning. Consequently, in highly interactive applications where a large number of participants are interacting, world daemons could experience scaling problems.

3.2.7. NetEffect - 1997

NetEffect ([61] and [62]) is an infrastructure for developing, supporting and managing large-scale virtual worlds for use by several thousands of geographically dispersed users using low-end computers and modems. The system partitions the world into so-called communities. Each community is associated to one server only, while one server can handle one or several communities. At any point in time, all users of a community are connected to the same server.

NetEffect is based on a graph of servers with one master server and a number of peer servers, as depicted on Drawing 9. The master server has two major goals. Firstly, it takes care of initial connection and distribution of clients and maintains each user's personal database. Upon initial connection, a client is handed the address of a peer server, responsible for the “nearest” community (in the virtual sense) and all further communication will occur between the client and the peer server. Secondly, the master server is in charge of load-balancing between the different peer servers. It is able to decide and migrate a whole community from one server to another if necessary. All communication between servers and client and servers is solely TCP based.



Drawing 9: The architecture of NetEffect is based on a graph of servers with one master server in charge of server load balancing and initial connection. Periphery servers undertake the major burden of network traffic.

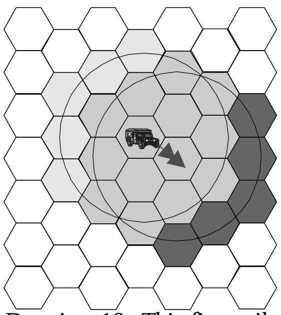
To reduce network usage between the servers and the clients, NetEffect uses a number of techniques. Firstly, it divides communities into a hierarchy of places and ensures that clients only receive updates for other clients that are within the vicinity of the same place. Secondly, it uses so-called “group dead-reckoning”. Using this technique, the server waits and accumulates movement updates for well-chosen groups of clients during a short period of time. Once it has expired, a vector is sent to all relevant clients. These will use the position and velocity information to approximate the position of other clients in real-time. Finally, NetEffect support bi-part audio communication in a peer-to-peer manner. Audio communication between two participants is mediated by the peer server, but all GSM-encoded traffic will transit directly between the clients.

NetEffect addresses the problem of scale from an enhanced chat perspective. While the movement filtering and the server hierarchy are suitable for such scenarios, the architecture would not be able to accommodate tighter interaction between the clients, since this would increase the load on the servers. Another drawback of the system is the way that it supports audio. As soon as larger groups of participants wish to talk, the peer-to-peer model will imply the unnecessary duplication of large numbers of audio packets to allow their transmission from the sender to all potential receivers.

3.3. Active Systems

3.3.1. NPSNET-IV - 1995

NPSNET [63] is the successor of SIMNET [64]. SIMNET intended to provide interactive networking for real-time, human-in-the-loop battle engagement simulation and war-gaming. SIMNET is based on a distributed architecture with no central server. Entities connected to the simulation broadcast events to the network (and thus all other connected simulation processes) at regular intervals. Receivers are responsible for deciding upon the relevance of the message and for calculating the effects according to a similar algorithm at all receiving processes to allow for fair-play. In between object position updates, receivers interpolate their position using dead-reckoning techniques. The type of information that is exchanged within a military simulation has been standardised in a standard called Distributed Interactive Simulation (DIS), a standard that emerged from the SIMNET effort. The standard describes the semantic of a number of Protocol Data Unit (PDU) and how these should be interpreted at the receiving side, more information can be found in section 3.6.2.1.


Drawing 10: This figure illustrates the principles of area subscription in NPSNET and what happens when a vehicle changes cell. When the Jeep above changes cell in the direction of the arrow, it will stop subscribing to the multicast groups that are associated to the cells which are light grey and will start subscribing to the groups associated to the cells which are dark grey. The origin and destination positions of the AOI of the vehicle are also depicted.

NPSNET IV is a network software architecture for solving the problem of scaling very large distributed simulations. NPSNET is the system that pioneered the idea to logically partition the space into regions. In NPSNET, regions are hexagonal cells. Hexagons are used because they are regular, have a uniform orientation and have uniform adjacency. Each region is associated to a distinct multicast group, therefore allowing a smooth transition from the broadcast model employed in SIMNET and previous DIS-based simulation. Each vehicle is associated to an Area Of Interest (AOI) which is typically defined by a radius. This is explained in more details in Drawing 10. The size of this radius depends on the type and functionality of the vehicle.

NPSNET is tuned for ground-level military simulations of real-life situations. The size of the cells and the division itself are calculated to accommodate military vehicles in normal situations. Using hexagons with a 2.5 km radius, a vehicle that advanced at the world record advance rate would only change cell every hour, leading to very few multicast group subscriptions and unsubscriptions.

The major drawback of NPSNET is precisely what it is tuned for. NPSNET is aimed at ground-level military simulations where vehicles move at normal speeds. There are number of situations where this is not adequate. For example, while being suited for open-spaces or environments where participants are evenly spread, the constant subdivision of the space into cells is not well suited for environments with a number of buildings and rooms which will typically attract a number of persons within a small area. For example, a virtual university campus would be approximately the size of one NPSNET cell but would have to accommodate thousands of participants within this very cell. This is where the network culling introduced by multicast groups would fail: client machines would have to process too many incoming messages and to render too many entities.

Finally, NPSNET enforces all connected entities to send their state frequently. This allows new participants or temporary disconnected participants to easily recover and catch up with the current state of the virtual world. However, this scheme puts a permanent load on the network, even if object states are not changed.

3.3.2. PaRADE - 1997

The Predictive Real-time Architecture for Distributed Environments, PaRADE [65], is a an experimental platform for the realisation of real-time multi-user close interaction applications. In particular, PaRADE tries to address the issue of maintaining sufficient causality and consistency within the constraints of real-time many-humans-in-the-loop interaction. The general idea behind PaRADE is that, through knowledge of time delays between all participating peers, a user's actions may be predicted locally and sent to other participants in advance so that each user may observe the actions as they occur. To make this possible, PaRADE needs help from the application. For example, the movement of controlled avatars may be restricted in a manner that allows the prediction of both movements and interaction with other entities, thus minimising the need for roll-back.

PaRADE builds upon a number of features to achieve the goals described above. Wall clock time is maintained on all hosts by clock synchronisation. This is achieved through robust estimation techniques mixing a novel algorithm for Round-Trip Time (RTT) estimation and NTP (Network Time Protocol). Replicated databases are maintained through the communication of non-deterministic event and local calculation of deterministic events. Causality and update control over the replicated databases is guaranteed through entity locking and the exchange of entity sequencers. Ownership requirement prediction is used to overcome delays incurred by sequencer exchange. To this end, PaRADE uses heuristics such as spatial proximity, collision detection, interest groups or explicit request.

All events are stamped with their actual or predicted time of commencement according to wall clock time. This ensures that events that represent continuous changes over time will result in consistent evaluation at each participating process once the event is delivered. Roll-back strategies are provided so that wrongly predicted events (and their results) are negated. PaRADE introduces the notion of sufficient causal ordering. It allows the application to dictate where the before-after relation of the standard causal ordering must be applied. This allows only a subset of the sequencers associated to objects to be encapsulated in the event delivery.

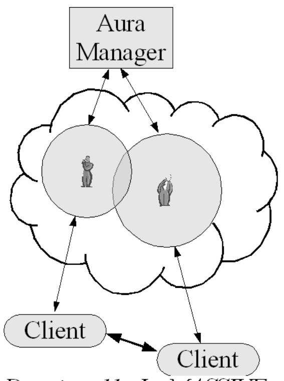
PaRADE does not address the problem of scale through the partitioning of space. However, this is not the purpose of this experimental system. PaRADE lets the application and the application programmer help the system in order to improve real-time performance in close collaboration. However, this requires a thorough knowledge of the implications of the design decisions. For example, deciding which objects should be part of the sufficient causal ordering is not an easy task. Neither is the decision upon the heuristics to employ to transfer the object locks to make sure ownership has been transferred in time. Finally, being able to balance user intent and application restriction to guarantee good prediction results will perhaps not always be possible.

3.3.3. The MASSIVE Family

3.3.3.1. MASSIVE-1 - 1995

MASSIVE [66] (Model, Architecture and System for Spatial Interaction in Virtual Environments) is a CVE system with a specific focus on large-scale multi-user virtual environments. MASSIVE distinguishes itself from other systems through a full implementation of the spatial model of interaction (see section 4.2.4).

The communication architecture of MASSIVE is based on typed peer-to-peer connections, which utilise a combination of RPC, shared attributes and streams. MASSIVE uses spatial trading to negotiate interactions between processes sharing the virtual environments. Spatial trading combines the virtual reality techniques of aura collision detection with the distributed systems concept of attribute-based naming service. Aura collision is performed within an aura manager. Upon aura collision, the manager passes out mutual interface references to the objects involved which enables them to establish a peer connection within which all further communication will take place. This is depicted in Drawing 11.



Drawing 11: In MASSIVE-1, the peer-to-peer communication between processes controlling participants is mediated through their auras and the aura manager, responsible for collision detection and for putting participants processes in contact.

An important example of information exchange between peers is focus and nimbus so as to compute awareness. MASSIVE also supports the concept of portals. Upon transportation from one world to the other, an object will notify its aura manager so that other objects can be notified. The aura manager will possibly transfer the responsibility for the object to another aura manager and the object will initialise a new connection.

In MASSIVE, all communication is via connection between pairs of typed interfaces. This restriction is a significant shortcoming and a source of potential network inefficiency. Indeed whenever a group of peers have to communicate, it would be more appropriate to rely on group communication such as multicast to transmit most of the information. Another potential problem of the approach of MASSIVE is the scalability of the aura manager. As the number of participants within one world grows, the aura manager will have to handle an increasing number of remote objects and compute their collisions.

3.3.3.2. MASSIVE -2 - 1999

MASSIVE-2 [67] is an implementation of an extension to the spatial model of interaction (see section 4.2.4). This extension introduces the concept of third party objects. Third parties can have two effects on awareness: attenuation or amplification of existing awareness relationships, and the introduction of new aggregate awareness relationships. Third party objects are medium specific in the sense that they can operate differently in the audio medium than the graphics medium. Such objects are embodied within the space and they might apply their effects recursively to one another.

Third party objects might be activated under three different cases. Firstly, the third party object's awareness of an object represents and expresses the degree of membership that the object has on the third party (regions, rooms, etc.). Secondly, the third party might be activated according to how aware other objects are of it and mediates mutual awareness of the other objects (common focus on an object). Thirdly, the effects of a third party might depend on how much one object is aware of it and how much it is aware of another object. In this last case, the third party consumes information from its members and make it available to external observers as an aggregate view at a lower level of awareness.

There are five main applications of third party objects: world structuring and regions, including nested spatial structures; aggregate views such as dynamic and mobile crowds of participants; common foci, i.e. objects which affect the mutual awareness of sharing participants; representational of group services such as non-local communication, subjective presentation; load management through the automatic creation and destruction of third party object by the system to cope with computational and network load.

In MASSIVE-2, each third party object is mapped directly to a corresponding group object, each group object managing one or more multicast groups (for various media for example). The spatial structure of third party objects within the virtual environment maps directly onto a hierarchy of groups managed by group objects. Participants send to one and only one group at any given time, it is their most local entirely containing group. Groups may send an aggregated stream of information upwards in the hierarchy. Reception of groups is controlled through the concept of auras around a participant or another process. Group reception is in the majority controlled by spatial overlap between auras and group representation. Consequently an observer will receive information from potentially many groups.

MASSIVE-2 makes use of two multicast groups for the discovery of objects and object data sending. The discovery group is used for a state transfer mechanism that requests and initialises the content of a group at a new incoming peer that has just joined. Apart for audio and video, MASSIVE-2 uses a reliable multicast protocol for all communication. This protocol uses sequence numbers on messages, which allows receivers to detect errors (NACK) and request retransmission using unicast.

Through the introduction of multicast and of third party objects, MASSIVE-2 improves greatly over its predecessor. However, while the concept of data aggregation is of high relevance to minimising network load, it introduces an non-negligible computing load. It is unclear where, in the current implementation, this processing is taking place and how much load it actually represent in typical situations. Through the use of unicast, the reliable multicast protocol of MASSIVE-2 is subject to repair implosion. While the wide use of partitioning techniques minimises the risks for repair implosion, it still might happen within aggregated information regions such as crowds of avatars.

3.3.3.3. MASSIVE-3 - 2000

MASSIVE-3 [68] is built to overcome some of the limitations of MASSIVE-2. For example, to support lower-bandwidth networks and to support extensibility and modification in better ways. MASSIVE-3 answers a number of requirements: the support for hierarchically structured virtual objects, the support for a number of consistency policies, a route to the support of modem-based users, a capable and flexible mechanism for interest management and the support for persistent virtual worlds.

MASSIVE-3 is based on the concept of an environment. An environment is modelled as a fully replicated distributed database. Each database can contain any number of hierarchically structured data items where the environment itself is at the root of the hierarchy. An environment is controlled by a single master process. Every change to an environment is represented within the system by an event object. Communication between replicated copies of an environment is logically multicast but realised using a TCP-based client-server model where the master process is the server. Normal applications never change their local copies directly, instead events are all copied to two sending and receiving queues and executed on reception. There are a number of standard data types that can populate an environment: 3D transformations, geometry or behaviours (coded in C++). Some of these data can be marked as local only and will not lead to any network communication.

MASSIVE-3 implements a consistency model pioneered by PaRADE (see section 3.3.2). Each data item has a designated owner. To separate reliable updates from unreliable ones, each item has two distinct sequencers increased in accordance to the reliability of the update. Every message includes a list of item sequencer values that must be reached before the event can be enacted. Ownership can be transferred in agreement with the current owner. Finally, events can be posted ahead of time.

MASSIVE-3 implements three different update policies. The first, “classic”, policy consists of transferring ownership from one client to another before being able to update a given data item. The second policy let all updates transfer and be decided upon by the current owner of the data item. Finally, the last policy, inherited from CIAO (see 3.5.3), combines both previous ones by requesting ownership and requesting the owner for the execution of an update simultaneously. Consequently, once the ownership has been transferred during the first update, all further updates will originate from the new owner, which guarantees the best interaction possible.

