# Multi-User, Interactive Virtual Reality via the Internet

**Thesis**

**Submitted in partial fulfillment of the**

**requirements for the Degree of**

**Honours in Computer Science**

**of Rhodes University**

**by**

**Clinton Brett Mclean**

**November 1997**

## Abstract

This thesis presents solutions to many of the problems associated with large-scale, widely distributed virtual reality, based on an architecture of heterogeneous devices and limited publically accessible bandwidth. The emphasis is on achieving multi-user, interactive virtual reality via the Internet, which inherently involves the associated problems, due to the nature of the Internet. The order of the discussion will be to first present a taxonomy of possibly scalable architectures based on their communication topology and data distribution. The next section will give an overview of related work in this field. Following on from this will be sections dealing with the problems involved with shared worlds of this nature and their possible solutions based on the author's research.

A final architecture will then be presented which encompasses these solutions. Finally, it will be discussed how to interface the architecture with VRML(the Virtual Reality Modelling Language) which has become the standard virtual world description language over the Internet.

## Acknowledgements

Thanks go to Shaun Bangay, my supervisor, who was always willing to help and expressed an interest in my research. Thanks also go to Shaun and other contributors to the development of RhoVeR, which I found of use in testing some ideas and from which I learnt a lot.

# 1.Introduction

Communication has evolved through many stages. From the basic interaction of simple organisms to primitive man making use of grunts and body posture there has been an evolutionary push towards a greater extension of self. In civilized man this process has gone through several major eras. The use of humans as the medium of propagation of the communication, in the case of messengers, to the use of the electronic medium, from the telephone to television and now the internet(which combines the nature of both television and the telephone, namely mass one-way visual communication and two-way individual communication, to achieve communication possibilities previously impossible). The essence of this process has been the extension of the persons self from a limited immediately surrounding area to a far greater area of influence, the entire planet.

It must be noted that this is not just a spatial increase, but is also a temporal one. Communication has been possible on a global scale for a long time now, but it is the speed at which this communication can now take place that is of great importance. The extension of a person' s self is dependant on the speed at which facets of their self can be communicated in order to be relevant to their current self(the self not being constant).

Due to the high speeds of communication now possible, more can be expressed in the same period of time than was previously possible. Long distance communication started with writing, then voice and now video and voice are possible. Each form is potentially more expressive than the last.

This brings us to the next major development.  Increases in bandwidth are supplying our demand for a greater extension of self. If audio can be more expressive than text, video and audio more expressive than audio alone then virtual reality is potentially a more expressive medium than video, audio and text as it includes these mediums as well as extends them. Virtual reality, as a communication medium, is a solution to expressing our self to the extent that we can in reality, but without the spatial limitations of reality. In virtual reality space can be manipulated.

The demand then to extend our influence has been a motivating force to take multi-user, interactive virtual reality to the Internet and, thus, it is currently a major area of research.

VRML(Virtual Reality Modelling Language) is the collective focus of this research. It is the standard which is likely to be accepted by the Internet community for describing virtual worlds.

VRML, though, is an evolving standard. At present, the VRML 2.0 standard allows for user interactions with the virtual world and descriptions of object behaviour through the use of scripts. It does not though have explicit support for multi-user interaction with each other and the shared world.

Much research into distributed virtual reality has already taken place, however, no agreed upon solution has been found for scaling virtual environments to the Internet. This is due to the nature of the Internet, namely it’s world-wide distribution and the possibilities of hundreds of users populating these virtual worlds.

Objectives which need to be addressed for scaling virtual environments to the Internet include:

- Allowing for possibly a large number of users simultaneously occupying the shared worlds.
- Keeping the shared worlds consistent.
- Allow for worlds which persist through time(i.e., changes made are recorded for extended periods of time).
- Minimizing latency( the time taken to update the shared world)
- Allow for heterogeneous internet devices(i.e., a user with a slow connection should not slow down the entire shared world network architecture).
- Allow for seamless navigation within the virtual environment and partitioning of large-scale worlds( i.e., regions could be downloaded as needed from a server or even allocated to a number of servers, with an apparent seamless connection between regions).
- Contention resolution(i.e., disputes when moving or grabbing an object).
- Collision detection inconsistencies due to latency must be solved.
- Creating a generic architecture that can be applied to various virtual world needs (i.e., trading consistency for greater interactivity and vice-versa).

I will deal with these issues under three main sections.

Firstly, the communication topology and shared world data distribution will be examined.

Following from this, shared world data consistency issues will be dealt with.

Finally, those problems dealing with shared world data persistence will be examined.

An architecture that can achieve the outlined objectives will then be presented.

# 2.Communication Topology and Shared World Data Distribution

## 2.1 Introduction

The following chapter deals with the structure of the shared world architecture in terms of the how the participant's computers communicate with each other and the manner in which the shared world's data is distributed, that is where does it exist?
After categorizing and an analysis of the most important architectures of relevance a brief examination of related work will be presented.
Lastly, the issue of navigating between shared worlds will be discussed.

## 2.2 Taxonomy

Various methods of categorizing virtual environment architectures according to different criteria have been attempted.
In this section I am interested in architectures classified according to their communication topology and shared world data distribution.
I define the communication topology as the way in which the hosts are connected to each other.
Two communication topologies will be examined.

- Centralized communication topology: This is a centralized form of network communication. All messages get sent through a central server and then to the destination host or hosts(Figure 1).
- Distributed communication topology: Each host is directly linked to every other host(Figure 2).

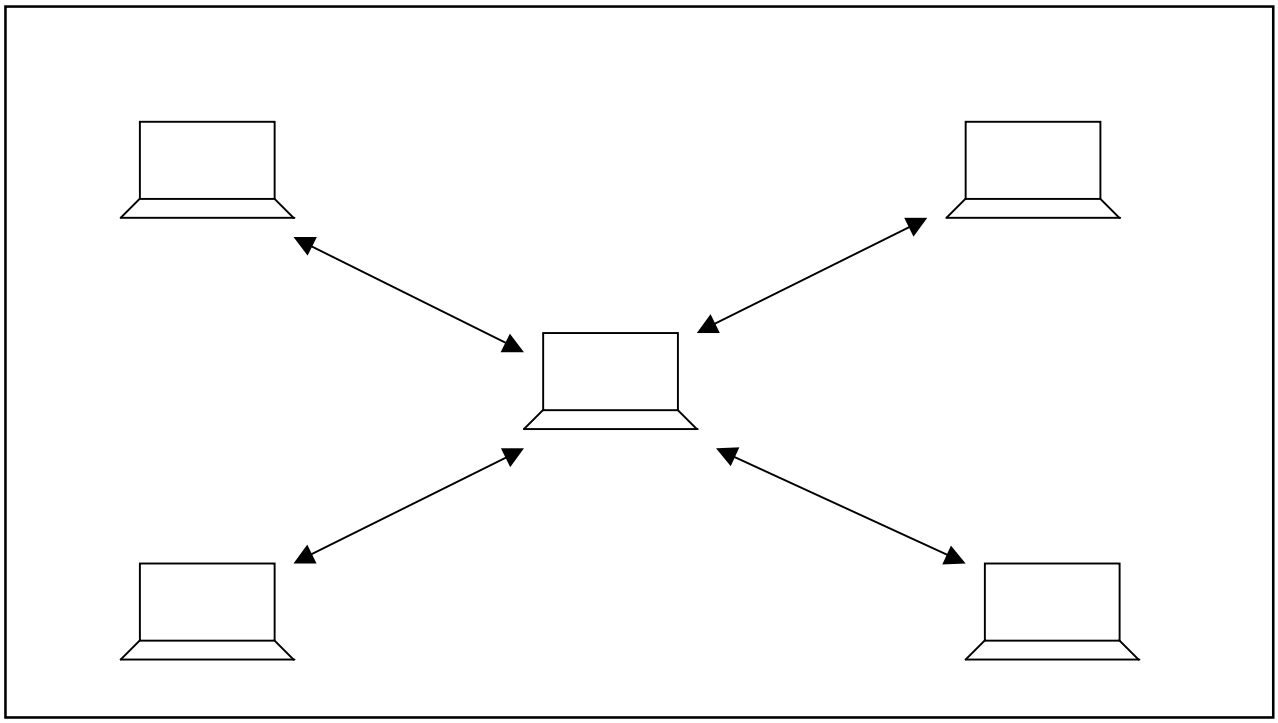

**Figure 1 - Centralized Communication Topology**

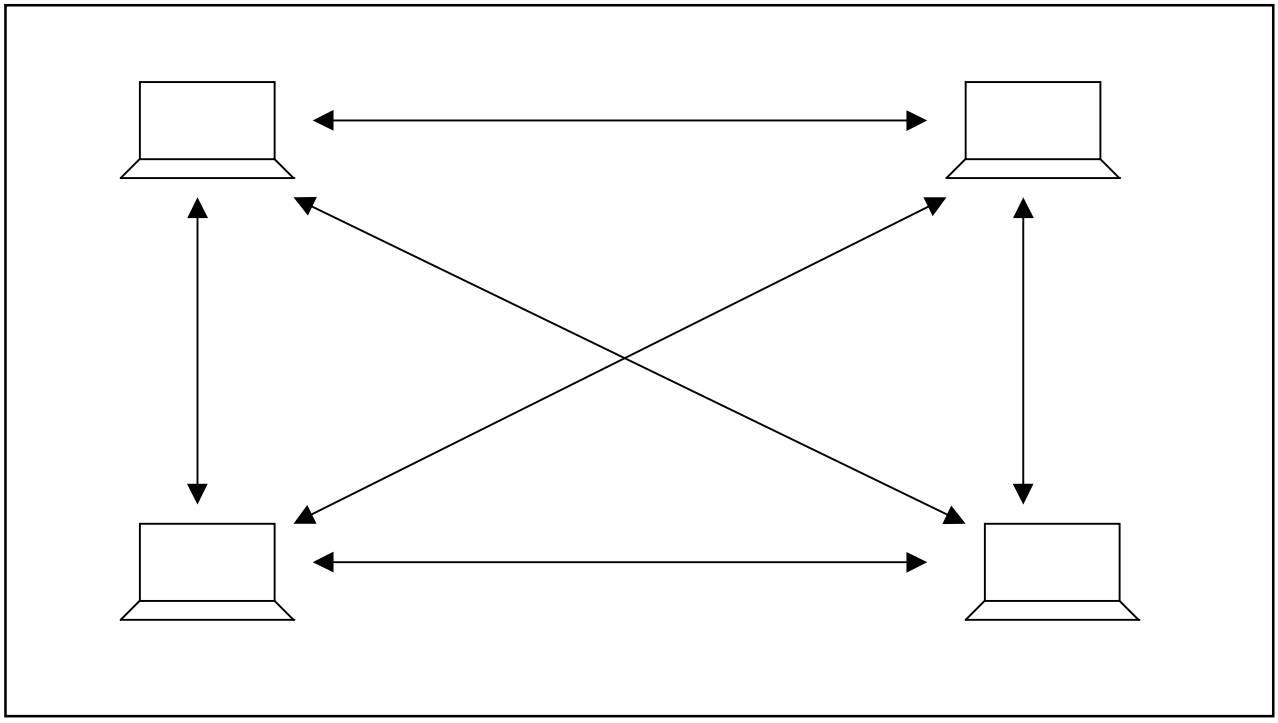

**Figure 2 - Distributed Communication Topology**

The shared world data distribution is concerned with where the data exists or the manner in which it is shared.

[Bangay 1996] Categorizes the data distribution in the following manner:

- A centralized database: the data exists on a single centralized machine. It can be updated and can send information to hosts when needed.
- A distributed database: the data exists on multiple machines and is always synchronized.
- A replicated database: the data exists on multiple machines, but is not necessarily synchronized. Periodic updates keep the data reasonably consistent.
- Distributed databases: the database is partitioned across multiple servers.

Since complete synchronization of data is an unlikely possibility when considering large-scale, widely-distributed virtual reality, replicated databases are more interesting for these purposes and I would add the following category derived from the above:

- Replicated databases: the database is partitioned across multiple servers with segments being replicated on multiple machines. Again synchronization is not a necessity.

I suggest that the following combinations of communication topology and shared world data distribution models are of interest and relevance to the subject under discussion and, thus, worthy of analysis.

- A centralized database with a centralized communication topology.
- A replicated database with a centralized communication topology.
- A replicated database with a distributed communication topology.
- Replicated databases with a distributed communication topology.

**Centralized database with a centralized communication topology.**

A centralized database with a centralized communication topology can be represented very much like figure 1, with the data being stored on the central computer.
This architecture is not likely to be scalable to large-scale, widely distributed virtual environments, especially those of a highly data-intensive nature, like a virtual reality graphical environment. The large amounts of data that would need to be communicated would exceed today's capabilities. The central computer, or server, would become a bottleneck in the system.
This architecture is viable in order to create text based MUD(Multi-User Dungeon) environments, but not the environments here under consideration.

**Replicated database with a centralized communication topology.**

Here the data is replicated on all participating computers, but communication is mediated through a single central server(as shown in figure 3).

