2. Design Issues for Distributed Virtual Environment Systems

Chapter 2

Design Issues for Distributed Virtual Environment Systems

"640K ought to be enough for anybody."

Bill Gates

When looking at existing systems we are interested in their solutions to two problems: how they tackle the problem of VE modeling and how they "execute" a given VE. The former is a much more abstract area and in theory may be independent of the underlying mechanism of distribution and VE support software. However, in reality this is rarely the case. Sometimes the implementation drives the modeling system used and vice-versa. Whilst treating these aspects separately is desirable, it is also very difficult since describing one aspect cannot be done without referral to features of the other. This chapter examines the issues that must be addressed when designing a VE system. Existing distributed simulation systems solutions are analysed with reference to the outlined issues and comparisons are drawn.

The term "distributed simulation" is very general and is open to many interpretations. "Simulations" can be broadly classified as either off-line/computationally intensive or interactive/low fidelity. The first class is the type of simulation that is often called discrete event simulation whereas driving and flight simulators would fall into the second class. A similar decomposition of "distribution" may also be attempted. It can be used to describe a simulation that is distributed over a number of tightly-coupled computational nodes with the intention to speed up the calculations. This fits well with the first class of simulation and when considering VR and interactive simulations this definition is also valid. However, the emphasis is more on the distribution of the simulation over some geographical distance such that multiple people may interact. Each of these types have their own requirements and hence their solutions cannot necessarily be applied to each other's problems. For example, the fact that there is a human being interacting with the simulation brings onboard a number of new requirements or, more realistically, constraints on how the simulation may behave.

2.1 Discrete Event Simulation Heritage

Before re-inventing the wheel it is beneficial to look at the historically largest form of simulation: discrete event simulation. There are two approaches for ensuring the correctness of a distributed simulation[1]: *optimistic* and *conservative*.

2.1.1 Optimistic versus Conservative

Initially, all simulations used a conservative solution to control their progression. Each simulation consists of a certain number of processes. Only when all processes have completed their work will simulation time increase and the next cycle commence. The obvious disadvantage to this approach is that those processes that take significantly less time to complete their work will be forced to wait. If each process was allocated to a physical processor then this would result in a considerable waste of the computational resources.

To overcome this weakness a different approach was sought. The optimistic solution permits each process to progress at their own rate. This would work fine if all processes were independent of one another. Unfortunately this is often not true and a situation may arise where a slow process communicates with a faster process indicating that their previous work was in error. Since all of the fast process' subsequent work was based on an invalid state, this must be abandoned and recalculated using the correct state. The method to restore this state is known as *rollback*. This solution is called optimistic because it works on the assumption that the situations requiring rollbacks rarely occur.

2.1.2 Time Warp

Time Warp (TW) is an optimistic policy simulation model that is structured as a number of processes that each maintain a Local Virtual Time (LVT) (Jefferson and Sowizral, 1985). Each process may progress at its own rate, advancing LVT as necessary. Each message that is sent between processes indicates the LVT of the sender and is used to decide whether a rollback is required of the receiver. Keeping a list of what has happened in the past soon eats into the resources of each process, so a mechanism for collecting old data has been provided.

At periodic intervals, the operating system interrogates each process for their LVT. Then the system's Global Virtual Time (GVT) is updated to show the progression of the simulation, taking into consideration the slowest process. When GVT is updated, any data previous to this time may be discarded since rollback may not occur before GVT. The choice of algorithm to calculate GVT is crucial to system performance and can make the difference between running or not running a simulation if large state lists are required (Bellenot, 1990; D'Souza *et al.*, 1994).

Further optimisation may be made by finding a way to reduce the amount of state saved in these lists. A basic mechanism would save the complete process state, however this is expensive both in terms of time taken to save the state and the time taken to perform a rollback. By performing incremental state saving (Cleary *et al.*, 1994), i.e. only saving the

[1] In discrete event simulation, distribution is almost always used just to increase the simulation's speed.

state that has changed, it is possible to improve efficiency. An alternative approach, called adaptive checkpointing, is to adjust the rate at which the process state is saved based upon the rollback behaviour (Rönngren and Ayani, 1994).

2.1.3 Discrete Event Simulation Summary

TW works well in discrete-event simulations and is a very popular model, but there are a number of problems. TW was not designed to be used for interactive applications which rely on completing all computations in a very small amount of time (~33 ms to achieve a 30 Hz update rate). In order to ensure that these strict deadlines are met, some notion of *predictability* must be provided. Rollback is a result of processes being allowed to continue at their own rate and can be seen as self defeating since the rate of progress is not controlled and the occurrences of rollbacks are *unpredictable*. In addition, one rollback may trigger another rollback in another process and so on until, potentially, each process has been rolled back to GVT.

However, there has been some recent work on the application of TW to real-time simulations resulting in the development of a Parallel Optimistic Real-Time Simulator (PORTS - Ghosh *et al.*, 1994). In PORTS, GVT is calculated continuously, i.e. after each event in the simulation, in order to speed the commitment of I/O operations. Incremental state saving is shown to be unpredictable and one proposed solution is to save the complete state every *n* events (where *n* is a constant for a particular simulation) in a similar way to adaptive checkpointing. This enables a bounded value for state saving and state restoration, thereby having predictable properties in the simulator. Deadline scheduling is also simplified because there is no event-migration or explicit load balancing and is done on a per-processor basis.

Despite this encouraging work, the application of PORTS to *interactive* simulations is unlikely. Take the case of a driving simulator where the driver is monitoring the environment and taking actions accordingly. Any rollbacks could interrupt the flow of time and would make it seem as though they are being controlled like a video recorder - pause, rewind, fast-forward and play - clearly defeating the goal of realism. In short, you cannot rollback a human being.

Conservative solutions ensure that situations that would require a rollback do not happen at all by, what proponents of optimistic policy would see as, restricting the progress of the simulation. This has the potential to under-utilise the available resources, but with a good load-balancing algorithm the impact of such an approach can be reduced. The perceived advantage of an optimistic mechanism is that if a process requires very little interaction with other processes in the simulation, faster progression may be made if it is allowed to go at its own rate (Lipton and Mizell, 1990). This may also be perceived as a waste of resources that may be better allocated to other processes in the simulation.

Therefore, since there may be many humans interacting with the simulations of VEs, a conservative system is the only workable solution. This will also aid predictability and scheduling to meet the real-time deadlines that are required of a VE system (discussed in section 3.3).

2.2 Issues

There are many problem areas to consider when building a VE system and there are even more implications. There is no established classification scheme available with which these areas can be examined and different solutions compared, so an attempt has been made to construct one. Separating one area from another was more difficult in some cases than others. Not breaking a problem area down into separate issues would make comparison difficult, on the other hand, splitting the area into too many issues would provide a distorted representation. There are a lot of interdependencies between these issues, but it is hoped that the divisions made will aid comparison rather than hinder comprehension. This section looks at each issue in turn and assesses the impact they have on system design.

2.2.1 Real-time

The largest single constraint on an *interactive* simulation is that it must operate in *real-time*. As described in section 1.4, a real-time system permits the generation of real-time displays which are updated fast enough to allow the participant to effectively interact with the simulation and other participants. *How* fast may vary depending upon the exact nature of the simulation, but the goal is to reduce the delays between human action and simulation reaction to an imperceptible constant duration.

A simulation is composed of a sequence of discrete time steps in between which the calculations to update the environment must be completed. Failure to achieve this could result in a breakdown of realism (if that is being striven for) or, at the very least, a reduction in the efficiency of the participant to interact with the simulation. While it is true that simulation time may continue at any rate if there is no human or time-dependent device involved in the loop, we are primarily interested in interactive simulations and therefore the actual time between each simulation time step must be *constant*. We live in a constant world and to require us to interact with anything other than this is contrary to all our natural skills and will present us with corresponding difficulties (Hawkes *et al.*, 1995). This is discussed further in section 3.3.

If these stringent deadlines are to be met then there must be a degree of predictability in the simulation's execution. An optimistic solution, as discussed earlier, is not very predictable whilst a conservative approach may be seen as a good basis to build upon. The design implications of real-time systems are discussed in section 4.3.

2.2.2 Communications

The structure of the communications subsystem is usually the most inflexible component of any system. The choice of platform and its location dictate what communications hardware is available. Consequently, the technique used to manage data is often directly influenced by this component.

2.2.2.1 Point-to-point

A point-to-point[2] transfer of information may be achieved by either establishing a link between sender and receiver at every transmission, or creating a permanent connection which is destroyed when there will be no more communications. Connection-oriented protocols such as Transmission Control Protocol/Internet Protocol (TCP/IP) are commonly used and provide a reliable service. Unfortunately, ensuring that the receiver gets all the information and in the right order generates a fair amount of overhead. Furthermore, each receiver must acknowledge receipt of the transmission.

2.2.2.2 Broadcast

One alternative is to "broadcast" the information on the network and hope that anyone interested in that information will hear the broadcast and pick it up. This is the exact opposite of the point-to-point mechanism and is supported in the User Datagram Protocol (UDP). This connectionless protocol uses self contained, addressed packets (or datagrams) which puts the onus on the application to ensure that the data is processed in the correct order. The major advantage of this method is that there is no need to maintain a large number of connections. Apart from being unreliable, its main disadvantage is that it is possible to flood a network with broadcast messages which are of no interest to other connected systems and thus degrade performance.

2.2.2.3 Multicast

An improvement on broadcasting is multicasting. This works in the same way except that the packets are only sent to a subset of the network rather than the whole. Nodes may belong to one or more multicast groups and hence will only receive transmissions that are intended for them. It was originally available on LANs such as Ethernet and Fibre Distributed Data Interface (FDDI) but is now available at the network layer through the Multicast Backbone (MBONE - Macedonia *et al.*, 1994). MBONE is a virtual network which runs on the same physical media as the Internet, but encapsulates multicast packets in normal IP packets and uses routers to forward them to their correct destinations. Multicast has yet to be standardised and consequently few implementations are available. More importantly, multicast *per se* is unreliable, although some research has been done on providing a reliable multicast service (Talpede and Ammar, 1995; Veríssimo and Marques, 1990). However, unless otherwise stated, any reference to multicast in this thesis is intended to describe the more common unreliable mechanism.

2.2.2.4 Bandwidth

The amount of data that may be transmitted in a given period of time has more impact on system design than any of these other factors. If only one network medium is being used then the task of designing an efficient protocol is relatively straightforward (but not simple). However, if multiple mediums are being catered for the problem becomes considerably more complex. A fast modem can manage approximately 28 Kbps, Ethernet has a bandwidth of 10

[2] Also known as unicast.

Mbps whilst FDDI and Fast Ethernet can offer 100 Mbps. It is quite common for this bandwidth to be shared amongst many other nodes thus reducing the effective data bandwidth considerably. There is also no way to *guarantee* a fraction of this bandwidth which adds to the problems. The evolving Asynchronous Transfer Mode (ATM) technology permits bandwidth to be reserved (channels), but this is currently even less available than multicast technology (Boisseau *et al.*, 1995).

