Analysis of Techniques used in Distributed Virtual Environments

Paul Okanda and Gordon Blair,
Computing Department,
Lancaster University,
Lancaster LA1 4YR,
United Kingdom.

Email: {okanda, gordon} @comp.lancs.ac.uk

Abstract

The manner in which a world is modelled upon a Virtual Reality system is very much reliant on (depending on the designer’s judgement) the critical and non-critical requirements of the model. An assessment of distributed Virtual Reality systems reveals a number of common constructs that are implemented in a wide range of standard ways. In this paper, we consider representative case studies and identify performance aspects of some common techniques for distribution control, data management and programming methodologies.

1. Introduction

Conceptually, a virtual world can be seen as a collection of objects, some of which are static (décor elements) and others that change state either spontaneously (like autonomous creatures in a game) or by users’ actions (like an avatar obeying users’ commands; an object undergoing modification in a co-operative design environment). Distributed applications that attempt to offer the illusion of immersion in shared 2D or 3D worlds for instance virtual shopping malls, multi-user online games and co-operative design commonly provide users with the ability to see and possibly interact (usually by means of the users’ avatars or computer driven objects) at run-time. Additionally, users viewing a given part of the virtual world at the same time must always get the impression of seeing the same objects in the same state.

Providing users with the above capabilities is an obvious challenge to designers especially so in massively multi-player DVEs in which thousands of users ought to potentially interact with each other and with autonomous entities over the Internet. This paper introduces, compares and briefly outlines an analysis of the performance aspects of a range of techniques used both at the application and platform level in contemporary distributed Virtual Reality systems. These techniques are indeed specializations of optimisation approaches some of which are specific to particular architectures.

2. Case Studies

Existing published works propose a range of ways to meet the challenges in design of virtual worlds. These works could be characterised into three main categories thus:

- Academic works
- Military Works
- Commercial Platforms

The works covered below are deemed to be a good representation of state-of-the-art DVEs.
2.1 Academic Works

2.1.1 DIVE

The Distributed Interactive Virtual Environment (DIVE) (Hagsand96), (Frecon98) is an internet-based multi-user software platform in which participants navigate in 3D space and see, meet and interact with each other. Originally a laboratory tool, it has developed into a system implemented on many university platforms for use in design of prototype virtual environments. It supports the development of virtual environments, user interfaces and applications based on shared 3D synthetic environments. It is especially tuned to multi-user applications, where several networked participants interact over a network with main focus put on:

- Scalable networking
- Interaction with individual objects

Virtual World Characteristics

A central feature in the programming architecture of DIVE is the shared, distributed world database. It runs on a peer-to-peer approach with no centralised server, where peers communicate by reliable and non-reliable multicast, based on Internet Protocol (IP) multicast (http://www.sics.se/DIVE/). Conceptually, the shared state can be seen as a memory shared over a network where sets of processes interact by making concurrent accesses to the memory. The virtual world in composed of objects, actors and views. The diagram below illustrates the DIVE run-time architecture – simplified to six processes interacting with one another in a single common world. Each process holds its own copy of the relevant objects, but the whole world can be thought of as residing ‘on the network’.

![Figure I: The DIVE run-time Architecture](image)

Coherency, Consistency and Concurrency

DIVE entities are protected from concurrent ‘write’ modifications by way of a simple object-based token-passing algorithm. Consistency and concurrency control of common data (objects) is achieved by active replication and reliable multicast.
protocols. That is, objects are replicated at several nodes where the replica is kept consistent by being continuously updated. Update messages are sent by multicast so that all nodes perform the same sequence of events.

Dynamic Interaction and Behaviour

The software model presented above forms the basis upon which essential functions of virtual environments are implemented. The dynamic behaviour of objects may be described by interpretative scripts in DIVE/Tcl that can be evaluated on any node where the object is replicated. A script is typically triggered by events in the system, such as user interaction signals, timers, collisions etc. DIVE entities interact and adapt to the changing environment. The degree of autonomy is achieved by associating Tool Command Language (Tcl) scripts with entities. Since Tcl is portable and interpretative, scripts are replicated along with entities and executed on any platform without compilation, which result in a set of actions. Powerful event and timer-driven behaviours are specified through the complete DIVE functional interface in combination with Tcl iterations and functions.

2.1.2 Virtual Society

The Virtual Society (VS) [Lea97] is an umbrella for a set of research projects within Sony CSL whose overall goal is to investigate how the future online community will evolve. Its research originates from the belief that future online systems will be characterised by a high degree of interaction, support for multi-media and most importantly the ability to support shared 3D spaces such that users will not simply access textual based chat forums, but will enter into 3D worlds where they will interact with the world and other users in that world.

