

DIVE: a scaleable network architecture for distributed virtual environments

This article has been downloaded from IOPscience. Please scroll down to see the full text article.

1998 Distrib. Syst. Engng. 5 91

(<http://iopscience.iop.org/0967-1846/5/3/002>)

View [the table of contents for this issue](#), or go to the [journal homepage](#) for more

Download details:

IP Address: 38.107.179.211

The article was downloaded on 20/02/2012 at 20:58

Please note that [terms and conditions apply](#).

DIVE: a scaleable network architecture for distributed virtual environments

Emmanuel Frécon[†] and Mårten Stenius[‡]

Swedish Institute of Computer Science, Box 1263, SE-164 28 Kista, Sweden

Received 6 March 1998

Abstract. We introduce the network software architecture of the distributed interactive virtual environment platform. The platform is designed to scale with a large number of simultaneous participants, while ensuring maximum interaction at each site. Scalability is achieved by making extensive use of multicast techniques and by partitioning the virtual space into smaller regions. We also present an application-level backbone that can connect islands of multicast-aware networks together.

1. Introduction

Our goal is to build a framework for controlling the large data flows that are generated by networked collaborative virtual environments. We investigate the techniques to enable multi-user environments that are spatially large, possibly infinite, contain many detailed objects and visualizations, and allow many simultaneous participants to interact with each other, objects, and processes present in the environment.

This paper is organized as follows. In section 2, we describe the conceptual abstraction through which distributed interactive virtual environment (DIVE) applications transparently communicate with each other, namely via virtual worlds. We then introduce the design choices that allow DIVE to scale to a large number of users interacting in real time. Following this, we present the communication architecture of the platform and the different techniques with which applications communicate event and streaming data. Then, we describe two key applications that allow DIVE to run on the Internet and to adapt itself to the heterogeneity of such a network. Finally, we compare the DIVE architecture to other approaches.

In the first sections, we describe the DIVE philosophy using an argumentation roughly parallel to the architectural discussion developed in [1] to describe the *Spline* system. Although DIVE and *Spline* have evolved separately, the platforms share many common goals and design choices. We discuss some differences and similarities between the two systems in more detail in section 6.2.

[†] E-mail address: emmanuel@sics.se

[‡] E-mail address: mst@sics.se

2. The world as a common interaction medium

DIVE [2] provides an architecture for implementing multi-user interactive virtual environments. The architecture focuses on software and networking solutions that enable highly responsive interaction at each participating peer, i.e. interaction results are immediately shown locally at the interacting peer but slightly postponed at remote peers.

By *peer* we mean an application process running at a specific *host* (computer workstation). Different hosts are connected via a *network* (here the Internet). An *application* or *process* is any active program interfacing the virtual environment by presenting and modifying entities (see below), monitoring and reacting on different types of events, and so on. A typical application is the visualizer process that handles interaction and visualization of the environment for a specific user. Other applications may for instance perform animations and complex simulations, and present three-dimensional user interfaces for different purposes.

A central feature in the programming architecture of DIVE is the shared, distributed *world database*. All user and application interactions take place through this common medium. This is illustrated in figure 1, which shows five applications interacting through two different worlds. Application processes 1, 2, and 3 interact with each other through world 1. Applications 3, 4, and 5 interact with each other through world 2. Note that application 3 is interacting with both worlds.

The world database acts as a world *abstraction*, since DIVE applications operate solely on the database and do not communicate directly with each other. This technique allows a clean separation between application and network interfaces. Thus, programming will not differ when writing single-user applications or multi-user applications running

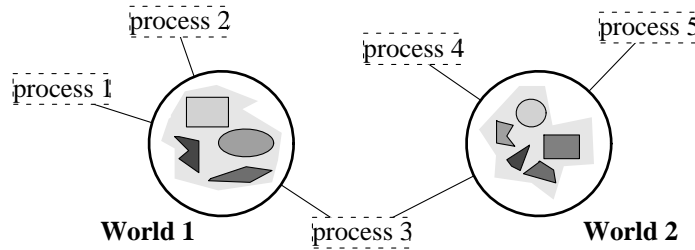


Figure 1. In Dive, separate processes interface and interact through one or more distributed world databases.