2.2.2.5 Latency

Communications latency is related to bandwidth and geographical distance. No matter what technological improvements are made, the speed of light will limit the transmission speed such that a latency of ~3 ms will be introduced for every 900 Km covered[3]. Thus design decisions are often based on the geographical distance over which the system will have to operate.

2.2.2.6 Shared Memory

This is a valid way of communicating between processes on the same node and the analogy can even be extended to operate over networks: distributed shared memory. However, underlying such functionality is always some form of message passing. Bandwidth and latency can still be applied to shared memory. Whereas a message-passing system has built-in concurrency control, a shared memory system must add this itself, usually in the form of semaphores.

2.2.2.7 Structure

There are three commonly used models for communication in distributed VR systems: client/server, peer and hierarchical. In a client/server model one or more physical processes are designated as a server whose responsibility is to receive and process requests from clients for any of its published services. A client of one process can also be a server to another. This model works well for operating system resources, e.g. the filing system, network manager and process manager, where there is a limited number of potential clients and the client and server are tightly-coupled. If the number of clients gets too high, however, the server soon becomes a bottleneck.

The peer model essentially makes every process in the system equal in terms of functionality. This does not mean that there is any duplication of work between peers although this is quite common.

The hierarchical model uses a system whereby processes communicate with other processes in the hierarchy by sending the message to their parent process. The parent checks the address on the message and either sends it to one of its other children or to *its* parent process. This repeats until the message has arrived at its destination. Messages entering the hierarchy from outside are sent to the root (master) process which forwards the message as per normal. As

[3] This calculation does not take into consideration the extra distance incurred as the light bounces off the interior of the optical fibre.

with the client/server model, this master process may become a bottleneck if the number of child processes increases too far, or there is a large amount of communication with other process hierarchies.

2.2.3 Data Management

If the whole VE was managed by one machine then data management is straightforward, every process has direct access to the information they need with little overhead. If the VE is distributed across more than one machine then the situation becomes more complex and requires a different solution. The overriding concern is to ensure that the integrity of the data is maintained at all times with minimal overhead. Other factors that affect solution selection are bandwidth and fault tolerance.

The nature of the target system and the geographical dispersion of the network dictates the type of management commonly used. All of the solutions currently offered fall within one of the categories shown in Table 2.1. Although general comparisons can be made between them, only those systems in the same category can be compared point for point.

2.2.3.1 Localisation

When the amount of data is small it is preferable that every process should have direct access to it. As the volume of data increases so does the burden on resources; memory and backing storage diminish rapidly and the amount of computation required to process the data rises dramatically. In a distributed system there is also an increase in network traffic as the data is

	Tightly-Coupled	**Loosely-Coupled**
Near	Parallel Processing *High Speed LAN*	Distributed Processing *LAN*
Far	Impossible?	Distributed Access *WAN*

LAN: Local Area Network
WAN: Wide Area Network

Table 2.1 Kleinrock distribution classification scheme.

moved around from one node to another.

It is therefore desirable to segment the data in some logical way such that any given process is only interested in one segment at a time (mostly). One common criteria used for segmentation is that of space. When the VE covers a large (virtual) distance it is broken up into a number of areas which are often allocated by and under the control of an Area Manager. The size of the areas can depend on many things, such as visibility, memory, speed of movement through the VE, etc., and upon the media, e.g. visual, aural, etc. The shape of each area is often kept uniform for simplicity's sake. Rectangular areas are often favoured although some work has been done with hexagonal areas (Macedonia *et al.*, 1995). However, some research has

examined the subdivision of model space based on visibility alone (Airey *et al.*, 1990). When applied to architectural models, the resulting binary space subdivision algorithm creates *cells* which are bounded by a number of splitting planes and can therefore be irregularly shaped.

2.2.3.2 Complete Distribution

This approach distributes the complete VE state between every node in the network. There is no duplication of information and any intention to change part of the VE state not under the control of a given process must be communicated to the process managing that data. Unless the state of the VE is distributed amongst all the nodes in the network sensibly, it is possible that such an arrangement could be detrimental to performance.

Since a given piece of data is only held in one place this solution is susceptible to machine failure or breaks in the communication paths. Such a technique can be applied at the near/tightly-coupled level and, perhaps, at the near/loosely-coupled level.

2.2.3.3 Partial Replication

When using partial replication only the parts of the VE state that will be modified by a given process will be held locally and only when needed. There are two sub-categories of partial replication: *active* and *passive*.

Active replication is where the process wishing to make the state change initiates the request for a local copy of the state. Modifications are made locally and the updated state is sent back to the originator.

Passive, or demand replication (Broll, 1995), requires an initial registration of interest in part (or all) of the VE state when the process is created. From that moment on it is sent copies of that subset of state when it has been modified by one of the other processes. Changes may be made locally and sent back or, alternatively, the owner is informed of the desired changes and then makes them itself. A variation on this method is that the remote process receives updates of the object's exported behavioural model (section 2.2.4.5). In which case the remote process is not expected to want to modify the object's state, just monitor it.

If changes are made locally it would be possible for multiple copies to be taken from multiple processes, altered and submitted simultaneously, therefore resulting in an inconsistent state. To prevent this from happening a system of read/write locks may be employed. Before obtaining a local copy of the state for modification, a write lock is requested. This will be granted once any outstanding write locks are relinquished. Either the requester must block until the lock is granted, or a time-out can be specified which will permit the requester to continue with other work. On submitting the changes the modifications are made and the write lock released. If multiple locks need to be acquired before proceeding then the problem of deadlock also arises. There are several variations on this approach but all are equally complex. However, if all changes are made by the owner there is no need for this complex system and the modification process is a lot more predictable. This technique is most commonly used at the near/loosely-coupled level, although it could be applied at the far/loosely-coupled level if bandwidth was high enough.

2.2.3.4 Total Replication

This solution requires the complete VE state (or the essence of it) to be held at *each* node in the network. The two possible reasons for storing the complete VE state are, firstly, that the node's calculations are based upon most or all of that information or, secondly, the distance between nodes is so great that latency has become a real problem (far/loosely-coupled). This method does not scale well since every node must keep each other informed of updates which soon consumes bandwidth. The allocation of locks is infeasible so passive replication must be used to receive continuous updates on each other's local VE state modifications.

2.2.4 Computation Management

Just as data is distributed, so can the computation. By computation we mean any work involving a specific object, whether it is an operation within or upon that object. Fortunately we can use similar categories to explore the options.

2.2.4.1 Complete Distribution

All operations on an object are performed on the same node that holds the object's data. If one process wishes to perform an operation on another then it must send a message to the other process. The allocation of processes to nodes may be optimised by enlisting the help of a load-balancing algorithm (section 4.3.1.4). By monitoring resource consumption and communication patterns the optimum allocation may be derived. This would permit most objects that often communicate with one another to be located on the same node - the movement of processes is commonly known as *migration*.

Such an approach works well on near/tightly-coupled systems but the latency and low bandwidth found in loosely-coupled systems can reduce its efficiency.

2.2.4.2 Partial Distribution

This method is similar to complete distribution except that the object's state is usually acquired using one of the passive or active partial data replication techniques and the changes made locally. There is, however, no duplication of computational effort.

2.2.4.3 Partial Replication

To compensate for slower communications links, e.g. near/loosely-coupled, it is possible to replicate some of the state computation on some or all of the nodes. These "ghost" or proxy processes are typically used to approximate the object's behaviour using a method called *dead-reckoning* (section 2.2.4.5). The process that performs the full simulation of that object also runs this model in parallel and when the two differ by a pre-defined amount, a copy of the real object's state variables is sent to all of the ghost processes. Subsequent approximations are then based on the latest update.

Dead-reckoning uses a simplified model of the object's behaviour. Typical key state variables used in this model are position and velocity which may be linearly extrapolated to provide a

low fidelity approximation. Higher fidelity may be achieved by incorporating other variables, such as linear acceleration and angular velocity, which are often needed by objects with highly dynamic behaviour, e.g. aircraft (Harvey *et al.*, 1991; Le Saché and de Medeuil, 1993; McCarty *et al.*, 1994).

This technique is very effective in reducing the amount of bandwidth required but the object behaviour produced in the ghost object can be sufficiently different from the normal to attract attention. This may, of course, be improved by increasing the complexity of the approximation, but there is a need to strike a careful balance between full and approximate simulation.

2.2.4.4 Total Replication

Simulating each object on each node may be required if the simulation is running over a very large distance (far/loosely-coupled). Receiving periodic updates from other processes when using partial replication is not practical when bandwidth is at a premium. Instead, only information that changes the behaviour of the mirrored objects is sent, thus permitting all calculations to be performed locally. Behaviour therefore appears correct everywhere (although maybe not at exactly the same moment in time) but at the cost of duplicated calculations.

2.2.4.5 Behaviour

What constitutes object "behaviour" and what form this takes is currently a topic of debate. In the strict object-oriented sense the data are the attributes, and the methods manipulate the attributes in a pre-defined way, e.g. modifying position over time. Therefore combining data and methods gives us the impression of behaviour. However, the computational load required to support this object behaviour can be quite high, e.g. flight dynamics for an aircraft.

It is possible to classify object behaviour as either *deterministic* or *non-deterministic*. In general, objects that do not sample input devices are deterministic, whilst those objects that do, including those under the control of humans, are non-deterministic. For example, the decisions made by a robot car can be determined in advance whereas the behaviour of a virtual car being driven by a human in a driving simulator cannot be predicted (Hawkes, 1993). The ability to predict behaviour means that it is possible to overcome communication and system latency.

Roehl (1995) has suggested a refined classification scheme whereby deterministic behaviour is split into two sub-categories: *static* and *animated*. Similarly, non-deterministic behaviour can be *Newtonian* or *intelligent*. The state of a static object is constant and therefore 100% predictable for any give time; an animated object changes state over time but this is still predictable. A Newtonian object interacts with its environment but does so in a straightforward manner, whilst an intelligent object can have a complex behaviour and may be as unpredictable as a human.

In a similar manner, Roehl presents 4 levels of behaviour which may be used to classify the type of distribution used:

0. Direct modification of an object's attributes (static).

1. Change in an object's attributes over time (animated).
2. Series of calls to level 1 behaviours to achieve a task (Newtonian).
3. Top-level decision making (intelligent).

The most basic form of behaviour distribution takes place between levels 0 and 1, when information such as position and orientation are transmitted at every simulation update. Dead-reckoning falls between level 1 and 2. Attempting to distribute behaviour any higher is problematic unless the state of the simulation at each node is guaranteed to be exactly the same at any given time. Indeed, levels 2 and 3 may not even be implemented in software, they could be provided by human interaction.

An example of a level 2 behaviour system is the *Two-Point Paradigm* (Bryson, 1991). It is based on interaction in classical physics which may be taken as due to the forces that act pair-wise between physical objects. While many forces may act on objects simultaneously, the net action of these forces may be represented as the sum of the individual forces on that object from the other objects. To keep track of all these interactions an *Interaction Matrix* is used whereby each row and column represents an object and the entries are lists of interactions between the objects for that row and column. For example, Figure 2.1 shows the simple case of a bouncing ball. The ball is acted on by the floor in two ways: gravity pulling it down and bouncing which reverses the z component of the velocity. The floor is not acted upon by the ball. The ball's cross-reference entry (bottom right) updates its velocity from its acceleration and its position from its velocity.

