7. Conclusion

Chapter 7

Conclusion

"In my end is my beginning."

Mary Stuart, Queen of Scotland

The final chapter of this thesis begins with a brief reminder of the work presented in the preceding chapters. Following this, the USS architecture is classified using the taxonomy presented earlier on and its most important features are highlighted. A few specific research areas that are relevant to distributed VE systems are also described, indicating the benefits they may provide. Finally, the current trendy topics in the area of distributed VE systems are related to the work presented here.

7.1 Thesis Review

The introduction to this thesis gave a brief introduction to the area of VR, highlighted the emphasis on interactivity, and described the two cornerstones of a system that would support this: real-time and consistency. The services of a real-time system enable the generation of real-time displays which are justified in chapter 3. Consistency reflects the need to ensure that everything in the VE appears in the right place at the right time, to one or more users simultaneously.

Chapter 2 began with an examination of the issues involved in the design of a system capable of distributing VEs. The solutions used by existing systems that have attempted to tackle this complex area vary quite substantially. In order to provide a way of comparing such systems a classification scheme was derived which strove to categorise each system on the basis of: real-time support, communications, data management, computation management, VE modeling, time management, fault tolerance and security. There is an intricate web of inter-dependencies connecting many of these categories which often makes examination of one difficult without referring to another, e.g. data and computation management working together to provide consistency. However, the author believes that this taxonomy is a good starting point and was applied to the seven distinct systems that were reviewed. The results of applying the classification scheme proposed in this thesis to the USS are presented later in section 7.2.

Chapter 3 questions the current way that VEs are modeled and highlights a particular aspect of human-computer interaction that is not addressed in most systems. To better understand how to model a VE, the structure of the natural environment was examined and several taxonomies of varying levels of detail were presented. Based on these attempts to classify natural and virtual environments, the author presented a suitable definition and abstract model for a VE. Essentially, current modeling practices take one perspective on the thing being modeled and concentrate on one medium, usually visuals. With this approach the model will function adequately until such time as another medium is considered, e.g. sound, or a different perspective has to be taken, e.g. infra-red instead of natural light. At this point the model will falter because some (or all) of the information that is now needed to simulate this perspective/medium will be missing. If a more ecological approach had been taken to modeling, then sufficient information would have been modeled initially such that similar changes would not require extra work. There are obviously practical limits to the amount of information that can be modeled at one time and these are discussed with relation to the modeling process as a whole. When looking at the design process it was noted that an integrated modeling and simulation system would enable development, experimentation and evolution. The ability to develop a simulation on-line provides much greater flexibility than is available with current systems and also a reduced development time cycle. These features will hopefully also encourage the VE designer to explore the different forms the model can take. Finally, evolution referred to the ability of each entity in the simulation to make changes to the model and create other entities.

Related to the issue of modeling a VE is its display. The purpose of a display is to take raw information from the environment, process it, interpret its meaning, and then present it in a form that enables the viewer to extract some meaning. A good display will permit the natural processing of the presented information and allow the participant to concentrate on the task at hand. A bad display will require the participant to expend extra effort and will probably degrade their performance. The second part of chapter 3 describes how variable-rate visual displays cause problems when judging time to contact with a virtual object. The example given is catching a virtual ball, but it could equally be braking in a virtual car to avoid a collision on a virtual motorway, or attempting to perform in-flight refuelling in a flight simulator. Essentially any task that requires the user to make judgements based on velocity and acceleration/deceleration can be affected if a constant-rate display is not used. Two methods of achieving such a display were presented: one requires special OS support, the other will work on normal operating systems.

Chapter 4 starts with the presentation of the requirements for a USS, a set of realistic design restrictions, and a little more detail on key aspects, e.g. distributed real-time systems. Having settled on a modeling process using specialisation through inheritance in chapter 2, a suitable representation of the VE abstract model is presented. Since the abstract model is derived from our universe, an appropriate naming scheme was adopted based around “universal”. A number of existing languages were examined before it was decided that none of them satisfied (or could be modified to satisfy) the requirements of a VE modeling language. The proposed language, UML, can be broken into two halves: data definition and instruction code. The structure of UML is important since it is an integral part of the USS architecture. Although UML code can be passed between USS processes, it could have any syntax or grammar. The data definition, however, influenced the mechanisms used to manage state within the architecture and *vice versa*. The design is dissected in section 7.2.