MASSIVE-3 attempts to establish an interest management model that is not based solely on space. It extends the notion of Locales from Spline (see section 3.2.1) by sub-dividing them into aspects, defined themselves by a number of functional classes, an organisational scope, a fidelity and a cost. MASSIVE-3 allows a locale to contain aspects with the same functional and organisational scopes, but with different fidelities (lower resolution, composite objects, e.g. crowds). An observer can then choose an aspect at a particular fidelity level according to some policy. This is done through the offering of a framework for the development of selection policies and the implementation of a number of standard policies based on topological distance (like Spline), Euclidean distance (like NPSNET), awareness (similar to MASSIVE-2) and benefit/cost. MASSIVE-3 uses different policies for replication, graphical rendering and audio rendering.

Apart from the questions raised over PaRADE, the major issue with MASSIVE-3 is how to select an actual policy for interest management. There might be ways to select these automatically, based on, for example, CPU and network load. But, this type of selection would require a carefully designed benefit/cost model or variations (subject to hysteresis), seeking for an optimal. In an example, the authors describe the effect of a number of manual selection policies. However, the complexity of this selection, with a number of parameters to assimilate, makes it an inappropriate mean for users with a medium technical level.

3.4. Active Toolkits and Kernels

3.4.1. MR Toolkit - 1993

The MR Toolkit [69] simplifies the development of VR applications by providing standard facilities required by a wide range of applications. The MR Toolkit is based on the Decoupled Simulation Model that structures VR application into four distinct components: the geometric model, computation, presentation and interaction. In this model, the computation component proceeds separately and asynchronously of the remaining components.

Based on this model, MR Toolkit provides support for common VR devices such as 3D trackers, HMDs or gloves and support the distribution of the user interface and data over several workstations. One typical example is the separate rendering of the two views necessary for stereo viewing in HMDs onto two different workstations. MR Toolkit categorises all the running processes necessary for a single-user application into master and slave, so that communication between processes always occur through the master process.

The MR Toolkit can be extended to allow multiple independent MR Toolkit applications to communicate with one another across the Internet. This is achieved through the MR Toolkit Peer Package [70]. Using this package the master process of an MR Toolkit application can transmit device data to other remote applications, and receive device data from remote applications. Application-specific data can also be shared between independent applications. The Peer Package is based on a peer-to-peer model where each peer maintains a list of all other connected peers. Peers are connected pairwise and a peer may send a message to any or all other peers at once.

Finally, the MR Toolkit is complemented by the Object Modelling Language (OML), a procedural and interpreted language with fundamental data types and operations for geometry, object-oriented programming and behaviour specifications. OML behaviours react to a combination of events, generate some changes in the state of the object and trigger other behaviours through the generation of new events. The Environment Manager (EM) [71] allows the creation of distributed environments through the sharing of a number of OML objects. Objects are replicated to a relevant subset of all connected applications. EM differentiates between shared object, local objects and ghost objects (the representation of local objects). EM offers different schemes for controlling concurrency, i.e. ownership token passing and ownership and access permissions. Whichever the scheme is, EM will ensure that the shared state is distributed at all copies of an object. The major contribution of EM is its support for “multi-user-different-content” applications as opposed to most systems. EM provides for simplistic subjective views mechanisms.

The various tools associated with the MR Toolkit do not provide any partitioning of the space in themselves. While this can probably be achieved at the application level, it is questionable whether these instruments would be sufficient to achieve partitioning for large scale multi-user environments. Furthermore, the Peers Package is solely based on UDP and does not offer any guarantees when it comes to data transmission. While this is not generally a problem on local networks, packet loss and reordering are common place on the Internet.

3.4.2. Urbi et Orbi - 2000

Urbi et Orbi [72] takes an alternative approach to the problems related to the realisation of multi-user shared environments. In this approach, the worlds have enough semantics for a terminal to represent it faithfully whatever its nature is (from textual commands to 3D rendering). To this end, Urbi et Orbi uses conceptual graphs to arrange the environments. The objects contained in an environment are linked to one another with relations that carry a number of semantics, both spatial and non-spatial. Examples of such relations are “is localised on”, “is adjacent to” or “is composed of”.

Urbi et Orbi is based on a fully distributed model where no connected process is aware of the current state of the whole world. Participants' processes are associated to cells and partitioning techniques based on the graph relationships are used to achieve scalability. The structure of the graph changes constantly, for example as a participant changes cell. The system makes use of group communication, on top of the Ensemble system [73]. Group communication lets the system choose different policies for the distribution of messages, e.g. causal ordering or unreliable multicast.

Urbi et Orbi is constructed around a novel programming language called Goal. Goal is used at two different levels: it is the interface between the system and the programmer and it is also used to transport information both between modules and between machines. Goal is distribution and replication oriented and carries semantics about distribution within the language itself. Goal is also a scripting language which eases code migration.

Urbi et Orbi imposes a new and different language onto programmers and environment designers. While the reasons for this introduction are understandable, they provide a possible hindrance to acceptance of the system as such. Furthermore, a new language also introduces limitations by not allowing the reuse of existing external code or migrating such code. Again, this can impair acceptance. Finally, even though it is based on Ensemble, a well-known system that is tuned for scalability, Urbi et Orbi has not been tested in real-life conditions, i.e. on the Internet with varying delays, congestions, etc.

3.4.3. Avocado - 1999

Avocado [74] is an object-oriented framework for the development of high-end distributed, interactive VE applications. Avocado provides programmers with the concept of a shared scene-graph, accessible from all processes forming a distributed application. Each process owns a local copy of the scene graph and the contained state information, which is kept synchronised. Avocado combines the familiar programming model of existing stand-alone toolkits with built-in support for data distribution that is almost transparent to the application developer.

All avocado objects are field containers representing object state information as a collection of fields. They support marshalling for the reconstruction of exact object copies remotely. Compatible fields can be connected so that whenever the source field changes, the destination field is set to the same value. Avocado uses a distributed shared memory model where processes can attach themselves to one or more distribution groups. The implementation ensures that all modification made to a local copy of an object are replicated across all its replicas. Avocado only replicates fields, while the orthogonal graph constructed by the field connectors is not distributed. This ensures that the result of connections are executed at the process that has created them and associates one process for the behaviours that can be described using this mechanism.

Apart from the C++ API, Avocado features a complete language binding to the interpreted language Scheme. Therefore, application programmers can take a two tracked approach to development. Complex and performance critical functionality is implemented in C++, while the applications themselves can simply be a collection of scheme scripts. This allows rapid development, since scripts can be tested and debugged while the application is running.

Avocado is built on top of Ensemble [73] and enforces a total message ordering to guarantee consistency in the presence of incremental state updates. This introduces an additional network latency, especially in Internet settings. Furthermore, it does not entirely account for the vast majority of applications where some actions can be conducted in parallel in various parts of the environment. Total ordering enforces the delivery of messages in exactly the same order while not taking into account the local causality of these separate group of actions. Another drawback of the Avocado approach is the impossibility to migrate execution ownership through the enforcement of local field connections. This is also valid at the programming level, where no apparent support for script and/or application migration is provided. Finally, through a focus on high-end systems, Avocado sets aside any partitioning techniques, which might impair some classes of applications.

3.4.4. DEVA - 2000

DEVA [75] intends to be a framework for the realisation of large-scale distributed virtual reality applications. Large scale raises a number of challenges: number of objects, complexity of the behaviour required of these objects, complexity of individual rendering techniques, number and geographical distribution of simultaneous users, and number of co-existing and interacting applications. To this end, DEVA divides its architecture into three distinct components. Within the DEVA framework, rendering and spatial management is handled by a specific system called MAVERIK [76]. Additionally, DEVA comprises two core services: 1) an execution environment, a means by which applications and user interactions can be integrated in a single run-time environment and 2) a distribution architecture, a means of distributing the actions of the user and the behaviour of the applications across a wide area network.

DEVA is based on a client/server architecture where distribution objects are mirrored at the clients. The server is in fact a cluster of processors running identical processes called server nodes. Together, all server nodes form a single multi-threaded parallel virtual machine capable of processing large numbers of objects. The intention of the server is to provide a computing resource for multiple virtual environments, and maintain a far heavier processing load than clients. All communication is based on TCP/IP and the communication strategy is based upon the assumption that inter server node communication is fast compared to client/server communication.

The programming model of DEVA is one of communicating “entities”, which can represent objects in the virtual environment, properties of the environment or more abstract programming concepts. Entities export a number of methods that can be called by other objects, and implement these using optimised imperative code (C++). Upon creation entities are associated to one of the nodes of the server nodes. DEVA provides means for entity migration to help balancing processing load if necessary.

Entities are decomposed into two different parts: an objective and subjective part, the former being located on the server and the latter on each client. Typically, an object represents what an entity does and a subject represents how the entity looks, sounds and feels. DEVA encourages low update rates on objects when communicating from the subject to the object and vice-versa. For example, changes caused by the manipulation of a subject (at a client) can make use of policies to only send a fraction of the changes at the object. Similarly, communication between the object and the subject might use techniques such as parametric curves to smooth out the effects of these discrepancies.

DEVA introduces the notion of components to describe the behaviours of entities. A component is a collection of methods and attributes relating to a single concept that can be attached or detached from an entity at run-time. A component is divided into a single object and multiple subjects, as described above. Entities contain two list of components, one inherited from the environment and the other containing their own innate behaviour. Components are written in C++.

DEVA attempts to minimise communication between the server and the clients by filtering away packets and making this visible for the application programmer through the concept of subjectivity. Confining to this model requires very careful design and will perhaps not always be possible. The introduction of a pool of servers taking care of the vital part of entities introduces inevitable network delays at interaction time (for example, for the acquisition of locks on entities). Finally, DEVA is solely based on C++ and requires the subjective parts of the components to be migrated and replicated at all clients, which impairs heterogeneity and environment dynamism.

3.4.5. Continuum - 2002

Continuum [77] is a Java middleware for building large-scale real-time networked virtual environment applications. Continuum is an open software framework from which several profiles can be derived, each addressing a specific application domain. The platform relies on a partial replication model and a configurable event communication mechanism that allows arbitrary consistency and synchronisation protocols to be implemented.

In Continuum, the objects populating the space encapsulate a state and a behaviour. The types of the objects are described using a so-called Object Definition Language (ODL). ODL supports basic types, classes and interfaces, and class inheritance. An interface or a class definition consists of typed attributes that describe the shareable state of the object and one-way methods that describe the external interface of the behaviour of the entity. ODL is seconded by a compiler that will generate stubs to be filled in by the programmer, apart from all system and communication oriented code. For the description of object behaviour, Continuum is interfaced to a reactive programming framework that provides programmers with primitives dealing with concurrency and event handling.

The replication model of Continuum is based on a master-slave relationship. Only one master can exist for an object at a given time. This is typically the creator of the object, but mastership is allowed to migrate if necessary. Continuum separates the behaviour of an object between its master and slave replicas. This allows the implementation of application-specific techniques to mask network latencies, such as dead-reckoning. Continuum offers a framework to plug in different consistency policies. For example, for highly dynamic objects, the master replica will constantly push in an unreliable manner the kinematic state of the object. Other objects can make use of the high-level event communication based on the shared methods described in the ODL. Concurrency control can be handled using an approach centralising decision at the master replica.

Communication within Continuum is based on events. An event represents an arbitrary change in the state of the object. Events are allocated to event channels and it is up to the consistency policies to decide the types of events to exchange between replicas. Continuum provides support for unicast and multicast UDP, as well as TCP. It also offers an SRM-like [78] reliable multicast protocol.

Continuum implements a default aura management policy for the spatial partitioning of objects. This policy is doubled with a resource mapping policy which determines the allocation and multiplexing of event channels to perform the filtering. A grid-based partitioning policy is an example of such a policy. Aura managers within each simulation use a control protocol to discover sets of objects that become visible and discard objects no longer visible.

Through the offering of a pluggable architecture, Continuum offers an attractive solution to the problem of large-scale virtual environments. For example, while the current aura mechanism does not take into account different media, extensions to the framework are made possible for this to happen. However, this offering has the price of complexity: deciding which consistency policies to use, which partitioning of the space to use and which concurrency policy to use is a difficult task for the application programmer.

3.4.6. NPSNET-V - 2002

NPSNET-V [79] attempts to put in place the technologies that will allow large-scale shared virtual worlds to adapt, scale and evolve continuously without being taken off line. To this end, NPSNET-V layers and aggregates plug-ins so that it is possible to create software in which the only functionality not provided as a plug-in is a minimal binding mechanism (or micro kernel). The system is entirely based on Java and structures host applications as component hierarchies, also called application graphs.

In NPSNET-V, the atom of composition is the module. Modules can form graphs using so-called module containers. Modules can be added or deleted from containers at any time, and removal operations are recursive. All modules are required to possess instance names, which allows the naming of any module using file-system path conventions. Modules are grouped in containers according to their functional role and are expected to maintain awareness of their neighbouring modules (via the containers). Modules have to be explicitly initialised, started, stopped and destroyed. They also provide seamless updating through specific methods to replace and retire them, with enough handshaking to migrate the state of an old module into a new one.

To communicate with one another, modules make use of a common interface layer based on a number of invariant Java interfaces. One type of shared interface is the property. Other modules can request to listen to property modifications. Another specific type of interface is the service. Services are placed in the hierarchy and announce a module's ability to perform a critical application role. Only one module can provide a given service within a given context, but services may be looked up upwards in the hierarchy, which allows their overriding if necessary.

NPSNET-V integrates a mechanism for XML-based configuration and serialisation of the module hierarchy. This allows all or part of the module hierarchy to be saved and recreated at a later stage. Together with an included configuration HTTP service, this mechanism allows the transfer of application components between clients and, for instance, to reproduce the entire state of a running application remotely.

In NPSNET-V, shared entities are organised in a hierarchical containment structure. Entities are modular constructs that conform to the model-view-controller pattern and that represent individual elements of virtual worlds. Additionally to the model-view-controller separation, the entity adds two crucial roles: observers and remote proxy. Observers allow models to notify their dependent view and controllers of changes to the model state. Entity state replication depends on the remote proxy interface, or ghost, which acts as a stand-in for a remotely owned master model.

In itself, NPSNET-V does not provides solutions to the problems of scale at run-time. It provides only few of the numerous techniques that will be necessary for the realisation of massive shared virtual worlds. NPSNET-V poses the grounds onto which standards for networking, physical modelling, graphics, etc. can possibly be developed.