The support infrastructure that the Virtual Society project aims to build is for a shared 3D virtual world (in the internet environment) allowing multiple users to interact in real-time in a range of settings (from shopping between friends from their own homes to cooperative work between geographically remote researchers). The proposed system has elements of Computer Supported Cooperative Work (CSCW), a virtual reality system and an online chat forum. The focus of this work is put on scalability issues especially from the distributed consistency and communication latency front. Since the model presented by the VS platform is a shared 3D virtual space, it must be seen consistently by all participating users which means that all actions occurring in the space must be propagated to all participants in a causal relationship. Additionally, conflicts between user actions ought to be either avoided or resolved. Since the aim is to scale the system across many sites, the choice of a centralised shared database cannot be retained (the main reason being performance overhead). Instead the database is distributed over sites while guaranteeing that any update on the database at one site is propagated to all other sites. This leads to the major issues of distributed consistency and communication latency which the VS project tackles using a hybrid approach. The approach adopted to solve the problem is the design of an architecture limiting the degree of sharing together with a framework of replicated proxies using group communications to handle consistency.

Database partitioning

The shared world is partitioned according to the notion of auras adapted for the purpose of defining the degree of sharing and, when necessary, of reducing sharing. Each object that exists in the virtual world specifies a dynamic aura that represents (by means of a 3D coordinate system) the portion of the virtual space in which it is interested. A separate unit, the Aura Manager (AM) constantly monitors objects as they move around the world and informs them when other objects collide with their
aura. In the current model, auras provide a means to control spatial interaction but they could be used for aural or sensory interaction as well.

**Latency Hiding with Caches**

Once two objects collide, there is need to support a degree of consistency so that they share the same view of the world. Latency is partially addressed by the use of auras which reduces the number of participants that should receive an update. Classification of data into three categories provides a means of increasing the speed of access to certain data types:

- Static data: read only and hence never changed
- Dynamic data: current value may be temporarily out of date
- Dynamic data: current value must always be up to date.

In a 3D space, many data values and actions (e.g. querying the owner of an object, changing the colour of an object) are read only or require only a delayed consistency. In such cases, the general notion of proxies could be used where the proxies act as local representatives of remote objects that can be viewed as a local software cache. When an object ‘A’ enters the aura of object ‘B’, it creates a proxy for itself then hands it to object ‘B’ through a communication link established beforehand. Object ‘B’ subsequently uses the local proxy as if it were object ‘A’ to perform any query or operation. Policies used in VS to cache and update cached information is as follows:

- Static data is cached into the proxy and any access to it uses the cached value.
- Requests to dynamic data whose values may be temporarily out of date return the currently cached value. The proxy further queries the actual data value at periodic intervals.
- Requests to dynamic data whose values must always be up to date result in the proxy actually requesting the data from the real object via a remote connection.

**Group Communication for Consistency Support**

Consistency support is built using groups to define who should be consistent and then using group communications to send the updates. Each object’s aura is mapped to a single group. When another object comes within that aura, it joins the group that represents the aura. When a change is made, any message sent to the group is propagated to all members in the group (i.e. within the aura) who then make the change locally. Consistency is guaranteed by a combination of message sending and locking. Further, consistency levels are assigned to groups for differing consistency requirements. For instance, a strictly consistent group, in which all updates are propagated immediately to the proxies would be fine for clients connected on a high bandwidth link. Instead, clients connected on a low bandwidth link would form a weakly consistent group in which updates are propagated at fixed time intervals in order to reduce the communication cost.

Consistency support relies heavily on the semantics of the send; the stronger/richer the semantics, the lesser the work that has to be done by the consistency algorithm.

### 2.1.3 DEVA

DEVA (Pettifer99) is a large-scale VR system developed by the Advanced Interfaces Group at the University of Manchester. It aims to address issues of scale in terms of number of geographically distributed users, extent and number of objects in the
environment, complexity of behaviour and graphical sophistication. The current version, DEVA4 consists of three main components:

- a dynamically configurable communication layer aimed at exploiting application level knowledge to optimise network usage.
- a distributed world-manager capable of dynamically changing between peer-to-peer and client-server modes, supporting a new object-oriented programming paradigm.
- a multi-threaded object orientated graphics kernel.

System Architecture

DEVA uses a logical client-server architecture. However, the server is in fact a cluster of processors, each in charge of different subsets of the world entities and this collection of servers implements a DEVA virtual machine as shown below.

![DEVA System Architecture](image)

**Figure II:** DEVA System Architecture

Object Model and Distribution

The virtual worlds are divided into environments within which there are entities. Since an environment may be considered an entity, it is possible to organise entities into a hierarchical tree such that the behaviour of an entity depends on its own properties and those of all nodes over it in the tree.

Entities of the world are created in and managed by the server. Client processes connect to the server to interact with and obtain state information about these entities. Clients obtain information concerning a specific entity by a point-to-point connection to the server managing the entity, but this distribution over several servers is masked at the application level, where a unique logical server is known. Object attributes and behaviour are described in the DEVA language. At a given time, an entity’s attributes and behaviour are a combination of properties proper to the entity, common to a range of entities and proper to the environment the entity is currently immersed in. DEVA’s language compiles to C++.