After outlining the USS design, a prototype implementation was described in chapter 5. Key to the system is a real-time distributed deadline scheduler which is difficult to implement with current hardware/software technology. The author had, prior to USS development, implemented a far less complex worst-case scheduler at the application level to help enforce a constant-rate graphical display. It was the author's experience that, even with a special-purpose operating system, use of such a scheduler was problematic due to the difficulty in accommodating actions beyond the application's control, e.g. network and disk access. Therefore the architecture's key elements were implemented without the scheduling functionality. The PML is used to provide a common interface to the various OS services that the USS processes require - mainly message passing. Following details of the PML, the structure of the UML interpreter was described, including a detailed explanation of the complex data structure used to hold the model description and its instance data. The remainder of the chapter dealt with each major software component in turn, starting with the UM, and highlighted key aspects of their implementation.

The implementation was evaluated in chapter 6 which started with a characterisation of the platforms used for testing. This was followed by a detailed examination of the UML interpreter, its performance and memory requirements. The impact inter-process communications have on performance was analysed in the section dealing with the PML. The rest of the chapter examined the simulation performance of the system as a whole, in single node, two node and three node configurations. In addition the process migration mechanism was demonstrated using the two and three node configurations. A number of enhancements that could be made to the design and implementation in order to improve the prototype's performance were also described. The chapter concluded with a discussion of the factors affecting the performance of the prototype and a number of general observations.

7.2 USS Classification

Table 7.1 replicates part of Table 2.3 in order to provide some basis for comparison of USS's features. USS is the only distributed VE system architecture out of those reviewed that has pursued the goal of interactivity through real-time displays and the application of real-time systems techniques.

7.2.1 Communications

Currently only point-to-point communications are used but there is scope for the utilisation of reliable multicast once it becomes available. Although USS was not designed with a specific bandwidth in mind, it is clear from the results presented in chapter 6 that anything below 10 Mbps would be unsatisfactory due to the associated latencies. Two communication structures have been adopted by USS. Firstly, a client/server paradigm is used between processes within the same node, but the communication paths are heavily influenced by a hierarchical organisation, e.g. messages to other nodes are routed through the UM. Secondly, communication between nodes is strictly hierarchical.

Feature		dVS	AVIARY	USS
Real-time Constraints	Supported?	No	No	Yes
Communications	Transport Mechanism(s)	Point-to-Point	Point-to-Point and Multicast	Point-to-Point (+ Reliable Multicast?)
	Targeted Bandwidth	10 Mbps +	10 Mbps +	10 Mbps+
	Structure(s)	Client/Server	Client/Server	Client/Server & Hierarchical
Data Management	Organisation	Passive Partial Replication	Complete Distribution	Passive Partial Replication (within USS) & Total Replication (between systems)
	Localisation Support?	No	Yes	Yes
Computation Management	Organisation	Partial Distribution	Complete Distribution	Complete Distribution (within USS) & Complete Replication (between systems)
	Behaviour Level	0	0	0, 1, 2
VE Modeling	Environment Management	Parallel	Multiple	Multiple
	User Support	Multiple, Decoupled with Representation	Multiple, Decoupled	Multiple, Integrated or Decoupled
Time Management	Progression Method	None	Implicit	Implicit (within USS) & Explicit (between systems)
	Node Synchronisation	None	None	NTP (SPS Idealised)
Fault Tolerance	Degree	0	0	2 through 4
Security	Method(s) Employed	None	Object Interface Level	Basic Access Control

Table 7.1 Comparison of distributed VE feature classifications including USS.

7.2.2 Data and Computation Management

The method of monitoring state updates which are only sent by the owner when changes are made can be classified as passive partial replication. This technique is used between USS nodes but all data is replicated in each system, with only system-unique data being transmitted between them. Localisation, which also has implications for computation management, is supported through the use of constraint functions in the UM. Complementing the choices of data management is the complete distribution of computation between processes within a system. Rather than distribute computation between systems, it is completely replicated in every system. Process migration is supported, thus increasing the scheduling options and hopefully efficiency. As discussed in section 4.5.4.10, arguments can be made for the use of

all 3 levels of behaviour distribution. Most of the systems reviewed supported the transmission of an entity's state variables, whether continuously, by request, or only when a change of value has occurred. Level 1 distribution (commonly called dead-reckoning) was used exclusively by WAVES and DIS. Despite the potential display side-effects of this technique it is quite effective in reducing bandwidth consumption. If necessary, it is possible to implement dead-reckoning with USS on top of the basic state management system.