3.5. Inactive Systems

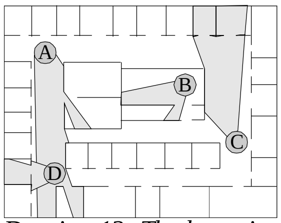
3.5.1. BrickNet - 1994

BrickNet [80] is a software toolkit based on the client-server model. A client can connect to a server to request objects of interest. These objects are deposited by other clients connected to the same server or by another server on the network. In BrickNet, virtual worlds are composed of objects that are self-contained and embody their graphic, behavioural and network properties. One particular feature of BrickNet is the possibility to have virtual worlds with different content. A virtual world manages its own set of objects, some or all of which may be shared with the other virtual worlds on the network.

BrickNet uses the servers as object request brokers and communication dispatchers. A server keeps track of client requests for objects and object updates, and services these requests when possible. Additionally, servers keep tracks of clients' status, their addresses and port information, and manage the sending or receiving of packets of information. All communication between clients has to transit via a server in BrickNet.

BrickNet is implemented using a mix of C and Starship programming languages. Starship is an interpretive language and can be used to describe objects' behaviour and share the execution of these behaviours on the network [81]. Starship is actually used throughout the system as the language for application development. Both the shared object hierarchies and their behaviour are created using the language. BrickNet distinguishes four classes of behaviours: simple, environment-dependent, reactive and capability-based. The first two categories execute in parallel without any synchronisation, except against local environmental conditions such as the frame rate. The two last categories use the ownership mechanisms of BrickNet to decide where the behaviours are to be executed and distribute their results.

BrickNet is based on the client-server model. As such delays are introduced when messages transit from the clients to the server in interactive situations. In large-scale multi-user situations, the servers then become the bottleneck of communication. Finally, the description of the system does not specify whether BrickNet uses some sort of reliable protocol above its UDP-based communication service.



Drawing 12: The key principle that drives RING is the recognition that in highly occluded environments such as above, only A and D need to exchange movement updates.

3.5.2. RING - 1995

RING [82] is a system that supports interaction between large numbers of users in virtual environments with dense occlusion. RING is built on top of a client-server architecture where clients exchange information through servers only and where several servers can be connected together to exchange information about their respective clients. At the servers, object-space visibility algorithms are used to compute the region of influence of each state change, as depicted in Drawing 12. Update messages are sent only to the small subset of workstations to which the update is relevant. In some cases, these updates are relayed through several servers to reach their client destination.

RING represents the environment as a set of independent entities, each of which has a geometric description and a behaviour. Entities with a dynamic behaviour are either autonomous or controlled by human input. Every RING entity is managed by exactly one client workstation. Clients execute the programs necessary to generate behaviour for their entities. Clients maintain “surrogates” for some of the remote entities, typically the entities which are visible to them according to the server.

Between updates, surrogate behaviour, in RING's case position updates only, is simulated by every client.

This server oriented design allows servers to process messages before propagating them to other workstations. Typical operations are culling, augmenting or altering messages. Culling is implemented using precomputed line-of-sight visibility information. This pre-computation is based on a subdivision of the environment into cells. During the simulation, servers keep track of which cells contain which entities by exchanging periodic updates messages when entities cross cell boundaries. Real-time position updates are propagated only to servers and clients containing entities inside some cell visible to the one containing the updated entity.

Depending on the capabilities of the available workstations and networks, clients can send messages to server(s) via unicast or multicast. Similarly, servers can exchange information using multicast. Each cell is associated to a multicast group. Entity updates are relayed to the multicast group representing the cell in which the update occurred. Each server listens only to the groups for the cell visible to one region of the environment. When a server receives a multicast message (from another server), it propagates it to clients with entities residing in the cell represented by the multicast group. This arrangement avoids the periodic messages that are otherwise necessary to detect cell boundary crossing.

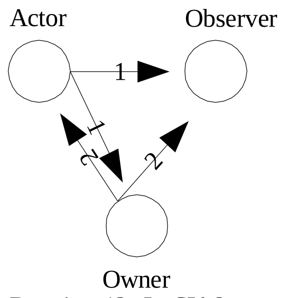
The author of RING has experimented with different server topologies. For example, in [83] it is shown that associating servers to regions of the environment and letting clients migrate from one server to the other as needed reduces the total amount of traffic.

The type of environments targeted by RING are highly-occluded environments. As such, the approach fits well for such environments. But it is not generic since it cannot be applied to more open spaces. Furthermore, the approach does not address the problems raised by other media. For example, audio reflects off walls and transits differently across space. Consequently, the pre computation of visibility information cannot apply to audio the same way. Furthermore, RING is based on a server approach. While the communication and the algorithms are tuned for high-capacity and high-speed filtering and forwarding of messages, this approach still needs the transmission of packets up to application user-space, the computation necessary to make decisions and the sending back of packets down to the network hardware. Typically, this will take much more time than a multicast-only approach where packets are effectively filtered at the network hardware level.

3.5.3. CIAO - 1999

CIAO (Collaborative Immersive Architectural layOut) [84] is a system that aims at achieving optimal responsiveness in multi-user virtual environments. This is achieved through a novel multicast-based optimistic concurrency control mechanism doubled by a user interface that attempts to show the effects of possible conflicts to let humans solve them using different means.

CIAO is based on an hybrid peer-to-peer and client-server model. The architecture of CIAO comprises three different types of nodes: participant nodes, a session manager and object information servers. Communication between participant nodes is multicast-based while communication with the servers is done through TCP. Participant nodes perform tasks such as human input processing, real-time rendering, avatar management and network communication. A session manager helps during the initialisation phase when new participants wish to join a new environment: it allocates multicast addresses and points at environment description files. The data for these files is obtained from the object information server. In CIAO, the content of the virtual environment is fully replicated at all sites.



Drawing 13: In CIAO, permission to perform operations is optimistically handled. The operation as well as the ownership transfer are sent from the interacting site. On reception, the owner will acknowledge the transfer through the sending of a validation message to both the interacting site and all its observers.

In CIAO, users manipulate objects without waiting and notify other participants of their actions immediately. If there are conflicting operations on the same object, an associated token is used for the maintenance of consistency. The semi-optimistic algorithm uses an incremental sequence number. When a participant starts manipulating an object, it assumes being able to acquire the lock on the object and multicasts the incremented sequence number, as depicted in Drawing 13. This operation is, later, validated or cancelled by the current owner of the object. To avoid conflicting operations confusing the users, a novel user interface is introduced. Only a ghost representation of the object is used during manipulation and the results of the manipulation are shown once it is terminated. During the manipulation, any manipulation by another user is also shown using another ghost. This allows users to solve the conflicts verbally or any other communication means.

CIAO focuses on minimising latency when manipulating common objects. As such, the algorithm that it proposes is only a “last resort” to keep consistency. However, the algorithm as described is sensitive to the non-reliability of multicast transmission. Indeed, inconsistencies will anyhow be introduced during token passing if some processes miss the reception of these messages.

3.6. Standards

3.6.1. RTP/I - 2001

Distributed interactive media are about managing the shared state of a medium. All participants are potentially able to change that state. Shared virtual environments of shared whiteboards are examples of such media. RTP/I (Real-Time Application Level Protocol for Distributed Interactive Media) [85] is a standardised application level protocol framework that addresses the problems of this specific class of application. RTP/I aims to make applications interoperable and allow the direct reuse of common functionality in the form of generic services.

RTP/I is based on a media model. This model is oriented around four concepts: the application, the environment, states and events. In the RTP/I terminology, the environment is the part of the information needed by the participants that remains constant over the course of a session. The state of the medium can change for two reasons, either by the passage of time or by non-deterministic events. The state itself is divided into sub-components, such as 3D objects or avatars in a CVE. Event and state transmissions form the basis of a distributed interactive media stream within which multiple senders can emit information.

RTP/I is especially designed to meet a number of requirements that are specific to this class of applications and generic enough to be needed in all applications of this class. These requirements are the framing of state and event data, the support for consistency and fragmentation, a standardised way to query the state of a sub-component, the ability to convey meta information, and a flexible protocol design. To this end, RTP/I reuses many aspects of RTP, including the concept of two distinct protocols for the transportation of data and meta information.

TeCo3D [86] is a shared workspace for dynamic and interactive 3D models which uses RTP/I as its transport mechanism and acts as a proof of concept. TeCo3D uses a completely replicated distribution architecture with reliable multicast as the transport layer. Inconsistencies caused by the operation delays are handled by deliberately delaying the local updates to match the transmission delays. Each node keeps a history, and if inconsistencies occurs, the situation is rolled back, the conflicting operation is carried out, and situation is rolled forward back to the current time. As a last resort, TeCo3D also includes a method for explicit state request.

3.6.2. DIS and HLA

Motivated by a requirement for simulation-based tools to support acquisition, planning, training and analysis efforts, the defence community (most notably the US Department of Defence) has heavily invested in research programmes on distributed simulation. This has resulted in a suite of protocols, collectively known as DIS (Distributed Interactive Simulation), a standard [87] for interconnecting large numbers of heterogeneous simulators across local and wide area networks.

Two important trends have emerged in the DIS community: a wish to use DIS technology for non defence-oriented applications, and a recognition of the fact that, while the DIS protocol constitute the low-level basis for distributed simulation, a set of standards and tools at a higher abstraction level is required to enable the timely and effective development of applications.

3.6.2.1. The DIS approach - 1995

DIS aims at the interconnection a large number of simulators using both local and wide area networks. It allows the combination of a wide range of features within one single shared synthetic environment: real-time interactive human-in-the-loop simulators, autonomous agents and numerical simulations of physical processes to name a few. DIS is based on three key principles:

- *object/event architecture*: Synthetic environments are composed of a collection of objects interacting with one another through the way of events. These events mediate the status and actions of these entities and result in the transmission of standardised network packets called PDUs (Protocol Data Units).
- *Autonomous simulation nodes*: Simulation nodes broadcast the events of all the objects that they are responsible for. It is the responsibility of the receivers to decide upon the effects of these events on their own state and local view of the environment.
- *Transmission of state change information only*: DIS is highly dependent on a technique called dead-reckoning and the existence of predictive modelling algorithms at the receivers. Continuous state modifications are transmitted at a reduced update rate and the receivers locally extrapolate future information based on past information (see section 4.3.2).

DIS is a proven concept and is extensively used as standard protocol for simulator interoperability. An advantage of the DIS concept is that all DIS compliant simulators, including networked VR, can operate within one virtual environment. However, DIS has three major drawbacks. Firstly, messages can get lost or arrive in the wrong order due to the use of the UDP/IP protocol. Secondly, the messages sent are part of standardised PDUs of fixed size, although generic PDUs exist to communicate any type of data. Finally, due to the broadcast mechanism the scalability is rather limited.

The fixed-sized PDUs in combination with the dead-reckoning algorithms are well suited to communicate state-updates and events between simulators (short messages). However, PDUs are relatively large and entire PDUs need to be broadcast even if just one attribute is changed.

3.6.2.2. The High Level Architecture (HLA)

The HLA standard [88] has a twin goal: easing interoperability of heterogeneous simulations and reusing simulations and their components. To that end, HLA focuses on interface specifications without making specific demands on the implementation of each simulation. The standard is based on four concepts:

- *The Run-Time Infrastructure* (RTI) is an implementation of a distributed operating system that will be the base software layer for future HLA applications. The RTI takes care of communication between all simulation models.
- *The Interface Specification* is a formal, functional description of the interface between the HLA application and the RTI.
- A set of *technical rules* is defined to which an HLA participant has to comply.
- The *Object Model Templates* are standardised formats to define the functionality of simulation models and the interaction between models. Thus the capabilities of all simulation models are defined before the actual simulation takes place.

HLA divides simulations into a number of federations concentrating on specific areas of the simulation. Federations are populated by a number of members called called federates. Federates may be simulation models, data collectors, simulators, autonomous agents or passive viewers. Simulated entities, such as vehicles or aircraft, are referred to as objects and live entities can be mapped onto objects by using surrogates. The Federation Object Model (FOM) describes all possible interactions between the federates and the Simulation Object Model (SOM) the capabilities of a single federate.

The state of each object is defined by its attributes. Objects interact with each other and with the environment via interactions which may be viewed as unique events, such as grasping an object, or a collision between objects. Attributes are own by federates, initially the object creator, but ownership may change during execution.

In order to reduce network traffic and limit the amount of computation each federate has to perform, HLA provides a mechanism of publication and subscription. Upon initialisation, each simulation registers (with the RTI) which objects and which attributes it will represent (publication). It also registers which attributes and interactions it needs in order to be able to perform its task (subscription). Federates can not only subscribe to attribute types, but also to (ranges of) attribute values, so as to maximise filtering at the source. Both publications and subscriptions are dynamic and may be changed during a session.

The Run-Time Infrastructure supports HLA compliant simulations through a number of services. The main categories of services are the management of federations, subscriptions, objects, ownership and time.

3.6.3. VRML - 1997

The VRML97 standard is a file format for describing arbitrary three-dimensional scenes and objects. Sometimes referred to as VRML 2.0, VRML97 is the defined standard that is derived from the VRML 2.0 format specification. A VRML97 world is a structured text file that is interpreted by VRML enabled browsers. The precursor to VRML97, VRML 1.0, enabled the sole description of geometry and materials; that is, any arbitrary three-dimensional structure using any texture or colour. The VRML97 standard complements the missing geometry primitives of VRML 1.0 and supports animations and behaviours through complex event routing and behaviour scripting.

VRML is aimed at single users exploring 3D scenes or worlds, which are distributed across the Internet. The file format of VRML is structured around the concept of nodes. The scene graph is defined though a hierarchy of nodes, where some nodes contain other nodes and others are leaf nodes (not containing other nodes). One of the advantages of VRML is that nodes can be defined through a URL (Uniform Resource Locator). This enables a single VRML world to exploit resources (3D data, textures, sounds and so on) across the Internet. Although the VRML97 standard is taken to be the basis of a number of server-based multi-user systems, the standard itself is solely single user.