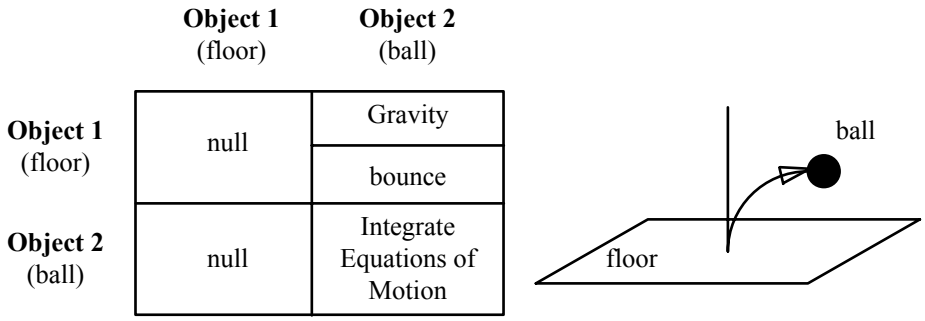

Figure 2.1 Example interaction matrix.

This technique can be extended to include other types of interaction including those with the user. With regards to distribution, it is only necessary to send changes in the interactions between objects and details of any new objects from one node to another. Each node can then calculate the evolution of the VE on its own, which reduces the network bandwidth required per object.

2.2.5 VE Modeling

The issue of modeling the VE will be fully discussed in the next chapter, however there are two aspects which can be usefully addressed beforehand. Firstly, whether the system can support more than one VE simultaneously and/or how multiple VEs are structured. Secondly, if any special provisions are made for users or participants in the VE.

2.2.5.1 Multiple VEs

Support for multiple VEs means that the system is effectively running parallel simulations using the same or different VE model. By using the same VE it could be possible to maximise the use of specific objects or areas of the VE (Roehl, 1995). For example, a virtual town hall could be used for meetings by different groups of people simultaneously.

If multiple, different VEs are supported then there is an opportunity to maximise the system's resources. Such an ability does, however, raise extra problems regarding scheduling, load balancing, etc. If the concurrent execution of VEs is available then a decision must be made as to whether an object may move from one environment to another and, if so, how this should be achieved.

Another possible use for multiple VEs is in the modeling process, where some or all of the properties of one VE are used to help speed development of another. The nature of the relationship between environments is important, as is the structure formed. One possible organisational technique is that of object-oriented inheritance where the attributes of one environment are inherited and augmented/extended by another environment.

2.2.5.2 Users

Typically, either the user is treated as a separate object or they are an integral part of the system. Also of interest, is whether multiple users can be supported or if only one may be present in a VE at a time.

Representing the user as an object has the advantage that it implicitly means that multiple users are supported, provided that there are enough input and output devices available. Added flexibility is provided if devices are not integrated into the user code directly, but exist as objects in their own right. The price of this object-oriented structure is, of course, performance - the extra communications overhead increases system latency.

Either the user's representation can be described in the same manner as every other object or some extra functionality is provided for just this purpose. The latter case is usually used when the user is integrated into the system or a part thereof.

2.2.6 Time Management

The relationship between simulation time and real clock time may be any function as long as it is constant. The simulation clock is used as the basis for synchronisation of the VE either explicitly or implicitly. Implicit progression is when simulation time is related to real clock time: as the system clock changes, so does simulation time. Explicit progression is change through notification from a remote source, e.g. a message timestamp or a special message that only occurs at the beginning of each time step.

An additional requirement in a distributed system when using implicit progression is to ensure that the real-time clocks on each node are synchronised. One possible option is the use of a Global Positioning System (GPS) receiver built into each node. A version of GPS was developed by the military - Precision Position System (PPS) - for keeping track of friendly forces. It works by sending a signal to 4 out of 21 active satellites which send back

information from which both positional and time information may be extrapolated. A commercial version is available, Standard Positioning System (SPS), with reduced positional accuracy - 100m horizontally instead of 17m, etc. Time accuracy with PPS is 100 nanoseconds (ns) and 167 ns with SPS. Detailed information can be found in Dana (1995).

Alternatively, a software algorithm can be used such as the one presented by Le Saché and de Medeuil (1993) where a client requests the time from a central source. The client synchronises itself on the time at this central source via a couple of timestamped messages. The synchronisation formula is shown in Figure 2.2. After clock synchronisation, delays can be measured as the difference between the send and receipt times of any given message.

$$t = \frac{t2 - t0}{2} + t1$$

t : new time for client
t0: send time of request time message (client clock)
t1: send time of response message (server clock)
t2: reception time of response message (client clock)

Figure 2.2 A Clock Synchronisation Formula

The problem of clock synchronisation is also of interest to the Internet community. The Network Time Protocol (NTP) is an extension of the client/server approach such that it may be applied in very large networks world-wide. For an in-depth description of the protocol the reader should refer to Mills (1992). On the general subject of clock synchronisation, Mills notes that the accuracy achieved is directly dependent on the time taken to achieve it. In other words, a few measurements will suffice for accuracy with a second or so, whilst dozens of measurements over many hours will be required to achieve millisecond accuracy. The number and frequency of these measurements is, however, perceived to be relatively low and unobtrusive to normal network operations.

However, Liskov (1993) notes that clock synchronisation algorithms are based upon assumptions about clock rate and message delay. Clocks are, therefore, only synchronised with some probability, albeit very high. Subsequently, she also states that algorithms should preferably depend on clocks for performance and not for correctness.

2.2.7 Fault Tolerance

Kim (1995) describes a fault tolerant computer system as "*... a system which can continue to operate reliably by producing acceptable outputs in spite of occasional occurrences of component failures, including those of both hardware and software components*". Fault-tolerance comes from *reliability* and *availability* (Milenkovic, 1992). System reliability can be provided by partial replication of important data and duplication of key hardware, whilst the availability of the system is ensured by keeping multiple copies of the system's resources. Furthermore, a system may be deemed *recoverable* if it can revert to a previous state and *robust* if it is capable of surviving a hardware failure. However, one does not imply the other.

Degree	Assumable Damages	Recovery Capabilities
4	No loss of visible actions (i.e. output of actions or database update actions)	Action-level fault tolerance (recovery of an interrupted visible action)
3	Loss of one or more visible actions	Slow recovery of a service function (no loss of hardware)
2	Loss of one or more service functions	Partial recovery of hardware (service degradation)
1	Loss of all but a core set of critical service functions	Minimum recovery of core hardware (minimum critical services)
0	Loss of critical service	No fault tolerance

Table 2.2 Degrees of fault tolerance.

Five degrees of fault-tolerance have been proposed by Kim which are reproduced in Table 2.2. Degree-4 is the highest level of fault tolerance (reliability) and ensures that all actions are completed successfully regardless of fault occurrences. It is possible that recovery from a fault may take so long that there is no choice but to abandon execution of visible action(s), restore the system to a previous state and then start again. Degree-3 caters for this case whilst degree-2 provides service degradation when some less-critical components fail and cannot be recovered. In the worst case, only the minimum critical services can be recovered and maintained which gives us degree-1 fault tolerance. If even this last stand is not possible then it is not a fault tolerant system.

A common way of providing fault tolerance is redundancy which may be applied to both hardware and software components. For the purposes of this thesis we are primarily interested in software components. The availability of broadcast communications on networks has been a great boon to the implementation of redundancy. One solution is to have a node which eavesdrops all inter-node communications to keep an up-to-date copy of *each* node's state. This means that if a node should fail (or a process on that node) then its state may be rebuilt quickly without replaying all the messages.

In distributed real-time systems it is common for the physical network to be duplicated, therefore providing a second physical communication path should the other fail. In such a system there is also a need for deterministic and reliable delivery or messages, which has provoked some researchers into investigating reliable multicast protocols (Grünsteidl and Kopetz, 1991).

2.2.8 Security

Current efforts in this area have typically been limited to the encryption of the data stream between nodes so that no unwanted party can listen in on the simulation. This could be done in software at the communications level or utilise special hardware. The complexity of the system protocol determines the degree to which security can be breached; a simple protocol may even permit unauthorised objects or people to participate in the simulation.

On another level, security also deals with access to sensitive information. Certain system services may need to be restricted, e.g. access to backing storage, or a group of objects may wish to share information with each other and no one else.

Obviously, such additions to the system architecture come at a price. Even data encryption hardware increases latency and a software-implemented access control system can eat away at CPU cycles.

2.2.9 Issues Summary

This section has presented a number of issues that must be addressed during the design process. Some of these are given higher priority than others and as such may not be accounted for in the final design. This is not because they are unimportant, merely because the field is new and the problems presented by the few issues addressed are quite significant. The next section looks at the current major system solutions.

2.3 Implementations

Ensuring a consistent and accurate environment must be the main goal of any human-in-the-loop simulator. Progressing the simulation at a constant rate, fast enough so that the participant may effectively interact with it, is the second goal. It is clear from the overview presented in the previous section that many of the design issues are entwined with each other. Deriving an architecture that correctly resolves each issue is a challenging task. This section examines some of the existing distributed VE systems and describes their overall structure.

2.3.1 SIMulation NETworking System (SIMNET)

SIMNET was the first system to prototype and demonstrate the feasibility of a distributed interactive simulation (Kanarick, 1991). It was initiated by the U.S. Defense Advanced Research Projects Agency (DARPA) and funded by the U.S. Army. This project involved many different companies but, to the author's knowledge, no academic institutions.

Some of the systems requirements were (Calvin *et al.*, 1993):

- Capable of supporting 100s to 100,000s of entities.
- Entities are geographically distributed.
- Simulations are heterogeneous.
- Computations are distributed (no central site).
- Operates in real-time.
- Must be low cost.

In order to meet the last requirement SIMNET was based around an Ethernet network. The maximum bandwidth of Ethernet is 10 Mbps and was one factor in the failure to support the large numbers of entities originally specified. 250 of the original SIMNET simulators (nodes) are currently in operation throughout the world although a node is capable of simulating more than one entity. A good example of this was the provision for *Semi-Automated Forces* (SAFs) which are semi-autonomous objects that have a certain behaviour and are directed from time to time by a human operator. To make the most of the available bandwidth and reduce the computational overhead of point-to-point links between nodes, messages are broadcast to all nodes regardless of whether they require the information or not. The SIMNET protocol is

designed such that if a node should miss a message, it will temporarily hold out-of-date information which will be amended upon the next transmission.

A host processor for a SIMNET node is typically an embedded single-board microprocessor-based system, or a workstation. Usually Local Area Networks (LANs) are used to link nodes within a single site and geographically dispersed sites are linked using Wide Area Networks (WANs). Due to the real-time requirement, the WANs are either private lines or packet networks with gateways that provide real-time allocation abilities. For example, the Defense Simulation Internet (DSI), which spans the U.S.A, is a dedicated network that uses the TCP/IP protocol but is not considered part of the Internet (Locke, 1992). The need to dedicate a network to a simulation indicates the problems of geographically dispersed simulations.

