DWTP — An Internet Protocol For Shared Virtual Environments

Wolfgang Broll

GMD —German National Research Center for Information Technology
Institute for Applied Information Technology (FIT)

Abstract

VRML (the Virtual Reality Modeling Language) has brought 3D objects and virtual worlds to a large number of Internet users. While it provides a suitable basis for the platform independent description of virtual worlds, an appropriate network architecture required to realize shared virtual worlds on the Internet is still an open issue.

In this paper we will introduce DWTP (the Distributed Worlds Transfer and communication Protocol). DWTP is an application layer protocol for shared virtual environments on the Internet. It provides a scalable network architecture for large-scale distributed virtual worlds.

CR Categories and Subject Descriptors: C.2.2 [Computer Communication Networks]: Network Protocols; C.2.4 [Computer Communication Networks]: Distributed Systems - *Distributed applications*; H.5.1 [Information Interfaces and Presentation] Multimedia Information Systems - Artificial Realities; I.3.2 [Computer Graphics]: Graphics Systems - *Distributed/network graphics;* I.3.7. [Computer Graphics]: Three-Dimensional Graphics and Realism - *Virtual Reality;*

Additional Keywords Virtual reality modeling language (VRML), distributed virtual environments, IP multicasting.

1 INTRODUCTION

Research on distributed virtual environments has been stretched out for several years with remarkable success on various aspects of it [8] [15]. Most systems realized did not achieve a large spread however, since they were limited to certain platforms or specific networks. VRML—the Virtual Reality Modeling Language— gives us for the first time the opportunity to develop large scale virtual worlds inhabited by world-wide distributed participants independent of a specific platform or operating system.

The current VRML standard [1] however, does not provide any support for sharing VRML worlds with other users. While a

GMD - FIT.CSCW, D-53754 Sankt Augustin, Germany
email:wolfgang.broll@gmd.de

number of approaches have been made to extend VRML in order to support multiple users [12], their representation by avatars as well as the sharing of objects [9], most of them rely on simple network architectures based on central servers [6]. An appropriate scalable network protocol for large distributed virtual environments on the Internet is still an open issue.

More recent proposals for new network protocols or architectures such as VRTP [5] and ISTP [2] indicate, that a flexible, scalable and universal network protocol for shared virtual worlds will be based on a heterogeneous approach including several basic internet protocols rather than using a single distribution scheme.

In this paper we will present our approach of an application independent network protocol for shared virtual environments. Our approach was guided by the following considerations:

- the protocol should be application (content) independent (not limiting it to a specific VRML version or VRML at all)
- the protocol should not rely on a particular underlying network layer
- the user (application programmer) should not have to deal with the underlying network layers
- the protocol should be scalable to a large number of world-wide distributed users
- the protocol should support the transfer of all data types required for sharing virtual worlds and realizing collaborative virtual environments

In the second section of this paper we will identify the requirements for network protocols for distributed virtual environments and compare those with existing application layer protocols for the Internet. In the third section we will introduce the basic architecture and components of DWTP—the Distributed Worlds Transfer and communication Protocol. The fourth section finally shows how the DWTP components can be used to create distributed VR applications.

2 REQUIREMENTS AND RELATED PROTOCOLS

In this section of the paper we have a look on the requirements on a network protocol for distributed virtual environments. We will further review some existing protocols for their suitability to be used for distributed VR on the Internet.

2.1 Requirements for DVE Protocols

In order to share virtual worlds across the Internet the two basic requirements are the transfer of the world contents to the individ-

ual participants and the distribution of all changes made to these contents. To transfer the worlds contents a scene description language such as VRML is required. The descriptions themselves will usually consist of one or several rather large files. Once these files have been transmitted and a local database has been created all changes to the worlds contents have to be transmitted to all other participants in order to keep the distributed scene data bases consistent and by that achieving the impression of single shared virtual world. These changes usually consist of rather small events, changing e.g. the transformation or the color of a object. If new objects can be introduced by participants, those have to be transmitted as well. The same applies if participants (users) are able to provide their own avatars. The latter have to be distributed to all other participants as well, but require usually a much larger amount of data to be transferred than events. In collaborative virtual environments additional services might be necessary. In order to support cooperation between participants audio and video streams might be transmitted between some or all participants.

In addition to the different types of data which have to be transferred, there exist individual requirements for the reliability of the services. While for example events containing the current transformation of a user's avatar are transmitted quite frequently and for that reason do not require a reliable transmission, other events might be essential for keeping the impression of a shared virtual world and therefore have to be guaranteed to be transmitted.