VRML97 pays specific attention to the ability to animate 3D scenes. An event propagation system is standardised. Animations are driven by the notion of linear key framed interpolation. Parts of the scene graph can be associated to a number of keys that will describe their value at some moment in time. Their real-time values will be interpolated between the key frames as time goes by. A number of graph properties can be controlled this way, for example position, orientation, colours and normals.

VRML97 also pays specific attention to the ability to write applications and bind them to the scene graph in adequate ways. For the application development, VRML97 allows scripts to be associated with parts of the scene graph. Two languages are specified, although script compliance is not a requirement from the standard: ECMA script and Java. Scripts are triggered by the event system and are provided with a standardised interface to modify the scene graph as the result of their execution.

Further work in the area of 3D on the Web includes the forthcoming X3D standard. Work in X3D is concerned with expressing the capabilities of VRML97 using the Extensible Mark-up Language (XML). A number of working groups within the consortium are looking at multi-user extensions of the format.

3.6.4. MPEG-4/SNHC – 1999

MPEG-4 [89] is the next generation standard for the coding of audio-visual data. The standard includes since 1997 an activity called SNHC, Synthetic/Natural Hybrid Coding, which deals with the coding of synthetically and naturally generated audio-visual information. As such, SNHC is primarily concerned with the compression of specific media streams beyond traditional audio and video, e.g. geometry, text, animation, text-to-speech. SNHC and MPEG provides therefore a totally different approach to compression, based on content, composition and the resource capabilities of the receivers.

MPEG has adopted VRML as its main scene composition mechanism and extended it for the support of 2D objects, face and body animation and 3D mesh coding. Another major difference is that MPEG uses a binary coding representation of the VRML scene, known as BIFS (Binary Format for Scene Description). BIFS supports content streaming and MPEG pays special attention to efficient scalable coding of a number of content types, e.g. VTC (Visual Texture Coding), geometry mesh compression, facial and body animation parameters, text-to-speech synthesis and structured audio. MPEG supports rudimentary interactivity through the processing of user input events. In version 2 of the standard, a back channel allows some information to be sent to the transmitter end, thus allowing for multi-user scenarios.

3.7. Multi-User Games

Multi-user games are one of the most successful applications of collaborative virtual environments. Their success justifies specific attention within this report. Multi-user games have their roots in so-called Multi-User Dungeons (MUDs) [90]. MUDs are text-based games which originate in the early 1980s. The games space is represented of rooms, doors and artefacts and is represented by a shared database. Additionally to browsing, manipulating the database, creating rooms and giving them behaviour through an embedded language, users can also communicate with each other in real-time.

Since MUDs, multi-user facilities have become an entire component of most modern games and a sales argument. The game industry is following the same trend as the academia and there is a current focus on massively multi-player games (MMGs). The premise of MMGs is a large shared game world inhabited by thousands simultaneous players. MMGs emphasise often on social interaction and on the story line. The number of registered users in games such as Lineage, The Sims Online, Star Wars Galaxies, EverQuest and Ultima Online records to millions. Most existing MMGs are based on a client-server model. The servers have to handle large data flows and multiplex these in real-time. Scalability within these architecture is ensured through overly dimensioned servers and through employing server clusters. In this context, highly interactive games such as Quake or Doom divide the worlds into many small isolated game sessions with a handful of players each. There exist a number of commercial middleware systems for server based game architectures, Terraplay [91], Butterfly.net [92] and TeraZona [93] to name some of them. Group communication and interest management are used in a number of academic distributed game implementation, including AMaze [94], MiMaze [95], Mercury [96], Continuum (see section 3.4.5) and [97]. People working in the entertainment industry have recently started to publish more openly their ideas and solutions, see [98], [99] and [100] for a few examples.

Game designers and developers have understood the potential of tightly integrating a scripting language to the game platform for a long time. As described above, MUDs were already based on an embedded language. Even though it is difficult to get much information from the entertainment industry, it seems that most games are doubled by a scripting language of some sort. For example, the Unreal engine contains UnrealScript [101]. Instead of inventing specific languages for the sole purpose of describing game logic [102], developers seem now to have moved to a more practical approach and start using well supported and more generic languages such as Python [103], LUA [104] or TCL [105]. The major commercial advantage of scripting languages for games is double. First, they allow to prolong the life of games through facilitating the later release of “extras” in the form of new levels, new worlds, etc. Second, game developers are driven by very tight development cycles and games that are being released are usually not entirely tested and debugged. Scripting languages allow to patch problems quicker so as to satisfy users through a “continuous” stream of game updates via the Internet.

3.8. Conclusion

Since the beginning of the 1990s, CVE systems have flourished and this chapter has isolated some of the most important ones. The design and implementation of so many systems have led to a number of learnings and these have started to be consolidated into a number of standards and proposals, as described in section 3.6. The next chapter will provide an overview of the major technical solutions that have been found so far.

Chapter 4 CVE Systems Trends

4.1. Introduction

CVE systems are complex to built and the issues that they need to address are many. This chapter describes the most important issues and solutions that have been provided and points at relevant systems whenever possible. Research in the field have been directed into two major directions. On one side, a lot of effort has addressed the issues that are related to CVE data distribution and the networking aspects that are directly relevant. On the other side, some more effort has been put on understanding the human aspects of the technology and how the sense of presence can be achieved. However, the issues of application development have been less scrutinised, even though a number of these issues go hand in hand with data distribution and human interaction aspects.

At the networking level, there are a number of issues that all systems are trying to address in a number of ways. Networked virtual environments try to ensure the illusion of a shared virtual world so that the effects of users' interactions are perceived at all remote peers within an acceptable amount of time. The time elapsed between an action and its perceived results conditions the illusion and the feeling that all remote participants actually share an identical simulated world.

Sharing a virtual world is more than just navigating around and talking to one another, it also consists of focusing on common objects, interacting with these in “real-time” and concurrently. This object-based interaction is an essential component of the human communication that can occur through the metaphor as it is instantiated by all these systems. Consequently, most systems have to incorporate mechanisms to handle concurrency and to make sure that users take turns or agree on a common goal in a natural manner. This will keep the virtual world consistent with all participant's will and ensure that the effect of their concurrent actions will be perceived at all participants in the same order and with the same results.

An additional issue is one of causality, or making sure that a synchronised course of actions is perceived in the same order by all participants. This issue is crucial when several remote participants are engaged in the course of several simultaneous actions. Ideally, all the effects of these actions have to be perceived at all participants in the same order, to avoid ambiguities in the gestural communication that emerges from their different interactions.

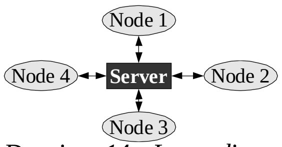
At the application development level, one of the crucial issues is to provide enough support while still offering flexibility and openness. For example, the structure of a number of systems have been influenced by the graphical aspects of virtual environments and the concept of a scene graph. The quest for flexibility is driven by the novelty of the field and the introduction of different programming interfaces or of a variety of programming languages has been seen as one possible solution. Finally, a later trend consists in the influence of middleware technologies on the field of CVE. This trend answers, again, the double quest for flexibility and openness, and aims at providing enough power and “tweaks” so as to best support the application or domain at hand.

4.2. Architectural Decisions

4.2.1. A Central Point or Not?

There are two general models between which all systems described in this chapter oscillate: these are the client-server and the peer-to-peer model. Both have advantages and drawbacks.

4.2.1.1. Client-Server

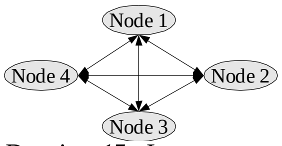


Drawing 14: In a client-server architecture (Nodes represent clients), information is sent by the clients to the server that chooses to forward it further to the other interested nodes.

The client-server model makes one process, the server, responsible for a (sub) part of the environment. In this model, clients send object updates to the server which is in charge of further delivery to other clients than the sender, as depicted in Drawing 14. Examples of such systems are AGORA, DEVA, BrickNet, RING, Living Worlds and NetEffect. The advantage of this model is that it gracefully solves the problems of concurrency and causality by letting the server decide upon the course of actions at a central and common point. The server is responsible for a number of objects, responsible for transferring ownership between all participants who wish to interact with the objects and responsible for the order of a set of actions. Another advantage of a server solution is the possible gain in bandwidth at the client side. The server is able to take a number of decisions upon which object updates should be transferred, at which pace, within which vicinity, as exemplified by RING. All these decisions can be made in concert with the clients and their known available bandwidth access. Consequently, client-server solutions are often used for community-oriented systems, which target consumer computers with modem connections.

The client-server approach has a number of drawbacks. From the human-aspects point of view, its main drawback is the introduction of arbitrary and unnecessary communication delays. Indeed, before any decision has to be taken at the client side, the client has to ensure that it will be allowed to perform the action. Furthermore, the server is responsible for the dispatching of object updates to all interested participants. Therefore, network packets will travel twice: once from the source client to the server and a second time from the server to the destination clients. On a congested Internet, this travel time can be measured in hundreds of milliseconds, if not in seconds in the worst cases. Server architectures are also facing the problem of scale. As the number of clients grows, as the number of objects grows, their processing and network load will increase. A solution, as employed in a number of systems, is the multiplication of servers in various ways (arbitrary, by virtual geographical position, by actual geographical position, etc.). This solution has a financial cost that might not be sustainable within all contexts. Finally, through the introduction of a central point, a server-centric solution introduces possible long-lived failures. As soon as one or several servers stop working, for hardware or software reasons, part of the virtual environment will also stop working and stop living.

4.2.1.2. Peer-to-Peer (Unicast)



Drawing 15: In a peer-to-peer unicast architecture, each node sends information to the other interested nodes.

In the peer-to-peer model, all participants' processes will communicate directly with a restricted and well-chosen set of other participants, as depicted in Drawing 15. Early systems such as MASSIVE-1 and the MR toolkit used this model. As opposed to the client-server model above, this model has the advantage to shorten network delays by suppressing an additional hop between the client and the server. Packets sent by a client will reach all interested clients at once, without the explicit help of a third party server. Since interaction is one of the key points justifying the very existence of CVEs, this solution is generally preferred for systems tuned for highly interactive environments. This solution has the advantage of not putting the burden of scale on any specific central point within the network. Instead, used in conjunction with partitioning techniques, peers will only have to communicate with a restricted set of peers. As consumer hardware is gaining in both communication and processing power, peer-to-peer systems are gaining importance.

However, there are a number of drawbacks to the peer-to-peer approach. For example, concurrency and causality of actions become more complex problems since they have to involve the arbitration of a number of remote peers. Additionally, filtering facilities such as those offered by servers to restrict the flow of information in the direction of specific clients in a concerted way are harder to achieve. One solution would be for each pair of clients to actually negotiate how this communication should occur, but such a solution requires some additional processing power at the sending client, which is not always compatible with the number of other tasks that it has to perform in real-time (graphical and audio rendering, for example). Finally, a pure peer-to-peer approach is the one that actually puts the largest burden on the network since packets have to be duplicated as many times as there are destination peers. To relieve this situation, multicast has been proposed and is in use in a large number of systems. This is discussed in section 4.2.2.

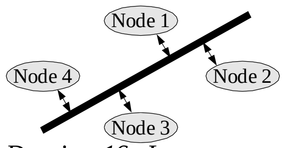
4.2.1.3. Mixing?

Recent research has acknowledged the complexity of actually finding a model that fits the needs of all applications. Therefore, a number of systems are seeking to combine the approaches in a number of ways to benefit in the best ways from both approaches. There are a number of situations where relying on a central point is crucial and simplifies the network architecture necessary to the establishment of large-scale multi-user virtual environments. One example of such a situation is the initial connection to the virtual environment. Relying on a server centralises the distribution of resources in an easy way. Since this server is only used at connection and possibly at disconnection time, its load is insignificant even when a large number of participants are involved. This sort of solution is used in systems such as CIAO and GreenSpace. Another example is how systems achieve persistence of the virtual worlds, be it with or without evolution. Persistence involves the necessary allocation of computer resources to store objects that have been created by remote participants and should be kept within the environment, and to keep these objects alive when participants have left. This allocation needs to be planned and organised and relying on a number of “central” nodes on the network is a necessary solution. An example system using this technique is Community Place and its so-called application objects (AO).

Other systems have experimented with the combination of both approaches at the protocol level. In those systems, the peer-to-peer approach is used to make sure that most packets reach their destination as soon as possible in an attempt to keep the best interaction results as possible. However, these systems use a number of servers to ensure the reliability of the transmission and possible retransmission when failures have been detected. An example of such a system is SmallTool.

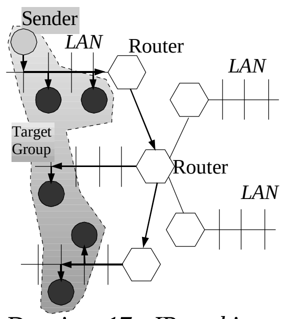
It seems to be difficult to find an approach that suits all possible applications. This recognition have led a number of developers to offer frameworks within which the different models can be mixed in harmony with the specific requirements of the applications. Continuum, and NPSNET-V to a lesser extent are such examples. However, these frameworks increase the actual work to be undertaken to develop applications since they only offer a number of more or less finalised building bricks that have to be refined and assembled in harmony. This leads to longer development cycles, especially since these issues are still not fully understood and since real-life trials are still important to show the appropriateness of the chosen approaches.

4.2.2. Unicast or Multicast



Drawing 16: In a peer-to-peer multicast architecture, nodes subscribe to a group and send information to the group. All members of the group will possibly receive this information.

In virtual environments, packets sent by participants have to reach a number of destinations. These destinations are typically participants that the system decides are interested in the packets. Partitioning techniques of all sorts are used to make the decision, this is covered in section 4.2.3. To reach their destination packets have to be duplicated. In client-server approaches, the server is in charge of the duplication. In pure peer-to-peer solutions, the peers (participants processes) themselves are in charge of the duplication. There is however an alternative choice, which is the one of multicast. IP multicast is a way to form groups of processes on the network. Each packet sent to a group will be received by all members of the group, as depicted in Drawing 16. The network implements "intelligence" to duplicate packets as needed instead of at the source. Duplication will usually happen within the Internet routers themselves, allowing for hardware acceleration and a faster delivery of the packet (see Drawing 17).



Drawing 17: IP multicast routes packets intelligently so that they are only duplicated at the routers connected to the local networks containing the receivers.