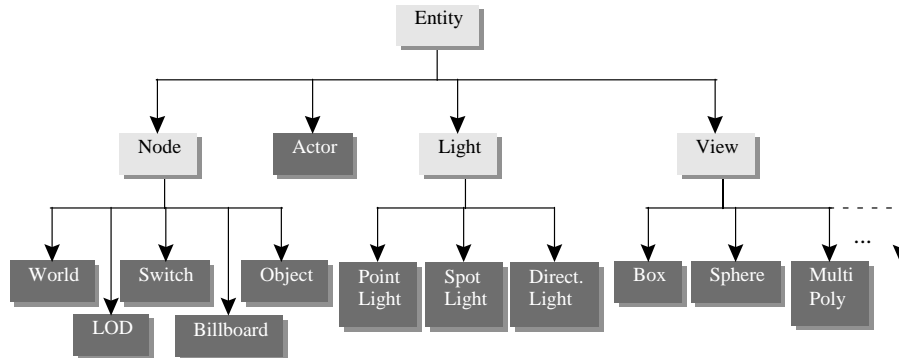


Figure 2. The simplified DIVE entity class hierarchy.

over the Internet. This model has proven to be successful while DIVE has changed its inter-process communication package three times and applications did not require any redesign, only minor code tweaking.

A DIVE world is a hierarchical database of what is called entities. DIVE entities can be compared with objects in object-oriented approaches, although DIVE is written in plain ANSI C. Entities are organized in a class hierarchy, depicted in figure 2. The database is hierarchical to ease ordering of information and rendering. In addition to graphical information, DIVE entities can contain user-defined data and autonomous behaviour descriptions. We believe that the three-dimensional rendering module should not dictate a structure to the content of the database. By making this distinction the system becomes more flexible, and can more easily be adapted to a large variety of rendering techniques.

Entity persistency is ensured as long as one application interacts with a world. When the last application ‘dies’, the world dies and entities will stop living. The next time an application connects to this world, it will reload its state from some initial description files. These descriptions are located by a file uniform resource locator (URL) [3], and thus may be located either on local storage, or remote servers on an internet. The current DIVE version handles persistency by running a monitoring process that periodically saves the state of the world to a file, from which the world database can be restored if the system needs to be restarted. These mechanisms are discussed in more detail in section 5.3.

3. Partial, active database replication

The DIVE architecture is based on active replication of (parts of) the database, so that a copy resides at each application process, as depicted in figure 3. This model allows applications to access the world database directly from memory, which provides low-latency user interaction.

Typically, entity additions, removals and modifications are done on the local copy first, then distributed to all connected peers through a network message and applied to the local copy at each receiving peer. By this we mean that the replication of the database is active. Conceptually, programmers can think of a ‘global’ central database residing somewhere on the network, as in figure 1, but the database is indeed replicated at each process, as in figure 3.

Traditional distributed database systems often enforce all processes to agree on the information state before committing updates. Such operations are difficult to combine with scalable real-time interaction. DIVE, on the other hand, focuses on highly interactive applications, and uses *partial* replication to remedy some of these problems.

- DIVE tolerates world copies that differ slightly and implements services that ensure their equality over time. Dead-reckoning techniques and run-time object update mechanisms are used to achieve this.

- DIVE divides the world into subhierarchies that are only replicated and used in between the small number of applications that have expressed an interest in a particular hierarchy.

Modifications to the database are applied locally first, then packed, transmitted and applied to all receiving copies. Thus, on the receiving side, world events are captured and transcribed into the database some time after they

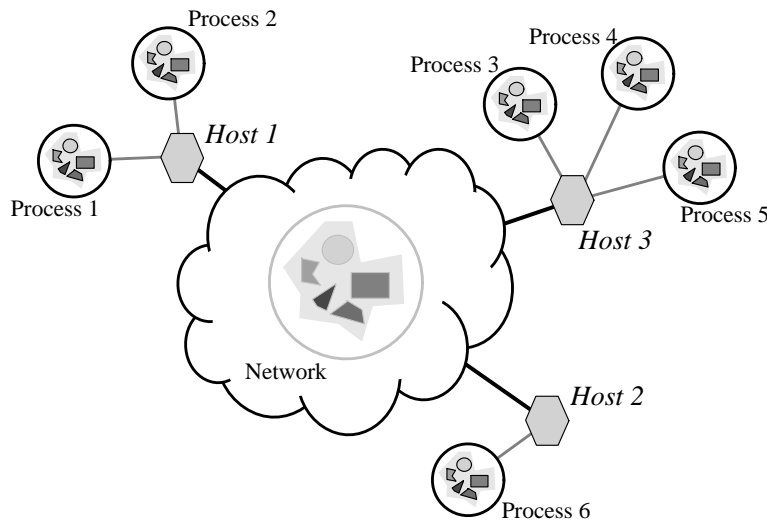


Figure 3. The Dive run-time architecture, simplified here to six processes interacting with one another in a single common world. Each process holds its own copy of the relevant objects, but the whole world can be seen as residing ‘on the network’.

occur. The length of this time is highly related to the network carrier separating the processes. Round-trip estimates measured in DIVE experiments over the Internet range from 25 ms within Sweden, 200 ms from Swedish to North American sites, and 800 ms to Japan. This is managed along with high packet loss rates. Despite this network latency, DIVE does not introduce differences between peers that are excessively large. The distribution principle fits with our experience of virtual environments: typically, entities will be modified in bursts and then stabilize into a common ‘inertia’, i.e. a common state that lasts. Differences introduced by latency and lost network packets are ‘healed’ over time by periodic synchronization using sequence numbers to keep track of entity versions, followed by possible update requests. As a result, with time, all connected peers will achieve the same ‘inertial’ state for these entities.