Thus we can identify the following data types to be transferred over the Internet for realizing shared virtual environments:

- reliable peer-to-peer transfers of (large) files
- reliable and unreliable transmission of (small) events
- reliable transfer of (medium-sized) files to a group
- unreliable transfer of stream data to a group
- unreliable peer-to-peer transfer (optional)

2.2 Related Network Protocols

We will now introduce some existing Internet protocols and review their suitability on supporting shared virtual environments.

2.2.1 HTTP/FTP

HTTP (the Hypertext Transfer Protocol) and FTP (the File Transfer Protocol) are the well established protocols for reliable file transfers over the Internet. Both protocols are based on TCP/IP connections which establish a direct (reliable) connection between two hosts. TCP/IP connections however, do not scale very well.

2.2.2 UDP Datagrams

In contrast to TCP/IP based protocols, UDP datagrams do not establish a connection between the sender and the recipient of a network package. That is why UDP is called a connectionless service. Thus each package transmitted might be routed differently to the destination. Neither the order of packages nor their reception at all is guaranteed. UDP datagrams can either be transferred by unicast (peer-to-peer) or IP multicasting [11]. IP multicasting is an experimental Internet protocol which allows

messages to be transferred to a group of recipients rather than a single host. To achieve this, the datagram has to be sent to one of the special IP addresses representing multicast groups. Hosts can join these groups in order to receive those messages. The messages are distributed via a subnet of the actual Internet called the MBone (multicast backbone). The MBone consists of multicast routers.

2.2.3 Multicast Based Protocols

IP multicasting is the only protocol to be scalable to a large number of distributed users. Due to the fact that only UDP datagrams can currently be transferred, it only provides an unreliable transmission of data. While this is sufficient for many multi-user applications such as audio/video conferencing tools, it is not acceptable for many other shared applications. For that reason a number of reliable protocols on top of IP multicasting have been developed during the past years. Most of these protocols however, were realized and optimized to support a particular application area. Among those protocols are SRM [7], RAMP [10], and RMP [16].

2.2.4 DIS

DIS [13] is a network protocol based on IP multicasting and used for distributed simulation of military scenarios. It is so far the only existing standard for shared virtual environments. DIS is used by the NPSNET [14] system. It defines a number of units (PDU's) which are transferred to all other participants of the simulation to transfer the state of each object. While the protocol is very suitable for the specific application area, most PDU's are not suitable for general purpose virtual environments. The concept of transferring the state of each object frequently allows new participants or temporarily disconnected participants easily to catch up with the current state of the virtual world. However it puts a permanent network load on the network, even if object states are not changed.

2.2.5 ISTP

ISTP (Interactive Sharing Transfer Protocol) [2] uses a heterogeneous communication infrastructure to support shared virtual environments. It is built on four existing underlying protocols: HTTP, RTP, TCP/IP and multicast UDP. While HTTP is used to transfer files and other large amounts of data, RTP is used for the transmission of streams such as audio. Short messages, such as state synchronization are realized via unreliable multicast UDP. Additional TCP connections between servers and clients are used for reliable message transmission and recovery from UDP failures.

3 DWTP

In this section we want to introduce DWTP —the Distributed Worlds Transfer and communication Protocol. DWTP is an application layer network protocol for shared virtual environments on the Internet [4]. It is based on top of standard Internet protocols such as TCP/IP and UDP/IP (unicast and multicast)(see figure 1).

DWTP enables a virtual environment to transfer and receive several types of data:
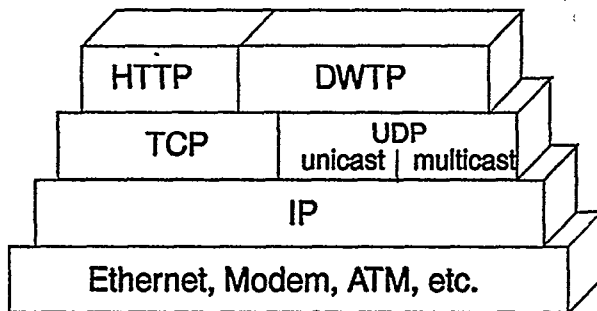
- events

Figure 1: Network Layers

- messages
- files
- streams

Events are used to keep distributed copies of shared virtual worlds consistent by transmitting appropriate synchronization data. Events may contain any kind of data and are usually rather small. The application can specify the required reliability for the transfer of events. Messages are actually a number of predefined events such as used for joining or leaving a shared virtual world. Some messages can contain additional data (e.g. for chatting, transmitting URL's, or sending requests). Files are heavy weight (large) objects, which require a reliable transfer. Examples are scene descriptions, avatar descriptions and VR applications. Files might be transferred peer-to-peer or to a group of recipients. Streams are used to transmit a continuous flow of data as used for audio or video. Streams do not require reliable transmission. DWTP provides a simple interface for these data types to the application or virtual environment, hiding the underlying network protocols.