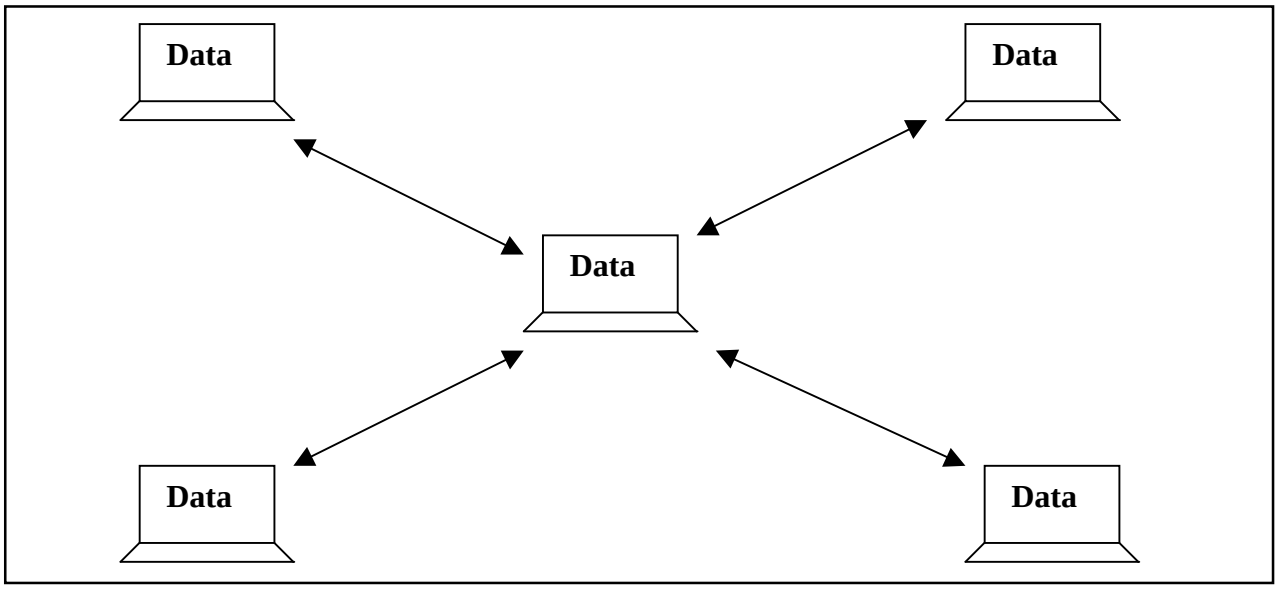


**Figure 3 – Replicated database with a centralized communication topology**

The problem with this architecture that the central server can also become a bottleneck of the system. Every update to the shared world that it receives must be communicated to all the other participants in order to update their data. This is problematic if a protocol like TCP is being used for each connection. UDP multicasting would suit this architecture, however, each message still needs to be processed by the central server before being communicated to the other participants. With many participants the server could still become a bottleneck as all communication is mediated through it.

**Replicated database with a distributed communication topology.**

This is an architecture that I feel has a lot of potential. Each participant can communicate with every other participant, without needing to go through a central server. They also each have a copy of the shared world data. Multicasting would be ideal for communicating updates in such a system.

However, the architecture that will be selected will need to have a certain degree of control or source of authority, as in the central server of the above architectures. It also needs a place to store the shared world data, as the participants will come and go. For these reasons I suggest that such an architecture like this will require a server computer. It will need to exist to accomplish tasks like allowing new participants to join the shared world and resolve issues like contention, data inconsistencies and reliability issues.

Such an architecture is demonstrated in figure 4.

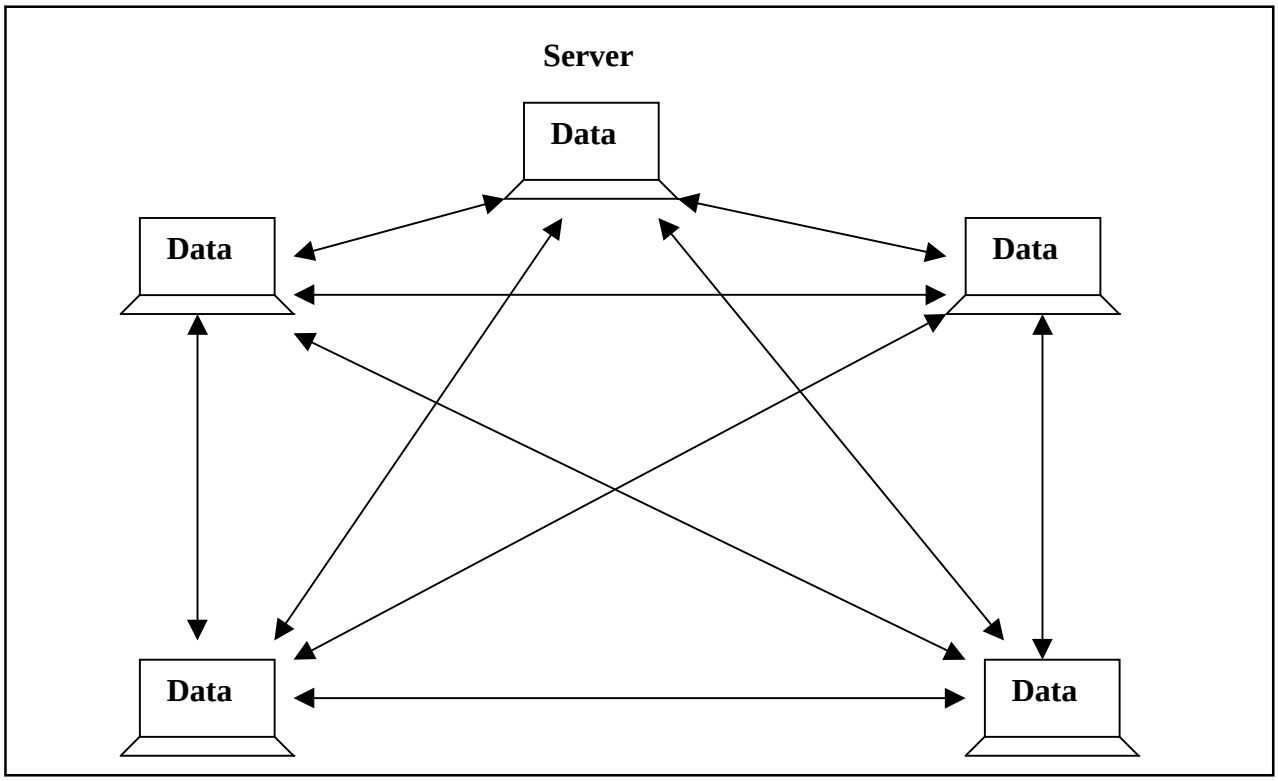


**Figure 4 – Replicated database with distributed communication topology and server**

**Replicated databases with a distributed communication topology.**

Here the database is partitioned across multiple servers. That is, each server is responsible for maintaining a particular segment of the shared world's data. Participants can communicate directly with each other and the server that is managing the particular segment of the world they are interested in. Participants who wish to navigate into a different segment will connect to the server responsible for that segment and communicate with the participants located in that segment. I like this model as it reflects the nature of the internet in the sense that users can navigate through a massively connected system. This architecture holds the same potential in that users could navigate through massive, seamlessly connected shared worlds with the servers containing and managing segments of these worlds.

Figure 5 represents this architecture.

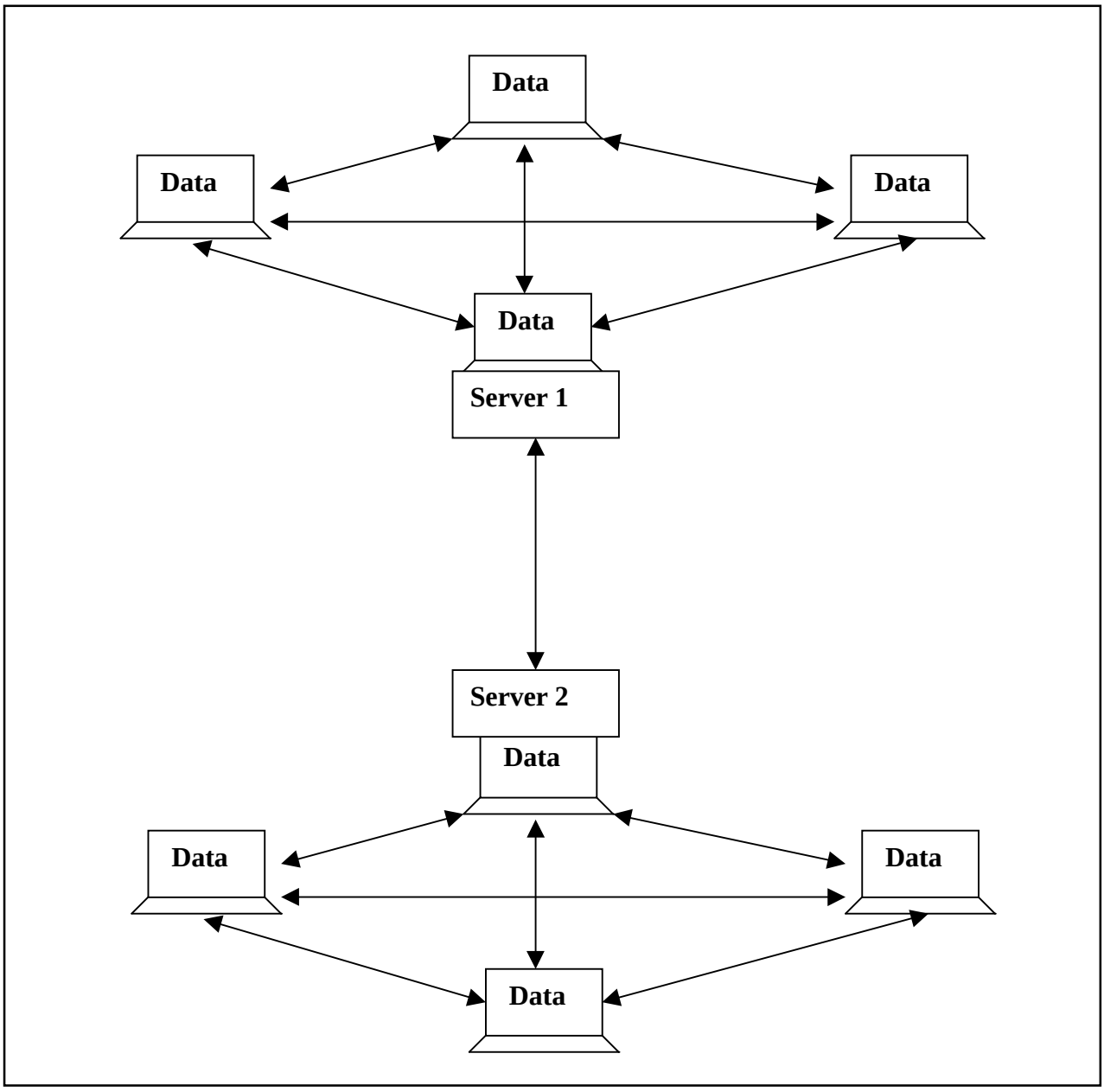


**Figure 5 – Replicated databases partitioned across two servers using a distributed communication topology for participants of each segment**

After examining the above architectures it is the last one that I find the most appealing. As I mentioned it seems to reflect the nature of the internet, an important consideration for any system that is going to make use of the internet's structure.
It also does not make unreasonable demands on the servers, as they are not used to mediate every update message. Since participants can communicate directly with each other they do not place the burden on the server to convey their messages.
A possible flaw in this architecture would be that since every host can communicate directly with every other host the number of connections would be n-1 for each host(where n is the number of hosts, that is participants and the server involved in a particular segment of the shared world).
This would be a serious problem if a TCP protocol were used for communication. When communicating messages to the shared world a separate TCP connection would need to be setup between the sender and every other host in the system. The update message would then be sent to every host one at a time.
What is required is multicasting. Since the same update message is sent to every participant it is ideally suited to multicasting. However, since these worlds are going to exist on the internet, normal IP multicasting as used on single network segments cannot be used.
Much effort, though, has gone into achieving multicasting over the internet since it will not only be useful for developing virtual environments over the internet, but will also allow distributed multi-media services, like video on demand, to become a reality.
The use of multicasting for achieving virtual environments is one of the issues discussed in the next section on related work.

## 2.3 Related work

This section will give an overview of related work in the area of large-scale, widely distributed virtual reality.

### 2.3.1 NPSNET

NPSNET is a networked virtual environment that was developed at the Naval Postgraduate School in Monterey, California. Important facets of the NPSNET system is that it was the first virtual reality system to use and extensively test the use of IP multicasting, through the use of the Multicast Backbone(MBone), as a means of scaling virtual environments. As a result of their tests it was found that IP multicast is a valid means of supporting large-scale virtual environments[Macedonia 1995].

Furthermore, techniques in partitioning the shared world and giving each segment it's own multicast group, were examined. The virtual environment is partitioned by making use of an area of interest manager(AOIM). The AOIM partitions the virtual environment into "a set of workable, small-scale environments or classes to reduce computational load on hosts, minimize communications on network tail links, and localize reliability problems."[Macedonia et al 1995].

In addition, dead-reckoning algorithms are used to reduce network traffic by making predictions regarding an object' s future state rather than always sending an update message.

The research undertaken with NPSNET has had a major influence on related research into large-scale virtual environments.

### 2.3.2 GMD - German National Research Center for Information Technology

Research that has had possibly the greatest influence on my own work, has been that of Wolfgang Broll of the GMD. This is especially true in my research into the selection of the communication topology and data distribution model that would make scalable virtual environments a possibility.

Like NPSNET, use is made of IP multicasting in order to realize a scalable communication infrastructure[Broll 1997a].

Large worlds are also sub-divided in order to reduce unnecessary network messages. However, this is not done in the same way as AOIM's in NPSNET, where the world is sub-divided into hexagonal regions(figure 7). Rather, a more generic approach is taken, which allows the sub-division of worlds into arbitrarily sized regions.