2.3.2 Distributed Interactive Simulation (DIS)

The experiences with SIMNET led to DIS which, unlike SIMNET, is being developed as a standard for networked, interactive simulation by a committee. An important distinction between the two is that SIMNET is a working system, whereas DIS is only a protocol definition with associated guidelines and does not specify how the implementation should be structured. Even though its applications are subject to security, the standard is not; version 1.0 is now a published standard: IEEE 1278.

Version 2.0 of the standard (DIS, 1994) summarises the DIS concept as:

> "*... a time and space coherent synthetic representation of world environments designed for linking the interactive, free play activities of people in operational exercises. The synthetic environment is created through real-time exchange of data units between distributed, computationally autonomous simulation applications in the form of simulations, simulators, and instrumented equipment interconnected through standard computer communicative services. The computational simulation entities may be present in one location or may be distributed geographically*."

2.3.2.1 Basic Architecture

The DIS architecture shows its heritage through its basic concepts:

- No central computer controls the entire simulation exercise.
- Autonomous simulation applications are responsible for maintaining the state of one or more simulation entities.
- A standard protocol is used for communicating "ground truth" data.
- Changes in the state of an entity are communicated by simulation applications.
- Perception of events or other entities is determined by the receiving application.
- Dead-reckoning algorithms are used to reduce communications processing.

When examining the communication services that DIS must provide (as dictated by the standards document), we find that data must be transferred between simulations in one operation, with or without first establishing a logical connection with the destination node.

Data may be sent using broadcast, multicast or point-to-point and, on the issue of unreliable service, no acceptable limit is set on the amount of data that may be lost. As a comment on the performance requirements of the communications architecture we are told that it "*...should provide a certain level of performance characterised in terms of throughput and delay. Both network delay and network delay variance should be minimised*". Another document (DIWG, 1993) states that the total network delay for tightly-coupled simulators, such as high-performance aircraft, should be less than 100 ms and less than 300 ms for other simulators, e.g. ground vehicles.

Each message, or Protocol Data Unit (PDU), has a 32 bit timestamp which specifies the time at which the contents of the PDU is valid as units of time past the current hour. This provides an accuracy of 1.676 microseconds and the timestamps used depend on whether system clocks are synchronised or not. If they are, then the timestamp is given in Universal Coordinated Time (UTC), if not, then the time is relative to the simulation application that issued the PDU.

Each PDU has an exercise identity field in the header which is an unsigned 8 bit number. A unique exercise identifier is assigned to each exercise occurring simultaneously on the same communications medium. In essence, DIS can support up to 255 (a value of 0 is not valid) parallel VEs.

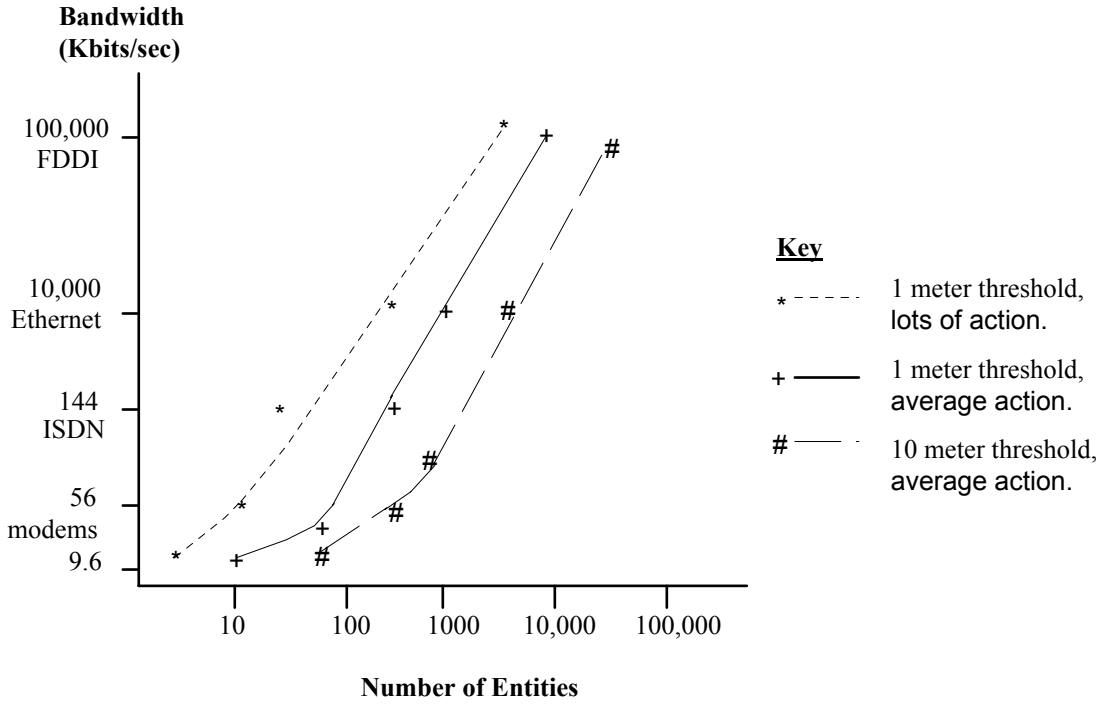

Figure 2.3 DIS performance with different dead-reckoning accuracies.

2.3.2.2 Performance

The total number of entities that may be supported is not only a function of the communications medium but the error thresholds which are an integral part of the dead-reckoning algorithms. Katz (1994) provides us with a graph (Figure 2.3) showing how the number of entities a medium may support can be reduced by decreasing the threshold (and

hence the computational load of the dead-reckoning algorithm) and increased by raising the threshold (which increases computational load). The results shown are part empirical data and part prediction based on those data.

A state-of-the-art DIS system is said to manage 8,000 entities on Ethernet (using a lax error threshold) and that the most expensive dead-reckoning algorithm in use consumes around 100 FLoating-point Operations Per Second (FLOPS) per remote entity. Interestingly, it is predicted that the original SIMNET goal of 100,000 entities will not even be reached using DIS over an FDDI (100 Mbps) network. In fact, the Close Combat Tactical Trainer (CCTT), which is being developed by the U.S. Army and Loral Federal Systems using DIS and FDDI, expects to ultimately handle only 851 entities plus audio communication traffic (Mastaglio and Callahan, 1995). All this assumes, of course, that the simulation node itself has enough computational power to simulate 100,000 entities.

2.3.3 Naval Postgraduate School Networked Vehicle Simulator IV (NPSNET-IV)

NPSNET is a research project in the Computer Science Department of the Naval Postgraduate School. The project's goal is "*... to promote the use, understanding and appreciation of VR*" (NPSNET, 1995). NPSNET utilises SIMNET databases, both SIMNET and DIS networking protocols and has a number of key functional components:

- *Terrain database* defining the 3D surface, e.g. ground or sea, and the various features, e.g. roads.
- *Static models* such as buildings, trees, etc.
- *Dynamic models* such as vehicles, aircraft, etc.
- *Display algorithms* which perform geometrical and rendering calculations on the complete VE from a given viewpoint.
- *Environmental effects* which included smoke, clouds, waves, etc.
- *Heads-Up Display* (HUD), a 2D overlay which may be used for superimposing information on the 3D view of the VE.
- *Networking* component which supports both broadcast and multicast.
- *Input options* allowing the device(s) to be matched to the application.

Despite being DIS-compliant, NPSNET only implements a fraction of the DIS Protocol, namely the Entity State, Fire and Detonation PDUs.

There are a number of software components unique to the NPSNET implementation (Zyda *et al.*, 1992b), notably the Physically Based Modeling package. The Physically Based Modeller (NPSOFF PBM) models rigid-body dynamics using a Newtonian framework (Zyda *et al.*, 1992a). Properties may include linear and angular velocities, mass and centre of mass, elasticity and location and orientation information.

2.3.3.1 Improving DIS

Macedonia, *et al.* (1995) correctly note that SIMNET was constructed for small unit training and has passed on this heritage to DIS. For this reason simulations do not scale well and are not currently suitable for large scale VEs. A number of problems are outlined:

- Bandwidth and computational requirements.
- Multiplexing media.
- Managing static objects.
- Database replication.

It is predicted that a VE with 100,000 players (entities) would require 375 Mbps of network bandwidth to each computer participating in the simulation. Since each node needs to maintain the state of every entity in the simulation (albeit using dead-reckoning models), they will require an inordinate amount of processing power. "*We conjecture that 1000 entities are the limit to which a single host can realistically manage despite future advances in computer and graphics architectures.*" These figures are in line with the performance graph in Figure 2.3 and also means that a more powerful network medium than FDDI will be required.

DIS goes to great lengths to prevent packet fragmentation by requiring that each packet is smaller than the maximum supported by the physical network. Unfortunately, this means that video and audio must be treated in the same way rather than in their more natural continuous forms. Support for these media at the transport or network layers, e.g. through the use of MBONE, relieves the application from the overheads of multiplexing and de-multiplexing.

The simulations usually contain large amounts of static objects, e.g. buildings, that must periodically send update messages even though their state has not changed, just in case somebody missed the last message. The entire simulation database must also be replicated at *each* node since there is no method of partitioning the database. These last two points show the expense of the DIS protocol, both in bandwidth and computational terms.

The reasons offered for these problems are four fold:

- **Event-State paradigm**. Since the simulation is stateless (a basic requirement for DIS) information has to be sent to every entity. This does not take into consideration the fact that the simulated systems "sense" the environment in different ways and therefore have different data requirements. Two geographically distant entities need not know what each other are doing until they are in much closer proximity to one another. By being stateless, the simulation is affected less by the unreliable transmission medium being used (broadcast).

- **Real-time trade-offs**. A real-time environment should avoid point-to-point communications between entities since this requires reliable communications such as the acknowledgement scheme used in TCP. Centralised databases cause I/O contention, so the only course left is to use a connectionless method of communication such as UDP.

- **Middleware**. There is no software layer to mediate between the simulation and the network. DIS must use bridges for large scale simulations which are an order of magnitude slower to reconfigure than routers and the number of nodes is limited to tens of thousands. A network using routers is limited only by the address space.

- **Small scale origins**. SIMNET and DIS were only used, until recently, for simulating small scale environments. This shows in the choice of transmission protocol and monolithic construction suitable for distribution over a single LAN. Past simulations have been packed quite densely with respect to the size of the environment and this

influenced the assumptions made about rates of activity and inevitably the DIS protocol itself.

Complete replication of the database is also grossly inefficient and some means of partitioning information is required. The proposed solution to this problem is an Area of Interest Manager (AOIM). The VE is split into a grid of hexagons - since they are regular in shape and have uniform orientation and adjacency. The division of entities amongst the hexagons is not strict and some entities may belong to more than one group at a time to avoid boundary and temporal aliasing. As the user moves through the VE, the groups behind them are paged out and more groups ahead of them are loaded in. The advantages of such a system include reducing the bandwidth needed to maintain the simulation, the localisation of reliability problems and the ability to make use of high speed networks such as ATM. ATM will probably support multicasting and its high bandwidth might permit the dynamic paging in and out of the hexagonal areas containing large amounts of simulation data.