Similar to other application layer protocols (such as HTTP) DWTP is based on different components: daemons and participants (peers). Daemons provide services to the participants of shared virtual worlds. In contrast to most other protocols DWTP uses several different daemons to realize the services required to support distributed virtual environments.
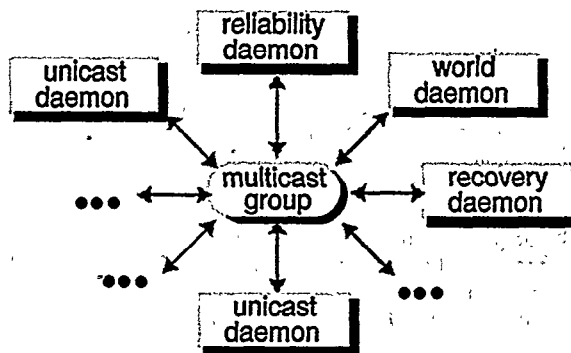

Figure 2: Network architecture of DWTP

Currently DWTP uses the following daemons:

- reliability daemons, to detect transmission failures (package loss) for unreliable protocol connections (UDP)
- recovery daemons, to provide unicast connections for recovering lost packages

- world daemons, to transmit virtual world contents (including users/avatars) to new participants
- unicast daemons, to realize a scalable architecture even for non multicast capable participants of the shared virtual world [1]

Using four different daemons seems to make DWTP rather complex. Nevertheless we preferred different daemons in contrast to one central daemon to make the overall approach scalable and tailorable (see also 3.4 Achieving Scalability). This does not necessarily mean, that four different programs on different hosts are required to provide the services for sharing a virtual world (see 4.3 The Prototype Implementation).

When using DWTP each shared virtual world is represented by one or several multicast groups. All daemons require access to multicasting to keep the overall architecture scalable (see figure 2). Each shared virtual world usually requires one reliability daemon, and at least one recovery and one world daemon (this assumes that participants do not necessarily have a copy of the world description before connecting to the world and that at least some messages require reliable transmission). Similar to HTTP daemons which can serve many HTML pages, daemons might serve more than one shared virtual world. Participants can be users (browsers/viewers), agents, (VR) applications, etc.

3.1 Data Transfers

In DWTP reliability is realized by the three daemons: reliability daemons, recovery daemons, and world daemons. DWTP splits application data to be transferred into small packages (see figure 3). While large objects such as files usually consist of a
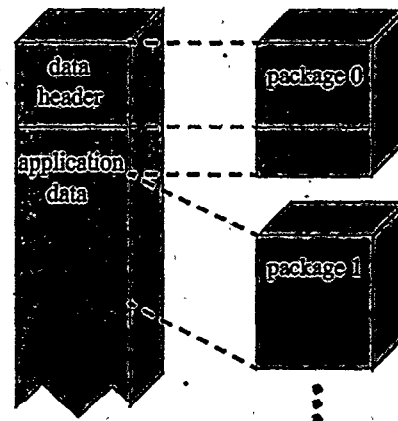

Figure 3: Splitting data into packages

large number of packages, most events as well as audio samples will usually fit into a single package. The package size is not fixed, but there is a maximum length for each package. The first package always contains a description of the data. This description is similar to a MIME type but specific to the needs of virtual environments.

Each package has a unique identifier. This identifier allows the recipients of the package to reassemble the complete message (if consisting of more than one package). Additionally this unique identification of a single packet is required for recovery purposes. The identifier consists of

- the host id,

- the sender id (identifying the application on the sending host, e.g. the process id),

- the message id (a number incremented for each message)

- and the package sequence number (identifying the package within a multi-package message)

Additionally each package contains the total number of packages of the transmitted data and a flag indicating if a reliable transfer of the package is required.

If reliability is required, the reliability daemon is used to detect transmission failures. In our protocol the reliability daemon sends positive acknowledge messages (ACK's) when receiving packages. We do not use negative acknowledge messages (NACK's) as used in ISTP [2], RAMP [10], or RMP [16], in order to prevent the NACK explosion effect (see figure 4). This ef-
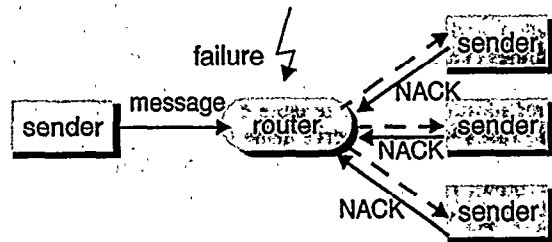

Figure 4: NACK explosion effect

fect usually occurs, if a large number of recipients are located behind a network router which fails to transmit a package. In this case all these recipients will send a NACK and by that lead to a partial congestion of the network. Since this already is the reason for most transmission failures, such a failure detection mechanisms would even intensify the problem rather than solving it. By using a single component (the reliability daemon) to send positive acknowledge messages the minimal network load for reliable messages is higher than with NACK's but does not have any significant peak values.

