ELSEVIER Parallel Computing 23 (1997) 991-1000

PARALLEL COMPUTING

Building the second generation of parallel/distributed virtual reality systems

Shaun Bangay [*], James Gain, Greg Watkins, Kevan Watkins

Department of Computer Science, Rhodes University, Grahamstown, South Africa

Received 1 June 1996; revised 7 November 1996

Abstract

We present the RhoVeR (Rhodes Virtual Reality) system and classify it as a second generation parallel/distributed virtual reality (DVR) system. We discuss the components of the system and thereby demonstrate its support for virtual reality application development, its configurable, parallel and distributed nature, and its synthesis of first generation DVR techniques.

Keywords: Distributed virtual reality; Ant Psychology

1. Introduction

The RhoVeR (Rhodes Virtual Reality) system is intended both as a flexible foundation for developing virtual environments and as a research testbed for investigating critical aspects of virtual reality (VR). The platform can be likened to an operating system in that it provides an environment and tools to support VR applications. We classify RhoVeR as a second generation VR system. In this scheme first generation systems are typified by an ad hoc approach, as different systems (such as VROS [3] and DVS [8]) explored different techniques with early VR hardware. By contrast, the second generation subsumes successful elements of the first generation and allows for greater structural expansion and alteration. There are four characteristics of RhoVeR which support this classification:

(i) *Configurable.* The RhoVeR system allows considerable flexibility within a well-defined structure. The benefits are threefold: as ever-more sophisticated VR equipment becomes available it can be incorporated into the system, the specific

[*] Corresponding author. E-mail: cssb@cs.ru.ac.za.

0167-8191/97/$17.00 Copyright © 1997 Elsevier Science B.V. All rights reserved.
PII S0167-8191(97)00040-9

topology requirements of an application can be catered for, and a spectrum of VR systems can be emulated for the purpose of simulation and fundamental research.

(ii) *Parallel.* RhoVeR supports both true parallelism, with multiple independent processes which communicate via message-passing and are spread across a network of processors, and pseudoparallelism (multiprogramming), with a number of processes hosted by a single processor.

(iii) *Distributed.* There are three techniques for distributing data amongst processes in a RhoVeR configuration. A sophisticated Virtual Shared Memory mechanism is used for data of global significance. Distribution is enhanced by shared tables and message-passing for faster, more direct communication requirements.

(iv) *Virtual reality support.* Virtual reality issues, such as latency and general performance, and the requirements of virtual reality entities, such as the creation and manipulation of worlds and objects, have received significant attention in the evolution of RhoVeR.

A discussion of the structure of the RhoVeR system and the realization of these four characteristics forms the remainder of this paper.

2. Structure

A central objective of the RhoVeR system is the rapid creation of VR applications. The challenge here is to balance the exigencies of versatility, so that the system can adapt to the burgeoning diversity of VR applications and support hardware, and structure, so that development is guided and accelerated by a ready-made framework. This balance is accomplished by extreme modularity, which allows the inclusion of additional features, such as support for new I/O devices, without sacrificing the existing structure.

A given RhoVeR application consists of a collection of event-driven processes, each of which executes independently and is an instance of a predefined module type. These module types define broad classes of VR system components, such as input, objects, worlds and output. The system supports multiple users, each of whom have access to a tailored collection of input and output module instances, but are not treated as a special case in the system. This makes the introduction of additional users simple and elegant. Communication between module instances is accomplished by three mechanisms: Event-Passing, which enables point-to-point communication between any two processes and is the substrate upon which the other two techniques rely, shared tables (ShapeData), which are cached within a process and distributed on demand, and a Virtual Shared Memory, which is a data structure accessible to all processes. The first generation system DIVE [1,2] makes use of a similar Event-Passing approach which is coupled with a replicated database visible as a shared memory block.