Each region will have the following fields:

- The transformation(specifies the position and orientation of the region)
- The radiation(defines the space in which contents of the region are realized)
- The horizon(defines the space which can be realized from within a region)
- The hull(specifies when to start loading the contents of a region)
- The external representation(the appearance of the region from outside it's radiation area)
- The contents of the region(specified by an URL, which is loaded when the user enters the hull)

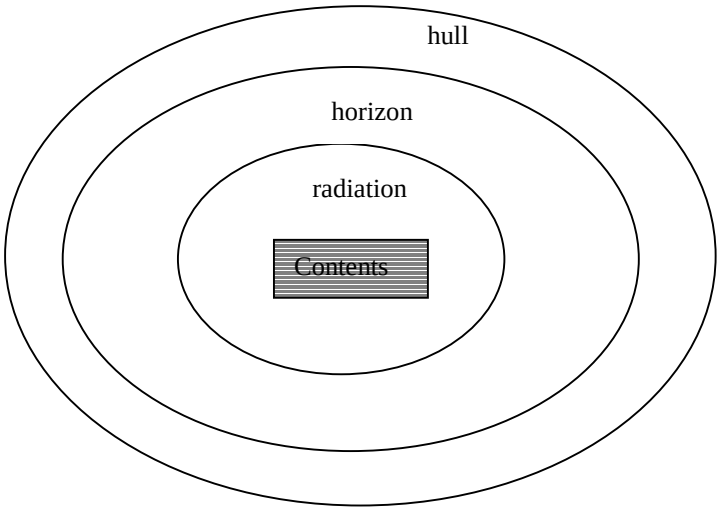


**Figure 6 - A region specified by fields**

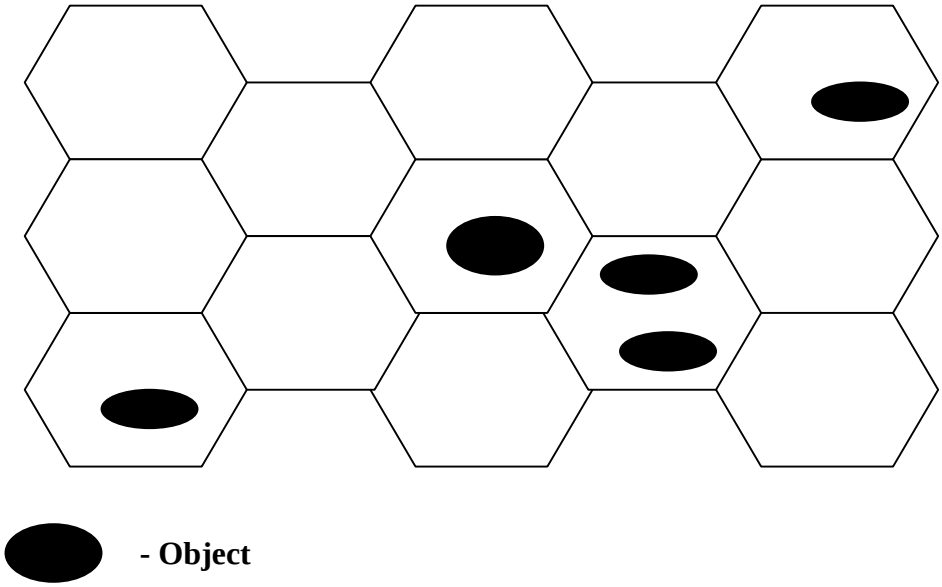


**Figure 7 – NPSNET area divided into hexagonal regions(Objects need only be aware of each other within a certain number of neighbouring cells)**

The communication topology and data distribution of the system can be described as a replicated database with a distributed communication topology, which is made scalable through the use of IP multicasting(as represented in figure 8).

The architecture includes a server or multi-user daemon(MUD).

Some of the specified tasks of the MUD[Broll 1997b] are the following:

- Connecting new participants to the shared world.
- Keeping the state of the shared world up to date.
- Enabling a certain degree of reliability(IP multicast is an unreliable communications protocol).
- Enabling consistency and contention resolution mechanisms.

In addition participants who are not IP multicast compatible( for example, those who do not yet have access to the MBone) can access the shared worlds through the use of unicast links to devices that do have access.

These devices are of two types, also described in [Broll 1997b].

- A proxy server: which, like the server, also keeps a copy of the virtual world data and can perform most of the functions of the main server, like connecting new participants and providing reliability services.
- A relay server: which simply allows unicast participants to communicate with the multicast group.

Although, if the architecture is to be truly scalable most participants will need to be multicast capable. This is a likely possibility with the growth of the MBone and protocols that allow efficient multicasting over the internet.

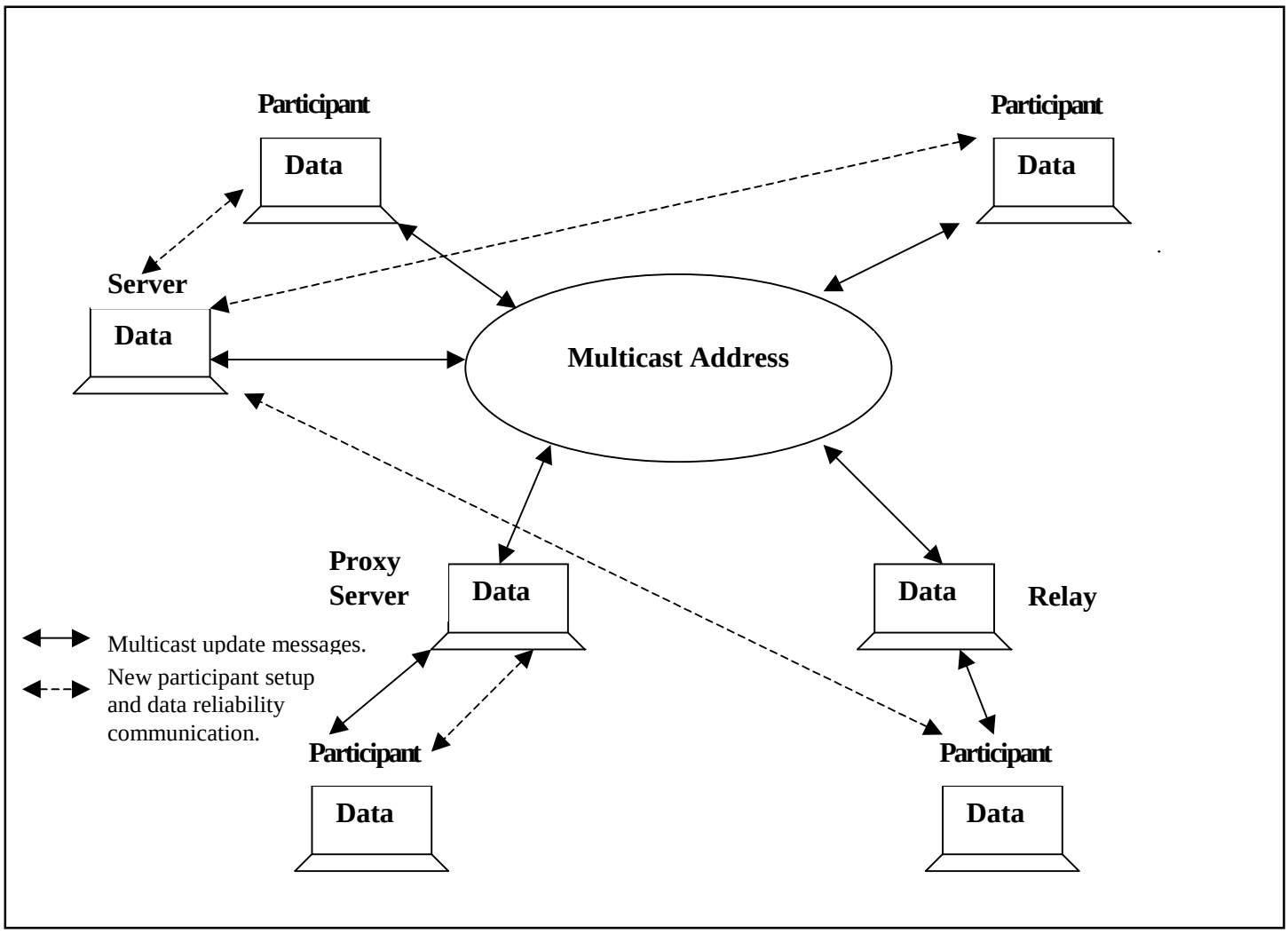


**Figure 8 – An architecture suggested by Wolfgang Broll of the GMD – German National Research Center for Information Technology.**

Much reference will be made to the above work as well as other related work in the following chapters. However, an issue directly related to the distribution of data will now be discussed. The following chapter deals with how virtual worlds and regions of large virtual worlds can be connected in order to allow the user to effortlessly navigate them.

## 2.4 Connecting worlds

In designing the architecture for virtual worlds it is with the intention that they can be navigated in much the same way that conventional internet browsing is achieved. That is, the user can navigate between pages with great ease, by merely clicking on hyperlinks, rather than entering the address of each document they view.

The objective is to produce a seamless virtual world environment, where the physical existence of the world data is not important. For example, a large virtual world could consist of regions whose data exists throughout the real world. This should be transparent to the user.

Two ways of achieving this will be discussed.

One is make use of portals, similar to hyperlinks in that you can click on an object or move the user representation into an area and be directly transferred to another virtual world or a different region within the same virtual world.

The second is to seamlessly link regions of large virtual worlds so that it is experienced as a collective whole, even though it’s data may be widely distributed throughout the internet.

### 2.4.1 Portals

Portals are objects or areas within the environment, which when clicked on or entered into result in a transference of the users viewpoint and representation to another position(in another virtual world or within the same one being navigated).

VRML already supports such a concept with its ability to define hyperlinks to other VRML files. The hyperlink can be activated by proximity or pointing device sensors as defined in the language[VRML ISO/DIS 1997].
Proximity sensors generate events when the user enters into the region of the sensor. Pointing device sensors are activated when the pointing device being used( i.e., mouse) is located over the geometry that is associated with the pointing device sensor. The hyperlinks to other VRML worlds are defined by URL's or URN's with the filename having a .wrl extension.
Portals, thus, do not pose implementation problems in VRML.

### 2.4.2 Regions

Large-scale worlds can be realized by seamlessly connecting their component regions. As the user navigates the world regions are loaded when needed.

They essential problem then, is to know when to start loading a particular region, as this should be done before the user enters the region in order to create a seamless navigation experience, since the user should be able to see the contents of a region before actually entering it.
Thus, the problem is determining when a region should become visible to the user.

This can be achieved through the use of proximity sensors. When a user reaches a certain distance away from the region, the region will be loaded and made visible. The distance could vary according to a level of detail value set for the browser in order to lessen the computational or network load. It could also be specified by the region itself. For example, a region containing large buildings will be visible from a greater distance than a region containing a sports field.

This could be realized by the regional spaces as proposed by Wolfgang Broll, described above in related work.
When the user enters the hull of the region then the contents of the region are loaded, ready to be displayed as the user enters the radiation area. From outside the radiation region an external representation of the area could be experienced. As in the example above of large buildings, the external representation could be a simple model of the buildings without much detail.
In order to avoid reloading of the area every time the user comes nearby, the region' s data could be cached.

Visibility sensors as defined by VRML could also be useful. Suppose regions are separated by barriers, like walls. The user may be within the required distance, but will not realize the region, due to the barrier. The barrier may also be quite extensive and, thus, the region will only be experienced after a certain amount of navigation, as in a maze environment. It will not be worthwhile then, to load the region based solely on its distance from the user. Rather, visibility sensors could also be used. Only when the region is visible and within a particular distance, should it be loaded.

# 3.Shared world data consistency

## 3.1 Introduction

Shared world data consistency is concerned with the degree to which the shared world's data appears the same. It is realised by researchers that complete synchronization of the data is an unlikely possibility, since the shared world's data consistency relies on the complete distribution of the event that caused the change and for this distribution to occur instantaneously, in order that the data is synchronized in time. However, certain types of events could occur simultaneously, like those that are synchronised to occur at a particular time, the knowledge of which has been or is inherently distributed throughout the system. Events could also be predicted, rather than communicated. If the prediction is accurate the event will occur simultaneously. These events will be discussed in greater detail.

However, for our purposes non-deterministic events will need to be communicated at the time of their occurrence. Thus, time delays in communication will make complete synchronization impossible.
Large-scale, widely distributed virtual reality demands that consistency be traded for interactivity, and vice-versa.

Shared world data consistency also concerns the problems of reliable event communication, contention resolution and collision detection as well as limiting the distribution of event occurrences to reduce network load.
These issues will all be discussed in this chapter.

## 3.2 Event synchronization

Event synchronization can occur in several ways. They may be explicitly communicated through the use of update messages. They may be predicted if sufficient information is available. They may also be implicitly synchronized, as in the case of time triggered events or in the case where event sequences are inherently distributed as part of the shared world data, as in the case of scripts or implicit knowledge of causation, i.e. A causes B, which is known to cause C.

### 3.2.1 Update messages

Update messages are required to communicate non-deterministic events, i.e., those generated by human users of the environment. They will also be required to communicate events generated by other agents of change in the system, like artificial intelligence entities, although, not always. If the artificial intelligence is completely distributed and of a deterministic nature, for example a robot which knows that if A then do B, then the event of the robot performing B will not need to be communicated if the robot's intelligence exists on all the replicated databases as well as the event or circumstance of A. This is due to the fact that the robot entity will behave in the same way throughout the system. Events of this kind will be discussed further in the section on implicit synchronization. However, I presented the example here in order to illustrate that this is not always the case.

If the robot's intelligence is not fully distributed then even such events will need to be communicated as the knowledge that, if A then the robot performs B, will not be known throughout the system. It may be a good idea then to distribute the robot' s behaviour, if it is deterministic, with its appearance when navigating shared worlds.

A typical update message will contain information like: the object's position, velocity and orientation. However, it is essential that the message format be generic in order to allow messages of various kinds. For example, it should be possible to transfer object geometry and appearance data in an update message. This would be the case where a user or artificial intelligence entity places an object into the world.

Messages could have a format resembling that in following figure.

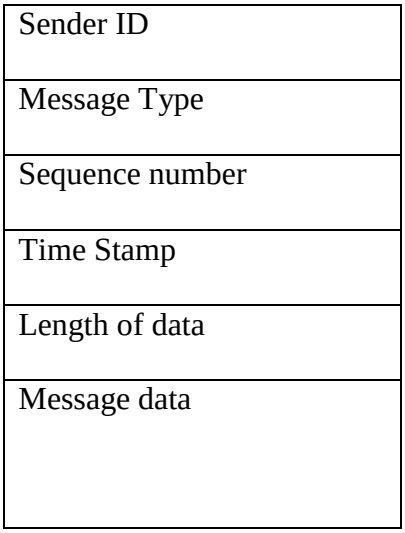


**Figure 9 – Format for update messages.**

Messages may be of various types. The most common are likely to be object state updates. The data contained in such a message is represented in figure 10.