2.3.3.2 VE Modeling

Since NPSNET is based on DIS there is little modeling infrastructure. The entities may be simulated in full any way the designer sees fit, the only requirement is that its behaviour can be approximated through a dead-reckoning algorithm. All nodes connected to the same network simulate the same VE.

2.3.4 Minimal Reality (MR) Toolkit

This toolkit is aimed at supporting work involving user interface design and may be split into three layers: low-level device support, data processing and high-level services (Figure 2.4).

2.3.4.1 Basic Structure

The device drivers are provided as a client/server pair, the server directly interfaces with the device and the client provides library routines that communicate with the server. The second level massages the data received from the device drivers into a more usable format as well as providing data sharing services between workstations. Complex tasks that are often performed have been encapsulated in a set of high-level functions to form the last layer. These include system initialisation and data synchronisation. All communications on the same machine uses TCP.

One application runs on a machine at a time. Each application has a *master* process that initiates the execution of other programs in the application which are designated as either *slaves* or *computation*. There may be many slave programs which perform simple tasks such as rendering images. Computation processes perform compute intensive work and are usually located on a dedicated machine connected to the master machine via a network.

2.3.4.2 Packages

To aid interface design a number of *packages* are provided to handle some of the more complex functions. There are currently four packages: Workspace Mapping, Panel, Data Sharing and Peer, but the latter two are of most interest in this context.

The data sharing package provides a way of managing a data structure that may be shared between machines by periodically sending an update copy to the other machines. The structure may be synchronous, in which case the receiving program controls its update, or asynchronous where the receiver does not have control (the default).

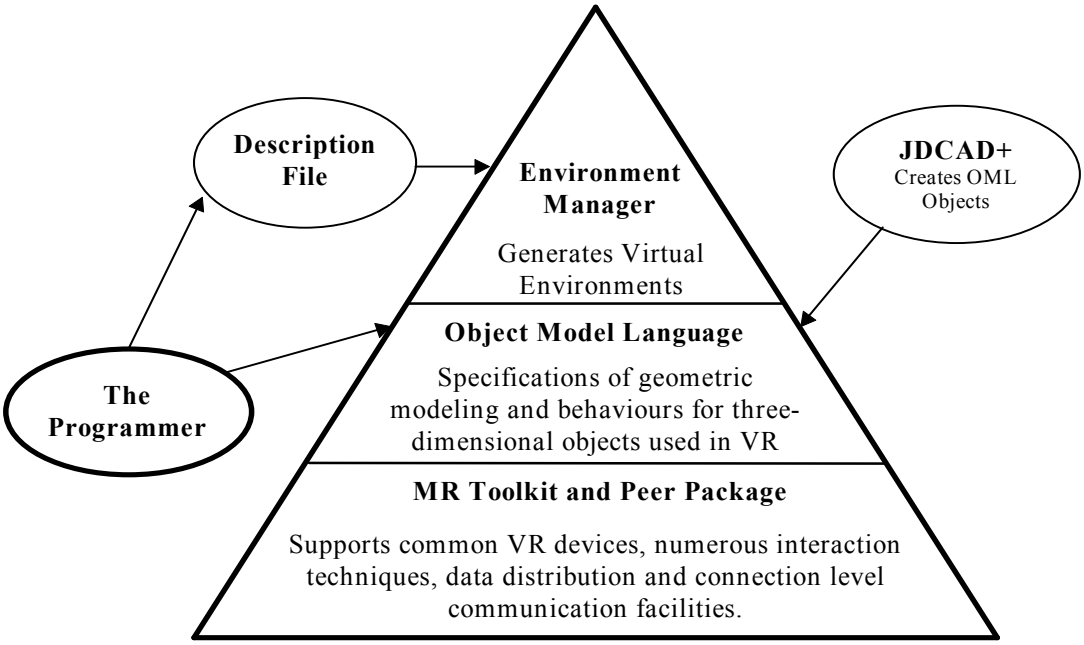

Figure 2.4 MR Toolkit component structure.

The peer package is a recent extension to MR Toolkit and provides the functionality to allow independent applications to communicate with each other via master processes (Shaw and Green, 1993). The slaves receive data from their peers via their master, i.e. slaves do not communicate directly with other slaves or computation processes. Application-specific information may be shared between machines using UDP to send messages to specific addresses. Each machine keeps a peer list which indicates their state, either active or inactive. A peer may become inactive deliberately, with the intention to join in later or not (as the case may be), or a peer may inadvertently become inactive. This happens when the local peer has not received any messages from the remote peer in the last 10 seconds. At this point the local peer attempts to re-establish communication. All peers are connected directly to one another which requires a lot of network traffic to maintain and, as a result, more than five networked machines is not recommended by MR Toolkit's authors.

2.3.4.3 VE Modeling

Platform independent object geometry and behaviour is described in a procedural programming language called Object Modeling Language (OML). An OML object contains code to generate the 3D geometry, controls how the object appears and code for implementing behaviour. The OML compiler produces an intermediate code that is executed by the OML interpreter which is embedded into the application program. An MR Toolkit program loads compiled OML descriptions, initialises devices, coordinates between devices and the objects, and calls the interpreter every graphical update. Therefore a program has to be written for every VE built.

To save time, a generic VE application has recently been added to the suite of programs in the form of the Environment Manager (EM). The EM is responsible for initialising the VE (using a script file), running both single user and multi-user VEs, and also provides facilities for monitoring the execution of the VE (Wang *et al.*, 1995). Each user in a multi-user VE runs an EM which handles calls to OML code. The distribution of the VE is transparent to the OML objects which just see one unified VE. The objects may be classed as *local* - managed by one EM only - or *shared* in which case other EMs may load them. To reduce bandwidth, only those shared variables that have changed state are transmitted and the EM also supports dead-reckoning by sending OML approximation functions to the other nodes. Unusually, it is possible to disconnect from the shared environment, perform some work and then reconnect at a later date.

The user is an integral part of MR Toolkit, in fact the whole system is built around the user. It is possible for multiple users to interact within the same environment when machines are connected using the peer package. Only one VE is simulated at a time.

2.3.4.4 Data and Computation Distribution

Two forms of concurrency control are supported through the use of ownership and access permissions; the choice of scheme is left up to the designer. A shared state variable may be owned by only one EM at a time and that ownership may, if needed, be transferred from one EM to another at run-time. The solution to the case where the transfer message is lost during transmission (possible when using UDP) is left up to the programmer to resolve. A shared variable also has one of two possible access permissions: *writable* and *readable*. If the variable is writable then EMs other than the owner, may write to that variable. If it is readable then they may only hold a copy of its value and its owner will send out updates when necessary.

Each EM has its own copy of the *entire* simulation including the shared variables. The identity of the owner is broadcast to every EM whenever ownership changes. When a remote EM wants to make a change, it requests ownership of the variable and then makes the change. In other words, each machine in the network that has a user wanting to interact in the simulation takes it in turns to run the simulation, whilst the others get the results and use dead-reckoning.

OML descriptions may be created and manipulated using the Jiandong Liang Computer Aided Design (JDCAD+) application which uses a hierarchical modeling system and a 6 degree of freedom (d.o.f.) input device.

2.3.5 Distributed Interactive Virtual Environment (DIVE)

DIVE was developed at the Distributed Systems Laboratory, Swedish Institute of Computer Science (SICS) to aid their research into the distribution, collaboration, interaction and multi-user aspects of virtual reality (Carlsson and Hagsand, 1993).

2.3.5.1 Distribution

The distribution model used in DIVE v2.2 can be conceptualised as a memory that is shared over a network. An old version of the ISIS Distribution Package (v2.1) is used to provide a mechanism for data sharing between systems (Birman *et al.*, 1987). Version 3.0 of DIVE was in beta-testing at the time of writing and no longer uses ISIS[4] which has been substituted for the SICS Distribution Package (SID2 - Hagsand, 1995) that provides similar functionality.

The database, which is completely held in memory, is partitioned into *worlds*. Worlds are implemented as ISIS process groups where each process actively manages its own replica of the database. A DIVE process can only be a member of one world at a time although it may travel between worlds. Each process consists of lightweight threads which are allocated a specific task, e.g. rendering, input/output management or updating the database. The consistency of the shared database is maintained by using mutually exclusive locks, multicast transmissions within the process group and distributed object locks. DIVE supports heterogeneous distribution and machines that are not equipped with graphics hardware can still run non-rendering components of an application.

2.3.5.2 Applications

Applications may be created using the provided C libraries and then run on one or more systems communicating over an Ethernet link using TCP/IP. Multiple applications (implemented as a process) may run simultaneously, modifying the state of the world database. The *visualizer* is a special application that uses selected input/output devices and enables the user to interact with the VE.

Objects in DIVE are allocated a globally unique identifier, a name and a position in 3D space amongst other information (Andersson *et al.*, 1995). They may also have one or more graphical representations. Composite objects are formed by grouping objects together hierarchically. Objects are stored locally in main memory, e.g. during creation, and may be shared over the network using a replication mechanism, i.e. after creation. Object information specific to an application is maintained by the application itself and is not distributed to other processes.

All DIVE processes communicate with messages which may change an object's state, a process' state, or inform the recipient of a specific event. Applications may register call-backs for these events which may be used to indicate errors or user interaction.

Behaviour in DIVE is implemented as a state machine with each arc referring to a particular *signal* type. A signal may be generated when a collision is detected, some form of user

[4] ISIS is now a commercial package and is no longer free to academic institutions.

interaction has occurred, on some input, or when an application wishes to trigger a behaviour directly. A random signal is also available so that some form of random behaviour can be simulated. Current supported behaviours are limited to manipulating the object's visual properties, spatial translation/orientation changes, generating a sound or triggering a behaviour in another object.

2.3.5.3 Users

Each user has their own personalised *body-icon* which is used to represent them in the world. The icon may be made of many parts, e.g. head, eyes, ears, hands and a visor. Each of these components serves a purpose. For example, each eye specifies a viewpoint from which the graphics display is generated and any object manipulated by the user is usually attached to one of the hands.

Vehicles provide a translation between data from input devices to actions in the VE. Several simple vehicles are provided with the system such as a mouse vehicle and one for monitoring head and hand movement when using an HMD. New vehicles may be created using the DIVE Application Programmer's Interface (API).

I/O handling and user representation is therefore integrated into the user object. Multiple users are supported as are multiple worlds which may be entered through *gateways*. Since each world possesses the same properties, there is no problem with object migration.

2.3.5.4 Time

Clocks in DIVE are not synchronised apart from system-level synchronisation using NTP and it is assumed that clock rates are equal on all machines.