However, multicast has a number of drawbacks. For example, until the début of IPv6, the number of available multicast groups was restricted. Therefore, schemes where each active object would be associated to a separate multicast group and where remote participants would join these groups as needed have been impractical. Such schemes are also impaired by the fact that joining and leaving operations cost a number of network and computing resources both at the client side and within the routers. This very problem does actually apply to all multicast solutions. Furthermore, the spreading of multicast on the Internet has been slow: network operators are reluctant to offer multicast to their customers, computer hardware only supports a handful of multicast groups in network cards[3] and operating systems have been slow to incorporate multicast capabilities. Finally, multicast packet delivery is based on UDP and is, thus, unreliable. Reliability is discussed in section 4.3.1.

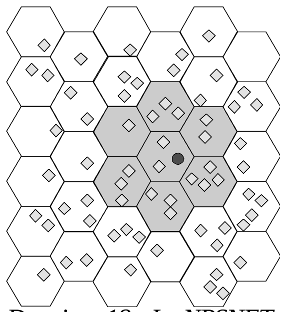
Travel time from the application to the network through the operating system is very much dependent of how the operating system is written and how it handles interrupts, especially in stressed situations. The travel time through the protocol stack is however non negligible. This has led to research such as [106] and [107], and better OS implementations.

Multicast has a number of advantages over unicast. For example, in unicast solutions based on the client-server model, packets have to travel all the way from the network hardware, through the operating system up to the application server before a decision can be made whether they should be forwarded to another participant or not. For the forwarding to happen, packets have to travel all the way back from the application, through the operating system, down to the network hardware. These travel times, under stressed situations can account for a large part of the delays introduced (see sidebar). These travel times are also of importance in pure peer-to-peer unicast approaches where packets are already duplicated at the clients. This is especially true since such clients have to perform a number of other computing intensive operations such as the rendering of the graphical 3D scene or the mixing of audio packets coming from the remote participants. On the other hand,

3 Most network cards support the multiplexing of less than two dozens distinct multicast addresses in hardware. Operating systems will take over the multiplexing when the number of subscribed groups have overtaken the amount supported by the hardware. When this happens multicast traffic can be slowed down due to the necessity to utilise CPU resources at the operating system level.

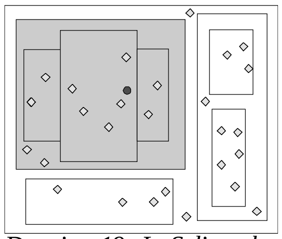
uninteresting multicast packets can already be discarded at the hardware level, or at the low-level software level.

To alleviate the slow spreading of multicast and its difficulty to reach consumers, a number of systems rely on mixed architectures, see section 4.2.1.3. Example of such systems are Spline, Community Place, SmallTool and Continuum. In these systems, servers are placed on a trusted network to glue together true clients and other multicast capable peers. Packets coming from the clients will be multiplexed at the application-level to all necessary clients of the servers and also sent to the multicast groups. Symmetrically, multicast packets incoming at the server will be forwarded as necessary to the clients. In a system like Spline, additional computing is performed at the servers in order to minimise the bandwidth used. An example of such processing is the re-encoding of audio streams or the pre-mixing and spatialisation of audio streams.

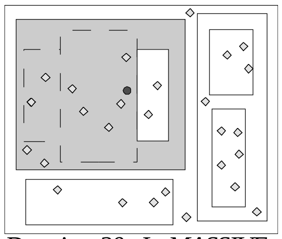

Drawing 18: In NPSNET, participants subscribe to the hexagonal partition that contains themselves and all the immediate neighbours of this hexagon. (The local participant is represented by a circle, remote participants by diamonds).

4.2.3. Dividing the Space

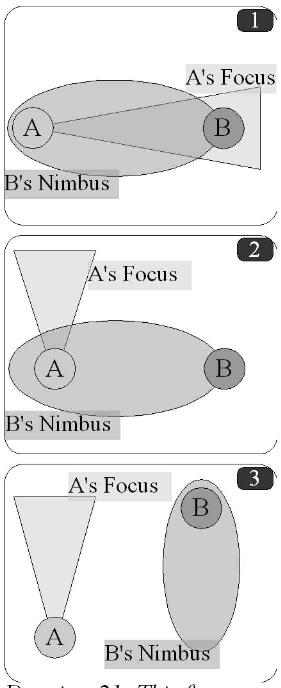
Human perceptual and cognitive limitations form the basis of the responses to the problems of scale. These solutions typically subdivide the virtual space so that each participant is not overloaded and perceives “enough” of the environment. “Enough” is defined in terms of their interest in the environment and its contents, and features such as solid boundaries or distance are used to restrict perception. For example, audio packets from distant enough participants can be discarded since audio spatialisation will render them inaudible. Position updates from an entity in another room can also be discarded under the condition that walls form a perfect visual obstacle. However, participants’ interests will change dynamically as they navigate and environments themselves are also deemed to change. Therefore interest management schemes need to be flexible and dynamic.


Drawing 19: In Spline, the spatial subdivision is based on regions of variable size. Participants subscribe to the partition that contains themselves and all its immediate neighbouring partitions.

- NPSNET divides the environment into hexagonal cells of fixed size, as depicted in Drawing 18. Each participant sends position updates to their current local cell but can choose to receive updates from several cells, as long as they are part of their area of interest (AOI). This type of subdivision is appropriate for applications such as battle simulations where objects move with predictable speeds and trajectories and where objects are spread rather evenly across the entire space.
- SPLINE divides the environment into so-called “locales” of variable size, as depicted in Drawing 19. Each locale possesses its own local coordinate system, which provides infinite geometric scalability. Each participant sends position updates to their current locale, and receives information from their current locale and its adjacent neighbours. Neighbouring is locally expressed within each locale to avoid any global knowledge. The use of variable sizes and shapes for locales provides additional flexibility when dealing with less predictable objects and environments and is more appropriate for indoor environments.
- MASSIVE-2 divides the environment into regions whose boundaries can provide different degrees of permeability for different media, as depicted in Drawing 20. For example, a wall may hinder all visual information but only attenuate audio information. Regions can also provide aggregate representations of their contents and move with it. For example, a region may synthesise a crowd of participants and move with them, offering a simplified representation at a distance (less position updates, pre-spatialised audio, etc.).


Drawing 20: In MASSIVE-2 regions add the notion of permeability. This figure is similar to the Spline example above. For the sake of simplicity, it is limited to demonstrating the concept of permeability (dashed lines) when it comes to artefact locations.

Cells, locales and regions use spatial properties and especially distance, to tackle the issues of scale and reduce the effects of movement and interaction. Typically, systems will associate dedicated and specific network resources to each sub-division of the space. These resources will be used by a restricted set of participants. Consequently, only a reduced number of participants will share those resources, which minimises the use of network bandwidth at all interested parties: both at the resources in question (whenever relevant) and at the participants. All the systems described above associate multicast groups to regions. However, dividing the space is also used in client-server solutions. Typically, such systems will associate servers to regions. Example systems using these techniques are Community Place (for its aura manager), RING and NetEffect.



Drawing 21: This figure exemplifies awareness mechanisms. It shows how even simple instantiations of focus and nimbus as discrete volumes around two participants A and B can be used to negotiate different levels of mutual awareness. In situation 1, A is fully aware of B. In situation 2, A has rotated 90 degrees and A is now only partially aware of B. Finally, in situation 3, B has also rotated and A is not aware of B any more.

4.2.4. Interest Management

There are also other models to tackle these problems at a higher level. The most prominent of these was pioneered in [108]. It proposes a model based on two components: the concept of *aura* and the concept of *awareness*.

Each active object of the virtual world has an aura for each medium in which it can interact (graphics, audio, etc.). This aura defines the extent to which interaction with other objects is possible and interaction between two objects can only happen when their respective auras collide. Auras facilitate scaling by limiting the number of object interactions that must be handled.

The second component of this model regulates interaction or communication between two objects once their auras collide. Interaction is controlled through the concept of awareness. One object's awareness of another quantifies the subjective importance or relevance of the other object in a given medium. In general, more resources (bandwidth, audio quality, etc.) will be dedicated to objects with high awareness.

To compute mutual awareness, two additional fields are associated to active objects: focus and nimbus. The focus characterises the observer's *allocation of attention*, while the nimbus characterises the observed object's *manifestation* or *observability*. The observer's awareness of the observed is a function of the observed nimbus and the observer's focus, as explained further in Drawing 21. These fields can be manipulated in various ways. For example, they can be used to model social behaviours such as shouting (large nimbus) or whispering (small narrow nimbus). This model has been implemented in a number of systems, namely an earlier version of DIVE [109] and MASSIVE-1. MASSIVE-2 also implements this model in an extended version where the effects of third party objects on mutual awareness and interaction are taken into account.

4.3. Network Protocols and Techniques

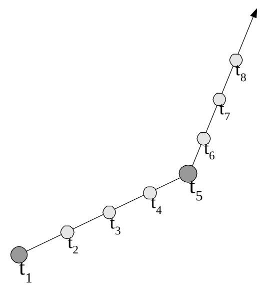
4.3.1. Reliability

In shared virtual environments, the state of objects has to be replicated in one form or another to the participants so as to ensure that they will be able to access them whenever needed. For graphical objects or their position, access will be necessary for every frame to be rendered. Consequently, it is of crucial importance that some of the objects and their state are distributed to all interested parties in a reliable way. This reliability is the key to the illusion to share an identical virtual world.

There are two ways of achieving the reliable delivery of packets: the “black-box” approach or in agreement with the application (Application-Level Framing). In the former approach, the application gives away packets to the protocol level and the protocol itself ensures their delivery to the recipients. TCP is a well-known example of such an approach. There are also a number of reliable multicast protocols that attempt to provide a similar service. For example, RMP [110] is used in a number of systems such as GreenSpace and Urbi et Orbi. In these protocols, the application can tune the delivery requirements. For example, it can request a total ordering of packets.

A number of systems use TCP or “blind” multicast protocols. Examples of such systems are Avocado, Urbi & Orbi and GreenSpace. This has the advantage of relieving the application from the intricate details of reliable delivery of packets. However, this approach has also a number of drawbacks. For example, TCP ensures the timely and ordered delivery of all packets that are sent, though it is not always necessary. A typical example is the position updates of an object. Since these occur very often, it does not make sense to ensure their ordered delivery as long as they are expressed in absolute coordinates. Instead, what is more important is the reception of the last sent update. Updates in between can gracefully be discarded if necessary. TCP does not support this style of communication. Furthermore, experience has shown that for highly-interactive applications, reliability has to be relaxed for the system to scale to a large number of users.

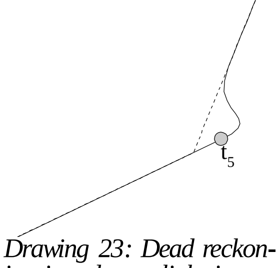
To address those problems, a number of systems have tried to let the application participate to such decisions at a finer granularity. One typical example of such a protocol is SRM (see section). In one of its profiles, Continuum provides support for SRM. Also, MASSIVE-2 uses a protocol which is similar to SRM but with a different acknowledgement scheme. The success of this approach and the recognition of different requirements when it comes to reliability of delivery have also led to a number of standard proposals. RTP/I [111] is one example of such a proposal. This view on relaxed reliability is also one of the driving forces behind frameworks such as Continuum and, to a lesser extent, NPSNET-V.



Drawing 22: Dead reckoning minimises network traffic through the use of a kinematic model. In this example, a new transformation and velocity vector are only transmitted at t_1 and t_5. This is to be compared to sending position updates at all intermediary timestamps.

4.3.2. Dead-Reckoning

Dead-reckoning is a technique to smooth out otherwise jerky position updates from participants. It was pioneered by SIMNET [64] and can be found in most systems The idea is to rely on a kinematic model (velocity, acceleration) to interpolate at all receiving participants the position of a moving object. This is exemplified in Drawing 22. Dead-reckoning has a number of advantages. It minimises the number of network packets necessary to describe the position updates of the participants, which accounts for the majority of data events within a number of applications (see paper). Additionally, it decouples the animation effects from the capacity of both the sender and the receivers. Providing a physical model allows the receivers to animate remote participants at each rendering frame. Symmetrically, the position updates of the sender are not conditioned to the frame rate in any way.



Drawing 23: Dead reckoning introduces slight inconsistencies because of network delays. The kinematic information sent at t_5 arrives later at remote peers. A dead-reckon path can be smoothed on reception to converge towards the path of the participant.

Because of network delays, the dead-reckoned path of remote copies will drift apart from the true movement of a participant. Consequently, while offering a number of advantages, dead-reckoning also introduces subjectivity when it comes to the real position of participants at a given time. This is further explained in Drawing 23.

Some client-server systems such as NetEffect or AGORA have extended the idea to a group of participants. The server cumulates position updates for a (small) time period and all information received within this period is forwarded to the relevant clients periodically. This optimisation has the advantage of cutting down the overhead of the protocol headers at various levels. Sending a group of updates within one single packet reduces redundant UDP or TCP headers and application-level headers. Since position updates can be expressed with very few bytes, aggregation forms an effective compression technique. This is especially true since, at the hardware and operating system level, there is very little difference between sending a UDP packet at its maximum size or at a smaller size (see sidebar).

In network communication, there are per-byte and per-packet costs. In some architectures such as buses or hardware forwarders the per-byte cost prevails. In other architectures such as CPU-based protocol processing or software forwarders the per-packet cost prevails. In personal computers, the per-packet cost is dominant at the protocol processing level, even though operations such as data copy and checksum computations are proportional to packet size. Consequently, in some UDP implementations, the discarding effects of checksums can be minimised and controlled at the application-level [112].

The technique of dead-reckoning can be taken a step further. The idea is to make sure all remote clients get a copy of a deterministic object behaviour. This can be achieved through the transmission of a parametric curve for example, but more complex behaviours can also be obtained. Systems such as DEVA or Continuum support the transmission of high-level events that will trigger the behaviour of objects in a deterministic fashion. This approach has the advantage of considerably decreasing the amount of information transiting on the network (assuming the behaviour has been transmitted previously). However, it poses some synchronisation problems since behaviour execution is subject to the separate processing power of all remote participants. However, when movement is a sole function of time and can be expressed as such, the technique is powerful. Such a technique is used, for example, in VRML97.