One main feature of DEVA is the fact that code describing environments or entities is dynamically loaded on demand. The server is persistent and processes any entities regardless of whether or not any clients are connected.

**2.1.4 Continuum**

Continuum (Frederic00) is a research project at France Telecom R&D, aimed at developing an adaptable and open platform for large-scale virtual worlds with a particular interest in real-time multi-player online games on the internet. It offers a software framework (currently Java) that allows various components to be
pluggable: the interest model and its underlying implementation, communication protocols, consistency protocols etc. The modular architecture enables deployment of a virtual world under given resource constraints, by choosing the appropriate set of components or even providing a new component for a given service, without having to reprogram other services or the application. Hence from the application programmer’s viewpoint, it offers a high-level of abstraction and distribution transparencies which mask (to some extent) the effect of network latencies.

**Virtual World Characteristics**

The Continuum platform proper is neutral with regard to the type of virtual world that can be designed on top of it. It has been used to build applications ranging from virtual communities with avatars to large-scale distributed simulations.

**System Characteristics**

The Continuum framework allows both client-server and server-less distributed architectures to be constructed while making the distribution topology transparent to the application programmer. In terms of interest management, a perception-based model has been implemented to allow a Continuum entity to be associated with an aura which quantifies its capacity to perceive and influence its environment. Scalability is achieved by multiple IP multicast event channels over which the full event traffic is partitioned.

**Object Model and Replication**

The Continuum object model allows both passive and active entities which are endowed with autonomous behaviour. Entities are represented by replicas distributed in several simulations. One of these replicas is called the master replica. An entity is created by a process creating a replica and binding it to a simulation. Then, mastership is transferred throughout simulations, to achieve persistence or for load balancing purposes. Entities inherit the SimObject abstract class, and must publish aura information, which allows the infrastructure to determine in which simulations to place the replicas. Slave replicas are created in simulations owning master entities likely to perceive them according to aura information. The infrastructure is in charge of creating/deleting slave replicas as entities move or change their perception capabilities. Synchronisation of replicas relies partly on the application programmer, who decides on a logical synchronisation policy (which messages to send when, with which quality of service etc), whereas the infrastructure ensures the diffusion of messages toward replicas of the same entity, taking possibly into account, the quality of service criteria.

### 2.2 Military Works

#### 2.2.1 DIS

The Distributed Interactive Simulation (DIS) (IEEE Computer Society95) results from an initiative from the US defence community to define and standardise an infrastructure that links different types of simulations at multiple locations to create realistic, complex virtual worlds for the simulation of highly interactive activities. The DIS protocol is geared towards ‘human-in-the-loop’ military simulations involving tanks, aircraft, ground forces, radars etc. It is now a proven concept which has been implemented in many public domains and commercial applications. The DIS protocol or protocols inspired by it are currently being used for commercial multi-player gaming projects.
The DIS protocol is not a DVE in itself but rather a communication protocol to be used in DVE systems. Thus its world characterisation criteria is almost meaningless. In terms of scalability, the DIS protocol has been tested successfully with large-scale simulations involving hundreds of entities (the largest DIS experiment involved 2000 entities).

System Architecture

Entity Distribution: The DIS protocol assumes a completely distributed architecture. There is no central server holding the full state of the simulation. Each individual simulation application taking part in a simulation is autonomous and responsible for some entities in the virtual world. The simulation's application maintains the state of its associated entities according to a given simulation model (e.g. dynamic model of fighting aircraft) and sends updates to other simulations.

Communication: The DIS protocol implements a broadcast model of communication in which each simulation sends messages to all other participating simulations. It is the responsibility of the receiving application to determine if the simulation event it receives is relevant or not (e.g. within its perception bounds). In terms of communication support, DIS has the following properties:

- best-effort non-reliable multicast,
- transport-to-transport maximum latency of 100ms.

Coherency: Dead-reckoning algorithms are used to reduce the rate at which a simulation issues a state update of an entity. The basic notion of dead-reckoning is agreement in advance on a set of algorithms that can be used by all simulations to extrapolate the behaviour of entities. It is also an agreement on how far reality should be allowed to diverge from these extrapolations before a correction is issued. Each simulation maintains an internal model of the entity it is responsible for together with a dead-reckoning model which represents the state of the entity as computed in remote simulations by extrapolation of the position/orientation state using a given dead-reckoning algorithm. Only when the difference between the internal model and the dead-reckoning model exceeds a given threshold does the simulation send a state update message so that other simulations can update their dead-reckoning model. The DIS protocol is an application-level protocol which defines a set of twenty-seven messages (called PDU for Protocol Data Unit) that a given simulation application may emit and needs to understand. The most frequently used PDU is the Entity State PDU (ESPDU) which carries information about an entity state (entity identity, location, orientation, dead-reckoning parameters etc.). PDUs are also defined for some special events or interactions between entities such as collision, fire (of munition) and detonation. The DIS protocol does not rely on reliable multicast. It addresses the issues of consistency and synchronisation between different simulations through the systematic use of absolute messages (no DIS PDU depends on preceding PDUs i.e. PDU\(_{n+1}\) supersedes PDU\(_n\)) and a heartbeat protocol which maintains a regular flow of information on the network. More precisely, a simulation application decides to emit an ESPDU for a given entity when one or more of the following conditions is/are met:

- the divergence between the internal model and the dead-reckoning model exceeds the predefined threshold,
- the appearance of the entity has changed (e.g. an aircraft burning),
- more than X seconds have elapsed since the last issue of an ESPDU (the default value of X is 5 secs.),
- the entity is about to disappear.
Thus, under a default DIS configuration, a simulation entity emits a state update every 5 seconds. These heartbeat messages take care of possible losses by resynchronising the receiving simulations. They are also used by a new simulation application joining an ongoing simulation to acquire the current state of the virtual world (it receives a message from each entity every 5 secs).

**Scalability:** The DIS protocol has been tested successfully with large-scale simulations involving hundreds of entities (the largest DIS experiment involved 2,000 entities). Its main advantage lies in its lightweight and decentralised approach. However, the broadcast model it uses in which each receiving simulation is responsible for discarding irrelevant data does not scale well and has an impact on network and processing load. To improve scalability of the DIS protocol and in particular to meet the scalability requirements of the large-scale simulations planned by the US military (e.g. the Synthetic Theatre of War with up to 100,000 simulated entities), several approaches have been proposed. Many introduce some form of Interest Management techniques which take into account the perception needs of each participating simulation so that a minimal amount of irrelevant data is received. All these Interest Management techniques make use of multicast (more precisely IP multicast) to partition the global simulation into disjoint communication groups.

- **Spatial based approaches** such as NPSNet (Macedonia95) partition the world into a grid of disjoint cells (a 2D grid of hexagonal cells in the case of NPSNet). Each grid is statically associated with a distinct multicast address. As a dynamic entity moves across the world, it joins and leaves the multicast groups depending on its location and domain of interest (typically its visual range). There is a trade-off between the size of the cells and the overhead of multicast group management. The smaller the size of the cell, the more precise the filtering but at the expense of a large number of multicast group ‘joins’ and ‘leaves’ since many ‘joins’ and ‘leaves’ per cell would be captured. These grid-based approaches have to cope with the current limits of multicast technology: the restriction on the number of distinct groups the multicast hardware can support and the time needed to join a new multicast group (up to a few secs).

- **Functional based approaches** filter simulation events based on the intrinsic needs of objects. Each simulation entity should be able to declare the simulation events that are relevant to it (ideally their domain of interest) through a subscription mechanism. These interest expressions typically include predicates about the attributes of entities. For instance, ‘send me all events from tank objects which are within a radius of 10 km’.

- **Temporal based approaches** are orthogonal to the two previous approaches. Regardless of how the data is filtered, an entity may not require updated information from other entities as frequently as it is available. Thus an entity may emit simulation events at various levels of resolution.

The DIS protocol and its successors go a long way towards providing an infrastructure for large-scale interactive simulations. However, it is highly biased towards a specific application domain – military simulations. Further, its usage in other fields (e.g. concurrent engineering) makes little sense. DIS is nothing more than a protocol. As such it does not provide an architecture that has a clear-cut separation between the low-level communication aspects and the application-specific simulation level. This is supposedly addressed by the HLA architecture which subsumes the DIS protocol.

### 2.2.2 HLA

The High Level Architecture (HLA) (Defence Modelling and Simulation Office99), (Van Hook96) is the result of an initiative from the US military to design, specify and
standardize a high-level simulation architecture that would integrate all kinds of simulations ranging from real-time distributed simulations (e.g. DIS protocol) to non real-time discreet event simulations. The HLA standard’s two main objectives are:

- to allow interoperability between heterogeneous simulations and
- to promote reuse of simulations and their components by specifying the general structure of a simulation in terms of interfaces without making specific demands on the implementation of each component.

**World Characterisation**

To achieve the above objectives, the HLA standard is made up of the following elements:

- Run-Time Infrastructure (RTI) which takes care of the communications between all local simulations.
- Interface Specification which is a formal description of the interface between HLA applications and the RTI.
- Rules that are defined to which a HLA participant must comply. These define the responsibilities between the federates of the simulation and the RTI.
- Object Model Templates that are standardized formats used to define the functionalities of simulation models and their interactions.