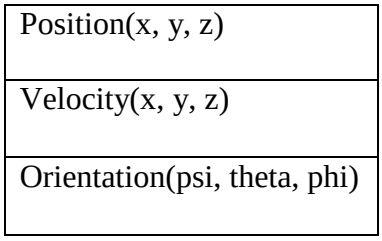


**Figure 10 – The data contained in an object state update message.**

Object state update messages should fit into the maximum transfer unit(MTU) of the data link being used in order to maximise real-time interactive use, as in the lightweight interaction communication of NPSNET[Macedonia et al 1995].

#### 3.2.1.1 Heartbeat messages

Heartbeat messages are update messages which are periodically broadcasted(after a specified number of seconds), regardless of changes in an object's state. For example, even stationary objects will send heartbeat messages. They are useful for ensuring consistency as a previous update message may have been lost. They can also help new entrants learn about the state of the shared world.

### 3.2.2 Prediction – Dead reckoning

It is often the case that object state updates could be predicted. If an object has a particular position and velocity, then its future position can be predicted based on that information. In most simulations objects do not show a high degree of randomness in their movement.

Dead-reckoning algorithms are the algorithms used to make such predictions and have shown a great deal of success in reducing network load, as experienced in the case of experiments with NPSNET[Macedonia 1995] and in my own work. They are also an essential part of the Distributed Interactive Simulation protocol(DIS)[IEEE 1993].

The algorithms can be tailored to a specified degree of accuracy. Thus, consistency can be traded for greater interactivity, and vice-versa, through the use of dead-reckoning algorithms.

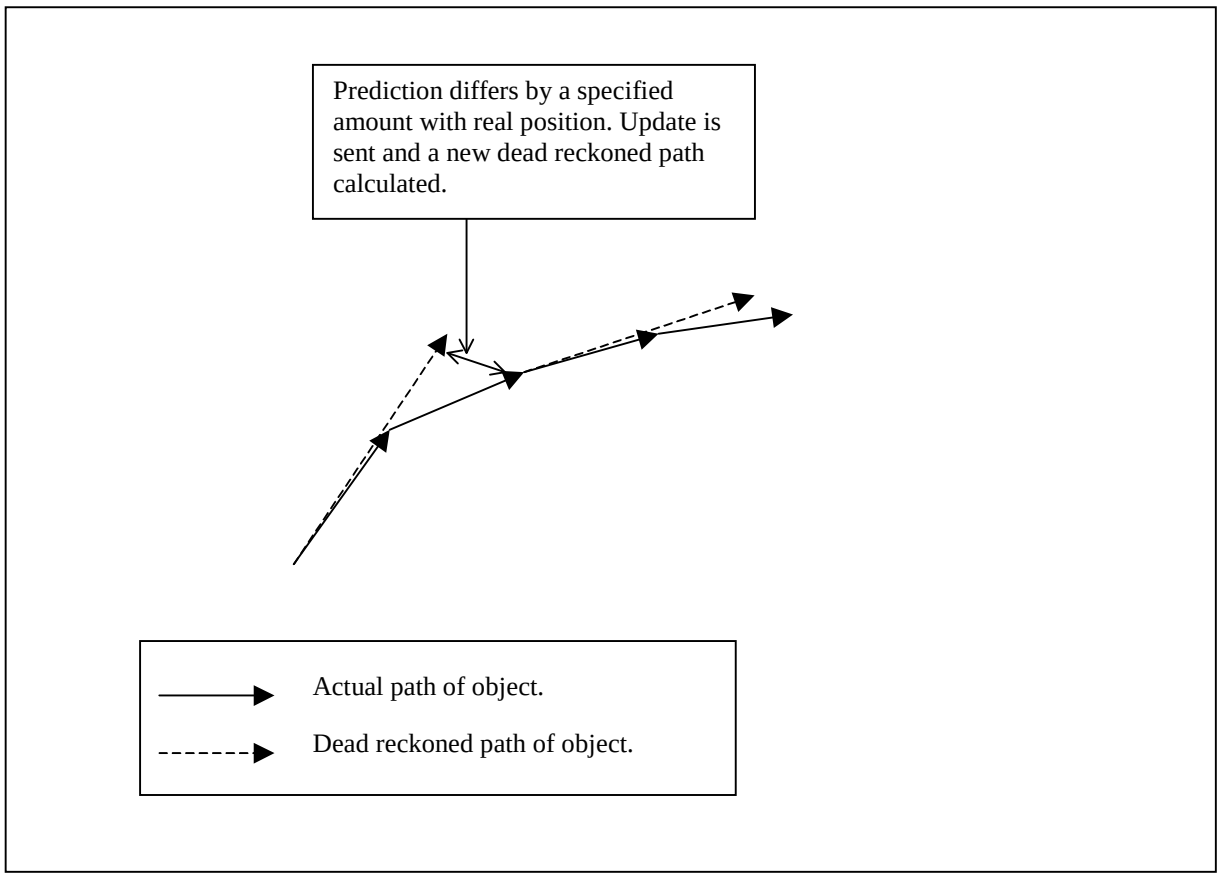


**Figure 11 – Dead Reckoning.**

It must be noted that the controlling process of the object's movement also uses the dead reckoning algorithm in order to determine whether its state is being correctly predicted. The objects real position and its predicted position can, thus, be checked and if necessary, an update sent. From this update new predictions can be made.

#### 3.2.2.1 Implementation

The basic dead reckoning algorithm is essentially simple. An object's predicted position is a function of its previous position, velocity and elapsed time.

That is,

| Predicted position = previous position + elapsed time * velocity |
|---|

Note that velocity must be defined as units per time period(i.e., m/s) and not per frame or execution cycle. This is to take into account processes or computers operating at different speeds. A faster computer will make a greater number of predictions over the same time period as a slower computer and, thus, if time is not taken into account the predicted positions will differ markedly.

Dead reckoning essentially consists of two components: a dead reckoning prediction checker and a dead reckoning prediction maker.

The dead reckoning prediction checker is responsible for checking that predictions being made reflect the actual state of the object. It has access to the actual state of the object as it resides on the same computer. If the actual state of the object differs with the predicted state by a specified amount, then an update is sent. The specified amount will differ according to the needs of the application(i.e., greater consistency will require a higher degree of accuracy and, thus, a smaller allowed difference will be specified).

The dead reckoning prediction maker will analyse the object's last transmitted state and make a prediction for its current state, that is, if the object is not receiving updates directly from its controlling process.

My implementation of dead reckoning was an extension to RhoVeR(Rhodes Virtual Reality). RhoVeR is a distributed /parallel virtual reality system. Its distributed nature is achieved through the use of Virtual Shared Memory(VSM) data tables and their corresponding VSM managers, which allows access to the VSM data by processes and ensures that any changes to the local data is propagated throughout the system, thus, ensuring the consistency of the shared world's data.

To achieve dead reckoning I interfaced the dead reckoning prediction checker and the dead reckoning prediction maker with the VSM manager. This is conceptually represented by figure 12.

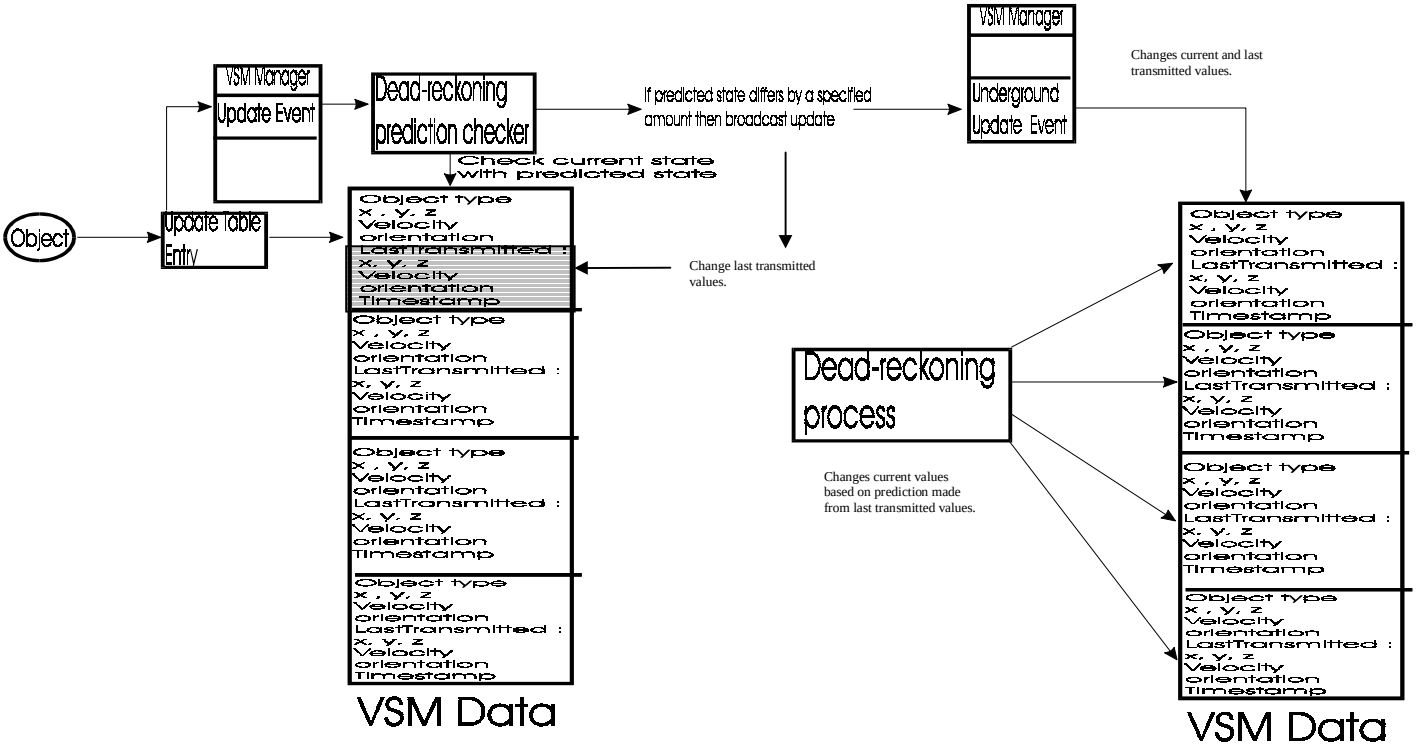


**Figure 12 – Dead reckoning with RhoVeR.**

Figure 12 demonstrates what happens when an objects controlling process updates it's state.

Firstly, a call is made to UpdateTableEntry(), which updates the objects current values in the local VSM's data. This, in turn, sends an event to the VSM manager, called EVENTUPDATE. On receiving the event the VSM manager calls the dead reckoning prediction checker to see if the objects data needs to be propagated throughout the system. Predicted values, based on the objects last transmitted values, are made. If the predicted values and the object' s real current values differ by a specified amount, then the objects real current values are transmitted. If they don't, we know that the predicted values for the objects current state are correct throughout the system, as the dead reckoning process uses the same prediction algorithm. Thus, no update message is necessary.

On receiving an update message, an event called EVENTUNDERGROUNDUPDATE is sent to the VSM manager. The VSM manager will handle this by updating the specified object's current and last transmitted values.

Meanwhile, the dead reckoning process is run periodically. This is achieved by registering the VSM manager with the timer unit module used in RhoVeR. It will then periodically receive TIMERTICK events, after specified durations. On receiving a TIMERTICK event, the VSM manager will call the dead reckoning process.

Each time the dead reckoning process is executed it scans through the VSM's data table and updates the objects current state, based on predictions made from the last transmitted values. If the objects controlling process is local to the machine, that object will be ignored.

### 3.2.2.2 Results

Experiments were conducted with RhoVeR to test the efficiency of the dead reckoning algorithm in comparison with the same application without it.

The application contained a single object. The object's behaviour is simple, in that it navigates on a flat plane. The navigation of the object is in a straight line and is limited to a square area. On reaching the border of the area the object is rotated and continues moving in the new direction, until it again reaches a border.

The world of this object is represented in figure 13.

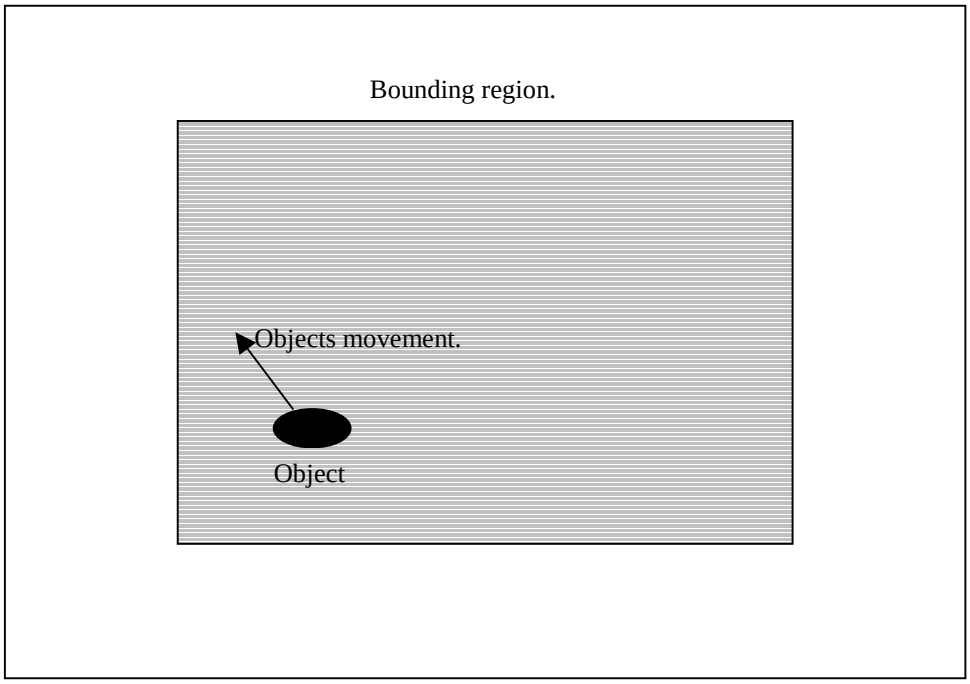


**Figure 13 – The object's world.**

It could be said that the object's movement and world is too simple to effectively test dead reckoning. However, this is wrong. By making use of bounding areas of different sizes, we can introduce different degrees of variation in the object's movement. A small area will result in a lot of variation, whereas a large area will result in less. We can, thus, easily test the dead reckoning algorithm at various levels of variation in the object's movement.

