SmallTool - a toolkit for realizing shared virtual environments on the Internet

Distrib. Syst. Engng **5** (1998) 118–128. Printed in the UK PII: S0967-1846(98)95808-9

SmallTool—a toolkit for realizing shared virtual environments on the Internet

Wolfgang Broll†‡

GMD—German National Research Center for Information Technology, Institute for Applied Information Technology (FIT), Sankt Augustin, Germany

Received 6 March 1998

Abstract. With increasing graphics capabilities of computers and higher network communication speed, networked virtual environments have become available to a large number of people. While the virtual reality modelling language (VRML) provides users with the ability to exchange 3D data, there is still a lack of appropriate support to realize large-scale multi-user applications on the Internet.

In this paper we will present SmallTool, a toolkit to support shared virtual environments on the Internet. The toolkit consists of a VRML-based parsing and rendering library, a device library, and a network library. This paper will focus on the networking architecture, provided by the network library—the distributed worlds transfer and communication protocol (DWTP). DWTP provides an application-independent network architecture to support large-scale multi-user environments on the Internet.

1. Introduction

Networked virtual reality (VR) may be a chance to provide a new interface metaphor to the Internet. While the WorldWide Web (WWW) is used by millions of people, it is currently impossible for these people to interact with one another. Shared virtual environments accessible from almost everywhere will be able to offer new types of cooperation.

The SmallTool-distributed VR toolkit includes a number of developments to enable worldwide-distributed users to participate in shared 3D worlds across the Internet, interacting with the objects of these worlds as well as communicating and collaborating with other participants. The development of this toolkit is part of a larger research initiative called *The Social Web*. Its goal is to develop the WWW into an active social meeting space in contrast to its current state as a more or less passive assembling of distributed artefacts. In the area of multi-user shared virtual environments we currently focus on two aspects: an adequate representation of users and their behaviours to extend shared virtual environments into a rich meeting and communication facility, and the network requirements to support a large number of worldwide-distributed participants on the Internet.

The realization of applications for shared virtual environments is based on a number of key technologies which make the development time intensive and expensive. An appropriate description for 3D objects is required. 3D scenes have to be rendered and input devices have to be connected to realize interactive worlds. Network connections have to be set up and mechanisms have to be established to keep copies of shared virtual worlds consistent.

The Virtual Reality Modeling Language (VRML) [1] provides us with a standard for the description of 3D objects on the Internet. Standard VRML browsers or plug-ins allow users to view 3D objects and navigate through VRML scenes, but they do not provide any support for shared virtual worlds. While browsers are available which support multiple users in VRML worlds, application developers cannot create their own applications independent of that particular browser. Additionally the network architectures provided for communication between the individual participants are rather inflexible and cannot be tailored to support different application areas.

In this paper we shall show how the SmallTool-distributed VR toolkit simplifies the development of shared VR applications. Since standard solutions for parsing, rendering and the connection of input devices exist, we will not discuss these features in detail. We will rather focus on the networking issues, which allow us to provide a flexible and scalable network infrastructure. This infrastructure is optimized for shared VR applications and provides a simple interface for applications and servers.

This paper is organized as follows. In section 2 we will introduce the basic architecture of the SmallTool VR toolkit. First we shall present the basic libraries that the toolkit is based on. Section 2.1 reveals the connections

† E-mail address: wolfgang.broll@gmd.de
‡ URL: http://orgwis.gmd.de/

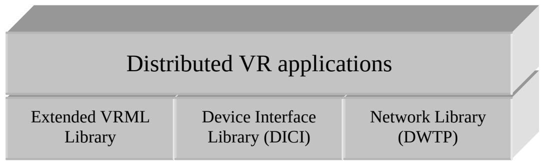


Figure 1. Realizing distributed VR applications with SmallTool.