**Object Model**

All simulation entities are referred to as objects in HLA. The state of each object is defined by its attributes and attribute values can be passed from one simulation to another. At one moment, one and only one simulation has the ownership of an object attribute and as such has the right to modify it. The ownership of an object attribute can be transmitted from one simulation to another. Objects interact with one another through so-called *interactions*. An *interaction* represents a unique event such as a collision between objects. A HLA application interacts with the RTI through a publication/subscription mechanism. Each simulation registers objects and attributes it will publish to the RTI. It also registers with the RTI attributes and interactions it needs to receive. A simulation may subscribe to a range of attribute values (e.g. 50<longitude<80). Based on this information, the RTI attempts to match the needs of all the participating simulations by setting up adequate communication resources and filtering the simulation data it conveys.

Amongst the services that the RTI provides are:

- **Federation Management**: create/destroy a federation of simulation, pause/checkpoint
- **Object Management**: create/delete an object, send/receive an interaction, update/reflect an attribute
- **Declaration Management**: publish/subscribe to object attributes and interactions
- **Data Distribution Management**: supports efficient routing of data ownership management; transfer ownership of object attributes
- **Time Management**: coordinates the advance of logical time and its relation to real time. The service specifies the following message ordering policies:

  Receive order – messages are passed to the simulation in the order that they are received.
Priority order – messages are passed to the simulation lowest time stamp first at the time the message was received (this does not prevent a message from being delivered to a simulation in its ‘past’).

Causal order – messages are passed to the simulation in a consistent order with regard to before and after relationships of the events represented by the messages.

Time stamp order – is the strongest ordering strategy and is equivalent to the one assumed by classical discreet event simulations.

Scalability:

As explained earlier, the RTI registers the publication and subscription statements of each participating simulation and is responsible for conveying and filtering the simulation information accordingly. The Data Distribution Management service within the RTI is responsible for routing the simulation data between the participating simulations via a routing space. A routing space (Van Hook96) is a multi-dimensional coordinate system in which simulations express interest to either receive attributes or interactions with other simulations. It is made of subscription regions and update regions. The RTI Data Management service detects the overlapping of subscription and update regions which indicates a dependency between simulations and establishes communication resources accordingly. The RTI makes a distinction between routing of data (establishing the necessary work connectivity) and actually sending the data. The objective is to minimize latency and maximise throughput by establishing the necessary network connectivity before it is actually needed. The following stages are involved in the RTI distribution framework:

- **expression of interest**: expressed by each federate. Receiving and updating interests are expressed through regions of routing spaces.
- **clustering**: the aim is to reduce the number of regions by unifying and combining them where possible. This reduces traffic on the network and as a result the cost of matching subscription and update regions with different algorithms involved in the process.
- **matching**: is the process of determining correspondences between subscription and update regions, the result being a list of receivers for each update region.
- **establishing connectivity**: connectivity on the network is established based on results of the matching stage. Multicast groups could be an example of this connectivity set-up.
- **sending data**: data is sent through communication channels and/or multicast groups from producers to consumers.

Routing spaces of the RTI are general and allow implementation of many multicast schemes. Grid-based filtering for example could be implemented using a region per cell. Filtering based on perception of entities, on geographic locality, frequency domain etc. are other examples of the kind of filtering that routing spaces are able to handle.

### 2.3 Commercial Platforms

#### 2.3.1 NetZ

NetZ (http://www.proksim.com) is a set of tools designed to simplify and accelerate the development of small and ephemeral online games. However, its basic concepts are quite general and could be used in the design of large-scale persistent worlds.
System Architecture

NetZ is based on a peer-to-peer architecture though it may be configured to feature a client-server architecture (all master replicas existing on the same server machine). Communications are point-to-point, using UDP and reliable UDP.

Object Model

NetZ proposes a high-level object Description Language (DL) used to describe world entities (called Duplicated Objects) and data-sets. The state of a duplicated object is composed of one or several data-sets. This language (the DL) allows the description of:

- Data-set: types and update policies (such as constant, reliable, unreliable, extrapolated etc).
- Duplicated Objects: data-sets, system call-backs, user call-backs and actions.

These definitions are then compiled into C++ classes, which the developer must extend to provide the implementation of call-backs and actions.

Object Distribution

All distributed objects are fully replicated on all replication sites. Amongst all replicas, one plays the role of the master. The platform propagates any action call on a replica to the master replica. Conversely, state changes on the master replica are propagated to other replicas when needed, in accordance with the dataset update policy description. Mastership can be transferred between simulation sites and its control is done by system call-backs.

Persistence and Fault Tolerance

Game reliability is the ability of a game to recover from faults. If a simulation participating in a game is disconnected due to a fault, there should ideally be no effect on other simulations. The disconnection of a station could be due to a player quitting a game, a network failure, power supply failure or some other fault. NetZ ensures game reliability by enabling duplicated objects to be fault tolerant such that when a station fails, all the duplicates of fault tolerant objects on the failed station are promoted to duplication masters on the game master where the initial game master is the station that initiates the game. In a situation where the game master fails, NetZ automatically elects a new game master. This way, stations may fail, but NetZ ensures that duplicated objects continue to persist in the virtual world. Hence persistence is ensured by mastership transfer. System call-backs allow the developer to control which entities are persistent and which ones are not (e.g. a player’s avatar is generally not persistent).