The following table represents the results obtained using different world sizes for the object.

| | *World size 1 X* | *World size 2 X* | *World size 3 X* | *World size 4 X* |
|---|---|---|---|---|
| ***Average number of updates/minute. Before dead reckoning.*** | 202 | 168 | 211 | 216 |
| ***Average number of updates/minute. After dead reckoning.*** | 24 | 14 | 6 | 4 |

**Table 1 – Results of dead reckoning using one object**

The results in the table above clearly show a dramatic decrease in the number of update messages needed when using dead reckoning. Even a very small world size with a large amount of erraticness in the object's movements, 1x in the table above, showed a great improvement from 202 to 24 updates. If we consider a world size of 4 X, then we see that instead of 216 updates needing to be broadcast per minute, only 4 were necessary.

Thus, it can be seen that a world with many such objects, without dead reckoning, could easily flood the network with update messages. But, with dead reckoning we can dramatically decrease the network' s load and, thereby, make scalable virtual environments possible.

### 3.2.3 Implicit synchronization

Certain events will be implicitly synchronized and will, thus, not need to be communicated, as they will occur naturally throughout the system.
These events are essentially of three types: time triggered events, deterministic script generated events and events that will naturally occur due to knowledge of causation being distributed throughout the system.

#### 3.2.3.1 Time triggered events

Shared world events could be synchronized according to time. That is, events could be specified to occur at a particular time, taking into account different timezones. If the participant's database has prior knowledge of these events and their time of occurrence, their occurrence would be completely synchronized. This is not possible through communication of event occurrences. It could, thus, be asked whether it would be better to rather synchronize all events to time. For example, a user will initialize an event like picking up an object, but before the event takes place, knowledge of the event will first be distributed. Once this is done, a time will be given for it's occurrence and, thus, the event will take place and be experienced at exactly the same time by all participants. This is not a reasonable approach for our general purposes as the costs of interactivity will outweigh the benefits of consistency for most applications making use of the system, that is, applications that have a high regard for human interactivity. Although, events of this type could be catered for in the system, in order to make it more generic to user's needs.

#### 3.2.3.2 Deterministic script generated events

Scripts that define the behaviour of objects in response to events are likely to be extensively used in virtual worlds on the internet. They have been catered for since the beginnings of the VRML 2.0 specification [VRML 2.0 spec] and extended by further advancements to the VRML specification[VRML ISO/DIS 1997]. It is, thus, possible to incorporate a scripting language like Java Script directly into VRML files.

A characteristic of scripts is that a single event can cause an event cascade. For example, event A could result in a script being executed which will generate events B, C, D … etc. If the script has been fully distributed to all participants, then the script's events will be generated by each instantiation of the script. This could be hugely detrimental to the network load if all these events are going to be communicated. In fact, the number of events communicated would be n * the number of events in each script(where n is the number of participants). This is clearly inefficient if each participant has an instance of the script, as they would already know about the events.

Therefore, scripts which are fully distributed and of a deterministic nature should not have to communicate their events that they generate. It is sufficient for only the initial event, which causes the script's execution, to be communicated(event A in the example above).

#### 3.2.3.3 Knowledge of causation

Knowledge of causation is analogous to the physical laws of the real world. For example, Newton' s laws of physics. If such laws are incorporated into the description of virtual worlds, then events that occur as a result of them will not need to be communicated, as they would be inherently distributed events.

For example, a law of gravity stated as follows: all objects have an attractive force, and distributed as part of the virtual world description, will result in the objects having a gravitational force. This force will then affect the behaviour of the objects, in the same way, throughout the system. Thus, events that occur as a result of the laws of the virtual world will not necessarily need to be communicated. The controlling process, in this case gravity, is fully distributed, unlike asymmetrical control in the case of human users or other asymmetric processes.

Such laws could be implemented in terms of VRML scripts, except their domain would be all objects in the virtual world.

### 3.3 Reliability of event communication

An architecture which is scalable cannot be based on an underlining reliable communication protocol, like TCP, as real time performance would be greatly degraded. If acknowledgements were used for each packet a participant receives, the network would be flooded as each message a participant sent would result in n –1 acknowledgements (where n is the number of devices which receive the message).

The architecture is likely to be based on IP multicasting, which is of an unreliable nature. It may, thus, be necessary to introduce a certain level of reliability into the system's communication, without seriously degrading its performance.

This could be achieved through the use of a server, which would also be connected to the multicast address.
The following reliability mechanism is suggested by [Broll 1997b].
Positive acknowledgement messages are used. Acknowledgments are sent, though, by the server only. Each message sent by participants includes: a unique sender ID, a message ID, a sequence number for each package making up the message and the total number of packages in the message. The server stores the packages and acknowledges each package it receives. All participants in the system receive the acknowledge. The sender keeps a copy of the message until it receives acknowledgements from the server for all the packages that make up the message. If acknowledgement messages are not received their corresponding packages are retransmitted. Receiving participants can also know if they received the entire message through the same mechanism. They will know that the server received the entire message if they receive all the acknowledgements. However, they may not

have. If not, the server can be requested to send the desired packages directly to them through a unicast link (UDP/TCP).

I would suggest, though, that the number of participants, who did not receive the message, should be taken into account. If it were a large number, then perhaps using the multicast address would result in more efficient communication, since a requirement of a scalable architecture is that unicast or point-to-point communication should be limited. Those participants who did receive the message could ignore it's rebroadcast.

Also, the server of the system will have other tasks to attend to, like handling new participants and dealing with issues like contention and inconsistency resolution. If it needed to rebroadcast a failed message to large numbers of users, then this could hinder it's functioning. The load should rather be distributed when it is beneficial to do so. In the case of a few participants not receiving the message, it may be of greater benefit to communicate directly to them through a unicast link. Multicasting would result in an unnecessary load on the other participants, without a justifiable benefit to the server.

Reliability of communication can, thus, be introduced into the system. However, the degree to which it is implemented should be adaptable to the virtual world's designers needs. The communication protocol should, therefore, be defined in a manner that takes this into account. It must also be possible to connect virtual worlds with different reliability requirements.

This is an issue which should be considered by the designers of the proposed VRTP(Virtual Reality Transfer Protocol).

## 3.4 Contention resolution

In all situations where a shared resource is being made use of by multiple users, there exists the possibility that the simultaneous manipulation of the data will result in violations of the integrity or consistency of the data. For example, shared databases that can be written to by many users.

Since the shared world data is a shared resource in this case, there exists the possibility of such violations occurring. For example, two users may attempt to grab and move an object at the same time. They may not have knowledge of the others attempts, due to time delays in the communication of the event. Thus, they could both move the object, possibly in different directions. The last movement that is communicated would overwrite the previous attempts by the other user. With many movements the object will appear to jump from one position to the next(whichever one was communicated last).

In order to solve this problem a locking mechanism needs to be in place so that a user can manipulate an object without conflict.

### 3.4.1 Single locking

Single locking could be achieved without much degradation to the real time performance of the system. Each entity, including the server, keeps track of the lock status for objects within its area. When an entity wishes to manipulate an object it checks the lock status of that object in its database. If it isn't locked it broadcasts the new lock status and does the desired manipulations. This is a weak form of locking, as it does not guarantee consistency. It may occur that another entity also wishes to

manipulate the object at the same time, as in the example before. In this case, it will also notice that the object is not locked and thinking that it can place a lock on the object broadcasts the fact that the object is now locked. It may only receive the other objects locking message after it had already communicated the lock. Both entities will now be in direct contention for the object.

This contention, though, can be resolved by using the server. If the server acknowledges locking requests and the unique ID of the entity placing the lock is associated with the lock status of an object, then the server will acknowledge both locking requests in the example above, however, it is the last acknowledgement transmitted that will be effective. The previous acknowledgement will be overridden and only one entity will have a lock on the object.

After the entity has completed its manipulation it broadcasts the removal of it's lock on the object. Locks can also be removed or reconsidered by the server after certain time periods, in case locks aren't removed after a reasonable time period or the lock release message was not received.

This is a weak locking system as manipulations may have occurred during this contention period. Hopefully, though, it will be too short, in most cases, to cause large inconsistencies.

### 3.4.2 Multiple locking

When a group of entities wishes to simultaneously manipulate an object, multiple locks may be required. But, multiple locking can be problematic in a non-synchronized system, such as is the case for any architecture which is of a widely-distributed nature and still wishes to display a high degree of interactivity. This is due to the time delays involved in communicating the manipulations. In order for a consistent interaction to occur, the order of the manipulations would need to be compared(perhaps by the server) and the result communicated. Such a solution may affect the required interactivity to an unacceptable degree.
However, perhaps suitable solutions could be found for certain situations which do not require synchronization and high degrees of consistency.

A typical problem would be a group of users attempting to grab and move an object at the same time. The difference with single locking is that we don't want any single user to take ownership of the object. The manipulation of the object must appear to be a collective result of the attempts by each user to move the object. Note that we are not concerned with it being an exact result, but rather a satisfactory result for users of the system.

An example situation will now be presented. It consists of four entities contending over a single object. That is, attempting to move it in different directions. We define each entity as applying an equivalent force on the object. The entities exist on different machines.

The initial situation is represented in the following diagram(figure 14).

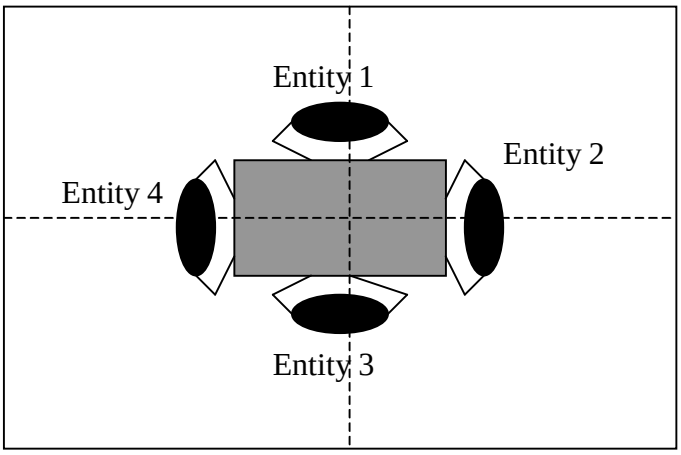


**Figure 14 – Four entities attempting to move an object.**

The entities are all attempting to move the object in different directions. If each entity does not take into account the attempts of the others, a situation will arise where the object true position is not known. The situation would appear to be different on each entity's machine, as shown in figures 15 to 18.

A solution is required, as we would like such situations to be possible in our shared virtual worlds. My suggested solution does not propose to keep the interaction consistent at all times, however, this would be an unreasonable requirement anyway, as a particular focus of this thesis is on interactivity.

The objective is to satisfy the expectations of the users by approaching a situation which would occur with a consistent solution.

My solution then is to suggest that the server be used to make judgements about what the resulting situation should be.

Firstly, when entities wish to manipulate an object, they check its lock status in their database, as in the case of single locking. However, certain objects could be defined to allow more than one lock. If there is a free lock the entity registers its lock with the server, which the server will acknowledge if the lock is possible. If in the case of an entity grabbing an object, its orientation will be judged relative to the object and vice-versa. If the object is moved by another controlling process, which has the required rights, then the entity will also be moved to the same extent. In the same way, if the entity moves, the object will move with it.

A suggested solution then, based on these definitions, is for the entities to behave, for a certain period of time, as if they had a single lock on the object. They should ignore the update messages they receive from the other entities that have a lock on the object and updates on the object's position due to the actions of these entities, during that period of time. Meanwhile, the server will receive and process all updates that it receives from the entities. Based on this information, it will determine a solution for the object's position which takes into account the manipulations exerted by the entities. This position will then be communicated. Since the server will have the required rights, the position of the object will be updated throughout the system. This will also result in the entities being moved as they are judged relative to the object if they have a grip on it. In fact, all the components of the interaction(the object and entities) can, in this case, be viewed as a single object.

It is, thus, the responsibility of the server to determine the position of this object based on its component forces(the entities) which are affecting its state.