Each sender keeps outgoing messages (requiring reliable transfer) until it has received an acknowledge message for all packages of the message. If appropriate acknowledge messages are not received within a certain time, the corresponding packages are retransmitted. To reduce the amount of retransmissions the sliding window technique (as used by TCP) is applied. This technique interrupts the transmission of packages after a certain number until acknowledge messages have been received. The number of messages (the size of the window), which are sent before waiting for an acknowledge message is dynamically adjusted according to required retransmissions. After a certain number of attempts package transmission is aborted. Other participants and daemons can detect transmission failures when receiving the acknowledge messages from the reliability daemon. To ensure that the connection to the reliability daemon is alive, it sends (empty) acknowledge messages if it has not received a message after a certain period. All acknowledge messages are sequentially numbered to allow participants and other daemons to detect lost ACK's. Usually the ACK's for several packages (even of different messages) are sent within a single ACK message. ACK messages however will always consist of a single package. Once a recipient or daemon has received all packages and the corresponding acknowledges the original message can be reassembled.

A participant detects the loss of a package by receiving an acknowledge for a package it has not received. It detects missing acknowledge messages by the sequence number of the acknowledge messages and a time-out mechanism based on the interval

for empty (alive) acknowledge messages. It then has to contact a recovery daemon (see figure 5). Recovery daemons cache net-

⟷ multicast connections
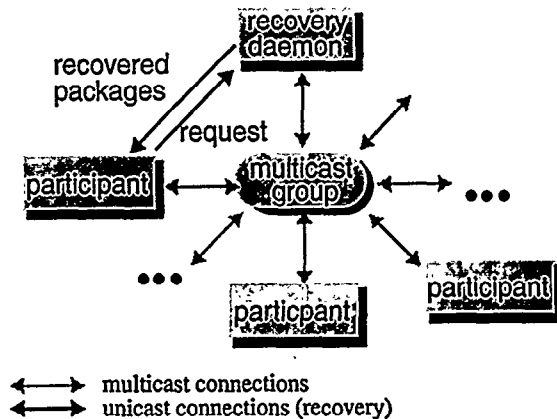⟷ unicast connections (recovery)

Figure 5: Recovering from transmission failures

work packages for a certain time. The requested packets are then sent to the participant or daemon by unicast. One problem of splitting the services between several daemons is, that all recovery daemons might detect that they missed a certain package. Although it has been acknowledged by the reliability daemon, it could not be recovered in this case. For that reason the reliability daemon always has to provide recovery services as well. To keep the load at the reliability daemon low, it might be configured to allow recovery from (recovery) daemons only rather than from arbitrary participants.

Sometimes a requested package has already been removed at the recovery daemon. This problem usually occurs when the network connection to one or several participants breaks down for a longer period. To overcome this problem, the participant has to connect to a world daemon. By sending the URL of the shared virtual world and a timestamp of the last received message, the world daemon will transmit all changes of this world which occurred after this timestamp as well as all pending messages (messages not completely received at this time).

3.2 Unicast Participants

Participants which are not multicast capable can communicate with other participants of a shared virtual world by unicast connections (UDP/IP or TCP/IP). Since all other participants as well as the daemons communicate via multicast groups, a special daemon is required to exchange messages between the unicast participants and the multicast groups of a shared virtual world. This task is performed by unicast daemons.

Participants get the network address of one or several unicast daemons from the world daemon when joining the world. This information is completely encapsulated within the protocol layer and thus not visible for the application. If the participant is not multicast capable it establishes a unicast connection[1] to one of the unicast daemons (see figure 6). Once this connection has been established, all messages from the participant are sent to this daemon. The daemon forwards these messages to the multicast group as well as to all other unicast participants connected to the same daemon. Messages received from the multicast group are distributed among all local participants. To guarantee

---

1. For UDP there is actually no real connection, but the participant is registered at the unicast daemon and is assigned a specific port where to send outgoing packages.