The components of the RhoVeR system can be separated into two categories: module types (Fig. 1) which can be likened to the bricks from which an application is constructed, and support libraries (Fig. 2) which provide the functionality to cement modules into a cohesive whole.

Name	Function
World	Centralized co-ordination and global attribute repository
VSM Manager	Control and distribution of shared memory structures
Input	Input data-capture and processing
Output	Rendering a view of the world
Object	Specification of attributes and behaviour of 3D virtual objects
Application Support	Collision Detection, Gesture Recognition and Timing

Fig. 1. Listing of module types.

2.1. Module types

– *World.* A world module provides centralised co-ordination of all modules within its domain. It is responsible for such housekeeping tasks as removing and merging processes, coordinating addressing services, and allocating system unique identifiers. The world module also stores global attributes, broadcasts global events and defines default responses to these events. For instance, gravity would be a global attribute and the world module would regularly transmit gravity events to which the default response would be falling. This type of separation of world and object attributes occurs in first generation systems such as AVIARY [10].

– *VSM Manager.* The VSM Manager enables access to Virtual Shared Memory data by all processes within a world and proliferates this VSM data across an application topology.

– *Input.* These interfacing modules are responsible for capturing and processing raw input device data and passing a stream of extracted information to the relevant module destinations. Currently implemented input modules include device drivers for the Polhemus InsideTRAK and Fifth Dimension Technologies 5thGlove.

– *Output.* RhoVeR output modules generate a view of the virtual environment. Each output module gathers information from the Virtual Shared Memory and locally cached

Name	Function
Control	Building hierarchies of Parenthood and Ownership
Event	Managing message-passing
ShapeData	Managing and distributing locally cached tables of object data
Process Management	Creating, merging and removing processes
Debug	Time-stamping and process-level debugging

Fig. 2. Listing of support libraries.

ShapeData as required. The principal advantage of not requiring that data be explicitly sent to an output module is that any number of such modules can be included with no other alteration of the system.

– *Object.* An object module describes both the behaviour and appearance of a three-dimensional shape within the virtual environment. The shape of an object is loaded from an OFF (object file format) file and then stored in a ShapeData table within the object process. Attributes which are subject to frequent change, such as position and orientation, are held in Virtual Shared Memory.

– *Application Support.* Other available modules include Gesture Recognition (which converts input data to a higher-level symbolic form), Collision Detection (which examines the VSM and generates events when objects collide), and a Timer Module (which enhances real-time control by generating tick events at specified intervals).

2.2. Support libraries

– *Control.* The control libraries support hierarchies of parenthood and links of ownership. A parenthood hierarchy stores the position and orientation of a child relative to its parent. An ownership link allows the controlling object (and no other) to alter aspects of a subsidiary object, such as position, orientation, and ShapeData.

– *Event.* These libraries implement a socket-based model of message-passing with capabilities such as inter-machine communication, buffering, and both blocking and non-blocking message receipt.

– *ShapeData.* Each module is provided with a locally cached ShapeData table, which typically contains high-volume externally accessible information such as an object's polygon-mesh structure. The ability to externally alter elements in a ShapeData table is also supported.

– *Process Management.* These routines cater for starting up and removing processes. Primitive load balancing is effected by killing a process on one processor and restarting it on another. Also, a merge function allows similar modules to be grouped within a single process which is controlled by simple multithreading, and consequently reduces the load of active processes. This merging is not automatic but must be instigated manually by the merging process. Merged behaviour is present in AVIARY [11], which employs Event-servers that allow a single process to control many objects.

3. Ant Psychology: An application

In this section a sample RhoVeR application is presented with the intention of illustrating the interaction of the various system components.

The Ant Psychology application consists of a collection of ants wandering around on a plane. These ants are generated by hives scattered across the plane. When a collision occurs the ants concerned are notified by a Collision Detection process and will tumble away from the point of impact.

This application was created as a testbed for RhoVeR. Each requirement of the application tests a different aspect of the system (Fig. 4). Fig. 3 shows the system in an entity-relationship format.