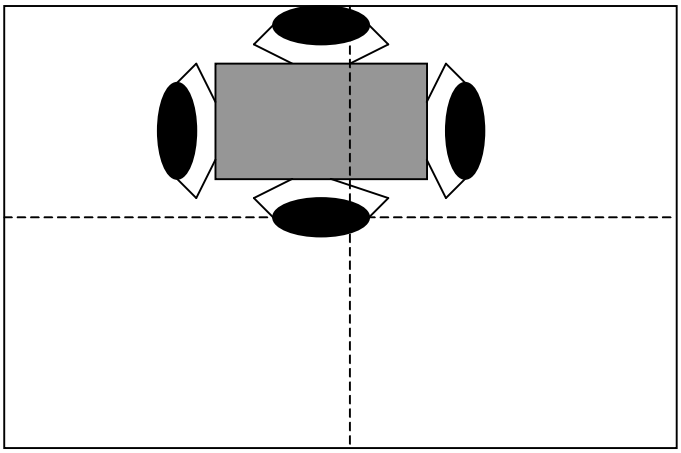

**Figure 15 – Entity 1's situation.**

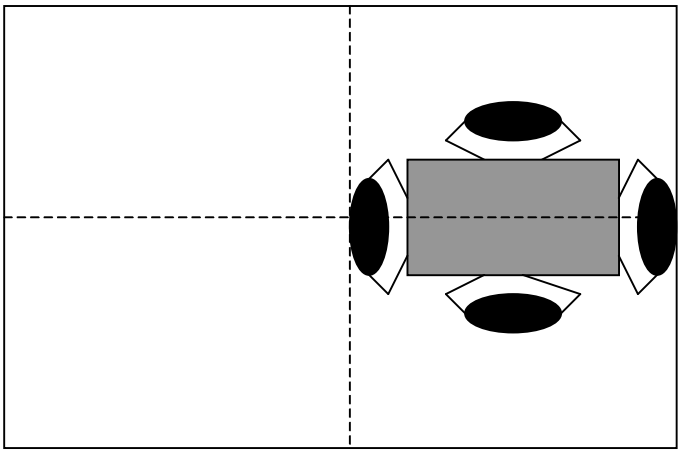

**Figure 16 – Entity 2's situation.**

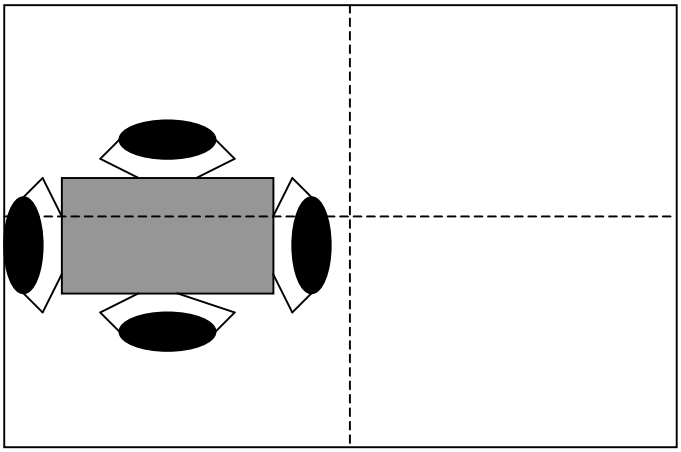

**Figure 17 – Entity 3's situation.**

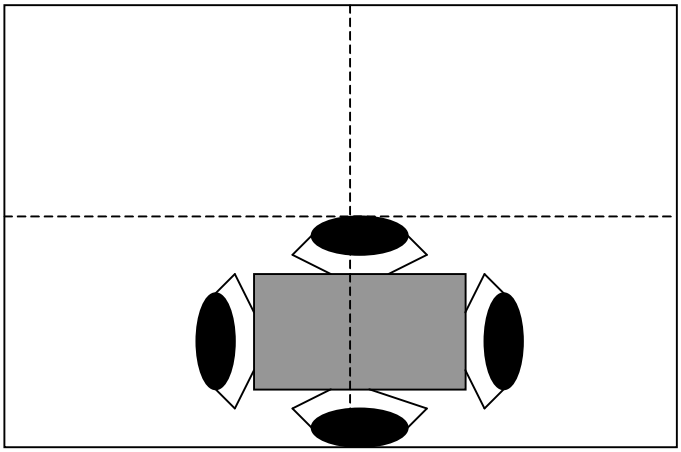

**Figure 18 – Entity 4's situation.**

Figures 15 to 18 represents the different views of the situation on each of the entity's machines. My solution, therefore, suggests that these four situations be processed in order to determine a collective result. If each entity applied an equal force in opposing directions, starting at the same time, then the result which should be communicated by the server should represent the starting position(figure 14) as the opposing forces would cancel each other out. In essence, then, my solution allows instant feedback from a user's interaction, which may, though, be altered by the server, but the server, in turn, ensures an expected result.

### 3.5 Collision detection

Collision detection is problematic due to the shared world's data not being synchronized at all times. If there where no inconsistencies, then conventional collision detection could be used on each of the participating machines, as each machine would have the same shared world data and could, thus, know that a collision which occurred locally, also occurred throughout the system.

This is not the case, though, for an architecture that is not always synchronized, like the suggested architecture in this thesis. The data inconsistencies could result in conflicting situations arising. For example, in games where the objective is for players to shoot each other, problems could arise out of the slight differences in the position of a player's representation, due to the expected latency. On each machine a player's position may differ by a small amount. This small amount, though, could result in a conflicting situation. This could be represented by the following diagram(figure 19).

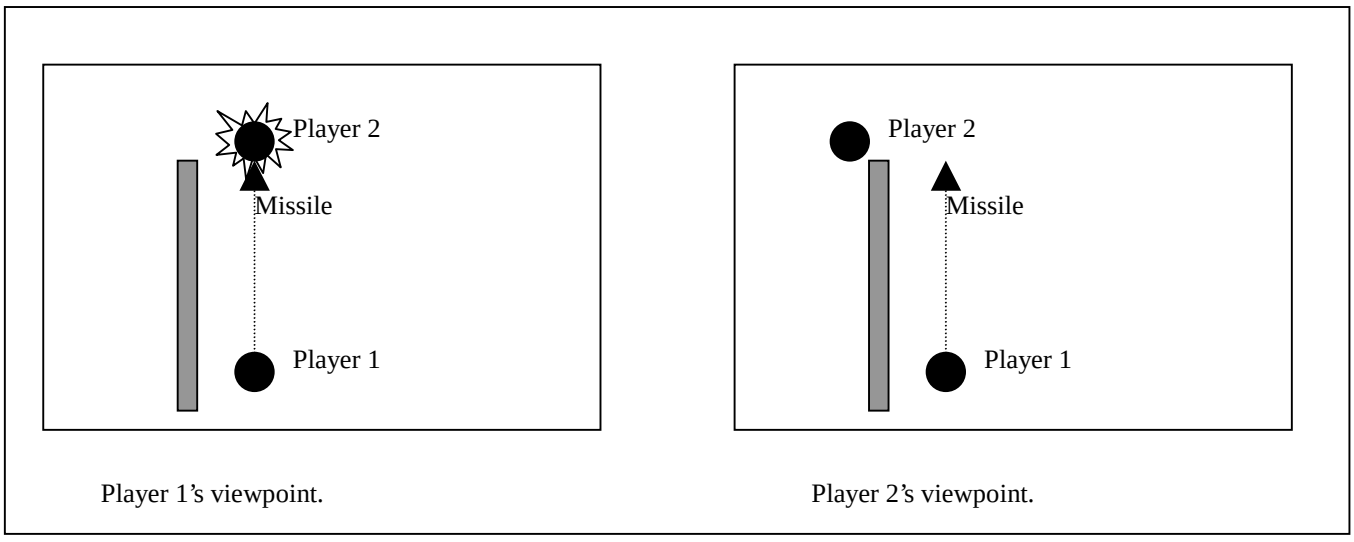


**Figure 19 – The viewpoint from two different player's machines**

Figure 19 shows what could occur due to data inconsistencies. Player 1 can see player 2 and fires a missile, which, according to player 1's viewpoint, is on target. Player 2, however, had previously moved out of sight, but this update message is still to reach player 1. On player 1's machine the missile hits its target. On player 2's, it doesn't.

The best solution may be, although with a certain loss in real-time response, to have the player broadcast when they are hit, according to their local data. In this situation, no broadcast will take place, as player 2 will not consider itself to have been hit. The loss in real-time response will occur, as the missile on player 1's machine, if it truly hit the player, will have to wait for such a message to be received before detonating. This solution introduces a certain amount of subjectivity regarding who hit what. In this case, the player's object has a higher priority than the missile object in deciding a result.

However, what would occur when objects of equal priority collide. For example, a sports simulation where one player tackles another. The same inconsistency may occur as in the above example. One player will register that the collision took place, the other won't. The solution, in this case, would be to only register the collision if both players broadcast the fact that a collision took place.

In general, many collisions could be dealt with in a localized manner. That is, if static objects are part of the world description, then their position will be consistent and, thus, conventional localized collision detection will result in the same situation throughout the system. However, where there are inconsistencies collision detection is a complex topic without a simple generalized solution.

### 3.6 Event Distribution

Another method of reducing network load is to limit the communication of events according to their particular area of interest. This method works on the premise that events will have an effect on a limited area of the world. For example, a user's representation will only be seen when within a particular distance. Thus, only other participant's within sight of the user will need to receive regular updates regarding his position. Of course, those participants will need to know if they are within sight of the user and will, thus, need to have some information regarding his position. This can be achieved by dividing the world into regions, like the hexagonal cells used by NPSNET(figure 7) or the regions suggested by Broll(figure 6), as described in the chapter on related work. Each region will be given its own network address. For example, the multicast address and their own unique port number. Objects will send and receive updates to the regions based on the range of their influence. The network addresses for each region will be managed by the server and broadcasted to the objects, when required.

A criticism of NPSNET's cells is that they are all of a uniform size. As a result they are not easily adaptable to the needs of particular virtual worlds. For example, a virtual world where entities cover large distances relatively quickly(i.e., aircraft simulations) will most likely make use of large cells. Their range of influence over short time periods will be large and, thus, using small cells will be inefficient, as the entities will rapidly move from one to the next. Likewise, a virtual world where the entities have a small area of influence and exist relatively close together(i.e., spectators in a sports stadium), will most likely require small cell sizes.

Both these situations, though, could exist in a single virtual world. For example, a jet flying over a sports stadium.

A solution, then, needs to be found to handle such situations.

The suggestion by Broll does allow division of the virtual world into arbitrarily sized regions. However, since these regions are circular in nature, problems may be encountered when covering areas of the virtual world by using such regions. The cells of NPSNET all fit together perfectly, leaving no space in-between. Broll's regions, though, can overlap. In this way all areas could be covered by a parent region whose extent is the entire virtual world area or regions which all intersect each other to the extent that no uncovered spaces are left. This is shown by figure 20.

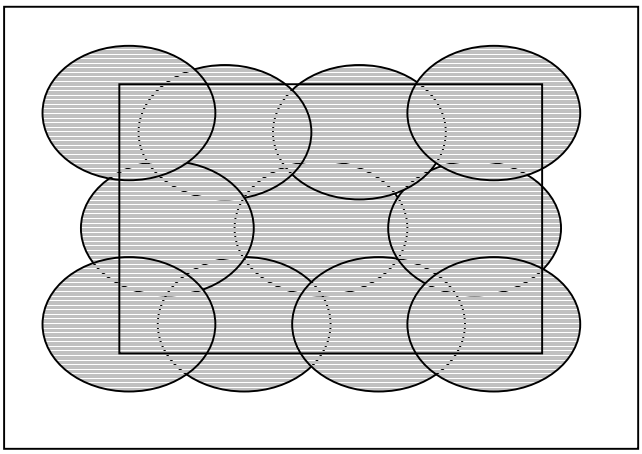

**Figure 20 – Area covered by concentric regions**

It is, thus, possible to cover the entire virtual world using such regions. The problem, though, is that it is up to the designer of the virtual world to ensure that it is completely covered as this will be, in most cases, necessary. Such a task will be quite difficult with regions of different sizes. It is open to error and also inefficiencies as large areas may overlap resulting in the unnecessary transmission of messages to regions they would otherwise not go to.

A solution could be implemented based on the bounding box technique used in many graphical applications. I suggest defining regions according cube shapes as they are easier to fit together and their shape can also be defined in terms of three dimensional magnitudes(length, width and height), rather than just a radius. They could be hierarchical and overlap, as in Broll's suggestion, and also have hull, horizon and radiation areas specified by cubes.
Entities could be given specific rights with regard to sending and receiving messages on entering a bounding region. That is, they may be allowed to listen for messages being broadcast to that region, but not send messages to it.

The example of a jet flying over a sports stadium could then be realised by having a bounding box which would include the stadium and the airspace above it. This bounding box could then have children regions, much smaller than itself, where the spectators could exist. The children regions could be specified to listen to messages from the parent region, but not send messages to it. The spectators within the children regions could send messages to it and receive messages from it.
Thus, the jet flying overhead will belong to the parent bounding region and have rights to send and receive messages through it. The jet's appearance and sound would, therefore, be communicated to the parent region. It will, though, not be able to see or hear the spectators, as they cannot send messages to the parent region. The spectators, on the other hand will be receiving from the parent region and will, thus, see and hear the jet.
Each region will receive it's own IP multicast address. In the above example, the spectators will send and receive messages through one address, while listening on another.

Another example is represented in the diagram below(figure 21).

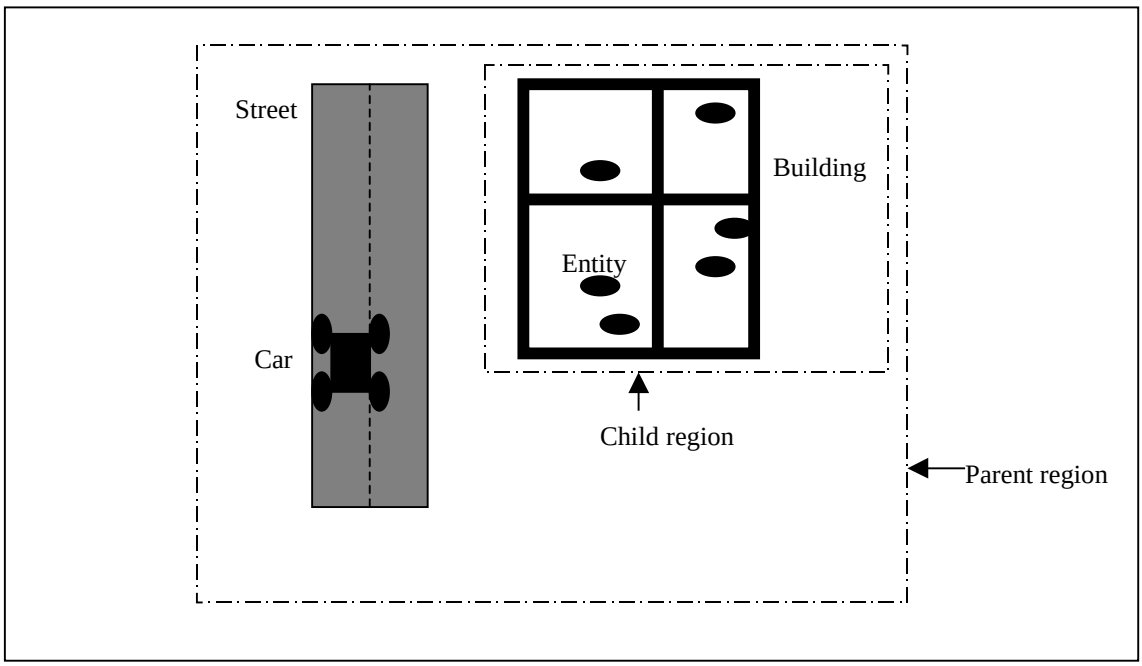


**Figure 21 – Sub-division of virtual world into parent and children regions**

The above diagram represents an example situation of entities in a building and a car driving past. The entities belong to a child region, which can listen to messages from the parent region. Therefore, it is possible for them to see and hear the car driving past. The car, on the other hand, belongs to the parent region, which sends messages to its child region, but it cannot receive messages from that region. The driver of the car, then, cannot see or hear the entities in the building. Note though, that the building can belong to the parent region, since the regions overlap. Thus, the building can be seen by the driver of the car, but not its interior.