7.2.2.1 Dead-Reckoning

Given an entity whose definition consists of a position vector and a velocity vector, a manager would monitor the velocity vector rather than the position. This would mean that rather than sending continuous position updates as the entity moved, the manager could extrapolate a position from the velocity vector. If velocity was constantly changing then this technique would give little improvement. However, if two velocity vectors were maintained by the entity then even this can be accommodated. One vector would be used internally for the entity's own calculations and the second vector would be its exported property - monitored by the manager. The exported version would fuel its own approximated behaviour model (the same as the manager's) and updated only when its approximated behaviour differed significantly from its actual behaviour. This now replicates the same functionality that conventional dead-reckoning systems have.

7.2.2.2 High-Level Behaviour

Level 2 behaviour distribution can also be supported through another basic USS mechanism, that of remote UML function invocation. A number of functions would be defined to achieve some high-level tasks, such as driving around a corner, and then executed at the appropriate time. This technique can be used to control a user's shadow on a remote system (section 7.4.5).

7.2.3 VE Modeling

Multiple universes may be simulated simultaneously by a USS, although the prototype only supports one. There can be many users interacting in a simulation and within a system there is no special distinction made between an entity representing a user and an automated entity. Input devices are sampled from within the user entity, however, whether this is mapped into a direct device access or through a server process is an implementation decision.

7.2.4 Time Management

Both forms of time management are utilised by USS. Explicit time progression is almost a by-product when a distributed deadline scheduler is used to coordinate the simulation. To ensure synchronicity between individual systems, an implicit progression model is used so that behavioural information generated by one system is valid in another.

7.2.5 Fault Tolerance

Fault tolerance is an expensive goal, best achieved by duplicating hardware and software components. However, there are a number of features of USS that lend themselves to at least a little reliability and recoverability - at a cost. The state held by a manager or entity may be reconstituted gradually through state updates or explicitly by request to the UM. If not enough information is held within a system to reconstruct the process, then it may be obtained from another system which is also simulating the same universe. If there is a problem with a particular node then entities can be migrated to another node. Alternatively, their state can be obtained from another system and started locally on another node.

7.2.6 Security

Security is another feature that can generally only be realised at a computational price. This aspect was not fully investigated because security measures can often hinder evaluation of other system features. However, there is basic access control support in that a process may locate the originator of any service request and the UML interpreter can limit access to OS services.

7.3 Important Features

The proposed architecture deals with a number of issues but there are a few aspects which are either worthy of note or unique to this solution.

7.3.1 Real-Time

A distributed real-time system forms the basis upon which the USS architecture is built. In order for the participants to efficiently interact with the environment and each other, it is important that they are provided with real-time displays. To keep in step with the constant update rate of the displays, it is necessary to ensure that all entities are also updated at a constant rate. Failure to meet this hard deadline is a system failure. If all updates are guaranteed to happen within a given time frame it is possible to start accommodating for lags in the system by performing predictive calculations. When the simulation is distributed over a number of machines the network must also have deterministic properties if it is not to upset the processing deadlines. Predictability at this level also presents the opportunity to compensate for communications latency. It is likely, however, that determinism will be realised at the cost of performance and the under-utilisation of resources - a matter of concern to the designers of ATM switches where guaranteed bandwidth and bounded latency are primary requirements.

7.3.2 Scaleability

All of the distributed VE systems reviewed chose one mechanism for handling computation and one mechanism for handling data within the system. It is not possible to scale the system up or down without affecting the performance of such mechanisms. DIS, for example, replicates the data making up the VE on *each* node and partially replicates computation on

each node through the use of dead-reckoning algorithms. Therefore if the VE has 10,000 entities, then each node must handle data and computation for *each* entity. Initially, when the number of entities in the simulation was in their low hundreds, this was not a problem. It was only when larger simulations were attempted that the idea of using localisation to reduce the workload of each node was suggested (section 2.3.3.1). In a similar vein, AVIARY uses a system model that works well when on a tightly-coupled network of workstations but will require some modifications if it is to support larger simulations. A similar story can be told for the other systems.