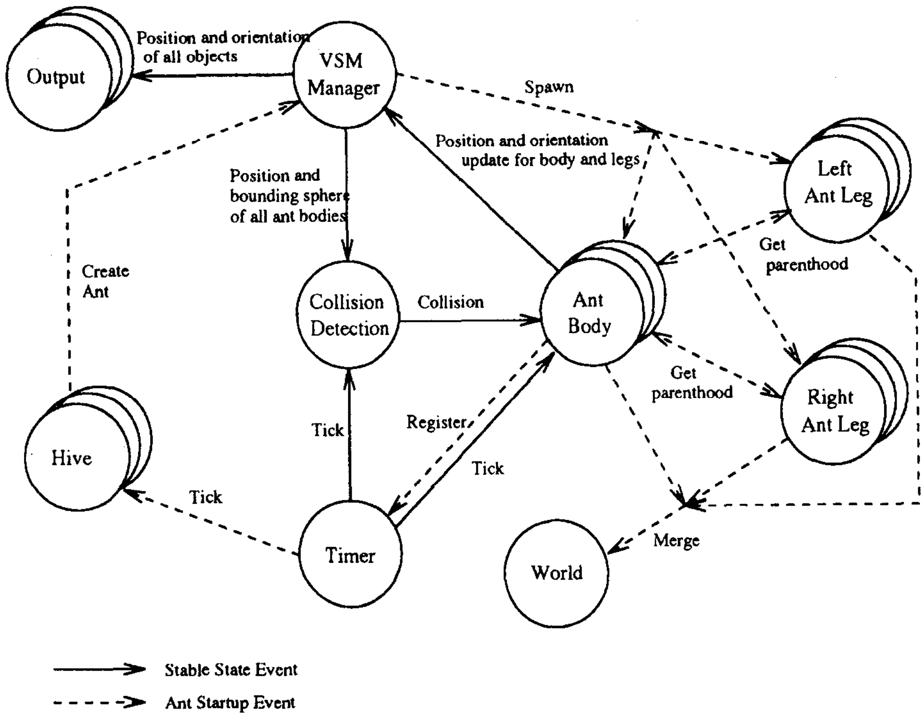

Fig. 3. Entity-relationship diagram for the Ant Psychology application.

On startup, World and VSM Manager processes are created. The VSM Manager in turn spawns a number of hive objects and output modules, a timer, a surface object, and a Collision Detection module. Only some of these modules are visible: the plane which demarcates the ant's world, a few pyramidal Ant Hives, and the Collision Detection module shown as an 'eye in the sky'. Each visible object has several ShapeData fields associated with it, including its colour, texture and polygon-mesh structure.

Ant Hives are represented by pyramids, the tips of which protrude above the plane. As new ants are born these pyramids move upwards revealing more of the buried shape and thereby denoting a larger hive. Initially each hive registers with the Timer module so as to be sent *tick* events at specified intervals. On receipt of a *tick* event, the hive requests the VSM manager to generate the three objects that form a new ant.

Application Requirement	System Component Tested
Arbitrary number of ants	Performance under variable load
Ant legs moved by ant body	Control
Hives spawn ants	Dynamic object creation
Collision detection	Data distribution

Fig. 4. Ant Psychology as a testbed for RhoVeR.

An ant consists of a box-like body and two legs. This two-legged ant was initially intended as the forerunner of a full six-legged ant, but we have become rather attached to the bipedal version. Immediately after being born, the body object, with which the ant's personality is associated, requests parenthood of its left and right legs, so that these become controlled by and positioned relative to the body. The new-born ant then requests the Timer module to transmit a *tick* event to it every few milliseconds. These are used to prompt the motion of the ant, a fraction of a step at a time. Ants walk in a fixed direction (which is occasionally randomly altered) and with a gait determined by a sinusoidal function. They continually monitor their position to avoid walking off the edge of the planar world.