2.3.6 Distributed Virtual Environment System (dVS)

Division build their own parallel processing computers which are currently based around INMOS Transputers, Intel i860 microprocessors and a number of ASICs. Their goal is to provide a seamless software environment to the VE designer which has resulted in the development of dVS (Grimsdale, 1993). Since its conception, dVS has been ported to Silicon Graphics, Inc. (SGI), Hewlett Packard and IBM workstations.

dVS v2.0.4 augments existing operating systems to try and provide the best possible performance over these platforms. It is organised into processes that perform certain tasks called *Actors*. There are actors for generating visuals, producing audio, performing collision detection, monitoring 6D trackers and many other tasks (Division, 1994). The user's application is also built from user supplied actors.

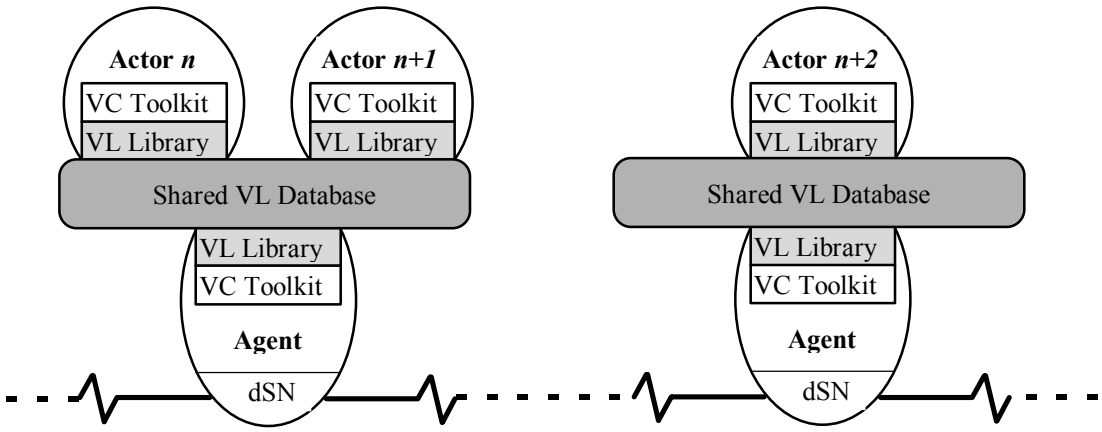

Figure 2.5 dVS system architecture.

The essential components of dVS are shown in Figure 2.5. At the core is a distributed database (VL) which may be accessed by actors through the VL Library. The VC Toolkit provides higher-level functionality for the manipulation of objects and makes calls to the VL Library to achieve this task. The *Agent* is a special actor which handles updates to the local database and informs remote systems of the changes. One agent assumes the role of the *Director* and is responsible for coordinating all database updates. Communications between agents are performed using the Division Session Network (dSN) software layer.

2.3.6.1 Database Structure

An object class in dVS is called an *Element*. An *Instance* of an element may be created and is the unit of communication between actors. Before elements can be defined and instanced, an *Environment* must be created. A root environment is always created by default when an environment database is created (owned by the Director) and subsequent environments may be arranged hierarchically. New environments may be created by any actor at any agent. *Containers* can be defined which consist of one or more elements and are treated as an atomic quantity. A new element definition is written using C-like syntax and passed through a pre-processor which produces the relevant VL data structures and library routines as C source code. These source code files are compiled and linked into the application executable.

2.3.6.2 Database Synchronisation

Actors *hold* an element and by *extracting* that element an actor may change the state and then commit it using an *update*. Any actor holding the element will be informed of the change in state through an *event*. An actor can register interest in (hold) either elements or instances, an action that is environment specific, i.e. updates to sub-environments are not reported. This process is complicated if the item of interest is part of a container. There are actually 3 cases that must be catered for:

1. Interested in a container and a sub-element changes => the whole container is reported as having changed.

2. Interested in a sub-element and the container changes => the sub-element is also reported as having changed.

3. Interested in a sub-element which is subsequently changed => report a sub-element change.

Application tasks have no direct access to VL to avoid contention when two applications try and access the same information. All data accesses to the databases are therefore made by copying. dVS provides a choice of three different synchronisation methods to help maintain database integrity.

1. None. Updates are sent asynchronously and any duplicate events detected before the event reaches its destination are folded into one, i.e. only the most recent update will be processed.

2. Local. Locks the event and associated data until all destination actors within the domain of the local environment database have processed the information. This event is also propagated to remote databases if required.

3. Global. Similar to a local synchronisation event except the lock is performed across all remote databases and as such can be time consuming when acquiring the resources.

Synchronous updates are not supported by VL. These are viewed as expensive, used in only a few special circumstances (although no examples are given) and not the way to maximise performance (section 2.3.6.4).

The agent monitors changes to the local database and distributes these changes to other agents on other machines if interest in those items has been previously registered. Only knowledge about other agents and their current interests is maintained by any given agent, which means updates are sent direct to the relevant agents thus avoiding the need for broadcast. Since only objects that are being held are propagated to remote databases, it is possible for one such object to reference another which does not exist locally. It is up to the application to ensure that it has registered interest in all necessary objects. Agents are allocated a port number which is held in a configuration file, allowing physical machines to connect or disconnect at run-time.

2.3.6.3 VE Modeling

The VC Toolkit supports a number of specialised elements which it calls *Virtual Objects*. The basic element is VCObject which may be decomposed further into other VCObjects and so on. The other standard elements which are held within a VCObject are VCAudio, VCBoundary, VCConstraints, VCLight and VCVisual. Each of these describes a certain number of logically related attributes and are often associated with a particular actor, e.g. VCAudio elements are monitored by the VSOUND actor.

The collision detection actor monitors VCBoundary elements and notifies the two relevant parties when a collision has occurred. Whereas the VIZ (visualisation) actor is interested in VCObject, VCLight and VCVisual elements.

Users are represented by a *Body* actor and therefore there may be multiple users in the same environment. The body actor is also abstracted away from the necessary I/O devices which exist as separate processes and can be assigned a special representation. It is unclear whether an actor from one environment can move into another.

2.3.6.4 Synchronisation

When a network of machines starts up, the first node to complete initialisation sets the time on the other machines to its own. No time synchronisation is performed thereafter. All messages are timestamped but this information is used to discard tardy messages that have already been superseded. dVS never waits for the arrival of a specific message and thus there is no lock-step synchronisation between nodes.

2.3.7 Waterloo Virtual Environment System (WAVES)

WAVES was formerly known as Highly Interactive Distributed Real-Time Architecture (HIDRA) and is targeted at low-cost platforms that use low-bandwidth media for communications, e.g. telephone lines (Kazman, 1993c).

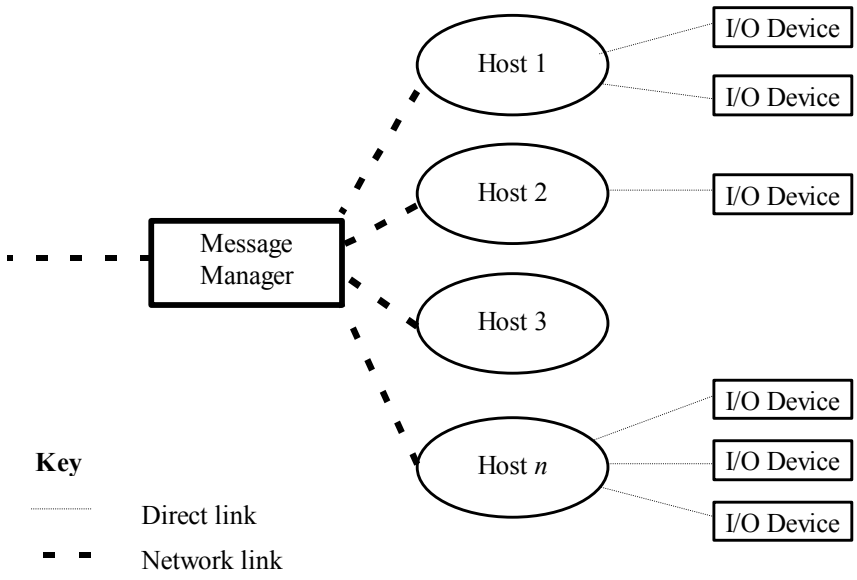

Figure 2.6 Basic WAVES architecture.

2.3.7.1 Basic Architecture

The basic components of the WAVES architecture are shown in Figure 2.6. Each *Host* simulates a subset of objects and provides certain services to each object, e.g. collision detection, rendering, etc. Whilst cyclically updating their set of objects, hosts periodically broadcast the state of their local objects to other hosts. Major I/O events, e.g. user input, are communicated each cycle to maximise fidelity. The communications between these hosts are done over virtual connections, mediated by a number of *Message Managers*. Connections may also be filtered so only messages of interest are sent to the hosts. The message managers

are also given the ability to delegate direct point-to-point links between hosts in special circumstances, e.g. a line carrying a video signal. Under WAVES, a VE may be distributed over a network of message managers, with the allocation of hosts to each manager being determined by a dynamic clustering algorithm. Objects have explicit behaviour models which aid load balancing, support dead-reckoning and may be used to predict an object's state in order to combat latency. The ghost objects that reside on a host are called *clones* in WAVES. As with other dead-reckoning systems, some fault tolerance is provided in that if one host should go down, then the others can carry on using their current behaviour models.

Load balancing is performed on each host based on several criteria: the host's processing power, the number of objects on the host and how closely related the objects are (Kazman, 1993b). The host sends its current load and their maximum possible load to their local message manager. When the host detects its load has risen above its maximum, it sends another message to the message manager indicating which object it would like to get rid of. Another host is found for the object or, if no suitable host can be found, the transfer request is refused.

Users should be representable as objects providing there are sufficient input/output devices on a host and this would also imply that multiple users can be supported. It is unclear whether it is possible to execute multiple environments.

2.3.7.2 Distribution of Responsibility

To solve the problem of area management, WAVES uses a special *Area Manager* which is paired with a message manager (Kazman, 1993d). The area manager maintains a list of viewable areas for a given viewpoint, one per host. When the list changes, the message manager's filtering criteria for a given host is changed so that only those objects in the host's viewable areas are sent to it. Since the area manager only changes message filters, it can be added or removed from a WAVES system without disturbing anything else in the system. To overcome rapid changes in area, WAVES proposes to use an object's behaviour model to anticipate the changes and send filter requests in advance. To avoid the problem of all users occupying a small number of areas and causing a bottleneck, there may be many managers in the system and they may balance their loads dynamically.

Interactions between objects are specified externally in *interaction detection* and *resolution* (IDR) agents (Kazman, 1993a). The *world view maintainer* contains the description (*world attributes*) of the environment that the objects operate in; a view controller which dynamically manages the inventory of agents which may be interfacing, and an inventory of all the objects which exist in the world (*world objects*). If IDR takes too long then the world may be broken into a number of areas, each with their own IDR facility. Each IDR server contains a production system, which allows the system designer to create arbitrary constraints on an object's state in the form of rules that are evaluated each execution cycle. Each IDR server contains a "theatre map" that plots the locations of all objects in the theatre and raises an exception when two objects attempt to occupy the same space. IDR servers can be designed to handle particular types of interactions: spatial, temporal or semantic. IDRs can therefore be used to detect interactions within a spatial threshold as a sort of prediction mechanism to accommodate lags in the system.