Adapting a design after the fact is always undesirable, because the end result is less attractive than it could have been if the design had taken a broader perspective to begin with. The architecture presented in this thesis is by no means perfect, but it does attempt to define a system that may be scaled from tightly-coupled multiprocessors through to large scale networking of machines over large geographical distances.

The decision of when to network a machine as a USN in a larger USS or as a separate system requires further investigation. It is clear, though, that there comes a point when the network bandwidth between two clusters of machines can no longer handle the amount of traffic generated within a system. In order for users on either end of this connection to participate in the same simulation, two systems must be configured from these nodes that are capable of replicating each other's simulation workload.

7.3.3 Bandwidth Reduction

A great deal of effort has gone into reducing the amount of bandwidth used between processes and nodes, thus increasing the number of nodes it is practical to have in a system. Only those portions of an entity's state information that are of interest to managers are transmitted and only when a change in this information has occurred. Managers may also specify constraint functions that are applied to the state data the entities transmit to their UM. These functions can filter out unwanted data before it is sent to managers resulting in unnecessary computation and, more importantly, sent over lower bandwidth communication links to other nodes. Further savings could be made if a multicast protocol was available.

7.3.4 Modeling

The premise with which the process of VE modeling was approached in this thesis was that the development of VEs should not be constrained by past technological standards.

The need to model a VE is relatively new and is presently more of an art than a scientific practice. It is an exploratory process that often requires many changes before the model has reached a satisfactory state. UML is integrated into the USS architecture in such a way that the initial VE model can be developed off-line and then modified on-line. Any changes are reflected instantaneously throughout the simulation. For example, a function describing the behaviour of an entity may be replaced by sending that entity a new UML definition for the relevant function(s). It is also possible to add or delete parts of the UML definition without affecting the existing state information for the rest of the definition.

The ability to build upon existing VE models is a powerful tool which can save time and cut development costs. Establishing a set of base environments with well defined core behaviours would ensure that VEs built by different designers would allow entities to move from one VE to another with reasonable ease. Although the movement of entities between universes was not implemented in the prototype, the ability to preserve those parts of an entity's state that are common in the source and destination universes is already in place.

Different perspectives on the same environment may be supported through the use of managers that monitor different components of the universe definition and display the contents in the desired manner. Alternatively, each manager may monitor the same information but only process those that meet certain criteria, probably with the aid of constraint functions to reduce bandwidth.

Any type of information may be modeled, subsequently the system has no knowledge of space *per se* and there is no requirement for it to be modeled. In fact, the UM and the core entity and manager libraries understand how information is structured, but do not expect any particular organisation, or look for any specific component in it. Consequently, only when a suitably dimensioned property (such as position) is added to the universe definition will space be modeled. Also, the relationship between simulation time and real clock time within the environment can be defined arbitrarily (section 4.5.5.1).

7.3.5 Flexibility

A minimal working system requires a RM, a UM and one or more ENT processes. In this state it is possible to run any non-interactive simulation. Although an entity may sample input devices, the user would not be able to see the consequences of their own or the simulation's actions unless a manager was present, connected to a display. A manager may be introduced to monitor state changes and generate a suitable display. For VE simulations the two most commonly used managers would be VIS and AUR. However, non-interactive simulation may simply require a text-based display of key simulation variables. Managers are not only used for generating displays; for example, the SIM checks for violations of an entity's space and informs the involved entities so they may resolve the situation.

The design of USS was driven by the desire to simulate interactive VEs, but due to its flexible structure it may be applied to other types of simulation. For example, artificial life simulators often use the model of a parallel processing, shared memory machine. Each "entity" within the simulation is a program whose instructions may mutate or, through breeding, become merged with another entity's code resulting in a hybrid. This process continues over and over again. USS lends itself well to this problem because:

1. There is a direct comparison between the beings in the artificial life simulation and entities.

2. An entity's code may be replaced at run-time and there is nothing to prevent the replacement code being generated by another entity.