the reliable transmission of messages, the reliability daemon connected to the multicast group(s) can be used. If the unicast connection to the participants is a non reliable connection (UDP/IP), the unicast daemon can provide its own reliability services for these connections. This reduces the load of the reliability daemon and allows us to detect transmission failures faster.
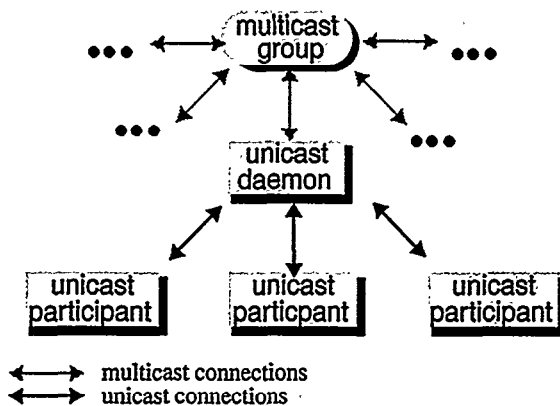

Figure 6: Unicast Connections

According to the unicast network parameters transmitted by the world daemon, the protocol layer might use individual unicast daemons for different data types (e.g. one unicast daemon for events and messages, and one for audio streams). Additionally one unicast daemon might be connected to several multicast groups at the same time. It can either transfer all messages between unicast participants and all multicast groups, or split messages according to the data type (i.e. one multicast group for audio, one multicast group for events and messages, etc.).

3.3 Connecting to a Shared World

World daemons can transmit a description of the shared virtual world on request and provide new participants with the required network parameters to connect to that world and communicate with other participants. By that mechanisms the persistence of dynamically changing shared virtual worlds is guaranteed. For that reason a local copy of the shared virtual world has to reside on the host of the world daemon which is updated according to received messages. The same applies to avatars and other participants currently connected to the shared virtual world. World daemons however, do not provide any application dependent features themselves, but allow application servers to communicate with the participants of a shared world. To make a shared virtual world persistent at least one world daemon is required.
New participants join a shared virtual world by connecting to a world daemon. This connection is specified by the world's URL (e.g. dwtp://world.daemon.com/worldname). This initial connection is a TCP/IP connection, since TCP/IP is appropriate for a one-to-one reliable transfer of large amounts of data. First of all the world daemon sends the network connection parameters (i.e. a multicast group and port as well as alternative network connections for non-multicast capable participants) to the new participant (see figure 7). The new participant connects to the multicast group or one of the unicast addresses and sends a reply to the world daemon. The daemon will then transmit all packages of incomplete messages it received until then and which might not have been received by the new participant at this point. This prevents the participant from missing any messages. Then the virtual world contents are transmitted from a file. In our prototype this is realized by transmitting VRML code. Since

the world's contents continuously change, this description usually is not transferred from a static file but created from the current state of the world by the application server (see "DWTP Server Interface"). The participant however might request additional parts of the shared world by specifying their local URL. This can include modifiable parts (e.g. descriptions of other participants/avatars), which have to be provided by the application server, or static files (such as texture images, or sound files) which can be transmitted directly by the world daemon. Afterwards the TCP/IP connection is released (by the new participant or after time-out) and all other communication is performed via the regular network connection established by the participant.
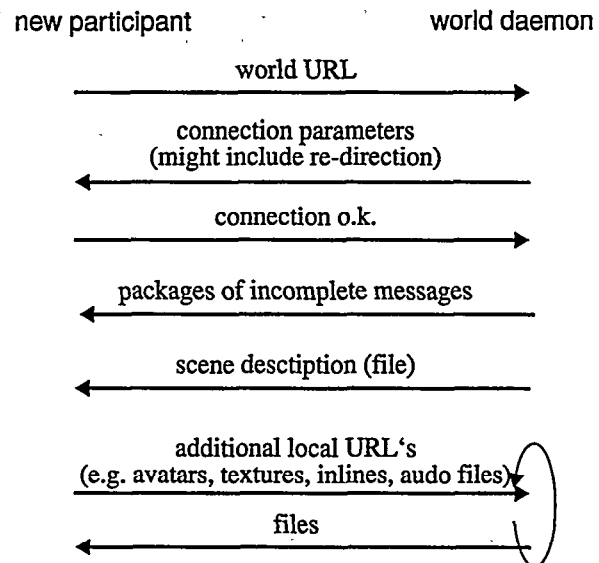

Figure 7: Connecting to a world daemon

If the participant has already connected earlier to the same world, it can add a timestamp to the requested URL indicating when its local copy of the world or participant description has been modified for the last time. Depending on the individual application server, the participant might rather receive a number of update events than a file describing the requests scene.

3.4 Achieving Scalability

DTWP provides support for two basic mechanisms to realize large-scale virtual environments.