To remedy problems with network congestion and traffic overhead, and to offer the possibility for hundreds of participants to share a common place, DIVE provides a mechanism for dividing the world into subhierarchies that are only replicated and used in between the small number of applications that are actually interested in them. Each subhierarchy may be associated with a multicast communication channel, called a *lightweight group*. As a result, processes that are not interested in these subhierarchies can simply ignore this branch of the entity tree and cut down network traffic. The top-most entity, i.e. the world, is itself always associated with a multicast channel, and every world member process must listen to it. Lightweight groups are discussed in more detail in section 4.3.

4. DIVE communication architecture

4.1. Peer-to-peer communication

DIVE focuses on peer-to-peer multicast communication as opposed to a ‘classic’ client–server model. A typical

client–server architecture would require DIVE processes to communicate with a server each time an entity of the database is to be modified. Thus, such an architecture would have negative implications on interaction time and would introduce lags.

There are two types of messages sent by DIVE, they correspond to the different kinds of data sent during a session.

- DIVE assumes that all entities are likely to be modified often. Thus, database modifications are sent using a reliable multicast protocol detailed in the next section. Multicasting is used to minimize the number of message duplications between the sender and the receivers. *Reliable* multicast is necessary to ensure the postponed equality of all world copies.

- Continuous data streams (i.e. audio and video) are sent using unreliable multicast. Here, the emphasis is not on maintaining consistency between different replicas but on achieving sound or visual continuity from a series of distinct messages.

DIVE messages comply with the real-time transport protocol (RTP v2.0) recommendation [4] so that DIVE sessions can coexist with other Mbone applications on the same multicast channel.

4.2. Networking and multicast protocol

For non-stream-based data communication, DIVE pursues an idea originally in scaleable reliable multicast (SRM) [5] to reduce the amount of messages passing and thereby minimize network load and increase scalability. The method uses multicasting heavily, makes communication entity-based, and bases reliability on a negative acknowledgment request/response scheme.

In our approach, communication objects are equivalent to DIVE entities. A sending protocol peer does not need to store messages until their arrival has been confirmed by all recipients, as in TCP for example. Instead, it may

given peer. The peer has joined G_1 and G_2 , but not G_3 and G_4 . The lightweight groups that are not joined are marked with a lighter background.

4.4. Data streaming

DIVE sends continuous streams of data (i.e. audio and video) using unreliable multicast. Here, the emphasis is not on maintaining consistency between different replicas but on achieving sound or visual continuity from a series of distinct network messages. Reliability cannot be a key issue since packets retransmitting would only introduce lags in the continuous media and have negative implication on interaction.

DIVE associates separate audio and video multicast groups to each world. These groups are used for live communication. The current implementation does not associate separate audio and video to entities that carry a lightweight group reference. This implementation hampers scalability for obvious reasons and we are planning to remedy this problem.

For audio and video transmission, DIVE inherits some techniques from the popular Mbone tools VAT and VIC [10]. In particular, several encoding and compression methods are offered, in order to achieve a satisfactory balance between audio/video quality and network load. Finally, audio streams associated to database entities (i.e. object and avatar point audio sources) are mixed and spatialized to render a ‘soundscape’ that corresponds to the three-dimensional structure of the world database [11].

5. DIVE run-time architecture

5.1. The DIVE name server

The key application that allows DIVE processes to enter a world is the *DIVESERVER*. The *DIVESERVER* is a name server that exchanges a world name into a ‘ticket’ that will allow an application to connect to a world. This ‘ticket’ is simply composed of a multicast channel on which all further world communication will take place, with the exception of specifically marked subhierarchies, i.e. lightweight groups. The workload of the *DIVESERVER* is thus proportional to the number of processes that are willing to *enter* a world at a given time point, *not* to the total number of *connected* users during the session. Thus, while the *DIVESERVER* is the necessary centre point with which any DIVE process needs to communicate before being able to enter a world, it scales easily to hundreds of simultaneous participants since it only needs to deal with initial connection requests.

Initial communication with the *DIVESERVER* can be established either on a fixed multicast channel, or using a traditional point-to-point client–server request. Several *DIVESERVERS* can coexist on the Internet, but currently, applications that obtain their tickets from different *DIVESERVERS* will not be able to share the same world since they are likely to receive different global world communication channels from the different name servers.