It conclusion, it can be shown that these measures do not reduce the realism of the virtual environment, as interaction is normally limited in a similar way in reality, while potentially greatly reducing the networks load.

# 4.Shared world data persistence

## 4.1 Introduction

It will be desirable to create virtual worlds that can be modified by users. Such changes should be kept and communicated to users whenever they enter the virtual world. This would allow us to create dynamically changing environments. It will heighten the users experience of existing in a real environment which they can interact with and, by doing so, change it. Virtual environments could become the collective result of many people's creativity. Collective projects could be undertaken. For example, groups of architects, spread out throughout the world, could collaborate on the design of a sports stadium. The structure could be manipulated and the changes would be immediately visible.

Thus, virtual environments will become places in which we can truly exist. That is, environments which we can be affected by and have an effect on.

## 4.2 Modifying the world

Modifying the virtual world could be achieved in various ways. For example, moving objects, changing them, creating new objects and even bringing new objects into the environment. In order for such changes to have a lasting effect on the virtual world, the worlds description will need to be changed and such changes communicated. For example, if a user brings a new object into the world, the object' s characteristics will need to be communicated, such as, it's appearance and behaviour. This would be achieved with an update message containing the object's description as its data.

The object would then be added to the participants and the server's world description.

The server, though, needs to also make changes to the virtual world description file(which should be in VRML) in order to make this new addition known to new users logging in to the world.

### 4.3 Entering the world

When a new user needs to gain access to a shared world, they will use the server as a dial in point. The server is then responsible for transmitting the latest world description to the new user. This will likely be a TCP connection, as reliability is needed in transferring the world's contents. This could take a certain amount of time, though, and during that time new changes could be made by the current participants.

In order to solve this problem the new changes will need to be cached by the server and transmitted directly after the original world's description has been communicated. Once the new participant has taken into account the effect of the latest changes, they can join the world. Heartbeat messages, as described earlier, will also be useful in resolving any inconsistencies that may have occurred.

### 4.4 Security Issues

The ability to manipulate shared virtual worlds poses some interesting security problems. Virtual worlds could be protected by passwords in much the same way that access to databases and user accounts are protected. For example, in the case of architects working on a collective design each architect could be given a login name and password. However, worlds that are generally accessible will also be desired. It should be possible for anonymous users to express their creativity in a shared virtual

world. However, such a situation is open to cyber-terrorism. It would be very easy for such a user, due to their anonymity, to destroy these virtual worlds, without any cost to themselves. Allowing only particular users to access a virtual world is not a general solution.

I suggest that there will be essentially six levels of security.

Level 1: Accessible only to registered users.

Level 2: Accessible and modifiable by registered users.

Level 3: Accessible to all, but only modifiable by registered users.

Level 4: Accessible to some and modifiable to different degrees by different users.

Level 5: Accessible to all and modifiable to different degrees by different users.

Level 6: Accessible and modifiable by all users.

Different degrees of modification would, for example, range from moving an object to changing its geometry.

Level 6 worlds would be very interesting in that they could be completely anarchic. Would some users mindlessly destroy such worlds? The characteristic of anonymity is, in general, quite foreign to humans, especially with regard to interaction. Anonymous interaction has in the past been very limited. Often the interaction being one sided, in that one individual or group is anonymous and the other not, i.e., terrorist organizations expressing political views to governments. With level 6 worlds, both the creators and users could be anonymous. What kind of social ethics could arise out of such a situation?

# 5. A proposed architecture

Based on my research and suggestions, I will now present an architecture that is scalable and can be widely-distributed.

The order of the presentation will be to first discuss the architecture's communication topology and shared world data distribution.

Next, an overview of the components of the system will be given. I will describe how the mechanisms that I have researched and made suggestions about, regarding the shared world data distribution, consistency and persistence, will be realized by the components of this architecture.

## 5.1 Communication topology and shared world data distribution

It is my opinion that a distributed communication topology with a replicated database is the most viable architecture, at this time, for realizing large-scale, widely-distributed virtual reality environments on the internet(the architecture, with a server, was represented in figure 4).

The advantages of such an architecture are:

- Each participant can communicate directly with all the other components of the system(participants, the server, etc …). This can be realized through IP multicasting, possibly making use of the Multicast Backbone(MBone).
- Communication can be realized by sending a single message to the multicast address. Therefore, communication does not have to be relayed through the server with the server having to address the message to every other participant individually. The server will not become the bottleneck of the system with large numbers of users.

- There are no “links in a chain” and, thus, participants who are using slow or faulty equipment will not pose a great risk to the system. A user with a slow machine will not individually slow down the entire system. Heterogeneous devices can, thus, be used.
- A replicated database ensures that processing can be decentralized. That is, data does not need to be processed by a server and communicated to the participants whenever updates to their virtual world are required. The participants can do the processing themselves, although, their world data must still be kept up to date by event messages. But, such messages are small in comparison to the information that would otherwise be communicated.

A disadvantage is that:

- A replicated database is not synchronized at all times. This, though, is to be expected if we are to create virtual reality applications which are of a large-scale and widely distributed nature. Insisting on synchronization would greatly limit the applications that could be realized.

This architecture’s communication topology and shared world data distribution is, thus, the most likely candidate for achieving the objectives of:

- Allowing large numbers of users who may be widely-distributed.
- Minimizing latency, as participants can communicate directly with each other while taking advantage of multicasting.
- Allow for heterogeneous devices.

## 5.2 Components of the architecture

The components of the architecture are similar to those suggested by Broll, as described earlier. They are: a server, participants, the multicast address, proxy servers and relays. The functions of the proxy server and relay are mainly to allow participants who do not yet have internet multicast capabilities, to connect to the virtual world. However, they are not an essential part of the architecture that I am suggesting, as IP multicast will become increasingly accessible. In fact, participants can transmit IP multicast packets within normal unicast packets to multicast routers, which then extracts the multicast packet and transmits it as appropriate. This is known as tunneling[Casner et al 1994]. The proxy server can, though, perform some of the tasks of the server(i.e., used as an additional dial in point or help with reliability issues). It can, then, be considered an extension, but not a necessary component of the architecture.

As such, I will deal with the following components, which are the essence of the architecture:

- The server.
- The participants.
- The multicast address.

They will be defined in terms of their responsibilities. That is, what functions do they perform in order to achieve the objectives of the architecture?

**The Server**

The server is the participant' s entry point into the virtual world. The first network address that the participant receives, will be that of the server. When a participant connects to the server, the server's first responsibility will be to communicate the description of the virtual world. The description must be up to date and, thus, the server will need to keep track of all changes that are made to the virtual world. The description should be in VRML code, which will need to be dynamically generated before transmitting to the participant, in order to reflect the current world situation. Changes that occur to the virtual world during this transmission will need to be cached. Once the VRML file has been transmitted, the latest changes will be communicated and then the multicast address that is being used.

These transmissions should be achieved using a reliable network protocol like TCP/IP.

Note that the participant still keeps the unicast address of the server, as this will be used for reliability purposes.

Mechanisms that I have discussed, which the server will also be responsible for, include:

- Seamlessly connecting worlds. Giving network addresses of servers managing other virtual worlds or regions when needed.
- Ensuring a certain degree of reliability in communicating events.
- Contention resolution(single locking, multiple locking).
- Event distribution. That is, managing the network addresses of regions to be given to participants when required.

- Modifying the world. Ensure that security measures are not violated and the persistence of the data.

The server could be composed of various components, it may not necessarily be one machine. Should the load on the server become excessive then its functions could be divided amongst a number of machines. A particular machine could handle new participants, by keeping track of and sending them the world information. Others could deal with contention resolution, reliability of communication and other causes of data inconsistencies.

**The Participants**

The participant's machine is responsible for ensuring that changes that occur to the participant's local world data are reflected throughout the system. It can achieve this objective through the use of the following mechanisms(which have been previously discussed):

- Communicating update or event messages to the multicast address when required.
- Use dead reckoning to make predictions for an object's movements and to check that local objects(that is, objects whose control is local to the participant's machine, like the user's representation object) are being correctly predicted on the other participants machines and the server.
- Efficiently handle implicitly synchronized events. Know when not to send event messages, as in the case of time triggered events, deterministic script generated events and events whose occurrence are a result of the "laws" of the world. Such events will be inherently distributed.

- Check for acknowledgements from the server after sending each message. If acknowledgements aren't received within a certain time period, retransmit the message.
- Manage locks on objects and broadcast when locking or relinquishing a lock on an object.
- Achieve collision detection that takes into account inconsistencies within the system.
- Distribute event messages according to the event's area of influence.

**The Multicast Address**

The multicast address is used to communicate updates and event occurrences to the server and participants of the shared world. Regions of the shared world can be given their own address(i.e., a unique port number). In this way partitioning can be achieved, where events are only broadcasted to their area of influence, thereby reducing the network's load. Multicasting could be achieved through using the multicast backbone, which has been shown to be useful for achieving large-scale virtual environments[Macedonia 1995].

The architecture is represented by the diagram on the following page(figure 22).

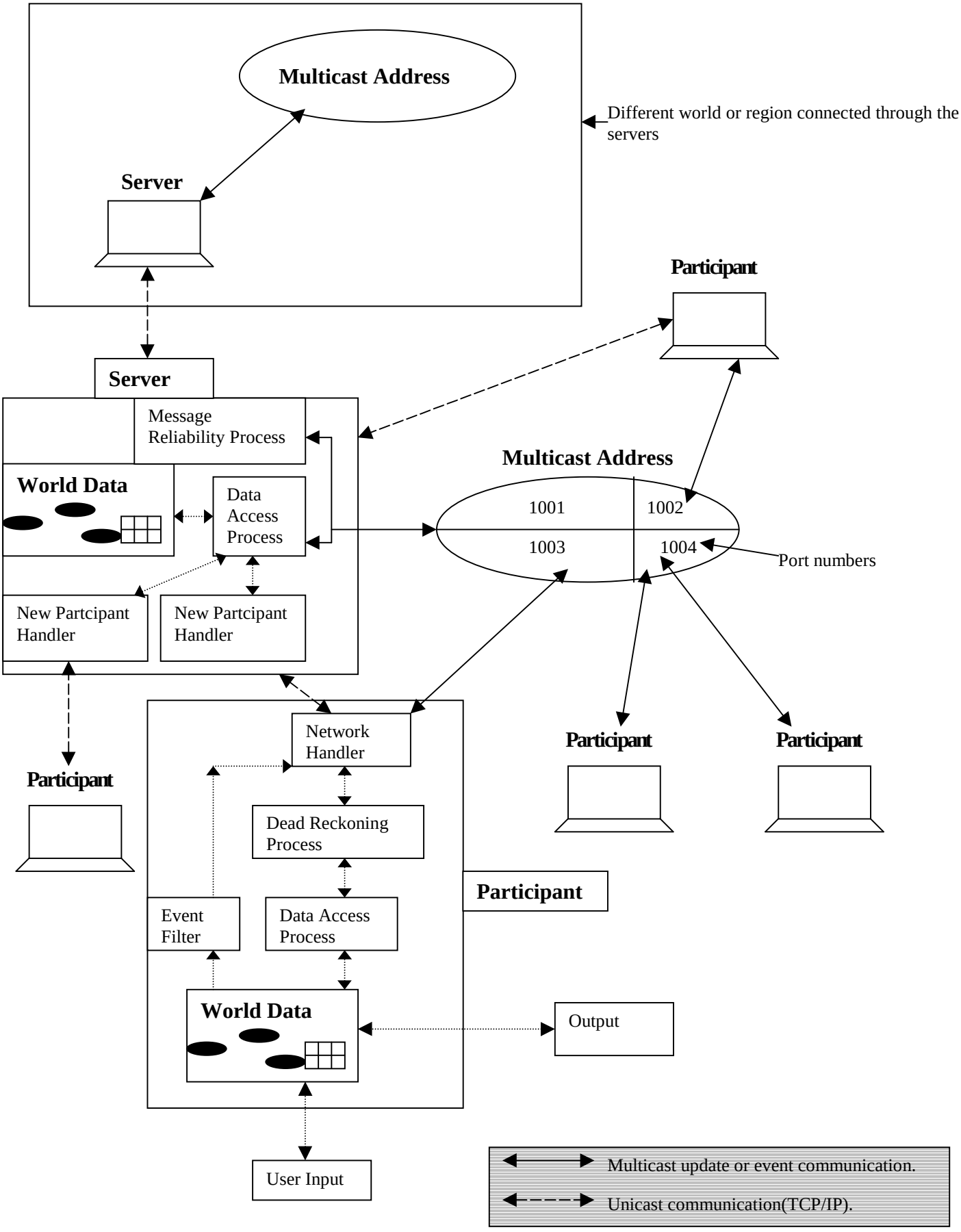


**Figure 22 – Components of a scalable architecture**

Figure 22 is a conceptual representation of the structure, components and workings of the suggested architecture.

A number of participants are shown which communicate with each other and the server through the multicast address. The participants are given particular port numbers depending on the region they are occupying and the extent of their influence. The server can listen to all the port numbers within its particular virtual world multicast address.

A second virtual world or region is shown. Both virtual worlds are connected through the servers. That is, participants can easily travel from the one to the other. Their server will recognize when they wish to do so and send them the address of the other server, which will handle their connection to its virtual world.