- adding additional daemons (on additional hosts) to reduce the load on the existing ones
- splitting worlds into smaller parts (regions), each part using its own network connections

Since there is no direct connection between the individual daemons (all daemons send and receive messages via the multicast connection) unless transmission failures occur, adding daemons usually reduces the load of the existing ones. Additional world daemons might be used to provide additional dial in points (initial downloads). Increasing the number of world daemons is useful, if a large number of users, which cannot be handled in time by the available world daemons, frequently join a shared virtual world. Instead of publishing additional dial in points as URL's for those world daemons, the original world daemon(s) can simply be configured to redirect requests to the new world daemons.

53

It is recommended to use a single DWTP address as dial in point and connecting all other world daemons of the same world by redirection.

Additional recovery daemons can be used to reduce the number of recovery requests on a single daemon. This might be necessary, if a large number of transmission failures occur. Adding unicast daemons provides the possibility to support a large number of participants even if they do not have access to multicasting. A single unicast daemon should not be connected to a large number of participants. Tests showed that the limit for the number of participants to be supported by a single unicast daemon is between five and twenty depending on the activity of the participants and the requested services (e.g. with/without audio). The network load of unicast daemons can also be reduced by configuring individual daemons for the transfer of events and stream data.
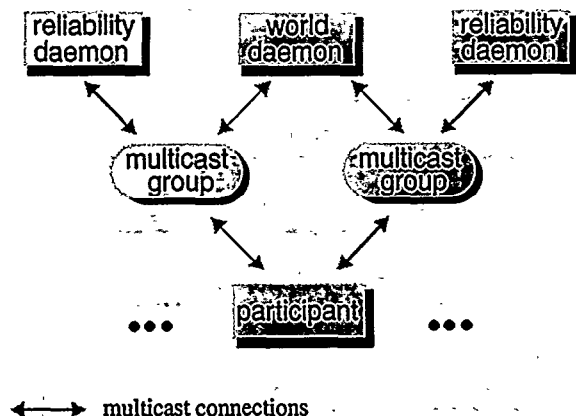

Figure 8: Splitting worlds into several parts

In addition to increasing the number of daemons for a particular world, it often is more useful to split the whole virtual world into smaller parts or regions. Each of these parts uses its own network connections (see figure 8). Thus the load of the daemons responsible for a certain part decreases. The VR application has to make sure, that the participant of such a shared virtual world is only connected to those parts, which are currently visible to him or her. Due to the update mechanism for world contents already transmitted earlier (as provided by world daemons), re-connecting to regions of a subdivided world is rather simple. Splitting a world between several multicast groups, does not necessarily mean that each part has to provide a full set of individual daemons. For a daemon splitting a single world into different parts is very similar to several independent worlds. Thus one daemon service might be split among several daemons for a single world, while other services for several worlds are realized by a single daemon.

4 USING DTWP TO CREATE SHARED VIRTUAL ENVIRONMENTS

In this section we will show, how the different components of DWTP can be used to support distributed shared virtual worlds or collaborative virtual environments (CVE's).

On the one hand the DWTP unicast, reliability and recovery daemons are completely independent of the realized shared virtual environment and thus can run stand-alone. On the other hand the DWTP peer component and world daemons have to communicate with the application or application server respectively.
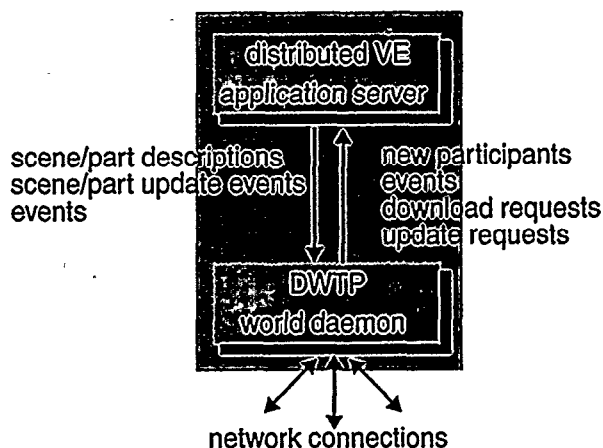

Figure 9: The DWTP world daemon interface

4.1 DWTP Server Interface

The DWTP server interface allows world daemons to be independent of a particular application or virtual environment. The reason for separating world daemons from the application server is that different applications will use individual description languages, file formats for worlds, sound, textures, etc., and different synchronization mechanisms (e.g. locks, master entities, tokens, etc.). Thus the world daemon receives and forwards all application or environment related messages to a distributed virtual environment server via its server interface (see figure 9). This includes:

- descriptions of new participants (e.g. avatars) as files
- events
- requests for world/participant downloads (connection)
- requests for world/participant updates (reconnection)
- messages (e.g. participant quit)

The application server will usually add new participants to the local data base and update them similar to the world scene according to the incoming events. In addition to changes to the world contents or participants, the application server will use events to receive requests for locks or other synchronization mechanisms (if provided at all). When receiving a connection request, the application server has to generate a world description which will then be transferred to the new participant by the world daemon. Reconnection is very similar, but the application server rather creates a list of events to be sent to the participant.