In summary, object behaviour is defined within the objects, interactions are defined within IDR servers and the environment is defined within the world view.

2.3.8 AVIARY

In the AVIARY model a distinction is made between objects that are presented to the user through different media: *Demons* are the pieces of software that implement an object and *Artifacts* are the manifestation of the demon in the VE (Snowdon *et al.*, 1993; Snowdon and West, 1994; Snowdon, 1995).

2.3.8.1 Basic Architecture

A virtual world is seen as a container for artifacts and a set of constraints on those artifacts and behaviour. The sole *World Object* represents a virtual world, acting as a container for artifacts, storing the identities of demons, details on the objects providing other services, and information shared by all objects. The actual artifact definitions are not held within the world object, but since the artifacts may be accessed through it, this information can be obtained indirectly.

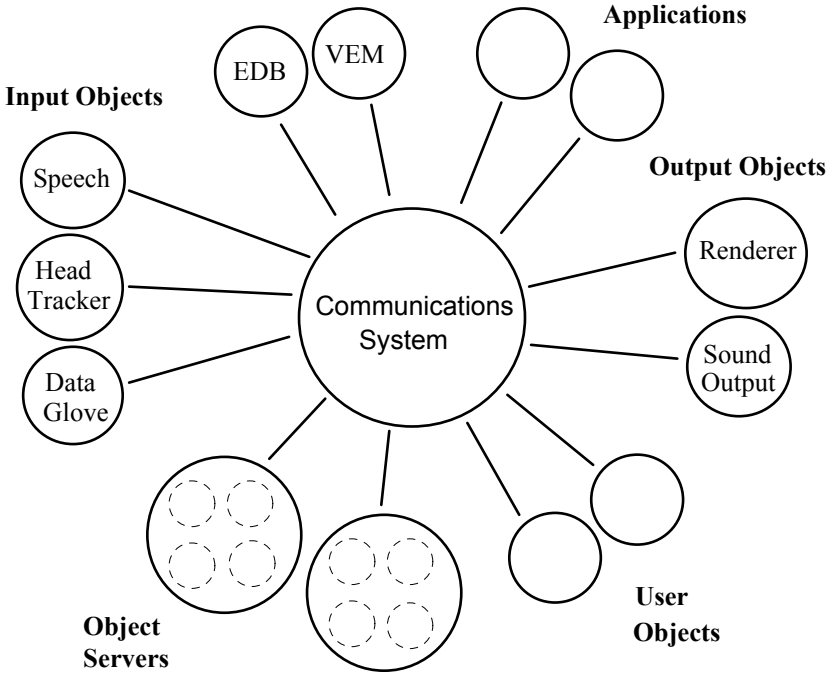

Figure 2.7 AVIARY component schematic.

The *Environment Database* (EDB) provides a spatial management service to other objects (Figure 2.7). When a demon moves, it sends a message to the EDB, which prompts a collision check for that object. The EDB then informs the relevant objects of the collision and they may then react as they please. To prevent the EDB becoming a bottleneck, it may be split into one or more new EDBs which share the existing workload (although this has not been implemented yet). In addition, separate EDBs may be employed for different media and therefore operate in parallel.

Object Servers provide an execution environment for demons, handling object creation/destruction, messages from other objects, memory management and scheduling. Inter Process Communication (IPC) between all types of objects is supported without restriction. One object server is allocated to each processor. Security-wise, each object controls access to its own data and may therefore protect any sensitive information.

Only one *Virtual Environment Manager* (VEM) is present in the whole system and provides services to ensure that the integrity of the VE is always maintained. This includes the assignment of identifiers to objects (aiding dynamic object creation) and also to classes and messages. This last mechanism ensures that objects that understand the same messages but have been implemented differently can still communicate with each other. Complementing the VEM is the *World Manager* which maintains a list of all the available services provided by objects. This enables any object to look for another object providing a service that it requires, e.g. visual rendering or collision detection.

The issue of time synchronisation is resolved in AVIARY by making use of real-time clocks on each node. Simulation or world-time can, however, be scaled relative to real clock time.

Both synchronous and asynchronous message passing is supported with both multicast and point-to-point links used to transfer the message. To prevent deadlock, the object server is multi-threaded so that it is always ready to respond to an external event.

2.3.8.2 VE Modeling

Behaviour of the artifacts is dictated by the methods defined for the creating demon which actually consists of two parts: artifact-specific and world-specific. All features that are shared by all objects in the world are held in the world-specific part and those unique to each class of demon in the artifact-specific part. This separation of attributes aids migration from one world to another. Demons can make use of services provided by any other kind of object and can inherit classes or define existing classes to extend its capabilities.

Multiple worlds are an important part of AVIARY's design since each may require a different interaction metaphor and it own laws and properties. The user is permitted to travel between worlds by using *Portal* objects that may appear as artifacts in each virtual world. When a demon moves between worlds the world-specific part of the demon is replaced by that of the new world whilst the artifact-specific part remains unchanged.

A demon may represent an application or a user, either way it is likely to need access to input and/or output devices. *Input objects* control input devices, sending data to all interested objects only when there is something new to send. *Output objects* monitor a particular location in the world and display a representation in the chosen media.

Users are represented by demons and are decoupled from the system and I/O devices. Although there does seem to be provision to integrate I/O into the user demon if performance dictates.

The current implementation is written in C with object-oriented features, including multiple inheritance added through macros. These macros create an internal data structure of class descriptions, object instances, etc. This data may be communicated to another machine thus

supporting object migration, although no load balancing checks are currently made to see whether this is required.

2.4 Summary

All of the systems examined here are trying to achieve interactivity, but none are real-time in the traditional sense and therefore do not support real-time displays. These are important aspects of a VE system and the impact of supporting them is discussed in sections 3.3 and 4.3. This classification has therefore been left out of the feature summary table (Table 2.3).

2.4.1 Communication Organisations

Typically there are *n* processes in a VE which need to communicate with each other. Using a point-to-point communications system, a link must be established between every process or a central server established, forming a hub. In the former case this will require *n(n -1)* links and in the latter, *n* links, although total centralisation can place a burden on the central server which can quickly become a performance bottleneck. Conversely, administrative processes often need to monitor most (or all) transmissions and make according actions, e.g. dVS's Director and the Message Managers in WAVES.

Whilst broadcast relieves the overhead of maintaining links, it floods the network with messages which are either an inconvenience (on shared networks), or wasteful (on dedicated networks) because in large VEs not every process needs to know what all the others are doing. Area management can be used to determine who needs to know what, but cannot use broadcast as the transport mechanism. Maintaining a number of point-to-point links is one solution but with the complications already outlined above. Multicast provides a way of overcoming these disadvantages (as demonstrated by the AOIM in NPSNET) whilst still retaining the low transmission overhead, but it is not widely available and is, like broadcast, unreliable.

Feature		DIS/SIMNET	NPSNET	MR Toolkit	DIVE	dVS	WAVES	AVIARY
Communications	Transport Mechanism(s)	Point-to-Point, Broadcast, or Multicast	Multicast	Point-to-Point within node and Broadcast between nodes	Multicast	Point-to-Point	Point-to-Point	Point-to-Point and Multicast
	Targeted Bandwidth	Unspecified	10 Mbps +	10 Mbps +	10 Mbps +	10 Mbps +	14 Kbps +	10 Mbps +
	Structure(s)	Peer	Peer	Hierarchical & Client/Server internally and Peer externally	Peer	Client/Server	Client/Server	Client/Server
Data Management	Organisation	Total Replication	Total Replication	Active & Passive Partial Replication	Total Replication	Passive Partial Replication	Complete Distribution	Complete Distribution
	Localisation Support?	No	Yes	No	No	No	Yes	Yes
Computation Management	Organisation	Partial Replication	Partial Replication	Total/Partial Replication	Total Replication	Partial Distribution	Partial & Complete Distribution	Complete Distribution
	Behaviour Level	1	1	0/1	0	0	1	0
VE Modeling	Environment Management	Parallel	Parallel	Single	Multiple	Parallel	Unknown	Multiple
	User Support	Multiple	Multiple	Multiple, Integrated (possibly)	Multiple, Integrated, with Representation	Multiple, Decoupled with Representation	Multiple, Decoupled (probably)	Multiple, Decoupled
Time Management	Progression Method	Implicit or Explicit	Implicit or Explicit	None	None	None	Unknown	Implicit
	Node Synchronisation	UTC or None	UTC or None	Programmer	None	None	Unknown	None
Fault Tolerance	Degree	3	3	3/0	0	0	3/0	0
Security	Method(s) Employed	None	None	None	None	None	None	Object Level Interface

Table 2.3 Distributed VE system feature classification summary.

2.4.2 Transport Mechanisms

Deciding whether to use a reliable message delivery service or not is a key decision in the design of a distributed VE system. NPSNET, MR Toolkit and other DIS-based systems use UDP between machines. DIVE uses multicast exclusively and AVIARY uses it for messages that need to be sent to many processes, but this is under the control of the application programmer. The only two systems that use a reliable service exclusively are dVS and WAVES. Both make use of a known network configuration and thus known addresses, to distribute the messages. If an unreliable service is used then the software protocols must reflect this decision and a degree of fault tolerance provided.

Even with the implementation of these two steps, loss of messages (or their delayed reception) will, inevitably, have an affect on the user interface. The effect could be anything from a slight glitch or jump in the display, to temporary loss of service. If these counter-measures are not taken then the designer is relying on a large number of variables holding true to keep things running, e.g. plenty of bandwidth available, network interfaces fast enough to capture packets, etc. Of those systems reviewed that use multicast/broadcast, NPSNET and MR Toolkit account for lost messages. Both use exported behavioural models but MR Toolkit actively encourages a machine to disconnect and reconnect during a simulation by providing appropriate API functionality. To the author's knowledge, DIVE and AVIARY do not make any provisions for lost messages, the consequences of which are unknown for both systems.

2.4.3 Bandwidth Implications

All of the systems use common networking technology so it is unsurprising that most are currently implemented using Ethernet. The DIS standard does not actually specify a bandwidth but the author does not know of any implementation using anything less than 10 Mbps. WAVES' target of 14 Kbps is laudable but there is precious little bandwidth to play with. Without compression, 14.4 Kbps will support a data rate of approximately 1.31 Kbytes/second[5]. The compression supported by modern modems could improve on this if there were repeating patterns in the data stream, like those found in ASCII text. However, the likelihood is that the messages sent between nodes will contain extensive binary data and thus compression will do little good. This figure does not, of course, include transport protocol overheads which may reduce the data transfer rate substantially (section 4.2.3).

The fact that the available bandwidth for a given process will vary during execution is a compounding factor. This affects both reliable and unreliable services and, depending on the criticality of the system, can at the very least wreak havoc on system performance. The author believes that the ability to allocate channels of fixed bandwidth for a fixed period (as supported by ATM), is essential to the development of distributed VE systems. Only then will communications become deterministic and thus release the designer to concentrate on other issues.