Once a process has received its ‘ticket’, it will no longer communicate with the *DIVESERVER*. Instead, it

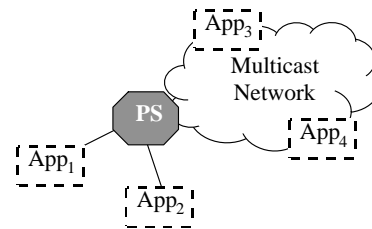


Figure 7. The DIVE proxy server as a simple message replicator.

will ask for the current state of the world by sending a request to the world multicast channel. If the process is the first one to connect, it will read the initial world status from several Internet locations[†]. Otherwise, it will receive an incremental state from already running processes, using regular point-to-point communication or multicast. As this state can represent large amounts of data, its total size is reduced using ZLIB compression [12]. DIVE uses a round-trip-time algorithm to find the nearest processes likely to answer.

5.2. The DIVE proxy server

For sessions involving peers located in different local networks, DIVE has long relied solely on the existence of the Mbone [10]—the IP Multicast backbone, a structure for interactive multimedia communication over the Internet. From the DIVE point of view, the Mbone presents some weaknesses.

- Despite its wide acceptance within the research community, the Mbone spreads slowly between subnetworks. Thus, running a worldwide DIVE session can require the intervention of several intermediate system staffs.
- Even if most of the operating systems now support multicast in their standard installation—sometimes as an option—some older versions do not or cannot be upgraded.
- In corporate networks, multicasting is still weak. As long as the corporate local network is composed of relatively new operating systems, it has an inherent support for multicast. However, Mbone connection, i.e. multicast connection with the outer world, is often missing.
- IP multicasting is not fully supported on different carriers such as ATM or ISDN lines.

Therefore, independence from the Mbone was recently gained by developing the *DIVEBONE*, an application-level backbone that can interconnect sub-islands with multicast connectivity and/or single local networks. The key application to the *DIVEBONE* is the DIVE proxy server. This application provides two different services on DIVE aware networks.

- The proxy server allows multicast unaware processes to join regular multicast sessions by acting as a message replicator for all connected DIVE applications. This is depicted in figure 7 and detailed below.

- The proxy server allows Mbone unaware networks to support DIVE multicast sessions by acting as a message

[†] Both the DIVE internal file format and VRML support inlining, which favours model description re-use.

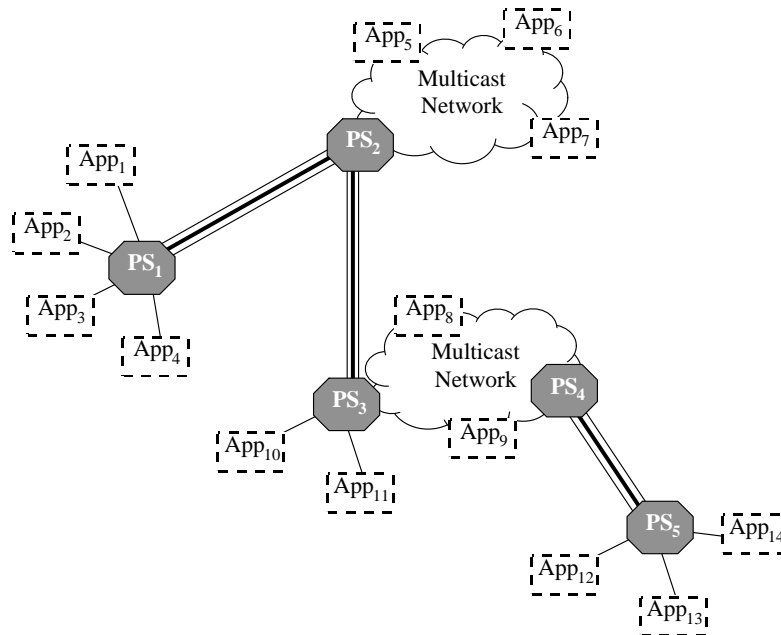


Figure 8. The Dive proxy server as an application-level multicast tunneller.

replicator for all connected proxy servers. This is depicted in figure 8 and detailed below.

The proxy server can act as a simple message replicator for individual clients. It catches all DIVE multicast messages sent by applications and replicates the messages to all connected clients. Similarly, each time a client sends a message, the proxy server will replicate the message to all other connected clients and to the multicast group. As an example, in figure 7, PS is a proxy server that will forward all data from App₁ to App₂ and the multicast network, and thus these messages will be caught and interpreted by App₂, App₃, and App₄. Similarly, all messages sent by App₂ will reach App₁, App₃ and App₄. PS will also forward messages coming from the multicast channel, i.e. from App₃ or App₄, to App₁ and App₂.