4.2 DWTP Peer Interface

Participants of shared virtual environments such as users (navigating a browser and represented by avatars), shared applications, agents, etc. use the DWTP peer component and associated mechanisms to connect to other participants and central services via DWTP (see figure 10).

The peer interface allows participants

- to connect to new shared virtual worlds
- to send requests for world parts/descriptions of other participants

- to receive these worlds parts/participant descriptions
- to send events to all other participants
- to receive events on connected worlds/parts
- to transfer local descriptions (e.g. the user's avatar) to all other participants
- to send stream data (audio/video) to some or all other participants

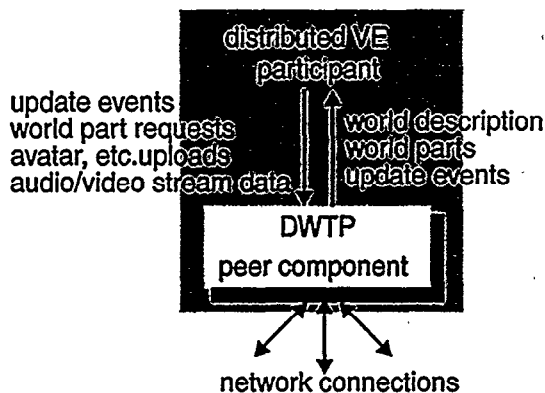Participants of shared virtual worlds will use the DWTP peer

Figure 10: The DWTP participant (peer) interface

component to connect to shared virtual world defined by an appropriate URL. They will then usually request additional (static) files and (dynamic) world parts or descriptions of other participants currently connected to the same world. If a local description already exists, they might alternatively request the update events necessary to resynchronize these descriptions with their current state. If the local participant wants to be represented in the shared virtual world, it will usually provide a representation to be uploaded and transferred to all other participants. For users this will be their avatar, for applications or agents these might be an arbitrary description. All modifications of the world based on local interactions of the participant have to be transferred to all other participants. This is realized by sending appropriate events over the network by the DTWP peer component. The participant might specify the required reliability level for each event. Currently DTWP supports only two reliability levels (reliable and unreliable transfer of data). Additional levels and types however, will be supported in future releases.

4.3 The Prototype Implementation

We have realized a prototype implementation of DTWP. In its second version the implementation is realized by a multi-threaded shared library. This library includes the DTWP peer component and the DWTP daemons (including the world daemon interface).

The DWTP peer component is used by our VRML browser SmallView [3] to connect and communicate with shared virtual worlds. Joining a shared virtual world is realized by specifying the appropriate URL (dwtp://...). When the world has been downloaded, it is parsed by SmallView for additional downloads (e.g. inlines, textures, avatars, etc.). When specified by a DWTP address, those will then be requested and transmitted by DWTP. Finally the browser will sent the local user profile (including the participant's avatar) to all other participants.

Consistency among the different distributed copies of the VRML world is realized by sending synchronization events. Synchronization events are sent by each individual VRML node to its replicated copies via DWTP whenever any of the node's fields is modified. To reduce the network load, a minimum time period is required between two synchronization events issued by the same VRML node. At the end of this period all fields which have been modified are transmitted. In addition to the field values, timestamps indicating the last modification are transmitted. When receiving a synchronization event, only those fields of the VRML node with timestamps older than the corresponding fields in the synchronization event are modified. To prevent several synchronization messages resulting from a single VRML event, all VRML events have been extended by an additional synchronization state variable (in addition to the transmitted value and the timestamp). The state variable shows, if an event has already been synchronized within the current event cascade. In the latter case no further synchronization is necessary, since the transmitted synchronization event will continue the event cascade in the replicated copies.

In addition to the VRML browser SmallView a universal configurable daemon based on the DWTP library has been implemented. This daemon includes all daemons provided by the DWTP library (as presented in the third section of this paper). thus it can be configured to realize any (arbitrary) combination of those daemons. It would even be possible to configure a single daemon to provide all daemon services for a shared virtual world. Using a single daemon for all services however, will make this server the bottleneck of the system. That is why we separate at least daemons providing peer-to-peer connections (unicast daemons) from all other services by setting up two or more daemons on different hosts.