4.3.3. Achieving Consistency

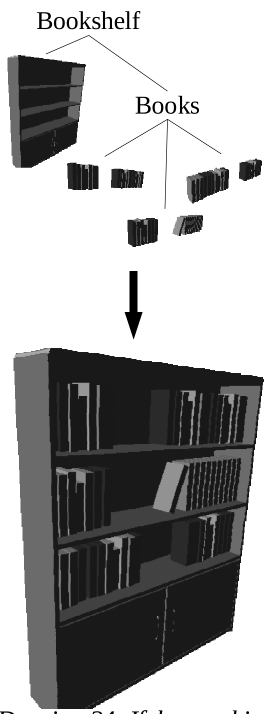
For the illusion of a shared virtual world to be entertained, the local copies of all participants have to be kept consistent across time and space. While participants might accept delays to reach the consistent state (see sidebar), consistency should happen within a “reasonable” amount of time. There are two major (and complementary) ways to achieve consistency: supporting roll-back mechanisms or making sure a user is allowed to perform an operation before it is actually executed.

Our everyday life in front of a computer is full of delays that we accept without too many questions: waiting for a Web page to appear in the browser, waiting for an application to start up, etc. Our experience has taught us to acknowledge these delays and to live with them. As time goes by, all hardware parts of a computer work faster, but the delays are still there.

The most widely used technique is the association of an ownership to each object of the virtual environment. Some systems do not allow for the transfer of ownership (residing for example, within a server). In systems that allow transfer of ownership, this transfer has to be done prior to any operation on the object itself. Even though finer-grained ownership is possible, e.g. at the level of object fields, most systems associate one owner to each object. Since ownership transfer is subject to network delays of all sorts and, thus, might impair interaction between participants, a number of systems have tried to minimise these delays using a number of techniques. For example, PaRADE uses prediction techniques to know in advance that ownership has to be transferred. These techniques are combined with application-dependent semantics about what can and cannot be done when participants approach an object. The system advocates the locking of users' possibilities just before an action can be performed, sends the “guessed” action in advance and offers some roll-back facilities if the action was not performed. There are other techniques that can be used, for example tracking the objects that are pointed at by the interaction device and requesting for ownership in advance. Such techniques have the drawback of introducing unnecessary network traffic.

In some other systems, such as CIAO, a more optimistic approach is taken. Users are more or less allowed to perform actions on objects at once. Ownership transfer is sent simultaneously and the system hopes that if there is conflict, it will be solved by other means such as the other communication channels between the various participants involved. To achieve a consistent state at all sites, the system incorporates roll-back mechanisms that allow all processes to get back to a previous state and advance to an agreed state.

In the real world, events happen according to their causal order. However, in CVEs causality may be violated due to non-deterministic message transmission delays in the network. Causality has been widely studied in parallel and distributed systems. Most of the work is focused on logical time systems and based on Lamport's *happened before* relation [113] and the vector clock. Logical time is not generally suitable for CVEs because not necessarily real-time. Consequently, wall-clock time is adopted instead to characterise the real-time behaviours in CVEs. Most of the work so far has been focuses on vector time, and adapting the concept to multicast [114]. The problem with this approach is that it reduces the potential concurrence of the system by ordering events that, from the applications perspective, have no relation. Consequently, in [115] the authors of PaRADE make a trade-off between the real-time requirements and causality through proposing a scheme aimed at maintaining only those important causal relationships. However, the scheme needs to know all the objects that an event will affect, which is not always applicable in CVEs, which are highly dynamic in their content and non-deterministic as such.



Drawing 24: If the graphical representation of a number of books, in the hierarchy, is placed under the graphical representation of a bookshelf, moving the bookshelf will have the effect of also moving the various rows of books positioned on the bookshelf.

4.4. Software Choices

4.4.1. Bringing Semantics to Data

The graphical aspects of virtual environments have in most cases dictated the software representation of the worlds themselves. At one or another level, most of the systems handle a hierarchy of objects well-suited for the creation of a scene graph. At the graphical level, this hierarchy has a number of advantages. An example is its ability to recursively move a whole sub hierarchy of objects through the modification of the transformation of the top object of the hierarchy to be moved, as described in Drawing 24.

In a number of systems, the scene graph is the only semantic aspect available to the programmer. For example, Avocado provides the scene graph as a sole abstraction and mechanisms to transparently replicate it at all sites participating to the environment.

There are, however, attempts to enhance this vision through a number of additional concepts. Urbi et Orbi is one such example. It is based on conceptual graphs where the links between objects carry a given semantic, instead of simply forming a hierarchy. Examples of such relations are “is localised on”, “is adjacent to” or “is composed of”. This semantic allows the application and the programmer to reason at a higher level of abstraction and to extract more information from the connections between objects. This can be beneficial in a number of situations.

Other attempts consist in bringing true object orientation to the structure of the data. Through the introduction of classification and inheritance, applications and programmers are able to reason at a higher level, for example to know whether some operations are available on a given object or not. Two examples of such systems are Continuum and NPSNET-V. In the latter, object orientation is complemented by a hierarchy to enhance the abstraction.

One of the effects of this search for more abstraction is the ability for several systems to provide for alternative visualisations of the environments themselves. For example, MASSIVE-1 provides a textual 2D rendering of the worlds and its inhabitants. This rendering (organised like a map) offers enough cues for textual participants to get a grip of where participants are and to understand a number of the activities that they are engaged in. Urbi et Orbi takes another approach by making available a shell in which commands can be executed to perform actions on the world components.

4.4.2. Behaviours

Behaviours give life to CVEs which would otherwise simply be static environments populated by avatars. While this theme of research is of high relevance and conditions the actual success of CVE as an application domain, it has been less investigated than other issues such as the architectural and networking aspects. This section does not consider behaviours as simple animation of the scene. It focuses on behaviours that can be triggered through human interaction and other unexpected events in the environment and how their results can be mediated to all necessary participants. There are, however, a number of questions which are recurrent across existing systems.

A few systems are based on a classification (arbitrary or not) of behaviours. This classification helps understand how the results of an execution can be seen at all remote participants. It works in conjunction with the networking mechanisms so as to try to minimise traffic as much as possible. The key direction of these models is that they try to separate “expected” behaviours from “unexpected” ones. Through a replication of all or part of the code that describes the behaviours, it should be possible to execute them in parallel, upon arrival of an unexpected event and until the next one. Two examples of such systems are SmallTool and DEVA. In SmallTool, explicit resynchronisation has to occur at application-chosen points in time. DEVA separates entities of the environment between an objective and a subjective part. The objective part is in charge of the unexpected, while the subjective part is in charge of its results. DEVA replicates the subjective parts and allows them to derive slightly.

The key problem behind this type of behavioural description is the one of synchronisation. Without any synchronisation, the replicated copies of the environments at all participants may diverge with time and lead to inconsistencies which might have disastrous effects on both the metaphor and the communication that occurs between participants through the environment.

To speed up the design and implementation process, a few systems have recognised the necessity to introduce interpreted languages. These languages, typically at a higher level, allow for quicker application development and the trial of various application interfaces and semantics, leading to better applications for the end user. An extreme example of such a system is Urbi et Orbi, where a specific language has been designed and around which most of the system is built. In general, there is little information available up to which extent it is possible to implement actual applications using those languages, or a combination of interpreted and imperative languages. Some systems such as BrickNet or Spline have taken unconventional languages, which increases the learning curve and minimises the benefits of such an approach.

4.4.3. Frameworks and Middleware

Designing and realising platforms for the deployment of CVE applications is a challenge. At the networking level, there are a number of flows to control and handle to minimise bandwidth usage and to maximise interaction between all the participants. Most early systems have been aimed at understanding these issues and finding novel ways to tackle the various issues. It has been found that, within some application domains, some models work better than other. For example, at the current time, a client-server model seems best suited for community-based applications. However, since this impairs interaction in general, new hybrid or pure peer-to-peer models have to be found.

To account for the complexity of the task and allow experimentation with different models and algorithms when it comes to data distribution and networking, a number of late systems are based on the idea of frameworks. Example of such systems are Continuum, NPSNET-V, JADE [116] or Bamboo [117]. The three last systems are oriented towards providing a run-time architecture that allows environments and applications to run non-stop and to be upgraded as needed during their (long lasted) life-time. The first system, Continuum is more geared to being able to mix network protocols, policies and architectures so as to suit the needs of particular application domains.

The drawback of frameworks is their very nature: they are frameworks into which components have to be plugged in. This means that the development of these components is a time consuming task, especially since these have to be developed in a concerted and compatible manner. Thus, their establishment has to undergo some sort of standardisation process or through their acceptance by a larger community. To this end, NPSNET-V has opened its source code to the Internet community. The later developments of the MPEG standardisation effort offer a more industrial approach.

4.4.4. Migrating lessons from 2D interfaces and CSCW

CVEs, through the primary use of a three-dimensional metaphor have had to develop an understanding of their possibilities and limitations. The dominant approach has been to start "from scratch", under the assumption that the metaphor was so much different from the traditional two-dimensional desktop. However, taking such an approach partly disregards years of research in the field of human-computer interaction. Consequently, a number of authors and systems have tried to bring back some of the lessons from traditional 2D interfaces into the world of 3D collaborative virtual environments. One such lessons is the recognition that the content of the shared environment does not necessarily appear to be identical as seen from all participants view point.

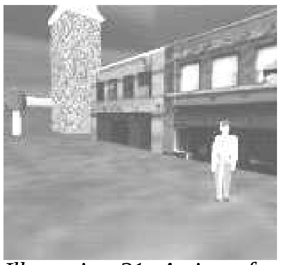


Illustration 21: A view of a town in a path finding scenario.

Earlier experiences in 2D interfaces have shown that this not always adequate. It has been shown that the strict WYSIWIS (What You See Is What I See) is too restrictive and "Relaxed WYSIWIS" [118] has been proposed to relieve these problems. Relaxed WYSIWIS acknowledges that there are inherent conflicts between the needs of a group and the needs of individuals and proposes to relax constraints along four different dimensions: display space, time of display, subgroup population and congruence of view.

Illustration 22: In this view of the same scenario, albeit from the other participant's viewpoint, arrows on the ground indicate the path to be followed and buildings that are not important for the explanation are made transparent.

Learning from 2D interfaces, a number of authors — [119] and [120] — have tried to migrate these concepts into what is generally called "subjective views". These subjective views can reflect the different roles and interests of participants. For example, a plumber and an electrician wish to see different aspects of a 3D architectural model, namely pipes and wires. In their most straightforward approach, subjective views allow the appearance and/or presence of artefacts within environments to be tweaked according to the needs and roles of participants. This type of scenario is exemplified by Illustration 21 and Illustration 22 (taken from the approach in [120]). However, the authors in [119] have also experimented with different positions for participants and objects.

There are also other examples of such migration among some of the other systems. For example, NPSNET-V requires all entities to fit to the model-view-controller abstraction [121] pioneered by Smalltalk. NPSNET-V requires that the abstract state of each object of the virtual world, its model, be separate from its views, which depict the object to the user, and from its controllers, which control the state of the object. While not explicitly stated as above, Continuum also takes a similar approach through the use of a specific language for the description of the objects (their state), their capacity (their methods), leaving apart their graphical representation.

4.5. Conclusion

Early systems such as MR toolkit have contributed to show that realising shared virtual environments was possible and that the idea had potential for a number of applications and domains. Later, and as the field matured, scaling in the number of participants and active objects has become a new problem that systems have had to tackle. The scaling issues partly originate from military usage of the technology and systems such as SIMNET. To deal with these problems a number of both network and software architectural solutions have been investigated. At a higher level, the key idea consist in modelling and using the human perceptual weaknesses to reduce the number of “interesting” entities to be taken in account. This has resulted in techniques such as awareness, world partitioning or automated summary of participants groups. At a lower level, and to ensure scalability for the remaining participating processes, the network architectural solutions depend on the requirements of the platform and the target applications. Currently, trends are trying to find the best models to intertwine peer-to-peer and client-server architectures. They also focus on offering open frameworks and middleware within which different replication and distribution strategies can be mixed and chosen from to best apply to the application at hand.

With time CVE applications have slowly migrated from technology demonstrators, through research prototypes into real applications in niched domains. As migration occurred, developers and designers have realised that more traditional software programming methods could be migrated into the field. Also, learnings from the past have been taken into account and experiences from collaborative 2D interfaces have transposed into CVE systems. However, the relative novelty of the field and the small amount of programmers and designers involved have slowed down this recognition and migration.

Chapter 5 Conclusion

CVE applications are highly interactive and recent trends show that they will soon have to support thousands of participants. As shown in this report, the research issues raised by such grand goals are many and complex. Here are some of the most important challenges, inspired by [48] and complemented by the survey conducted in this report.

CVEs accommodate varying numbers of geographically distributed users. All participants have to be kept updated with changes in the virtual environment. They will also converse using means such as network audio and video communication. Supporting these users at the networking level raises many issues and CVE systems handle distribution in significantly different ways. There are three major network architectures being used: client-server, peer-to-peer unicast and peer-to-peer multicast (see section 4.2). Current research is looking into new ways of combining these architectures to better support various applications and media over mixed infrastructures (networks and computers).

Home connection to the Internet is improving, but users are also becoming more demanding. Current trends show the development of home networks, computers that will always be powered (media centres, personal video recorders) and the popularity of applications that constantly access the network (P2P applications). All these trends point at a future where a number of applications and computers will constantly compete for external access to the Internet. In short, bandwidth will continue to be a scarce resource, even if the problem has evolved under the past five years.

The scalability of CVEs refers to two distinct aspects: the graphical and behavioural complexity of the environments and their content; and the number of simultaneous participants and active entities that can be hosted within these worlds. All participants have to be kept updated with changes in the environment. As the number of participants and active entities increases, network traffic to mediate messages describing those changes as well as audio and video communication will also increase. Whichever the distribution architecture is, the major bottleneck is the so-called “last mile”, the connection of end users to the Internet (see sidebar) and the processing power available at their computer.

Human perceptual and cognitive limitations form the basis of the responses to the problems of scale. These solutions typically subdivide the virtual space so that each participant only perceives “enough” of the environment. “Enough” is defined in terms of their interest in the environment and its contents, and features such as solid boundaries or distance are used to restrict perception. For example, audio that attenuates with distance can simply be cut off at a given distance. The recurrent theme of these solutions consists in dividing the space in smaller areas and associating separate software and hardware resources to these subdivisions.