Load Balancing

The expectation is that master replicas are more CPU demanding than slave replicas. Thus, it may be of interest that master replicas are not in a single simulation but spread all over processes. The system may automatically transfer mastership in order to obtain the same number of masters on each site. However, this is not suitable for entities like avatars which must remain under control of the player. System call-backs allow specification of whether or not an entity can be migrated.
Data Synchronisation

NetZ synchronisation is defined at two levels:

Static: the data-set description level allows definition of a given data-set type as constant, reliable (data-sets that do not change often), unreliable (data-sets that change often and of which the loss of a state change is considered acceptable), extrapolated (data-sets that change continuously like positions for which the loss of a state change is considered unacceptable).

Runtime: whenever the method update is called on a distributed object, the system checks the state of data-sets and communicates state changes according to specified policies:

- constant datasets are never communicated
- reliable dataset changes are sent using a reliable protocol
- unreliable dataset changes that are sent using an unreliable protocol
- extrapolated dataset changes that are checked against a prediction model; changes are sent only if there is a significant change or a continuity break. Dead-reckoning techniques can be customised by the developer.

The game logic may in principle be programmed without being aware of the distributed nature of world entities, since the platform transparently takes into account the creation/deletion of replicas in any simulation. It also redirects action calls to master replicas and state changes to replicas.

2.3.2 Blaxxun

Blaxxun Community server (http://www.blaxxun.com) is a VRML server that permits incorporating 3D, multi-user interaction into a web site. Its aims at co-operating with a HTML server to create user communities for business, education and entertainment. The Community server supports the operation, administration and usage tracking of virtual worlds. Applications include entertainment (e.g. canal plus second world, ’Virtual Paris’), e-commerce (e.g. Virtual Munich airport centre) or business applications.

World Characterisation

A server can support a virtual world with several thousands of simultaneous users. The world is partitioned into ’rooms’, a given client is aware of only one room’s contents. In terms of interaction, Blaxxun users are able to see each other’s avatars and communicate via text chat channels, as well as maintain the state of shared objects (like a chess board). There is no support for real-time interaction.

Object Model

The object model sticks to VRML in which a VRML scene corresponds to each ’room’ of the world. All client sites seeing a given scene share the same initial VRML description and warn the server of any avatar movement. The server keeps track of all avatar positions and forwards position updates to ’interested’ users. The server dynamically computes the closest neighbours to the user and updates their positions more frequently than those that are far away. This process keeps the number of avatars the user gets updated positions of to a fixed minimum.
Object Distribution

Blaxxun uses both TCP/IP and UDP for communications between the client and the server. TCP/IP is used for infrequent, low priority, but delivery critical information while UDP is used for frequent but non-critical information. For instance, download of the 3D world is done via TCP/IP while motion updates are transmitted via UDP.

3. Categorisation of Techniques

For ease of comparison, we divide the techniques into two broad categories thus:

- Application-specific techniques
- Platform-specific techniques

3.1 Application-specific Techniques

These approaches have been employed by application programmers as a contribution to optimisation of DVEs they design. It could further be sub-divided into two parts:

3.1.1 Object Management Approaches

Object Servers:

The use of separate processes for each object can be inefficient since it often requires substantial resources if each process is to be hosted on a separate processor. As a result, this technique proposes that a number of processes be run concurrently on a single processor. In some systems such as AVIARY (West92) and RhoVeR (Bangay96b), there is support for a single process to run the code corresponding to a group of object processes. This reduces resources required, decreases the quantity of context switching and allows migration of object processes from one object server to another for load balancing. However, the cost of inter-node communication does not change significantly in proportion to the decrease in processors and as such, the advantage of localised processes is offset by increased inter-node communication.

Synchronous Databases:

This technique is used as a solution to the problem of keeping a distributed world representation consistent across all nodes. It involves the use of database locking to ensure consistency with changes to objects requiring distribution to all nodes. In early versions of DIVE (Frecon98), the ISIS toolkit (Birman90) uses a multicast protocol to distribute changes and set locks. All nodes are guaranteed to have seen the same sequence of events, which while good for system integrity, provides limits on scalability for instance in DIVE where an upper limit of ten peers was set.

Asynchronous Lossy Databases:

The use of synchronous databases involves maintenance of identical copies for all nodes. Though the locking mechanism causes performance loss, many current VR applications are content if provided with the most recent version of data regarding the state of the world; consistency between simulations is not of paramount importance. The Virtual Environment Operating Shell (VEOS) (Coco93) system for instance uses this principle when performing updates between nodes.
The latency associated with messages between nodes experiences some variation when communication time dominates over the processing time in a node. Discontinuities occur in the value of the latency suggesting the possibility of sudden changes in the performance of a system using this approach.