[5] 14.4/ 11 = 1.309 Kbytes/second (assuming 8 data bits, 1 start bit, 2 stop bits and no parity).

2.4.4 Distribution & Scaleability

Communications latency affects all systems, regardless of architecture, however, it is the largest enemy of scaleability. As the distance between nodes increases so will the latency and unless the system protocol and structure is modified to account for this, performance will degrade beyond acceptable levels.

Each of the current systems reviewed address one of Kleinrock's classes with a possibility of application in another if the conditions are right. None attempt to address more than two and certainly no changes are made to the system architecture to help it adapt. Each form of data and computation distribution has advantages and disadvantages. All can be applied successfully in a near/tightly-coupled system but as we move through far/tightly-coupled into the far/loosely-coupled classification, so the solution weaknesses become more apparent.

Complete distribution of both data and computation is a victim of increased latency since all accesses and data modifications have to be communicated to their source. The worst-case task would be the monitoring of a piece of information, performing an action when it reaches a certain value and then modifying it. This would require a message to get the latest value and possibly another to modify it *every* simulation step. If this task was performed on many objects it could saturate the network. Both AVIARY and WAVES use this approach. Active partial replication, as used partly by MR Toolkit, also has the same problem. Whereas data would be accessed via an object interface with complete distribution, copies of whole chunks of object state can be distributed with active replication.

Partial replication of data provides slight relief from this symptom by supplying a mechanism that will send any interested party a copy of the relevant portion of state (or behavioural model) only when it *changes*. Not only does this reduce bandwidth consumption, but also the computational load because the task function is only executed when an update is received, rather than at every simulation step. Modifications can be made by sending the instruction to change data back to the source. A slight variation on this is partial computational distribution where changes are made locally and communicated back to the owner, or, as in dVS, committed to the shared database. If the latter method is used, locks must be used to preserve data integrity. Lock acquisition and release can lead to deadlock (section 2.2.3.3) and are inherently undeterministic and thus unsuitable for a real-time system.

Complete data distribution has the advantage that data is only stored in one place, while partial data replication duplicates parts of the environment's state, thus consuming more resources (DIS/NPSNET). This pales into insignificance against total replication where the complete environment state is duplicated. In a high bandwidth configuration this is a waste of resources, but it is the only solution when the distance between nodes is large and latency is high. The largest challenge in this case is to keep the replicated databases in synchrony. Transmitting modified segments of environmental state between databases is not a viable option. Partial computational replication would seem to be a possible solution.

The usefulness of exporting behavioural models can be shown clearly by once again considering the goal of distributed VEs over a 14.4 Kbps telephone line. A level 0 behaviour system would likely send a position and orientation update for each simulation time step. Assuming 6 x 32 bit floating-point numbers (3 for position and 3 for orientation) plus, say, another 16 bits for an object identifier gives a total of 208 bits or 26 bytes. Using our previously calculated data rate of 1340 bytes/second we can determine that 1340/26 = ~51.5

messages that can be sent per second. Assuming a modest 15 Hz update rate, this permits us to send updates to ~3.4 objects. If more bandwidth is available initially then this is quite a tempting, easy solution and is used by DIVE, MR Toolkit (at its lowest level), AVIARY and, to a lesser extent, dVS. If a higher level behavioural model was supported, such as dead-reckoning, then messages would be sent at a much lower rate (depending on the object's behaviour) thus permitting more objects to be supported.

However, level 1 behaviours still require messages to be sent quite often and it would be quite easy for the databases to get out of synchrony considering the latency. Instead of informing each other of deviations from the predicted behaviour, it would be more sensible to totally replicate the computation and only inform each other of changes in object behaviour. This could be an update of the behavioural description effected by software, e.g. level 2 behaviour, or by a user, e.g. level 3. Bryson's two-point paradigm (2.2.4.5) is representative of the kind of information that could be sent.

Load balancing and process migration are best applied in a tightly-coupled system. There is obvious application for these techniques when using complete computational distribution and they can also be applied to systems using partial replication. With a large number of ghost processes and area management there are likely to be those that are accessed more frequently than others. Spreading the computational load evenly whilst minimising the distance between communicating objects could greatly improve performance.

2.4.5 Time

Most of the systems reviewed do not seem to have any policy on time management. AVIARY uses the implicit model for clock synchronisation which is less than full-proof. Clock oscillators can drift (as any network administrator will testify) and need to be constantly corrected. The most common method for doing this is NTP which is adequate for non-time-critical work where second accuracy will suffice. When dealing with multiple updates per second this clearly will not do. With extra effort over a longer period of time it is possible to synchronise clocks to millisecond accuracy using NTP, but the author feels that this may be inadequate when dealing with 33 ms time spans (for a 30 Hz update rate). Ideally, each node would be equipped with a Standard Positioning System which would ensure that all machines throughout the world were synchronised to within 167 ns. Unfortunately, the current cost of this technology would probably be prohibitive so solutions like NTP are the best remaining choice for systems using implicit time models. Indeed, if clock synchronisation is needed in MR Toolkit or DIVE, the designers have assumed that NTP would be used.

The explicit time model uses timestamps in messages for various purposes such as informing them of the send time, the time at which the message is valid, etc. DIS/NPSNET uses a timestamp format which can specify a time up to an hour after the current hour, to within an accuracy of 1.676 microseconds. However, there seems to be no suggested methodology of ensuring that each node has the correct current time. In this instance there would seem to be a requirement for both models to be used together to manage simulation time.

It might be possible to use explicit time progression exclusively within systems that use complete/partial computational distribution or partial replication, but when total replication is used a common reference is required.

2.4.6 Fault Tolerance

Those systems that export behavioural models (section 2.2.4.5) implicitly support a notion of *reliability* (degree-3). Failure to receive an updated model, because the source host is down, can be remedied when the host rejoins. DIS ensures this by requiring that no one machine controls the simulation. MR Toolkit permits a node to leave and rejoin the simulation but this does not really constitute robustness since leaving and rejoining relies on using the correct protocol. WAVES does export behavioural models, but there is no mention in the available documentation that states fault tolerance as a design goal.

None of the systems pursue the goal of *availability* through duplication of resources, probably because they are at a premium. Total replication of both data and computation is done by DIVE which would put it in the best position to provide fault tolerance, although this is not a stated goal. When the faulty node recovers, another node in the simulation can send it a complete copy of the current environment state. *Recoverability* is not supported by any of the systems and the only true *robust* systems are those based on SIMNET/DIS.

If interaction is a high priority then degree-4 fault tolerance is the most desirable and may even be considered as the only usable type. Any faults managed at a degree below this would be reflected as a disconcerting change in the VE display. This may manifest itself as anything from a small "jump" in continuity (degree-3) to a total loss of realism (degree-1).

Rather ironically, the least reliable transport mechanism - broadcast - is also the best way of providing fault tolerance: through redundancy. The incorporation of a special process/node in the network that listens into every message and maintains a state backup using point-to-point links would place an unacceptable overhead on communications. It would require two messages to be sent for every communication rather than just one. Fortunately the reliability issue is being dealt with (sections 2.2.2.3, 2.2.7) which will remedy one of the weaknesses of any system that uses broadcast techniques.

2.4.7 Security

This is an issue that none of the current systems fully address. This is not too surprising since all of these systems are used as tools for researching the field and security can get in the way. An encrypted data stream is not particularly helpful if you wish to monitor message passing, nor is access control when you are experimenting with object interaction metaphors. AVIARY makes a token gesture by putting each object in control of its own data. This is not an added feature, this ability comes with the adoption of an object-oriented structure. An object's methods may be coded in such a way to vet access but AVIARY provides no built-in/automatic security layer.

2.4.8 Modeling

With the exception of MR Toolkit and WAVES, all of the systems support the concept of multiple VEs in one way or another. DIS supports multiple exercises which take place in the same environment, whether these exercises can interact is not clear. DIVE assigns a multicast group to each environment so messages are not processed unless the user is present in that environment. dVS can support different environments but there is no evidence to suggest that

elements in one environment can move to another at run-time. All objects in AVIARY occupy one of the available VEs which are designed as a hierarchy of worlds, each one building on the properties of the parent. Objects may also migrate from one world to another, a feature shared by DIVE. However, in AVIARY worlds may possess different properties whereas DIVE worlds would need to be programmed identically to facilitate migration.

All the systems support multiple users in differing ways. MR Toolkit might support more than one user if each had their own workstation and was sharing the same database. The WAVES literature does not specifically state that it can support many users, but its general structure of hosts and I/O devices infers that it does. In DIVE and MR Toolkit, the user is an integral part of the system, in fact they are built around the user. DIS, dVS, WAVES and AVIARY do not distinguish a user from any other object except that it may have various I/O devices connected to it. All of these can be used to simulate VEs with no human participation whatsoever. Despite this treatment, dVS does seem to emphasise the ability to specify a special user representation in the VE in a manner similar to DIVE. The latter, however, also uses this representation to configure the required I/O devices.

2.4.9 System Summaries

2.4.9.1 DIS-based Systems

DIS and SIMNET would have originally been classified as near/loosely-coupled but DIS is now trying to move on towards far/loosely-coupled. The problems with such a move have been discussed in this summary and in section 2.3.2. NPSNET is being used by the Naval Postgraduate School as a testing ground for new ideas and concepts to help DIS make this transition. Despite the DIS community's advocation of the protocol's applicability to non-military VEs, the author feels that it will always be of restricted use due to its constrictive definition. All messages sent between objects have to be defined in advance and of the dozens already defined only one of them is of general use: the Entity State PDU. The other PDUs deal with explosions, logistics support, etc., which are inherently military-application specific.

2.4.9.2 MR Toolkit

This system is used to aid research into user interfaces and is accordingly designed around the user. It does its task well but its lack of generality limits its applications in the same way as DIS-based systems.

2.4.9.3 DIVE

DIVE is more flexible than DIS and MR Toolkit, but its use of total replication and an unreliable message delivery system make scaleability a real issue.

2.4.9.4 AVIARY

Of all the systems reviewed, AVIARY is the most flexible but shares another problem with the others in that it will have problems scaling up to larger VEs. The use of complete distribution has limits and must be supplemented with other forms of data/computation management, requiring changes in the system's architecture.

2.4.9.5 WAVES

There is only limited information available on this distributed model although the literature states that a prototype implementation is being developed. The inclusion of low bandwidth communications is cause for concern and catering for this could compromise the design

2.4.9.6 dVS

A restriction shared by all of the systems presented here is the difficulty with which the VE definition is changed. dVS requires the basic components and structure of the environment to be scripted off-line, pre-processed, compiled and linked in with the Actors.

Its exclusive use of point-to-point links may also prove to be detrimental to performance when larger networks of dVS machines are attempted.

2.4.10 A New Architecture

From the analysis presented in this chapter, it is possible to extract those features that effectively resolve the presented issues and derive a new architecture for distributed VE systems. This is presented in chapter 4 following a closer look at a couple of aspects which deserve more attention: modeling and displaying VEs.