For simplicity, in the previous example we have considered that all the running DIVE applications had joined the same session, and thus were communicating through the same world database. However, the DIVE proxy server is able to perform software, high-level multiplexing in between different groups.

The current version of the DIVE proxy server is not aware of the worlds within which the different applications interact. It simply acts as a message dispatcher at the multicast level. However, a straightforward evolution is to build intelligence into the proxy server and use it to compress and preformat data before sending it to its direct clients, allowing them to run on low-bandwidth networks such as modem lines. Examples of preformatting include audio spatialization and mixing. Another evolution is to automatically duplicate the proxy servers in order to cope with the additional computational and network load introduced by new clients. Computational load is likely to increase if compression and preformatting are introduced.

The proxy server can also act as a multicast tunnelling application. Additionally, to serve directly connected DIVE applications, proxy servers can also connect to each other in order to bind together Mbone unaware multicast networks. Figure 8 depicts a complex example of five different proxy servers, named PS₁ to PS₅, that allow communication between 14 different applications, named App₁ to App₁₄. For simplicity, in this example we consider that all the running DIVE applications have joined the same session, and thus are communicating through the same world database. Again, however, proxy servers are able to perform software, high-level multiplexing in between different groups and serve different applications on different groups. Tunnelling proxy servers with one another constructs the DIVEBONE.

Because of the complexity of setting up a DIVEBONE architecture, proxy servers are guarded against cycles. Therefore, packets entering the DIVEBONE are marked with the identifier of the proxy server that introduced them and proxy servers drop packets that are received from two different sources. The DIVEBONE uses principles that are similar to the first versions of the multicast routing daemon, i.e. DVMRP [13].

A straightforward evolution of the DIVEBONE is again to introduce compression and preformatting in between proxy servers. Another interesting application of the DIVEBONE is application-level network analysis. The current version of the proxy servers can log network traffic and a companion application shows graphical statistics in real-time. The most interesting statistics gather separate logs for each message type sent by DIVE applications. Using these statistics, it has been possible to reduce the packet size for the most common operations, and thus to increase scalability.

5.3. Persistency managers

The current DIVE version handles persistency by running an impassive process, called PERSISTENT, that keeps its own active copy of the whole world database and periodically saves its state to a file, in order to achieve failure-less restarts. In order to strengthen persistency, several PERSISTENT processes can be run in parallel, preferably on different hosts and networks. This arrangement utilizes DIVE's active replication to its best by not requiring that restarting PERSISTENT processes read their initial state from their associated file. Instead, restarting PERSISTENT processes are likely to receive that state from another PERSISTENT process that is still running.

The existence of PERSISTENT processes is managed by another process called AUTOPERSISTENT. Its intention is double.

- AUTOPERSISTENT detects requests that are made to the DIVESERVER and automatically runs a PERSISTENT process for each world that is requested, if such a process is not already running on the host that AUTOPERSISTENT runs on. Automatic starting can be restricted to a set of worlds, enabling institutions to offer persistency for the worlds that they wish to make public.

- AUTOPERSISTENT supervises its associated PERSISTENT processes and restarts them on failure.

However, these techniques have still to be refined—many issues are still to be investigated in depth, such as how to delegate responsibility for persistence of different parts of a world, and how to handle object ownership in a general way.

6. Related platforms

The systems that are most related to DIVE are those that have been experimenting with the same ideas, such as loose consistency, no central servers and world subpartitioning.

6.1. The ancestors—SIMNET and DIS

In order to achieve scalability, i.e. hundreds of simultaneous participants, DIVE takes an approach that was pioneered by SIMNET [14] and DIS [15]. The key to this approach is that a distributed scaleable system can only achieve real-time performance if the view of the common data is not fully consistent throughout all the peers that participate in the simulation. DIVE shares some commonality with SIMNET and DIS.

- DIVE makes wide use of peer-to-peer communication.

- DIVE does not rely on any central service, for improved scalability and fault tolerance (except for the DIVESERVER, which supplies the initial ticket).

- DIVE uses dead-reckoning mechanisms to reduce the number of messages generated when objects move.

However, DIVE goes beyond DIS on some key aspects.

- DIVE is a complete platform for implementing interactive multi-user virtual environments. DIVE is not only a communication protocol.

- DIVE supports both audio and video communication. We believe that spatialized audio, and in particular live communication, is a key to the success of virtual environments.

- Introducing new objects at one site does not require that the other sites can access an external description of the objects, typically a file containing three-dimensional data. When a peer introduces new entities in the world, these are automatically transmitted to all connected sites, using DIVE specific mechanisms.

- Peers do not necessarily own objects. As a result, a peer will be able to connect, introduce an autonomous object and leave. The autonomous object will continue living, even after the peer has left.