3. In the same way, one entity may spawn another and define its behaviour through UML code generation.

The simulation would still operate within fixed deadlines but the update frequency could be reduced to sub-interactive rates. There may well be more efficient task-specific methods for the other types of simulation but USS at least provides a platform for testing ideas before developing the project further.

7.4 Areas for Investigation

A few improvements to the prototype were presented at the ends of chapters 5 and 6, but there are a number of areas encroached upon by distributed VE systems in general that the author feels need further attention.

7.4.1 Reliable Multicast

As a distributed system is scaled up, so the feasibility of using point-to-point communication links rapidly disappears. Multicast communications present the only practical solution: the overhead of a single transmission is incurred despite sending to multiple destinations. Unfortunately, the multicast systems that are becoming available now are, like their predecessors, unreliable. For data such as audio streams the occasional loss of a packet is acceptable. However, if state or event data is lost making its way from one machine to another then this will affect the state of the simulation. The consequences of this range from an event occurring on one node and not another, to users making a decision based on incorrect information. At the operating system level the consequences could be more severe, e.g. invalidation of a fault tolerance redundancy mechanism. Research into reliable multicast protocols is underway and the author believes that this work should be encouraged.

7.4.2 Guaranteed Bandwidth

Distribution over large areas not only increases communications latency between system components, but the latency also varies by greater amounts. Although it is impossible to totally eliminate latency, steps can be taken to account for it, but only if sensible estimates can be made. Fortunately, ATM permits the reservation of channels of fixed bandwidth between the communication's endpoints. Adoption of a technology that provides this kind of service at all levels, from LANs through to WANs, would also seem to be an essential component of future large-scale distributed VE systems.

7.4.3 Time Synchronisation

In order to synchronise time between machines there would appear to be two basic options: use a software protocol, such as NTP, or a satellite-based system such as SPS. If synchronisation over many hours is unacceptable, then the accuracy obtained using software protocols is quite low: within a few seconds. If simulation protocols can be developed that cope with this level of accuracy then this is sufficient. However, the author believes that the same amount of care given to estimating communications latency should be applied to that of time synchronisation. There is a solution available in the form of SPS which is currently prohibitively expensive (section 4.5.5.2) but, given a mass market and a little time, there is no

reason why this technology would not become cheap enough to incorporate into every machine.

7.4.4 Real-Time Operating Systems

At the time of writing there are very few operating systems that can be used for real-time applications and are therefore expensive in comparison to the plethora of general-purpose operating systems. There are even fewer that support deadline scheduling and address the problems of distributed scheduling. The popularity of real-time systems research has risen somewhat since the widespread availability of multimedia workstations, but significantly more work is needed in this area before they may be effectively utilised for interactive real-time VE simulation.

7.4.5 Shadowing the User

There comes a point when latency is so great that simply reflecting every single change in a user's state to all other machines becomes impractical. A solution is to only transmit actions between machines and let the user's shadow processes effect these changes in the mirrored environments. This presents three problems that must be resolved. Firstly, how these actions are recognised; secondly, how they can be described in a form suitable for transmission and, thirdly, how these actions are interpreted. The first and last problems will be heavily influenced by the type of simulation in that the nature of the actions exhibited will vary. For example, parameterised actions in a networked driving simulator may be reduced to accelerate, decelerate, turn left, turn right, navigate roundabout, park, etc. Whereas a Computer Supported Cooperative Work (CSCW) application might involve more intimate interactions between users. Consequently actions may even be required to mimic human gestures and facial expressions, e.g. approval, disapproval, happy, sad, etc. The format used for transmission of these actions may be as simple as executing a parameterised function remotely, or something more complex.

The choice of technique has implications for maintaining the integrity of the simulated environment. Consequently, more work is needed to assess the additional system functionality required to aid action recognition, representation and interpretation.

7.5 Outlook

The USS architecture has been dealt with in a rather isolated manner over the past few chapters. This section attempts to relate it to a few of the current popular topics in the area of VE systems.

7.5.1 Internet

The work in this thesis is not applicable to the Internet as it stands today: variable delays are experienced between communication endpoints and the available bandwidth may vary, to name but two problems. IPng (or IPv6) is essentially IPv4 (the current version) with some modifications (Bradner and Rankin, 1995). Aside from introducing techniques to reduce

message fragmentation, preallocation of network resources is supported, allowing establishment of connections guaranteeing bandwidth and latency. Multicast has also been added as a standard addressing option for IP datagrams; in fact it has replaced broadcast as the base service abstraction, which is now a special case of multicast. Combined with a suitable transport mechanism from desk to desk, such as ATM, it should be possible to apply the USS architecture to the future Internet and certainly improve upon the prototype.