to VRML and presents extensions provided by the extended VRML library. In section 2.2 we will introduce the universal device library of the SmallTool toolkit. Sections 2.3 and 2.4 discuss two sample applications, realized to demonstrate the use of the toolkit. In section 2.3 we will give a brief introduction of the SmallView browser, and in section 2.4 we will present the SmallServ application server. Section 2.5 shows how shared VR applications can be realized with SmallTool. In section 3 we will discuss the networking architecture in detail as used within the networking part of the toolkit. After reviewing the requirements for networked virtual environments in section 3.1 we will introduce the distributed worlds transfer and communication protocol (DWTP). The following sections describe the various aspects of the protocol including the support for heterogeneous network connections, reliable transfers on unreliable protocols, and the realization of persistent and consistent worlds. In section 3.6 scalability issues are discussed. Section 4 relates our work to others, and sections 5 and 6 discuss future directions and give conclusions.

2. The SmallTool VR toolkit

SmallTool is a toolkit for realizing shared virtual environments on the Internet. It is based on a number of libraries to build (VRML-based) networked VR applications. The SmallView VRML browser and the SmallServ application server are such applications which allow users to participate in shared VRML worlds.

Figure 1 shows the overall architecture for building VR applications with the toolkit. The toolkit consists of the following three libraries.

- The extended VRML library for reading and writing VRML files and for rendering VRML scenes. This library is based on OpenGL and is independent of any particular window system.
- The device interface library for connecting external devices to such applications.
- The network library to provide a flexible and scalable network architecture for shared virtual environments on the Internet. The network architecture used within this library is discussed in detail in section 3 of this paper. These libraries are currently available for SGI IRIX, Sun Solaris and Linux.

2.1. VRML and more

This section will introduce the extended VRML library. The extended VRML library provides a simple interface to read, parse, visualize and write VRML files. Additionally it processes input events to modify the VRML scene and can automatically generate events for synchronizing shared copies of the world. The extended VRML library supports the VRML standards 1.0 and 2.0 (VRML 97) as well as extensions for multiple users [2] and arbitrary input/output devices. Rendering is based on the OpenGL API. We will now focus on the extensions going beyond the current VRML standard.

2.1.1. Representing and managing users. The extended VRML library provides support for the representation of users by avatars. Avatars can be defined by extensions within VRML files, either as part of a user profile or within the shared virtual world. A user profile is a VRML file identified by a special header line and will usually be loaded by a multi-user browser or applications at start-up time. It provides user-dependent settings to the system as well as to other participants in distributed worlds. These include the unique id of the participant, his or her nickname for chatting and audio communication, and additional personal information such as their e-mail address or the home page URL. The user identification provided by the user profile may also be used by the VR application for individual set-ups (e.g. calibration of input devices). All user information is located in a *user* node, which has to be the root node of the user profile. As part of a user description, one or several avatars can be defined by *avatar* nodes. Avatar descriptions are based on standard VRML for geometry and behaviours. Additionally user- and/or avatar-dependent values specifying individual spatial ranges for interactions, chatting and audio communication may be specified either in the user or the individual avatar node. These spatial limitations are based on a mechanism similar to the spatial model for interactions [3].

In order to specify the current participants of a shared virtual world, the URLs of all current participants are defined as inlined URLs as part of a scene description. The number or type of participants of a particular world might have to be restricted. Thus our extensions enable providers and authors of a shared virtual world to restrict the number of participants, their type (users, agents, applications), avatar type and avatar size.

2.1.2. Subdividing virtual worlds. The extended VRML library supports a mechanism for subdividing large virtual worlds into several regions [2]. Regions define parts of a virtual world which might be temporarily invisible to a participant when his or her viewpoint is outside a particular area. When the contents of this world part cannot be realized by the user, they do not require updating immediately upon changes based on object behaviour (e.g. animations) or interactions of other participants.

However, the subdivision mechanism realized by the extended VRML library provides much more than that. It can be used to nest several regions and by that realize a region hierarchy. A landscape for instance might be subdivided into several regions residing on the same level. Within these regions, however, there may be subregions representing buildings. Some of these buildings may be very large and thus they are subdivided again into smaller regions representing one or several rooms. One of these rooms may be a biologist's laboratory and some of the bacteria visible through a microscope might also represent an individual region. Thus, when looking into the ocular of the microscope the user might enter the world of the microbes and scale down to their size. The extended VRML library provides these functions by a special node to inline regions into VRML files. This node (*region* node) defines a transformation for the region within the current frame, its contents (as URLs), its external representation, and an optional transformation of the viewpoint when entering the region. Additionally, three spaces are used to determine the mutual visibility of participants inside or outside the specified region.