The Collision Detection module functions as follows: each time it receives a *tick* event from the Timer module, it compares the bounding spheres of all object pairs and sends collision events to pairs that lie too close together. This prototype is intended as a test of the Virtual Shared Memory and is by no means a definitive Collision Detection method.

As the number of ants and hives increases there is a proliferation of processes and their corresponding overheads. To improve system-wide efficiency, a scheme for merging similar objects is implemented. Thus each of the following classes of object are subsumed into a single process: hives, left legs, right legs, and bodies from the same hive. The distinction that ant bodies not be merged indiscriminately allows ants from different hives to behave differently. There are some intricacies in this merging process: when two processes with different ShapeData or state variables are merged, these must be accommodated in the merged result. Once merging is accomplished the redundant process can be eliminated, thereby freeing up system resources.

Applications such as this Ant Psychology example can be overlaid on the basic RhoVeR system very easily.

4. RhoVeR as a second generation VR system

By definition the development of a second generation VR system must involve an analysis of the similarities and differences in existing (first generation) systems. RhoVeR goes further by allowing the use of these different approaches within a consistent environment with the intention of judging their relative performance characteristics.

The Event-Passing layer in RhoVeR is a basic support structure on top of which a variety of distribution techniques can be implemented. An examination of VR systems [4] has revealed that most are distributed using some form of message-passing for inter-process communication and synchronization. Using Event-Passing it is possible to implement constructs such as the client–server approach [3,9], the master-slave setup used in the MR Toolkit [6] or the dead-reckoning found in NPSNET [7].

Some high level distribution mechanisms have already been built into RhoVeR to facilitate research into virtual reality. These are the ShapeData extensions, which are unique to RhoVeR and Virtual Shared Memory, which is modelled on shared memory schemes found in other VR systems [1,2,5]. Both are built on top of an Event-Passing substrate.

5. Parallel aspects of RhoVeR

The RhoVeR system consists of a number of processes capable of executing concurrently, on one or more machines. To simplify discussion, the aspects of the system running on a single machine will be referred to under the heading of parallelism, while those involved with communication between machines will fall under the section on distribution.

On a single machine, usually running a variant of the UNIX operating system, RhoVeR processes are implemented as standard UNIX processes. Some measure of data sharing between these processes is required and this is implemented through a message-passing mechanism. The processes also communicate through a Virtual Shared Memory area, which on a single machine is easily implemented directly as a shared physical memory block. Every process has read-only access to the shared memory block, with limited write permission being determined by the ownership hierarchy.

Each process is event-driven, and acts in response to some external event. This has the advantage of simplifying scheduling, since once an event has been serviced the process will sleep until the next event. Load on the processor is consequently greatly reduced.

Since RhoVeR requires relatively large UNIX processes which typically consume significant resources even for trivial tasks, it is expedient to limit the number of active processes in the system. Object modules are generally the most abundant processes and are thus prime candidates for a system of process merging. This merging is accomplished after process creation and so provides a potential load balancing facility. Once in place, modules can migrate to less heavily laden processors, or even be moved in response to the analysis of traffic patterns.

An advantage of the event-driven nature of the system is that, provided a few conditions are enforced, the chance of deadlock can be significantly reduced, to levels typical of standard sequential programs.

A skeleton of the Event-response loop of a RhoVeR process is listed below:

```
loop
  Event=BlockGetEvent ();
  Call service routine for Event
forever
```

This process will remain dormant until an event arrives, after which it responds with an appropriate service routine. This routine will consist of either standard sequential code, or a call to the *SendEvent* or *GetEvent* routines. Neither the *GetEvent* nor the *SendEvent* procedures will ever block since communication is completely asynchronous, but *GetEvent* is often called repeatedly in a tight loop until an event arrives.