CVEs are slowly migrating from the research sphere into the industry. This shift generates stronger requirements on software quality and modularisation. It has resulted in the emergence of a number of software frameworks that seek to provide pluggable architectures where modules, possibly written in various languages, can be assembled to form an application. The relative novelty of CVE applications and the necessity to experiment with various designs and approaches have also led to the slow integration of interpreted languages into systems and toolkits. As these languages do not require any recompilation, they shorten development time and allow designers and programmers to take a more iterative approach.

CVE collaboration usually assumes that each participant sees the same content, still from a different perspective. Earlier experiences in 2D interfaces have shown that this is not totally adequate. This has led to the introduction of “subjective views” (see section 4.4.4). These subjective views allow users to perceive environments slightly or radically differently, reflecting the different roles and interests of participants. More generally, a “space-vs-place” debate [122] as also agitated the community. Although not exclusive, these opinions have led to different types of environments. Space has resulted in fully navigable CVEs with avatars. Place has resulted in environments that are not necessarily three dimensional or where means to ease and constrain navigation are provided.

Chapter 6 Acknowledgements

Some of the illustrations in this report originate from elsewhere:

- Illustration 1 is courtesy of the former Fraunhofer CRCG, now imedia - The ICPNM Academy.
- The model used for Illustration 3 originates from Lightscape.
- Illustrations 4 and 5 are courtesy of the Georgia Institute of Technology.
- Illustration 10 is courtesy of the MIT.
- Illustration 13 is from the paper *A Two Pass Solution to the Rendering Equation: a Synthesis of Ray Tracing and Radiosity Methods* by John R. Wallace, Michael F. Cohen and Donald P. Greenberg (1987). The image appeared on the cover of *Computer Graphics: Principles and Practice* by Foley, van Dam, Feiner and Hughes.
- Illustration 16 is courtesy of Id Software.
- The helmet shown on Illustration 17 is from the former Division, Inc.
- Illustration 18 is courtesy of Fifth Dimension Technologies.
- Illustration 19 is courtesy of the center for advanced information processing at Rutgers, the state university of New Jersey.
- Illustration 20 is courtesy of Immersion Corporation (formerly Virtual Technologies Inc.)

Chapter 7 Bibliography

[1] Jaron Lanier's Home Page, http://www.jaronlanier.com/.

[2] Myron W. Krueger, Artificial Reality II, Pearson Higher Education, ISBN: 0201522608 , January1991.

[3] William Gibson, Neuromancer, Ace Books, ISBN: 0441569595, March1984.

[4] Emmanuel Frécon and Anneli Avatare-Nöu, Building Distributed Virtual Environments to Support Collaborative Work, Proceedings of the ACM Symposium on Virtual Reality Science and Technologies, Taipei, Taiwan , pp. 105-113, November 1998.

[5] Dave Snowdon, Elizabeth Churchill and Emmanuel Frécon (Eds), Inhabited Information Spaces: Living with your data, Springer Verlag, ISBN: 1852337281, January2004.

[6] Elizabeth F. Churchill, David N. Snowdon and Alan J. Munro (Eds), Collaborative Virtual Environments - Digital Places and Spaces for Interaction, Springer-Verlag, ISBN: 1-85233-244-1, May2001.

[7] ActiveWorlds, http://www.activeworlds.com/.

[8] The Palace, http://www.thepalace.com/.

[9] there.com, http://www.there.com/.

[10] Gavin Bell, Rikk Carey and Chris Marrin, The Virtual Reality Modeling Language, ISO/IEC 14772-1:1997, 1997

[11] The Web3D Consortium, http://www.web3d.org/.

[12] Steve Benford, Dave Snowdon, Chris Brown and Gail Reynard, Visualising and populating the Web: Collaborative virtual environments for browsing, searching and inhabiting Webspace, Computer Networks and ISDN Systems, Vol. 29, No. 15, pp. 1751-1761, November 1997.

[13] Navinchandra K. Patel, Simon P. Campion and Terrance Fernando, Evaluating the Use of Virtual Reality as a Tool for Briefing Clients in Architecture, Proceedings of the Sixth International Conference on Information Visualisation (IV'02) , London, England , pp. 657-663, July 2002.

[14] Christian Knöpfle and Gerrit Voβ, An intuitive VR user interface for design review, Proceedings of the working conference on Advanced visual interfaces, Palermo, Italy , pp. 98-101, 2000.

[15] Steve Bryson and Creon Levit, The Virtual Wind Tunnel, IEEE Computer Graphics and Applications, Vol. 12, No. 4, pp. 25-34, July/August 1992.

[16] Dorothy Strickland, Larry Hodges, Max North and Suzanne Weghorst, Overcoming phobias by virtual exposure, Communications of the ACM, Vol. 40, No. 8, pp. 34-39, 1997.

[17] Larry F. Hodges, Rob Kooper, Thomas C. Meyer, Barbara O. Rothbaum, Dan Opdyke, Johannes J. de Graaff, James S. Williford and Max M. North, Virtual Environments for Treating the Fear of Heights, IEEE Computer, Vol. 28, No. 7, pp. 27-34, July 1995.

[18] Azucena Garcia-Palacios, Hunter Hoffman, Albert Carlin, Thomas A. Furness III and Cristina Botella, Virtual reality in the treatment of spider phobia: A controlled study, Behaviour Research and Therapy, Vol. 40, No. 9, pp. 983-993, 2002.

[19] Mel Slater, David-Paul Pertaub and Anthony Steed, Public Speaking in Virtual Reality: Facing an Audience of Avatars, IEEE Computer Graphics and Applications, Vol. 19, No. 2, pp. 6-9, March/April 1999.
[20] Russ Zajtchuk and Richard M. Satava, Medical applications of virtual reality, Communications of the ACM, Vol. 40, No. 9, pp. 63-64, 1997.
[21] Don Allison, Brian Wills, Doug Bowman, Jean Wineman and Larry F. Hodges, The Virtual Reality Gorilla Exhibit, IEEE Computer Graphics and Applications, Vol. 17, No. 6, pp. 30-38, November/December 1997.
[22] Maria Roussos, Andrew E. Johnson, Thomas G. MoherJason Leigh, Christina A.Vasilakis, and Craig R. Barnes, Learning and Building Together in an Immersive Virtual World, Presence, Vol. 8, No. 3, pp. 247-263, June 1999.
[23] Detlev Schwabe and Mårten Stenius, The Web Planetarium and Other Applications in the Extended Virtual Environment EVE, Proceeding of the 16th Spring Conference on Computer Graphics, Budmerice, Slovakia , pp. , May 2000.
[24] Michael Zyda, John Hiles, Alex Mayberry, Casey Wardynski, Michael Capps, Brian Osborn, Russell Shilling, Martin Robaszewski and Margaret Davis, Entertainment R&D for Defense, IEEE Computer Graphics and Applications, Vol. 23, No. 1, pp. 28-36, January/February 2003.
[25] John Vince, Virtual Reality Systems, ACM Press, ISBN: 0-201 87687-6, 1995.
[26] Ivan E. Sutherland, Sketchpad---A man-machine graphical communication system, Proceedings of the AFIPS Spring Joint Computer Conference, , pp. 328-346, January 1963.
[27] Ivan E. Sutherland, The Ultimate Display, Proceedings of IFIPS Congress, New York, USA , pp. 506-508, May 1965.
[28] Ivan E. Sutherland, A Head-mounted Three-dimensional Display, AFIPS Fall Joint Computer Conference Proceedings, Vol. 33, No. 1, pp. 757-764, 1968 .
[29] Henri Gouraud, Continuous Shading of Curved Surfaces, IEEE Transactions on Computers, Vol. 20, No. 6, pp. 623-628, June 1971.
[30] Bui-Tuong Phong, Illumination for Computer-Generated Images, PhD Thesis University of Utah, Department of Computer Science, July1973
[31] P Jerome Kilpatrick, The Use of Kinesthetic Supplement in an Interactive System, PhD Thesis University of North Carolina at Chapel Hill, Computer Science Department, 1976
[32] F. Raab, E. Blood, T. Steiner, and H. Jones, Magnetic position and orientation tracking system, IEEE Transactions on Aerospace and Electronic Systems, Vol. 15, No. 5, pp. 709-717, September 1979.
[33] Thomas Zimmerman and Jaron Lanier, A Hand Gesture Interface Device, Proceedings of CHI'87, Toronto, Canada , pp. 235-240, April 1987.
[34] Gary J. Grimes, Digital Data Entry Glove interface device, Patent 4,414,537, AT & T Bell Labs, November1983
[35] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile, Modeling the interaction of light between diffuse surfaces, Proceedings of SIGGRAPH '84, Minneapolis, USA , pp. 213-222, July 1984.
[36] Wenzel, E.M., Wightman, F. L. and Foster, S. H., A virtual display system for conveying threedimensional acoustic information, Proceedings of the Human Factors Society, 32nd Annual Meeting, , pp. 86-90, 1988.
[37] Carolina Cruz-Neira, Daniel J. Sandin and Thomas A. DeFanti, Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE, Proceedings of SIGGRAPH '93 Computer Graphics

Conference, , pp. 135-142, August 1993.
[38] Jannick P. Rolland, Larry D. Davis and Yohan Baillot, A Survey of Tracking Technologies for Virtual Environments, in Woodrow Barfield and Thomas Caudell, Fundamentals of Wearable Computers and Augmented Reality, Lawrence Erlbaum, ISBN 0805829024, 2001
[39] F. Raab, E. Blood, T. Steiner, and H. Jones, Magnetic position and orientation tracking system, IEEE Transactions on Aerospace and Electronic Systems, Vol. 15, No. 5, pp. 709-717, September 1979.
[40] Mark Ward, Ronald Azuma, Robert Bennett, Stefan Gottschalk and Henry Fuchs, A Demonstrated Optical Tracker with Scalable Work Area for Head Mounted Display Systems, Proceedings of ACM Symposium on Interactive 3D Graphics (I3D 92), Cambridge, MA, USA , pp. 43-52, March 1992.
[41] How does the DTI display work?, http://www.dti3d.com/technology.asp.
[42] Daniel Gomez, Grigor Burdea, Noshir Langrana, Integration of the Rutgers Master II in a virtual reality simulation, Proceedings of the IEEE Virtual Reality Annual International Symposium (VRAIS'95), Research Triangle Park, North Carolina, USA , pp. 198-202, March 1995.
[43] Robert J. Stone, Haptic Feedback: A Brief History from Telepresence to Virtual Reality, Proceedings of the First International Workshop on Haptic Human-Computer Interaction, Vol. 2058, No. , pp. 1-16, September 2000.
[44] Ronald Azuma, Yohan Baillot, Reinhold Behringer, Steven Feiner, Simon Julier, Blair MacIntyre, Recent Advances in Augmented Reality, IEEE Computer Graphics and Applications, Vol. 21, No. 6, pp. 34-47, November/December 2001.
[45] Ronald Azuma, A Survey of Augmented Reality, Presence: Teleoperators and Virtual Environments, Vol. 6, No. 4, pp. 355-385, August 1997.
[46] Ronald Azuma, Tracking requirements for augmented reality, Communications of the ACM, Vol. 36, No. 7, pp. 50-51, 1993.
[47] Nathaniel I. Durlach and Anne S. Mavor, Editors, Virtual Reality: Scientific And Technological Challenges, NATIONAL ACADEMY PRESS, ISBN: 0-309-05135-5, 1995.
[48] Steve Benford, Chris Greenhalgh, Tom Rodden and James Pycock, Collaborative virtual environments, Communications of the ACM, Vol. 44, No. 7, pp. 79-85, July 2001.
[49] Richard C. Waters, David B. Anderson, John W. Barrus, David C. Brogan, Michael A. Casey, Stephan G. McKeown, Tohei Nitta, Ilene B. Sterns, William S. Yerazunis, Diamond Park and Spline: A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtime Modifiability, Presence: Teleoperators and Virtual Environments, Vol. 6, No. 4, pp. 461-480, August 1997.
[50] John W. Barrus, Richard C. Waters, David B. Andersson, Locales: supporting large multiuser Virtual Environments, IEEE Computer Graphics and Applications, Vol. 16, No. 6, pp. 50-57, November 1997.
[51] Richard C. Waters, David B. Anderson and Derek L. Schwenke, Design of the Interactive Sharing Transfer Protocol, Proceedings of the 6th IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'97), Cambridge, MA, USA , pp. 140-147, June 1997.
[52] Jon Mandeville, Thomas Furness, Masahiro Kawahata, Dace Campbell, Paul Danset, Austin Daul, Jens Dauner, Jim Davidson, Jon Howell, Kigen Kandie and Paul Schwartz, GreenSpace: Creating a Distributed Virtual Environment for Global Applications, Proceedings of the IEEE Networked Reality Workshop, Boston, MA, USA , pp. 95-117, October 1995.

[53] Brian Whetten, Todd Montgomery and Simon Kaplan, A High Performance Totally Ordered Multicast Protocol, in Kenneth P. Birman, Friedmann Mattern and André Schiper, Theory and Practice in Distributed Systems, Lecture Notes in Computer Science 938, Springer Verlag, ISBN 3-540-60042-6, 1994

[54] Paul Schwartz, Lauren Bricker, Bruce Campbell, Tom Furness, Kori Inkpen, Lydia Matheson, Nobutatsu Nakamura, Li-Sheng Shen, Susan Tanney, Shihming Yen, Virtual Playground:Architectures for a Shared Virtual World, Proceedings of the ACM Symposium on Virtual Reality Software and Technology 1998, Vol. , No. November, pp. 43-50, 1998.

[55] Rodger Lea, Yasuaki Honda, and Kouichi Matsuda, Virtual Society: Collaboration in 3D space on the Internet, Computer Supported Cooperative Working, Vol. 6, No. 2/3, pp. 227-250, 1997.

[56] Hiroaki Harada, Naohisa Kawaguchi, Akinori Iwakawa, Kazuki Matsui and Takashi Ohno, Space-sharing architecture for a three-dimensional virtual community, IEE Distributed Engineering Journal, Vol. 5, No. 3, pp. 101-106, September 1998.