- The *border* defines the actual border of the region within the current frame. When crossing the border of a region, the user becomes a member of that region.
- The *radiation* defines the space (outside the region) from where the contents of the region are visible. If no radiation space is defined, the contents of the region are not visible until entering the region (as specified by the border space).
- The *horizon* specifies the visibility of objects from within the region (beside the contents of the region itself). If no horizon is defined, other parts of the world outside the region are not visible after crossing the border of the region.

In addition to these spaces, the *hull* of a region defines a fourth space to determine when the contents of the region require to be loaded or updated.

2.1.3. Achieving consistency among distributed copies. Support for shared VRML worlds using the extended VRML library is based on the concept of replicated copies of virtual world contents. In our model there are no particular shared objects as used in the living worlds approach [4]. Thus, all modifications of any object in the virtual world are distributed to all other participants of the world. The extended VRML library realizes the consistency among different copies of a shared virtual world through some basic functions provided by its event model. The event model represents all events by objects. There exists one object class for each input device and for several VRML node classes (e.g. sensor nodes). As soon as an instance of such a node class is modified, it creates a synchronization event to forward the changes to all replicated copies of the node. To reduce the number of events transferred over a network connection, a maximum frequency for events sent by a single object may be specified by the application. The event model ensures that the most recent event is always transmitted. Additionally the application might specify whether particular events require reliable distribution or not. Many events such as avatar movements do not require reliable transmission, since they are updated frequently, while other events (e.g. opening a door) will lead to permanent inconsistencies, if they get lost during transmission. The event model provides support for the serialization of events to transfer them over a network connection.

VRML events as used in VRML 97 consist of a field value and a timestamp. The recipient of an event is defined by the routing mechanism. A single VRML event might modify a large number of nodes within the scene graph. The reason for that is that a single event caught by a VRML node might create new events and so on and by that create what is called the *event cascade*. Thus, it is important to synchronize events at the beginning of the event cascade and execute them locally on each site rather than synchronizing each node modified individually. To achieve this, synchronization is performed by the sensor nodes, which catch an external event and make it available to the VRML scene. Additionally all events based on timers (e.g. the execution of animations), which are independent of user input, are not transferred at all.

2.2. Connecting VRML scenes to the real world

In VRML support for input devices is limited to a simple pointing device. VR applications, however, often realize complex user interactions which require support for keyboards, six degree of freedom (6DOF) input devices, etc. To connect any type of additional input or output devices to a VRML world, the external authoring interface (EAI) [5], which is not yet part of the VRML standard, has to be used.

SmallTool provides a flexible mechanism to connect any kind of input or output device to a VRML world. This mechanism is based on:

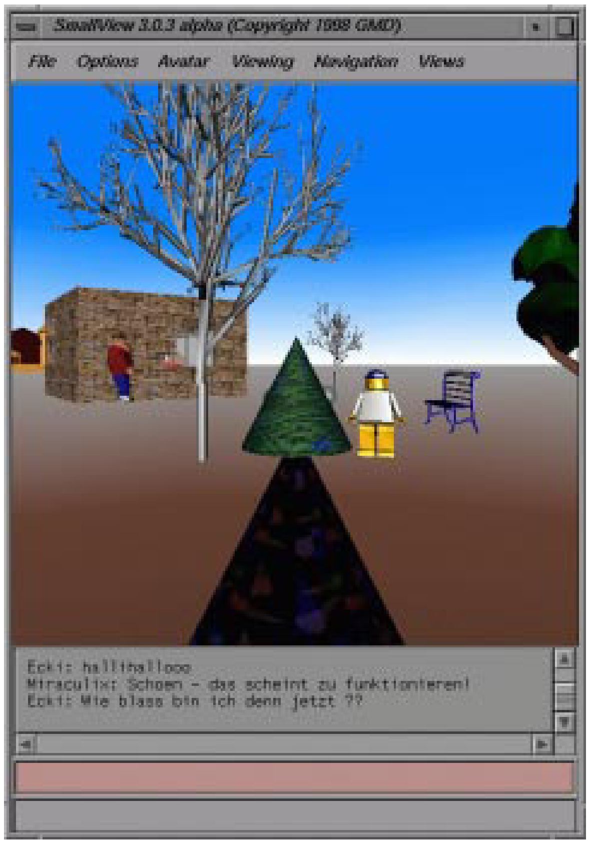
- a small VRML extension providing a universal sensor node,
- the device-independent communication interface (DICI).