Finally we have realized an application server (the place where always a copy of the shared world is kept). This application server (SmallServ) is connected to DWTP by the DWTP server interface. In addition to providing file descriptions or update events of the worlds contents, SmallServ provides a mechanism to resolve access conflicts between multiple participants of the shared world.

5   CONCLUSIONS AND FUTURE WORK

In this paper we introduced DWTP—the Distributed Worlds Transfer and communication Protocol, an application layer protocol for connecting large scale virtual worlds and multiple users on the Internet. DWTP provides a set of daemons in order to realize the individual requirements for realizing distributed VR applications. Based on different network protocols it provides the basis for a universal protocol for shared virtual worlds.

Our future work will further enhance DTWP in order to support participants connected via low bandwidth connections such as modems and to provide additional mechanisms for peer to peer communication. We will additionally work on more sophisticated mechanisms to reduce the amount of recoveries. Especially smart mechanisms to select the appropriate daemon for a certain service considering the load of the daemon and the network connection will be subject to further investigation.

Acknowledgments

butions to the implementation of the first version of DWTP. We further would like to thank the unknown reviewers of this paper for the helpful comments.

References

[1] L.A. Ames, D.R. Nadeau, and J.L. Moreland, *The VRML 2.0 sourcebook*, John Wiley & Sons, New York, 1997.

[2] R.C. Waters, D.B. Anderson, and D.L. Schwenke. Design of the Interactive Sharing Transfer Protocol. Proceedings of *Sixth IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* (June 18-20, 1997, MIT, Cambridge, Massachusetts), pages 140-147. Los Alamitos, California: IEEE Computer Society Press, 1997.

[3] W. Broll. Populating the Internet: Supporting Multiple Users and Shared Applications with VRML. *Proceedings of the VRML'97 Symposium*, pages 33-40. ACM, Feb. 1997.

[4] W. Broll, and D. Schick. DWTP-A Basis for Networked VR on the Internet. Proceedings of IS&T/SPIE's *Symposium on Electronic Imaging: Science & Technology 1998 (EI'98)*. (San José, January 24 - 30, 1998).

[5] D. Brutzman, M. Zyda, K. Watsen, and M. Macedonia. Virtual reality transfer protocol (vrtp) Design Rationale. *Proceedings Sixth IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* (June 18-20, 1997, MIT, Cambridge, Massachusetts), pages 179-186. Los Alamitos, California: IEEE Computer Society Press, 1997.

[6] Cyberhub, Blaxxun Interactive. [www] http://www.blaxxun.com

[7] S. Floyd, V. Jacobsen, C. Liu, S. McCanne, and L. Zhang. A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing, Scalable Reliable Multicast (SRM). ACM SIGCOMM 95.

[8] T. A. Funkhouser. RING: A Client-Server System for Multi-User Virtual Environments. *ACM SIGGRAPH Special Issue on 1995 Symposium on Interactive 3D Graphics*, pages 85-92. New York, 1995.

[9] Yasuaki Honda, Y., Mitra, B. Rockwell, B. Roehl. Living Worlds, Draft 2.0, April 13, 1997, [www] http://www.living-worlds.com/draft_2/index.htm

[10] A. Koifman, and S. Zabele. RAMP: A Reliable Adaptive Multicast Protocol. *Proceedings of IEEE INFOCOM '96*, San Francisco, CA., March 1996. [www] http://www.tasc.com:80/simweb/papers/RAMP/ramp.htm

[11] V. Kumar, *MBone: Interactive Multimedia on the Internet.* New Riders, Indianapolis, Indiana, 1995.

[12] R. Lea, Y. Honda, K. Matsuda K., S. Matsuda. Community Place: Architecture and Performance. *Proceedings of the VRML'97 Symposium*, pages 41-49. ACM, 1997.

[13] J. Locke, "An Introduction to the Internet Networking Environment and SIMNET/DIS", [www] http://www-nps-net.cs.nps.navy.mil/npsnet/publications/DISIntro.ps.Z

[14] M. R. Macedonia, M. J. Zyda, D. R. Pratt, et al, "Exploiting Reality with Multicast Groups: A Network Architecture for Large-Scale Virtual Environments", *Proceedings of the IEEE VRAIS'95*, pages 2-10. IEEE Computer Society Press, Las Alamitos, CA, March 1995.

[15] Q. Wang, M. Green, and C. Shaw. EM —An Environment Manager For Building Networked Virtual Environments. *Proceedings of the IEEE VRAIS'95 Conference*, pages 11-18. IEEE Computer Society Press, Las Alamitos, CA, March 1995.

[16] Brian Whetten, Todd L. Montgomery, and Simon Kaplan. *A High Performance Totally Ordered Multicast Protocol, Theory and Practice in Distributed Systems*. Springer Verlag LCNS 938.