New participants are connected to the virtual world by communicating with a new participant handler process, using TCP/IP unicast links. Note that the participant handler process could exist on a separate machine, thus, distributing the load of handling new participants.

A message reliability process also exists on the server to implement a certain degree of reliability into the multicast communications. It too, could exist on a separate machine. That is, a separate machine could be completely devoted to ensuring a level of reliability in the system.

This ability to define components of the server in terms of modules could be very advantages in spreading the servers processing load over a number of machines. All changes to the world data takes place through a data access process, which could implement the required levels of security.

On the participants' side, components exist to filter events which may not need to be communicated, such as the implicitly synchronized events previously mentioned. The dead reckoning process is also shown. It can make changes to the world data based on its predictions. It also observes local objects whose movement is being predicted on the other participants' machine. If there is a difference between the objects real position and its predicted position, then an update is sent.

The following section will deal with the suitability of such an architecture in achieving the required objectives.

## 5.3 Objectives achieved

In my introduction I outlined the objectives that the architecture would need to address. In this section I will repeat those objectives and comment on the suitability of this architecture for achieving those objectives.

- Large numbers of users should be allowed to simultaneously occupy the shared worlds.

The use of IP multicasting and the ability to limit the number of event messages through the use of mechanisms like partitioning of the shared world(messages are limited to an area of influence) and dead reckoning, as well as determining which events will be implicitly synchronized, allows the architecture to scale well.

- Keeping the shared worlds consistent.

Perfect consistency is not achievable by any architecture that wishes to realize the other objectives. However, this architecture will achieve a reasonably high degree of consistency through the use of update or event messages and mechanisms which achieve a level of reliability in their communication, as well as mechanisms which deal with other causes of inconsistencies, like contention resolution.

- Changes to the shared worlds must persist through extended periods of time.

This is achievable by means of the server keeping track of all changes and updating the world description accordingly.

- Minimize latency.

The communication topology and data distribution of the architecture will minimize latency as bottlenecks in the system are unlikely, due to the distribution of the network load to the internet's multicast mechanisms. Also, communication is direct. Events do not need to be processed through some centralized relaying device. This is also appropriate for the following objective.

- Heterogeneous internet devices should be allowed.

Since there are no "links in a chain" a slow participant' s device cannot individually slow down the entire system.

- Allow for seamless navigation within the virtual environment and partitioning of large-scale worlds.

The mechanisms of portals and regions solve this objective. Large-scale worlds can be sub-divided into regions, which can be downloaded as needed from the current server or other servers. Portals allow users to jump from one region to the next. Regions can also be seamlessly connected as previously described.

- Contention resolution(i.e., disputes when grabbing or moving an object).

The mechanisms of single and multiple locking where described to achieve this objective. However, due to the unsynchronized nature of the system, these mechanisms must be weak, although, hopefully sufficient for our needs.

- Collision detection inconsistencies due to latency must be solved.

Collision detection is another issue that is complicated by an unsynchronized system. But, as mentioned in the appropriate chapter, collision detection inconsistencies can be dealt with by assigning objects different rights in deciding who hit what, when inconsistencies occur.

- Creating a generic architecture that can be applied to various virtual world needs.

The architecture allows consistency to be traded for interactivity. For example, the degree to which predictions are made using dead reckoning, can be adjusted from a level of complete accuracy(updates are always sent) to a very low level of accuracy.

In conclusion, some of the objectives make achieving the others more difficult. The architecture cannot be synchronized at all times if any degree of interactivity is required. However, this is to the detriment of the shared world's data consistency. The situation arises, therefore, where a tradeoff between consistency and interactivity must take place. Although, through the use of the mechanisms described, high degrees of interactivity can be achieved while still resulting in acceptable levels of consistency.

# 6.Interfacing with VRML

## 6.1 VRML – Background

VRML(Virtual Reality Modelling Language) is now considered the standard language for describing virtual worlds for transmission over the internet. Not only is it capable of describing the appearance of the world(in terms of its geometry, textures, etc…), but it can also be used to describe the behaviour of objects in the virtual world. This is done by associating scripts with an object. The scripts can describe how the object behaves in response to certain events. Scripts are written in a scripting language like Javascript.

Event routes can also be defined in terms of source and destinations for the events. That is, an event can be defined to be the output from a particular object and the input to another.

VRML has evolved from describing worlds purely in terms of their appearance to incorporating complex behaviours and interactivity with the user navigating the world. However, the final major advancement has not yet taken place. This is the objective of allowing multiple users to navigate these worlds simultaneously and interact with each other and their environment. In terms of the VRML 2.0 this could be realized in terms of complex scripts, which would communicate updates and perform other required tasks. However, this would not be an efficient and generalized solution, as an underlying architecture is needed to effectively achieve objectives like the ones I have suggested.

What is needed, then, is a means of interfacing VRML with an architecture like the one that I have suggested.

## 6.2 The multi-user interface

The Living Worlds initiative[Living Worlds 1997] has suggested interfacing shared objects to the shared virtual world by making use of a multi-user extension to VRML worlds. These extensions are based on prototyping and scripts. They can be vendor dependent as no specific network protocol is defined. The vendor supplies their own multi-user technology(MuTech) which defines how network communication takes place between participants and their server.

It could, thus, be possible to interface VRML worlds to the architecture suggested in this thesis by making use of such multi-user extensions. My criticism of the MuTech is that no network protocol is defined. This may be acceptable for initial experiments, but in order for virtual worlds to be easily and seamlessly navigated a network protocol will be required, such as the suggested VRTP(Virtual Reality Transfer Protocol). Without such standardization a situation will occur where it is necessary to have a particular browser application in order to navigate a particular type of virtual world.

However, how could VRML worlds be interfaced to the architecture suggested here by using a multi-user interface?
Note that my definition of the multi-user interface is not the same as Living Worlds. It can perhaps be considered a simplified version for demonstration purposes.

Essentially, the multi-user interface would be responsible for observing the events that are generated in the VRML world(i.e., by the user and scripts). It could incorporate all the mechanisms that I have previously described, such as dead reckoning, in determining whether the events need to be transmitted. It will also be the interface between incoming events and the VRML world.

This is demonstrated in the following diagram.

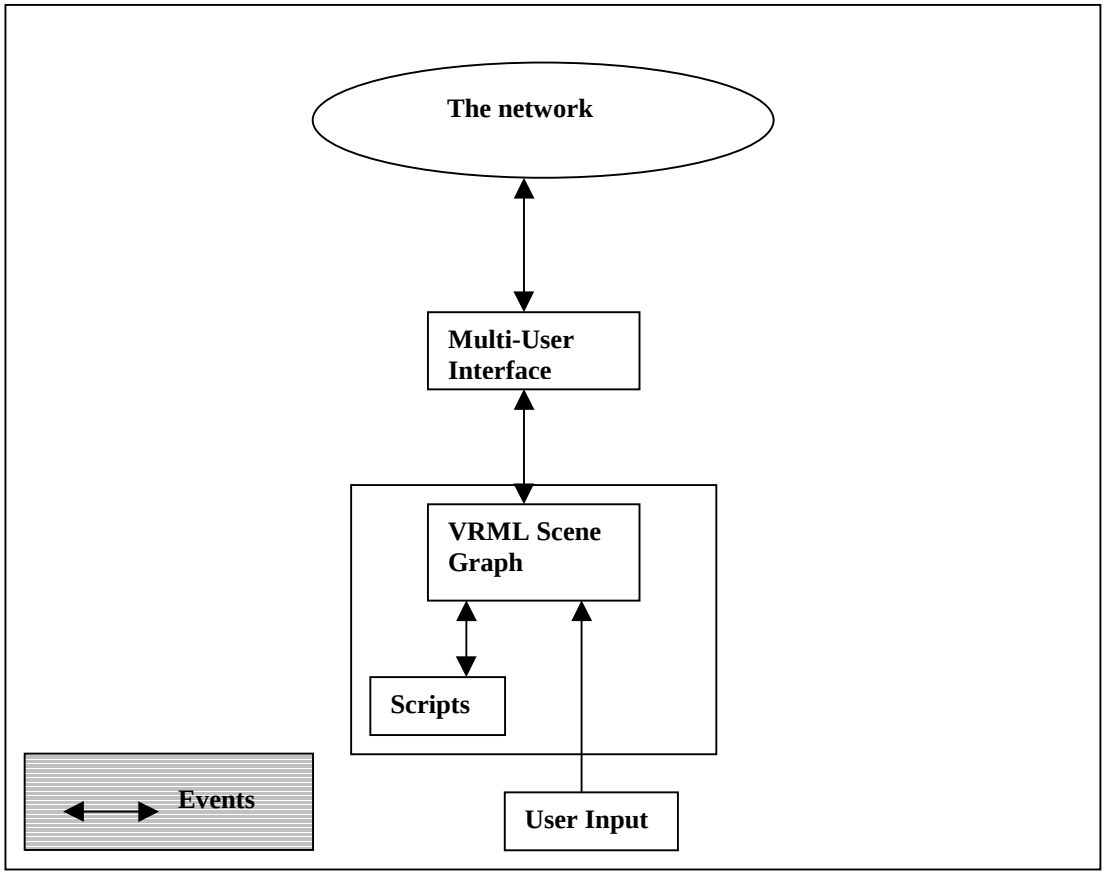


**Figure 23 – Using a multi-user interface to broadcast VRML events.**

Note that VRML events and the events previously described in the suggested architecture are very similar. As such, the architecture is not at all inconsistent with VRML and should be readily adaptable.

# 7.Conclusion

I have presented an extensive investigation into achieving large-scale, widely distributed virtual environments on the internet. Objectives were stated that such an architecture would be required to meet.

These were:

- Allow for possibly a large number of users simultaneously occupying the shared worlds.
- Keeping the shared worlds consistent.
- Allow for worlds that persist through time.
- Minimizing latency.
- Allow for heterogeneous internet devices.
- Allow for seamless navigation within the virtual environment and partitioning of large-scale worlds.
- Contention resolution.
- Solve collision detection inconsistencies due to latency.
- Create a generic architecture that can be applied to various virtual world needs.

It was shown that an architecture with a distributed communication topology, made possible through IP multicasting, and a replicated database would be the most suitable option to achieve the objectives of:

- Allowing large numbers of users who may be widely-distributed.
- Minimizing latency.
- Allow for heterogeneous devices.

Following this, mechanisms were described which dealt with keeping the shared world's data consistent while achieving the other objectives.

Mechanisms were investigated which limit the number of update or event messages. It was shown that dead reckoning can dramatically reduce the network's load.

Methods of achieving a certain level of reliability in communicating messages with an inherently unreliable protocol(IP multicasting), were found.

Contention resolution and collision detection were investigated. It was found that perfect solutions are not possible with an unsynchronized architecture(only an unsynchronized architecture will be viable in solving most of the objectives). However, sufficient solutions for the general needs of the architecture(that is, focusing on interactivity while still producing results approaching those expected from a synchronized architecture) were found.

It was shown that events may only have a limited area of influence. They need only be communicated to the regions of the world that they are likely to have an effect on. Partitioning of the world in this way can, thus, also greatly reduce the network load. It was shown how the server could be used to keep track of changes to the virtual world, thereby ensuring the persistence of the data.

An architecture was then presented which makes use of the mechanisms previously described and was shown to be an acceptable solution for the stated objectives.

## References

[Bangay et al 1996] Bangay, S., Gain, J., Watkins, G. and Watkins, K., *RhoVeR: Building the Second Generation of Parallel/Distributed Virtual Reality Systems*, Department of Computer Science, Rhodes University, May 1996.

[Bangay 1996] Bangay, S., *Modelling Parallel and Distributed Virtual Reality Systems for Performance Analysis and Comparison*, Phd Thesis, Department of Computer Science, Rhodes University, November 1996.

[Broll 1997a] Broll, W., *Populating the Internet: Supporting Multiple Users and Shared Applications with VRML*, Proceedings of the VRML '97 Symposium, Monterey, California, February 1997.

[Broll 1997b] Broll, W., *Distributed Virtual Reality for Everyone – a Framework for Networked VR on the Internet*, Proceedings of the IEEE Virtual Reality Annual International Symposium (VRAIS '97), IEEE Computer Society Press, 1997.

[Casner et al 1994] Casner, S., Kristol, D.M. and Schulzrinne, H., *Frequently Asked Questions(FAQ) on the Multicast Backbone(MBONE)*, http://www.mediadesign.co.at/newmedia/more/mbone-faq.html, April 1994.

[Gossweiler et al 1994] Gossweiler, R., Laferriere, R.J., Keller, M.L. and Pausch, R., *An Introductory Tutorial for Developing Multi-User Virtual Environments*, Computer Science Department, University of Virginia, Charlottesville, 1994.

[IEEE 1993] Institute of Electrical and Electronics Engineers, International Standard, ANSI/IEEE Std 1278-1993, Standard for Information Technology, *Protocols for Distributed Interactive Simulation*, March 1993.

[Living Worlds 1997] Living Worlds, *Concepts and Context*, http://www.livingworlds.com/draft_2/lw_ideas.html, Febuary 1997.

[Macedonia 1995] Macedonia, M.R., *A Network Software Architecture for Large Scale Virtual Environments,* Phd Thesis, Naval Postgraduate School, June 1995.

[Macedonia et al 1995] Macedonia, M.R., Zyda, M.J., Pratt, D.R., Brutzman, D.P. and Barham, P.T., *Exploiting Reality with Multicast Groups: A Network Architecture for Large-Scale Virtual Environments,* Proceedings of the IEEE Virtual Reality International Symposium '95, North Carolina, March 1995.

[Macedonia et al 1995] Macedonia, M.R. and Zyda, M.J., *A Taxonomy for Networked Virtual Environments*, Naval Postgraduate School, Monterey, California, 1995.

[VRML spec 2.0] *The Moving Worlds VRML 2.0 Specification,* Draft #2b, http://vrml.sgi.com/moving-worlds/spec/concepts.html, June 1996.

[VRML ISO/DIS 1997] *The Virtual Reality Modeling Language,* ISO/DIS 14772-1, April 1997.