- DIVE divides the world into subregions that are associated with lightweight communication groups. While NPSNET (Naval Postgraduate School Net) [16] also supports subpartitioning, it is less flexible since the subregions have a predefined shape, i.e. tiles of hexagons.

- DIVE network messages do not contain whole objects, but rather self-contained modifications to be applied to entities.

- DIVE does not require that peers send keep-alive messages. Instead, we have based our approach on SRM and are able to detect missing data packets on reception.

Much research on networking and protocols is being done in the context of DIS and DIS-like systems. Of particular relevance to the DIVEBONE architecture is the currently planned *CBone* (*Cyberspace Backbone*) [17], which also aims to serve as an application-driven replacement for the MBone, providing new services such as bandwidth reservation and dedicated multicasting. The existence of the CBone is bounded to the emerging effort for a standardized network protocol for large-scale virtual environments: the virtual reality transfer protocol (VRTP) [18].

Furthermore, among current work on reliable multicasting is *LBRM* (*log-based receiver-reliable multicast*) [19], which provides a way for detecting data loss on the receiving side, by employing 'heartbeat' messages and distributed logging.

6.2. A comparison: the spline system

The scaleable platform for large interactive network environments (Spline) [1,20] is one of the software platforms that resemble DIVE most. Spline has now evolved into a proposed open standard for multi-user virtual worlds: open community [21], along with its communication protocol: the interactive sharing transfer protocol (ISTP). However, the principles that drive open community are similar to those that lie behind its ancestor.

Similarly to DIVE, Spline uses peer-to-peer communication and a derivative of SRM. As with DIVE, Spline has evolved from a pure multicast approach to a mixed client-server and multicast approach, in order to cope with low-bandwidth networks. The role of the Spline session manager can be loosely compared to that of the DIVESERVER, even though the role of the DIVESERVER is not as interactive, but rather a mostly passive name mapping service.

Spline divides the universe, which is called the *world model*, into subregions called *locales*, each associated with a multicast group, a technique that can be compared to the lightweight groups of DIVE. DIVE supports coarse-grained partitioning of the whole universe by introducing worlds and gateways between worlds. DIVE also supports fine-grained partitioning of the worlds by introducing the lightweight group, a flexible application-dependent mechanism for implementing subregions. Spline, however, does not introduce this partitioning mechanism, but introduces locales as a complete concept for interrelating and subdividing worlds and regions.

Spline has abandoned Scheme—a scripting language—as its language for autonomous object description; instead, Java, a strongly structured language will be used. We believe that scripting languages are more appropriate, since they allow rapid and interactive prototyping. DIVE uses the tool command language (TCL) [22] as its main behaviour language and has directly benefited from the release of version 8, which introduced a byte-code compiler and offered an improved execution speed. DIVE supports Java through a prototype external Java-to-TCL interface, which hides TCL under a strongly typed class hierarchy.

6.3. Community place

Community place [23] is a system developed at Sony, that partially stems from a research collaboration with SICS [24]. The system uses a layered communication architecture where client-server communication is used on low-bandwidth networks, i.e. on modem lines to the actual users, and peer-to-peer communication is used for long-distance communication between servers. The browser is tuned for low-cost PC hardware, to promote a wide spread of the platform, uses VRML 2 for world descriptions, and features server-based Java as a behaviour language.

6.4. MASSIVE-2

MASSIVE-2 [25,26] was developed at the University of Nottingham. It is based on the use of *third-party objects* [27], which act as mediators of awareness and relationship between other objects in the environment. The third-party objects are used as a general mechanism for filtering, structuring and partitioning of the shared environment, with IP Multicasting as a basis for communication. MASSIVE-2 addresses issues such as nested world structuring, dynamic view abstraction and aggregation, and the management of common foci for several participants.

6.5. Distributed worlds transfer and communication protocol

The distributed worlds transfer and communication protocol (DWTP) is an application layer protocol [28]. Like most of the systems listed above, including DIVE, DWTP is heavily based on multicast and offers a layer on top of UDP/IP to ensure reliability of data transmission. Furthermore, DWTP follows a trend that is common to both DIVE and Spline, i.e. the introduction of specific processes, which scope and set of tasks are restricted to a few operations.

As compared with the communication layers of both Spline and DIVE, the major disadvantage of DWTP is that it realizes reliability using positive acknowledge messages (ACK), sent from specific processes in an attempt to lower the network traffic overload generated. Spline and DIVE use both a variant of SRM, which is based on negative acknowledge messages (NACK). DIVE avoids NACK implosion by sending these protocol messages using multicast. On missing packet discovery, the sending of NACK messages is randomly postponed. Consequently, all the peers that have discovered the same failure will cancel their own NACK sending, since packet resending will be done using multicast and is then likely to reach them as well.