This allows the deadlock scenario outlined below:

```
Process P1
  Send P2 request
  Block until event from P2
Process P2
  Send P1 request
  Block until event from P1
```

Under certain conditions two processes can each be expecting a reply from the other which each is unable to provide until their requests are satisfied.

Some restrictions on the nature of events circumvent this problem. Consider a situation which involves communication between only two processes and at least one must wait for a response. We classify events which trigger a response as *server events*. By definition the process providing a response is the Server and the waiting process is the *Client*. Now consider the following restrictions:
- Clients may only block when waiting for a response from a Server.
- Blocked Clients must respond to Server events.
- Servers must return with no possibility of blocking.

Given these rules, a process can only block when waiting for a response from a server, and every process will always eventually respond to server events. The last stricture can be relaxed, provided the server acts as a client while it is blocked. It is a useful condition to include however, since it improves efficiency by limiting possible message paths and disallows infinite cyclic client–server chains. Communication involving more than two processes has to date been rare but can easily be remodelled as a client–server chain.

6. Distributed aspects of RhoVeR

RhoVeR is designed to be extremely versatile and applicable to any equipment configuration. A module may be located on any host, with the exception of the VSM Manager which is replicated on every host in use. Thus modules can be matched to machines with the appropriate computation and communication capabilities. For instance, output modules can be placed on graphics machines capable of rendering in hardware and computation intensive modules on high-end hosts.

The inter-process communication facilities offered under RhoVeR fit into three categories. The most basic form is direct Event-Passing. This is used to implement the other forms of distribution, as well as for any specialised communication required by a particular application. Part of the RhoVeR design called for a system suitable for research into distributed virtual reality (DVR) systems in general. The Event-Passing layer allows techniques from other DVR systems to be quickly and easily emulated. To qualify as a DVR system in its own right, RhoVeR provides more specialised communication facilities that simplify the creation of virtual reality applications.

The ShapeData extensions focus on sharing data amongst a subset of processes. This data is typically fairly large and is changed only at irregular intervals. It includes the polygon-mesh shape of an object (which the output modules access extensively) and colour and texture information, although it is not confined to visual aspects alone.

A particular ShapeData structure is owned and maintained by a single process, but duplicate portions are stored where required. When the owner changes an entry in its ShapeData, it sets a flag in the VSM and so signals associated processes to update their local versions.

The Virtual Shared Memory is the third inter-process communication facility. A block of memory is provided whose contents are distributed across every host in the system. It is structured as an array with a single small record element for each process. These records store details that are in demand by a spectrum of processes, such as the type, position, and ShapeData modification status of a process.

The VSM is implemented on each host as a block of shared memory. The local VSM Manager collects update events from processes with the appropriate permissions. It then changes the local copy and propagates these changes to VSM Managers on other hosts. The possibility of overwriting records has been eliminated by allowing only the owner process to write to a record but the synchronisation of VSM caches is not guaranteed. This, however, is acceptable within the virtual reality problem domain.

The Event-Passing mechanism is implemented using TCP/IP. This protocol has several advantages: it allows RhoVeR to function across the Internet and it guarantees the transmission of events. Other Internet compatible DVR systems rely on UDP, which does not support event acknowledgement and is hence capable of superior performance. However, its use mandates either additional checks for dropped messages, or the occasional loss of data. The implementation of UDP-based Event-Passing is a possible future enhancement to the system.

7. Performance issues

There are two main factors that affect the performance of a parallel/distributed system: the efficiency of data propagation and process computation. The focus of this section is on measures undertaken to improve these aspects in the RhoVeR system.

7.1. Data propagation

Each of the three data distribution methods (VSM, ShapeData, Event-Passing) is optimised for a different form of propagation. The VSM is intended for global access to small frequently altered elements of data. Since the VSM is resident on each machine and is implemented as a shared memory block, machine-wide update is immediate. The only delay is caused by data transfer across machine boundaries. The ShapeData mechanism supports large data structures that are only needed by a subset of processes and are changed infrequently. Retransmission of ShapeData is circumvented by cacheing local duplicates. Dedicated change counters within the VSM are used to signal that portions of the original ShapeData have been altered. In this way only the changed fields are retrieved. Low-level Event-Passing is restricted to direct point-to-point communication. An experimental flow-control feature is in place to reduce latency by preventing a glut of events from being sent from high-traffic processes.