3.1.2 Application Programming Methodologies

Dead-reckoning:
This is an extremely useful technique in distributed VR systems that is used for data distribution in systems such as Distributed Interactive Simulation (DIS) (IEEE Computer Society95), Distributed Interactive Virtual Environment (DIVE) (DIVE Website, http://www.sics.se/DIVE/), and applications of the Minimal Reality (MR) Toolkit (Green92). It involves compensation for low communication bandwidths with increased processing by each entity in the system. Instead of just transmitting positional information, additional information such as the direction of movement and speed is included. This information is used by each entity to continuously update its estimate of the position of the others. Fewer updates are sent, since intermediate values can be ‘dead-reckoned’. Objects calculate their own dead-reckoned position and issue updates when this diverges substantially from their actual position. The Figure below illustrates a dead-reckoning algorithm.

![Dead-reckoning algorithm](image)

Experiments with dead-reckoning have proved that it uses infrequent updates and causes a decrease in the cycle time. It produces a linear dependence on the number of processors when using a shared communication medium, provided the number of processors is below a threshold level, determined by the communication time, and the interval between updates. It increases the processing overhead in each node due to the extra computation required.

Token passing:
This approach addresses the problems of distributed control and contention for shared resources within VR systems. Control of shared databases and the resolution of disputes becomes the responsibility of the process holding the token. Its use is illustrated in a MR Toolkit application in which players at different physical locations play a game of handball. The effect of relocating the larger portion of processing to the process carrying the token produces an improvement in the average cycle time, provided processors do not need to remain synchronised. While synchronous token passing can be used as an access control mechanism, asynchronous token passing is used as a distributed...
control mechanism for load balancing, and to improve interactive performance on a single processor. The latter application benefits from the use of dead-reckoning to counter saturation of communication links though frequent movement of the token eliminates most benefits.

Object Behaviour:

An important feature for persistent worlds is the ability to add new types of objects or modify entities’ behaviour at run-time. This feature is not common amongst contemporary VR systems. Attempts at exploitation of application level object behaviour include the use of scripting languages such as in DIVE where Tool Command Language (Tcl) scripts are associated with entities. This enables behaviours to be replicated along with entities and be executed on any platform without recompilation. The use of scripting languages also allows non-expert end-users to build complex behaviours by assembling elementary behaviours described in the language using for instance a graphical interface to associate a set of conditions to the triggering of events and actions. Recent research by INRIA has resulted in a purely graphical Java coded interface in which complex behaviours are built by ‘click-drag-and-drop’ actions on iconic representations of behaviour (Boussinot96). Other systems such as DEVA and Continuum enable dynamic code loading of object models and behaviours described in a compiled language. DEVA’s object types are described in a specific object-oriented language, the ‘DEVA language’, which is then compiled partly to binary code and partly to pseudo-code executed on a virtual machine. The DEVA virtual machine starts with a very poor set of built-in classes after which entities’ code is dynamically loaded on demand. Continuum uses the Remote Method Invocation (RMI) class loader, enabling new types of objects to be loaded on demand, provided certain conditions of accessibility and security are met. The code path is accessible to the hosting process, either on its own class path or provided with the new object, and the hosting process accepts to load the new piece of code.

3.2 Platform-specific Techniques

These are techniques that have been used at the middleware (platform) level to support real-time interaction between users that are dispersed geographically. Amongst the ultimate goals that have driven their application are scalability, heterogeneity and reliability. We classify this category into two:

3.2.1 Architectural Approaches

Distributed Architectures:

These are based on a spread-out topology and could be considered to exist in two main classes:

- The peer-to-peer model in which each peer can send messages directly to the other peer. There is no centralised server in the system and as such, each peer has a symmetrical role in relation with other peers. Objects that are instantiated at one site are automatically replicated at all other remote sites. There are two approaches that could be used in this kind of architecture. The first one is a logical abstraction which simplifies DVE application development at the cost of performance. Typically, in these implementations, a newly connected client must form point-to-point connections with all participating clients. Hence for $n$ participants, the number of connections required is $n(n-1)/2$. In addition, if the environment involves the sharing of enormous scientific data sets, the data set is fully replicated at each site. Unless the data sharing policy is modified to account...
for large data sets, this scheme is not scalable. The second approach reduces the impact of the performance overhead by dividing clients into multicast groups with unique addresses to which they subscribe. There are obvious advantages of the use of distributed architectures such as the potential they provide for scalability, their robustness and a lower network delay. However, systems in which they are used have to cope with complications in maintaining a consistent state of the virtual world; their reliance on multicast protocols that are not widely available or accessible (e.g. in modern internet connections). Further, the fact that there is no centralised server raises serious security issues.