[57] Mike Wray and Rycharde Hawkes, Distributed Virtual Environments and VRML: an Event-based Architecture, Computer Networks and ISDN Systems, Vol. 30, No. 1, pp. 43-51, April 1998.

[58] Living Worlds, http://www.web3d.org/WorkingGroups/living-worlds/.

[59] Wolfgang Broll, Populating the Internet: Supporting Multiple Usersand Shared Applications with VRML, Proceedings of the VRML’97 Symposium, Monterey, CA, USA , pp. 87-94, February 1997.

[60] Wolfgang Broll, DWTP - An Internet Protocol for Shared Virtual Environments, Proceedings of the International Symposium on VRML, Monterey, CA, USA , pp. 49-56, February 1998.

[61] Tapas K. Das, Gurminder Singh, Alex Mitchell, P. Senthil Kumar and Kevin McGee, NetEffect: a network architecture for large-scale multi-user virtual worlds, Proceedings of the ACM symposium on Virtual reality software and technology, Lausanne, Switzerland , pp. 157-163, September 1997.

[62] Tapas K. Das, Gurminder Singh, Alex Mitchell, P. Senthil Kumar and Kevin McGee, Developing Social Virtual Worlds using NetEffect, Proceedings of the Workshop on Enabling Technologies Infrastructure for Collaborative Enterprises (WET-ICE '97), MIT, Cambridge, MA, USA , pp. 148-154, June 1997.

[63] Michael R. Macedonia, Michael J. Zyda, David R. Pratt, Donald P. Brutzman and Paul T. Barham, Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments, Proceeding of the 1995 IEEE Virtual Reality Annual International Symposium (VRAIS'95), RTP, North Carolina, USA , pp. 2-10, March 1995.

[64] James M. Calvin, Alan Dickens, Bob Gaines, Paul Metzger, Dale Miller and Dan Owen, The Simnet Virtual World Architecture, Proceedings of the IEEE Virtual Reality Annual International Symposium, Seattle, Washington, USA , pp. 450-455, September 1993.

[65] David J. Roberts and Paul M. Sharkey, Maximising Concurrency and Scalability in a Consistent, Causal, Distributed Virtual Reality System, Whilst Maintaining the Effect of Network Delays, Proceedings of the 6th IEEE Workshop on Enabling Technologies Infrastructures for Collaborative Enterprises, Cambridge, MA, USA , pp. 161-166, June 1997.

[66] Chris Greenhalgh and Steve Benford, MASSIVE: A Distributed Virtual Reality System Incorporating Spatial Trading, Proceedings of the 15th

International Conference on Distributed Computing Systems (DCS'95), Vancouver, Canada , pp. 27-34, June 1995.

[67] Chris Greehalgh and Steve Benford, Supporting Rich And Dynamic Communication in Large Scale Collaborative Virtual Environments, Presence: Teleoperators and Virtual Environments, Vol. 8, No. 1, pp. 14-35, February 1999.

[68] Chris Greenhalgh, Jim Purbrick and Dave Snowdon, Inside MASSIVE-3: flexible support for data consistency and world structuring, Proceedings of the third international conference on Collaborative virtual environments, San Francisco, California, United States , pp. 119-127, 2000.

[69] Chris Shaw, Mark Green, Jiandong Liang and Yunqi Sun, Decoupled Simulation in Virtual Reality with the MR Toolkit, ACM Transactions on Information Systems, Vol. 11, No. 3, pp. 287-317, July 1993.

[70] Chris Shaw and Mark Green, The MR Toolkit Peers Package and Experiment, Proceedings of the IEEE Virtual Reality Annual International Symposium (VRAIS'93), Seattle, WA, USA , pp. 463-469, September 1993.

[71] Qunjie Wang, Mark Green and Chris Shaw, EM - An Environment Manager for Building Networked Virtual Environments, Proceedings of the IEEE Virtual Reality Annual International Symposium, Research Triangle Park, North Carolina, USA , pp. 11-18, March 1995.

[72] Didier Verna, Yoann Fabre and Guillaume Pitel, Urbi et Orbi: Unusual Design and Implementation Choices for Distributed Virtual Environments, Proceedings of the 6th International Conference on Virtual Systems and MultiMedia, Gifu, Japan , pp. , October 2000.

[73] Mark Hayden, The Ensemble System, Technical Report, TR98-1662, January 1998

[74] Henrik Tramberend, Avocado: A Distributed Virtual Reality Framework, Proceedings of the IEEE Virtual Reality Conference, Houston, Texas, USA , pp. 14-21, March 1999.

[75] Steve Pettifer, Jon Cook, James Marsh and Adrian West, DEVA3: Architecture for a Large Scale Distributed Virtual Reality System, Proceedings of the ACM symposium on Virtual Reality Science and Technology , Seoul, Korea , pp. 33-40, October 2000.

[76] Roger Hubbold, Jon Cook, Martin Keates, Simon Gibson, Toby Howard, Alan Murta, Adrian West and Steve Pettifer, GNU/MAVERIK: a micro-kernel for large-scale virtual environments, Proceedings of the ACM symposium on Virtual reality software and technology, London, United Kingdom , pp. 66 - 73, December 1999.

[77] Frédéric Dang Tran, Marina Deslaugiers, Anne Gérodolle, Laurent Hazard and Nicolas Rivierre, An Open Middleware for Large-Scale Networked Virtual Environments, Proceedings of the IEEE Virtual Reality Conference, Orlando, Florida, USA , pp. 22-29, March 2002.

[78] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne and Lixia Zhang, A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing, IEEE/ACM Transactions on Networking, Vol. 5, No. 6, pp. 784-803, December 1997.

[79] Andrzej Kapolka, Don McGregor and Michael Capps, A Unified Component Framework for Dynamically Extensible Virtual Environments, Proceedings of the 4th International Conference on Collaborative Virtual Environments, Bonn, Germany , pp. 64-71, October 2002.

[80] Gurminder Singh, Luis Serra, Willie Png and Hern Ng, BrickNet: A Software Toolkit for Network-Based Virtual Worlds, Presence, Vol. 3, No. 1, pp. 19-

34, Winter 1994.
[81] Gurminder Singh, Luis Serra, WUlie Png, Audrey Wong and Hern Ng, BrickNet: Sharing Object Behaviors on the Net, Proceedings of the IEEE Virtual Reality Annual International Symposium, Research Triangle Park, North Carolina, USA , pp. 19-25, March 1995.
[82] Thomas A. Funkhouser, RING: A ClientServer System for Multi-User Virtual Environments, Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics, Monterey, CA, USA , pp. 85-92, 209, April 1995.
[83] Thomas A. Funkhouser, Network Topologies for Scalable Multi-User Virtual Environments, Proceedings of the IEEE Virtual Reality Annual International Symposium, San Jose, CA, USA , pp. , April 1996.
[84] Un-Jae Sung, Jae-Heon Yang, Kwang-Yun Wohn, Concurrency Control in CIAO, Proceedings of the IEEE Virtual Reality Conference, Houston, Texas, USA , pp. 22-28, March 1999.
[85] Martin Mauve, Volker Hilt, Christoph Kuhmünch and Wolfgang Effelsberg, RTP/I - Toward a Common Application Level Protocol for Distributed Interactive Media, IEEE Transactions on Multimedia, Vol. 3, No. 1, pp. 152-161, 2001.
[86] Martin Mauve, TeCo3D - A 3D Telecooperation Application based on VRML and Java, Proceedings of MMCN/SPIE '99, San Jose, CA, USA , pp. 240-251, January 1999.
[87] , IEEE Standard for Distributed Interactive Simulation, IEEE Standard 1278.1, 1995
[88] , The IEEE HLA Standards, IEEE Standards 1516 series,
[89] , Moving Picture Experts Group, ISO/IEC International Standard 14496, January1999
[90] Richard Bartle, Interactive Multi-User Computer Games, , December1990
[91] Terraplay Network Solution for Gaming, http://www.terraplay.com/prodpdf/terra_prod_system.pdf.
[92] Butterfly.net: Powering Next-Generation Gaming with Computing On-Demand, http://butterfly.net/platform/technology/idc.pdf.
[93] TeraZona White Paper, www.zona.net/whitepaper/Zonawhitepaper.pdf.
[94] Eric .J. Berglund and David R. Cheriton, AMaze: A multiplayer computer game, IEEE Software, Vol. 2, No. 3, pp. 30-9, 1985.
[95] Christophe Diot and Laurent Gautier, A distributed architecture for multiplayer interactive applications on the Internet, IEEE Networks magazine, Vol. 13, No. 4, pp. 6-15, 1999.
[96] Ashwin R. Bharambe, Sanjay Rao and Srinivasan Seshan, Mercury: a scalable publish-subscribe system for internet games, Proceedings of the first workshop on Network and system support for games, Bruanschweig, Germany , pp. 3-9, April 2002.
[97] Stefan Fiedler, Michael Wallner and Michael Weber, A communication architecture for massive multiplayer games, Proceedings of the first workshop on Network and system support for games, Bruanschweig, Germany , pp. 14-22, April 2002.
[98] Jonathan Blow, A look at latency in networked games, Game Developer, Vol. 5, No. 7, pp. 28-40, July 1998.
[99] Rick Lambright, Distributing object state for networked games using object views, Game Developer, Vol. 9, No. 3, pp. 30-39, March 2002.
[100] Paul Bettner and Mark Terrano, 1500 archers on a 28.8: Network programming in Age of Empires and beyond, Proceedings of the 2001 Game Developer Conference, San Jose, CA, USA , pp. , March 2001.

[101] UnrealScript Language Reference, http://unreal.epicgames.com/UnrealScript.htm.
[102] Robert Huebner, Adding Languages to Game Engines, Game Developer, Vol. 4, No. 7, pp. , September 1997.
[103] Bruce Dawson, Game Scripting in Python, Proceedings of the Game Developers Conference, , pp. , 2002.
[104] Roberto Ierusalimschy, Luiz Henrique de Figueiredo and Waldemar Celes Filho, Lua-an extensible extension language, Software: Practice & Experience, Vol. 26, No. 6, pp. 635-652, June 1996.
[105] Tux Racer, http://tuxracer.sourceforge.net/.
[106] Craig Partridge and Stephen Pink, A faster UDP, IEEE/ACM Transactions on Networking, Vol. 1, No. 4, pp. 429-440, August 1993.
[107] Bengt Ahlgren, Per Gunningberg, and Kjersti Moldeklev, Increasing communication performance with a minimal-copy data path supporting ILP and ALF, Journal of High Speed Networks, Vol. 5, No. 2, pp. 203-214, 1996.
[108] Steve Benford, John Bowers, Lennart Fahlén and Chris Greenhalgh, Managing Mutual Awareness in Collaborative Virtual Environments, Proceedings of ACM Symposium on Virtual Reality Software and Technology, Singapore , pp. 223-236, August 1994.
[109] Lennart Fahlén, Chris Brown, Olov Ståhl and Christer Carlsson, A Space Based Model for User Interaction in Shared Synthetic Environments, Proceedings of InterCHI'93,, Amsterdam, The Netherlands , pp. 43-50, April 1993.
[110] Brian Whetten, Todd Montgomery and Simon Kaplan, A High Performance Totally Ordered Multicast Protocol, Theory and Practice in Distributed Systems, Vol. 938, No. , pp. 33-54, September 1994.
[111] Martin Mauve, Volker Hilt, Christoph Kuhmünch and Wolfgang Effelsberg, RTP/I - Toward a Common Application Level Protocol for Distributed Interactive Media, IEEE Transactions on Multimedia, Vol. 3, No. 1, pp. 152-161, March 2001.
[112] Lars-Åke Larzon, Mikael Degermark, Stephen Pink, UDP Lite for real-time multimedia applications, Proceedings of the IEEE International Conference of Communications, Vancouver, British Columbia, Canada , pp. , June 1999.
[113] Leslie Lamport, Time, clocks, and the ordering of events in a distributed system, Communications of the ACM, Vol. 21, No. 7, pp. 558-565, July 1978.
[114] Kenneth Birman, André Schiper and Pat Stephenson, Lightweight Causal and Atomic Group Multicast, ACM Transactions on Computer Systems, Vol. 9, No. 3, pp. 272-314, August 1991.
[115] David J. Roberts, Benjamin G. Worthington, Paul M. Sharkey and Johannes Strassner, Influence of the Supporting Protocol on the Latencies Induced by Concurrency Control within a Large Scale Multi User Distributed Virtual Reality System, Proceedings of the International Conference on Virtual Worlds and Simulation (VWSIM), SCS Western Multi-conference, San Francisco, CA, USA , pp. 70-75, January 1999.
[116] Manuel Oliveira, Jon Crowcroft, Mel Slater, Component framework infrastructure for virtual environments , Proceedings of the third international conference on Collaborative virtual environments, San Francisco, CA, USA , pp. 139-146, September 2000.
[117] Kent Watsen and Michael Zyda, Bamboo - A Portable System for Dynamically Extensible, Real-Time, Networked, Virtual Environments,

Proceedings of the IEEE Virtual Reality Annual International Symposium, Atlanta, Georgia, USA , pp. 252-259, March 1998.
[118] M. Stefik, D. G. Bobrow, G. Foster, S. Lanning and D. Tatar, WYSIWIS revised: early experiences with multiuser interfaces, ACM Transactions on Information Systems (TOIS), Vol. 5, No. 2, pp. 147-167, 1987.
[119] Dave Snowdon, Chris Greenhalgh and Steve Benford, What You See is Not What I See: Subjectivity in Virtual Environments, Proceedings of Framework for Immersive Virtual Enviroments (FIVE'95), QMW University of London, UK , pp. 53-69, December 1995.
[120] Gareth Smith, Cooperative Virtual Environments: lessons from 2D multi user interfaces, Proceedings of the Conference on Computer Supported Collaborative Work'96, Boston, MA, USA , pp. 390-398, November 1996.
[121] Glenn E. Krasner and Stephen T. Pope, A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System , Journal of Object Oriented Programming, Vol. 1, No. 3, pp. 26-49, 1988.
[122] Steve Harrison and Paul Dourish, Re-place-ing space: the roles of place and space in collaborative systems, Proceedings of the 1996 ACM conference on Computer supported cooperative work, Boston, MA, USA , pp. 67-76, November 1996.