6.6. Other recent work

Recently, much practical work on shared virtual environments on the Internet has focused on the development of the virtual reality modelling language (VRML) [29]. Most of these systems support a static background scene and loose sharing of avatars. Furthermore, most platforms rely on simple client-server communication, often together with a single centralized server. As we have pointed out before, interactive client-server based systems suffer from higher latency and from a difficulty in scaling. Finally, these systems usually support only distribution of avatar motion, at a filtered low-frequency update. Under those conditions they are able to claim high numbers of simultaneous participants. DIVE, in contrast, focuses on an architecture that supports unfiltered free motion of any entity, including avatars and other reacting autonomous objects, in order to achieve high-quality interaction and communication between participants.

7. Conclusion

In this paper, focus has been placed on how to support a system that is scaleable both with respect to the number of users, and to the network distance between different users on heterogeneous networks. In particular, two such techniques stand out.

- *The DIVE proxy server and the DIVEBONE* provide tools for an application-level virtual multicast backbone, that can be used in heterogeneous settings where connectivity in other ways is limited, and to create distributed, but separate, application-specific DIVE networks. Furthermore, DIVE becomes a hybrid system that can both be viewed as a client-server and a peer-to-peer system, with increased flexibility for varying demands.

- *Lightweight groups* are a database-level mechanism that allow a partitioning of distributed environments into subsets that can be requested or disregarded based on custom semantics. This technique can be used as a general high-level application-dependent scheme for experimenting with issues such as optimizing data transfers, database sizes, view culling, object and world abstractions, and so on.

Achieving scaleable virtual environments requires an adequate architecture such as the one described in this document. However, we believe that application-level

techniques are also required to reduce the number of network messages sent and received by peers that are connected to a common environment. The common theme of the techniques that we are experimenting with is that they intend to replace a group of low-level unary operations by semantically rich operations that only require a few network messages for their transmission. We are currently planning to implement or are implementing the following.

- Operation aggregation controlled from the application. This is particularly useful when realistically animating virtual humans: all limb rotations can be grouped together into a single operation.

- Adding an animation language that is triggered on event reception. Animations are distributed to all peers and thus can be executed locally and simultaneously at all peers.

- Extending scripting language support. Scripts are distributed together with the database and can, in some circumstances, be executed locally and simultaneously at all peers. While animations are tuned for visual authoring tools, scripts can be used to add intelligence to the animations and glue together different animations. Support for customised scripting languages is also under development.

The DIVE system has developed from a lab tool to an evolving, general-purpose platform for networked, shared virtual environments. While this paper has focused on the database and network levels, the system is used for experimentation with application, user interfaces and interaction in a wide variety of settings. DIVE is an open research platform. Binaries are available for downloading, for noncommercial use, at <http://www.sics.se/dive>.

Acknowledgments

We would like to thank three key persons without whom DIVE would never have existed. Olof Hagsand has been the primary architect of the DIVE platform for a long time and many of the principles and design choices described in this article are direct results of his research. Olov Ståhl has been working on DIVE almost since the project started and his knowledge and understanding of all the internal mechanisms are extremely valuable. And without the visions and strong will of Lennart Fahlén, the platform would never have become the general testbed for shared virtual spaces it is today. Finally, we gratefully acknowledge the EU and the different international companies that have sponsored our work through the years. In particular, the DIVEBONE would never have existed without the support from EU's ACTS Programme (COVEN AC040).