7.5.2 Virtual Reality Modeling Language

The Virtual Reality Modeling Language (VRML) is an attempt to bring interactive, 3D VEs to the Internet via the WWW (SGI, 1996). From a modeling standpoint, VRML is a classic example of a visual-centric approach. SGI's Open Inventor was chosen as the starting point for the format which, over the past two years, has been adapted to fit the role of a general format for describing VEs. After reconciling the representation of visual information with the need to model behaviour and the demands on the client browsers, it was decided to alter the way that the Open Inventor scene graph is used. This has been just one of many changes to the file format. Consequently, VRML has the same basic look as Open Inventor but is used in a different way. Audio has been added to the language and at the time of writing the more important problem of encapsulating behaviour is being addressed. Most people in the VRML community are agreed upon the fact that some form of programming language is required to describe object behaviour but no consensus has been reached on which language. The fact that this debate is happening at all reflects the problems of completely isolating information representation from simulation execution. It is exactly these problems that the USS architecture seeks to relieve through integrating the modeling process with the system that will execute the VE model.

7.5.3 Java

Java has been proposed as a language suitable for object behaviour representation within VRML. Java is interpreted, platform independent and increasing in popularity every day. Unfortunately for VRML browser writers, source code for a Java interpreter is not available requiring a lot more development work just to simulate a VRML scene. On the positive side, native translators are beginning to appear which greatly reduce the execution times of Java code. However, although Java can load classes (in byte-code form) at run-time, existing classes/functions cannot be redefined and there is no way of modifying data structures at run-time. Without these abilities, the VEs modeled using VRML will be very static in nature and require considerable amounts of time to develop and maintain. Specifically, if a VE is to be "upgraded" then all users will have to disconnect whilst the new one is installed, possibly followed by a conversion of old state data to the new format. Certainly not a quick or easy procedure to schedule when the server is accessed by clients throughout the world.

7.5.4 Consequences

Some of the problems with the WWW and the Internet have already been described (section 6.7.1). In addition to these, VRML is being developed incrementally from a visual file format with the intention that it should one day also guide how machines should be networked to realise interactive VEs over the Internet. By approaching the problem in this way, the author

believes that, in its current form, VRML will not fulfil the expectations held by so many in the VRML community. For example, moving entities from a VE served by one machine to a different VE served by another is not possible unless some standardised structure for the information has been adhered to. Currently this is not possible unless all the designers agree to conform to a given structure and even then there is no way of enforcing such an agreement. A modeling mechanism such as inheritance and a common set of base VEs would resolve this problem.

The problems of distributed VEs are so many and varied that they must all be addressed simultaneously to reach a well-rounded solution. Inevitably, however, the lessons learnt by developing VRML will reinforce the validity of applying certain techniques to distributed VE systems and may possibly even disprove others.

7.6 Summary

This thesis has attempted to fuse research in distributed systems, real-time systems, modeling, languages and human-computer interaction into one system capable of distributing real-time interactive simulations. Those issues examined (to varying degrees) just within the area of distributed systems support were: message passing, marshalling and unmarshalling, naming and name resolution, heterogeneous nodes, scheduling, process migration, configuration management, performance management, time, synchronisation, security and persistence.

The problem domain is so complex that the exploration of the issues and their inter-dependencies within the time permitted was relatively limited. Many decisions had to be made during the design process, all of which were biased towards a system capable of supporting multi-user, interactive, VE simulations. Interactivity demanded a real-time system and multiple users required a distributed architecture with comprehensive techniques to maintain the integrity of the shared VE. Of the requirements presented in section 4.2, applicability was represented by the modeling language and its integration into the system, whilst fault tolerance and security took a back seat.

The architecture's structure is based upon the philosophy that the right tool is used for the right job. The combination of different distribution techniques, integrated with an expressive, flexible modeling language, has resulted in a scaleable system that can be used to both develop and simulate VEs in a heterogeneous, distributed computing environment.