7.2. Computational efficiency

An event-driven paradigm is computationally efficient, since processes sleep between handling events and do not consume significant processor time. This is further enhanced by merging similar objects, so as to reduce the load of processes on the system.

8. Conclusion

This paper has discussed four key facets of the RhoVeR (Rhodes Virtual Reality) system: the control mechanism, parallelism, data distribution, and performance issues.

- The control sections of RhoVeR allow the assembly of VR objects into parenthood hierarchies and support the notion of ownership.
- From a parallel viewpoint RhoVeR consists of a collection of independent event-driven processes, with facilities in place for merging processes and preventing deadlock.
- RhoVeR also has a sophisticated three tier approach to data distribution drawn in part from first generation systems: Event-Passing, ShapeData tables and a Virtual Shared Memory. Each of these levels caters for different communication requirements.
- The real-time considerations inherent in virtual reality make performance issues critical and the subsystems of RhoVeR have been analyzed and designed with this constraint in mind.

We believe that these aspects of the RhoVeR system support its classification as a second generation parallel/distributed virtual reality system.

There are several areas of possible expansion and research in the system: alternatives to TCP/IP as the underlying network protocol should be explored, extensive benchmarking and monitoring of RhoVeR performance is necessary and a scripting language would be helpful in specifying startup conditions.

References

[1] M. Andersson, C. Carlsson, O. Hagsand, O. Stål, DIVE – The Distributed Interactive Virtual Environment Tutorials and Installation Guide for DIVE version 2.2, Swedish Institute of Computer Science, Kista, Sweden, 1993.

[2] M. Andersson, C. Carlsson, O. Hagsand, O. Stål, DIVE – The Distributed Interactive Virtual Environment, Technical Reference for DIVE Version 2.2, Swedish Institute of Computer Science, Kista, Sweden, 1993.

[3] S. Bangay, Parallel Implementation of a Virtual Reality System on a Transputer Architecture, M.Sc. Thesis, Department of Computer Science, Rhodes University, 1993.

[4] S. Bangay, A comparison of Virtual Reality Platforms, Unpublished paper, available via anonymous ftp from cs.ru.ac.za as /www/vrsig/SDB04.ps.Z, 1994.

[5] W. Bricken, G. Coco, The VEOS Project, HITL Technical Report TR-93-3, Human Interface Technology Laboratory, University of Washington, 1993.

[6] M. Green, L. White, Minimal Reality Toolkit Version 1.4: Programmer's Manual, Department of Computer Science, University of Alberta, Edmonton, Alberta, 1995.

[7] M. Macedonia, M. Zyda, D. Pratt, D. Brutzman, P. Barham, Exploiting reality with multicast groups: A network architecture for large-scale virtual environments, in: Proc. of the IEEE Virtual Reality Int. Symp. '95, North Carolina, 1995.

[8] D. Pountain, ProVision: The Packaging of Virtual Reality, Byte, October, 1991.

[9] D. Schmalstieg, M. Gervautz, P. Stieglecker, Optimizing communication in distributed virtual environments by specialized protocols, in: Proc. of the 3rd Eurographics Workshop on Virtual Environments, Monte Carlo, Manaco, 1996.

[10] D. Snowdon, A. West, T. Howard, Towards the next generation of Human-Computer Interface, Informatique '93: Interface to Real and Virtual Worlds, March, 1993.

[11] A. West, R. Howard, R. Hubbold, A. Murta, D. Snowdon, D. Butler, AVIARY – A Generic Virtual Reality Interface for Real Application, Virtual Reality Systems, May, 1992.