- The multiple-server model in which the database is distributed amongst multiple servers. Clients connect to an appropriate server as needed and clients then subscribe to different multicast addresses to listen to broadcasts from servers. As mentioned above, client-server systems have been considered as having only a single system-wide server. In light of the advantages of grouping used in multicasting, the use of multiple servers which each provide a specific set of data have been used in systems such as VS (Lea'97). This topology provides a particularly effective way to handle large numbers of connected clients distributed over a wide virtual space. Each geographic region of the virtual space can be maintained by a separate server. The servers share the load of sustaining the state of the virtual world by handling only the subset of the connected clients that are in their geographic region. This could be considered an attempt at tackling the low levels of scalability, less robustness and high packet/message delays inherent in centralised single-server architectures.

Centralised Architectures:

As opposed to Distributed Architectures, these are based on a client-server model. One (or several servers) is responsible for handling message distribution for their clients. This client-server approach to data and distribution control is used in a number of VR systems. Centralised architectures have the advantage of making consistency and synchronisation issues easier to deal with when a single server holds a global state of the simulation. It greatly simplifies management of multiple clients, especially in situations requiring strict concurrency control. Additionally, in centralised architectures, persistence and security functionality are easier to introduce. However, its role as an intermediary for delivery of data can impose an additional lag in the system. They have a higher network delay than distributed architectures since there is always an extra latency incurred in going first through the server. Compared to distributed architectures, centralised architectures have a lower potential for scalability since a server (or servers) represent a bottleneck or message distribution and object state management. They have lower robustness since in the event that the server fails, the whole simulation halts. Despite these disadvantages, this architecture is still useful for support of small groups of collaborators.

3.2.2 Distribution Approaches

Broadcasting:

This technique is normally combined with dead-reckoning and involves transmission of updates in a single message that is received by all entities in the system. Replicated homogeneous topologies (using this mode of transmission) are classical of military VR simulations (as in DIS, SIMNET (Locke'93), HLA (Van Hook'96), NPSNET (Macedonia'95c)). In such topologies, each client holds a fully replicated database of the shared environment and state information is shared by broadcasting messages to all participating clients. This system has no centralised control whatsoever, hence any new client joining a session must wait and gather state information about the world that is broadcasted by the other clients. Broadcasting tends to require a dedicated
network since it is considered to be an undesirable communication technique when sharing a communication medium with other applications. It impacts on the performance of other nodes attached to the same network by forcing them to process packets that are not necessarily intended for them.

**Multicasting:**

This technique could be considered a variant of broadcasting with the advantage that messages can be sent to a subset of nodes. This makes it possible to send a single message to a set of processes. It uses the same principles as broadcasting, but allows selective communication between groups of nodes. Multicasting is becoming increasingly popular, particularly with large systems running over the Internet where many participants are distributed over large areas e.g. DIVE (DIVE Website, http://www.sics.se/DIVE/). A simple model of multicasting involves treatment of each multicast group as a broadcast system that does not affect other groups. Using the measurement of packet latency between two nodes in a network, the effect of broadcasting/multicasting in a Wide Area Network (WAN) is dependent on the propagation delay, provided sufficient bandwidth is available. However, as bandwidth decreases, latency increases substantially, particularly when message queues exist in the network. If local computation time is large compared to the network performance, then latency depends only on the computation time.

**World Partitioning:**

In a bid to achieve high levels of scalability, VR systems have adopted a variety of world partitioning approaches ranging from simple static ones to complex interest management techniques. These techniques are not exclusive to each other and amongst the most common approaches used are static coarse-grained partitions, perception-based approaches, interest management and information aggregation depending on level of details. As used in DIS, all these partitioning techniques make use of multicast (more precisely IP multicast) to partition the global simulation into disjoint communication groups. There is always a trade-off between the size of the partitions and the overhead of multicast group management since smaller the size of the partitions, the more precise the filtering but at the expense of a large number of multicast group ‘joins’ and ‘leaves’ since many ‘joins’ and ‘leaves’ per partition are captured. Additionally, these grid-based approaches have to cope with the current limits of multicast technology: the restriction on the number of distinct groups the multicast hardware can support and the time needed to join a new multicast group (up to a few secs).

**4. Summary**

This paper has introduced, compared and outlined a brief analysis of the performance aspects of techniques both at the application and platform level in distributed Virtual Reality. It has considered their use in contemporary DVEs and their respective performance aspects. It is evident that the main focus of current implementations is on design, testing and evaluation of scalability, responsiveness and persistence requirements imposed by distribution of user interaction over the public Internet. This however reveals a design gap whose fulfilment involves incorporation of flexibility. One way that flexibility could be achieved is through provision of support for dynamic adaptation strategies. These techniques, apart from introducing flexibility have the potential of providing a framework on which scalability, persistence and responsiveness requirements can be met.
Bibliography


(Blaxxun) ‘Blaxxun Interactive Website’, http://www.blaxxun.com


(DIVE) ‘DIVE Website’, http://www.sics.se/DIVE/


(Macedonia95c) Michael R. Macedonia, Donald P. Brutzman, Michael J. Zyda, David R. Pratt, Paul T. Barham, John Falby, John Locke, ‘NPSNET: A Multi-Player 3D Virtual