References

- [1] Waters R, Anderson D, Barrus J, Brogan D, Casey M, McKeown S, Nitta T, Sterns I and Yerazunis W 1997 Diamond park and spline: social virtual reality with 3D animation, spoken interaction and runtime extendability *Presence* **6** 461–81
- [2] Hagsand O 1996 Interactive multi-user VEs in the DIVE system *IEEE Multimedia Mag.* **3** 30–9
- [3] Berners-Lee T, Masinter L and McCahill M 1994 *Uniform Resource Locators (URL)* Internet RFC 1738
- [4] Schulzrinne H, Casner S, Frederick R and Jacobson V 1996 *RTP: A Transport Protocol for Real-time Applications* Internet RFC 1889
- [5] Floyd S, Jacobson V, McCanne S, Liu C-G and Zhang L 1995 A reliable multicast framework for light-weight sessions and application level framing *Proc. ACM SIGCOMM 95* (Cambridge, MA: ACM) pp 342–56
- [6] Pink S, Saulsbury A and Hagsand O 1995 OS 6—a distributed operating system for the next generation of computer networks *Proc. IEEE IWOOS'95: Int. Workshop on Object-Orientation in Operating Systems* (Piscataway, NJ: IEEE) pp 48–51
- [7] Fahlén L E, Ståhl O, Brown C G and Carlsson C 1993 A space based model for user interaction in shared synthetic environments *Proc. INTERCHI'93* (Reading, MA: Addison-Wesley) pp 31–7
- [8] Hagsand O, Lea R and Stenius M 1997 Using spatial techniques to decrease message passing in a distributed VE *Proc. VRML'97: The Int. Symp. on the Virtual Reality Modeling Language* (Monterey, CA: ACM)
- [9] Snowdon D, Greenhalgh C and Benford S 1995 What you see is not what I see: subjectivity in virtual environments *Proc. FIVE'95: Framework for Immersive Virtual Environments* (London)
- [10] Kumar V 1995 *MBone: Interactive Multimedia On The Internet* (Indianapolis, IN: Macmillan) ISBN 1-56205-397-3
- [11] Adler D 1996 Virtual audio—three-dimensional audio in virtual environments *Swedish Institute of Computer Science (SICS) Internal Report* ISRN SICS-T-96/03-SE
- [12] Deutsch P and Gailly J-L 1996 *ZLIB Compressed Data Format Specification Version 3.3* Internet RFC 1950
- [13] Pusateri T 1997 *Distance Vector Multicast Routing Protocol, IETF—Work in Progress* draft-ietf-idmr-dvmp-v3-05
- [14] Calvin J, Dickens A, Gaines B, Metzger P, Miller D and Owen D 1993 The SIMNET virtual world architecture *Proc. IEEE VRAIS'93: Virtual Reality Ann. Int. Symp.* (Los Alamitos, CA: IEEE Computer Society Press)
- [15] 1993 *Standard for Information Technology, Protocols for Distributed Interactive Simulation* American National Standards Institute, DIS-ANSI/IEEE Standard 1278-1993
- [16] Macedonia M, Zyda M, Pratt D, Brutzman D and Barham P 1995 Exploiting reality with multicast groups: a network architecture for large-scale virtual environments *Proc. IEEE Comput. Graphics Applicat.* **15** (5) 38–45
- [17] Brutzman D, Zyda M and Macedonia M 1996 Cyberspace backbone (CBone) design rationale *Proc. 15th DIS Workshop on Standards for the Interoperability of Distributed Simulations* (Orlando, FL: Institute for Simulation and Training)
- [18] Brutzman D, Zyda M, Watson K and Macedonia M 1997 Virtual reality transfer protocol (VRTP) design rationale *Proc. Workshops on Enabling Technology: Infrastructure for Collaborative Enterprises (WET ICE): Sharing a Distributed Virtual Reality* (Cambridge, MA: MIT)
- [19] Holbrook H W, Singhal S K and Cheriton D R 1995 Log-based receiver-reliable multicast for distributed interactive simulation *Proc. ACM SIGCOMM '95* (Cambridge, MA: ACM) pp 328–41
- [20] Barrus J, Waters R and Anderson D 1996 Locales: supporting large multiuser virtual environments *IEEE Comput. Graphics Applicat.* **16** 50–7
- [21] 1997 *Open Community Web Site* <http://www.meitca.com/opencom/>
- [22] Ousterhout J 1994 *Tcl and the Tk Toolkit* (Reading, MA: Addison-Wesley)

- [23] Lea R, Honda Y, Matsuda K and Matsuda S 1997 Community place: architecture and performance *Proc. VRML'97: The Int. Symp. on the Virtual Reality Modeling Language* (Monterey, CA: ACM)
- [24] Lea R, Honda Y, Matsuda K, Hagsand O and Stenius M 1997 Issues in the design of a scaleable shared virtual environment for the internet *Proc. 30th Hawaii Int. Conf. on System Sciences (HICS-30)* (Maui, HI: IEEE)
- [25] Greenhalgh C 1996 Dynamic, embodied multicast groups in MASSIVE-2 *Technical Report* Department of Computer Science, The University of Nottingham NOTTCS-TR-96-8
- [26] Greenhalgh C 1997 Analysing movement and world transitions in virtual reality tele-conferencing *Proc. ECSCW'97: The 5th European Conf. on Computer Supported Cooperative Work* (Lancaster: Kluwer Academic) pp 313–28
- [27] Benford S and Greenhalgh C 1997 Introducing third party objects into the spatial model of interaction *Proc. ECSCW'97: The 5th European Conf. on Computer Supported Cooperative Work* (Lancaster: Kluwer Academic) pp 189–204
- [28] Broll W 1998 DWTP—An internet protocol for shared virtual environments *Proc. VRML'98: The Int. Symp. on the Virtual Reality Modeling Language* (ACM SIGGRAPH) pp 49–59
- [29] 1997 *The Virtual Reality Modelling Language* ISO/IEC Draft for International Standard 14772-1, <http://www.vrml.org/Specifications/VRML97/DIS/>