The universal VRML sensor allows the author of a 3D scene to specify an arbitrary input device. Events received by this sensor can be used to modify the scene by routing the sensor fields to other VRML nodes similar to the mechanism used by standard VRML sensors. Additionally such sensors can be used as an output interface to external devices by routing VRML node fields to the sensor nodes. This sensor node is realized as part of the extended VRML library discussed before.

The DICI consists of a client and a server part. An application based on this SmallTool library (such as the SmallView browser) will use a client to communicate with external devices. Each external input or output device requires a tailored server. Several sample servers are provided to connect standard input devices such as trackers, gloves and SpaceMouse (Magellan). Each server can send or receive events by TCP or UDP connections, depending on whether reliable transmission of events is required or not. An arbitrary number of clients (not necessarily using the same VRML world) can be connected to a single server.

This flexible architecture allows us to:

- connect complex 3D input devices such as trackers and data gloves to VRML worlds
- use VRML objects as indicators for the state of real world objects (e.g. a door is opened, somebody entered a room, etc)
- manipulate real world objects from VRML worlds (e.g. switch the light on or off).



```
Ecki: hallihalloo
Miraculix: Schoen - das scheint zu funktionieren!
Ecki: Wie blass bin ich denn jetzt ??
```

Figure 2. Screenshot of the SmallView browser.

2.3. The SmallView browser

The SmallView browser (figure 2) enables users to participate at shared VRML worlds. To support multiple users it provides the possibility to specify user profiles and user avatars. Communication among distributed users is supported by spatial chatting and audio. Similar to an HTML browser or other VRML browsers or plug-ins, shared virtual worlds can be specified by a local name or a URL. The URL may either point to an ordinary hyper text transfer protocol (HTTP) server or be a DWTP address (e.g. `dwtp://multiuser.gmd.de/worlds/meetingSpace.wrl`) to participate at a shared VRML world. In the latter case a network communication to other participants is established in addition to the download of the scene description and the local avatar is transmitted. The local avatar can either be selected from a user profile, maybe provided by a shared virtual world, or can be imported from an ordinary VRML file.

The subdivision of shared virtual worlds as provided by the extended VRML library is used within SmallView to subdivide worlds into several regions, each using its individual network connection. When the viewpoint or the avatar of a user enter the hull of a region, the browser will start a new connection to the DWTP URL specified for the contents of that region. If the participant has not used this part of the world recently, all contents will be transferred. If the participant has already downloaded this part before, a timestamp of the last change of the local copy is added to the request to receive an incremental update. After the avatar or the user's viewpoint has left the hull of the region, the browser will usually cancel the network connection to reduce the local network traffic. However, a timeout mechanism is required to avoid the overhead of several connect, break up, reconnect cycles when the user navigates along the border of the specified hull. By this mechanism each participant will only be connected to those regions relevant for his or her current viewpoint within the shared virtual world. Nevertheless the region hierarchy can be used for additional reduction of network traffic. In addition to regions on the same level, parent regions (or their sister regions) will often not be visible after entering a region due to a limited horizon. Thus those parts can often be unloaded as well.

The first prototype of the SmallView browser was developed shortly after VRML 1.0 was presented. At this time it used a simplified network architecture based on Internet protocol (IP) multicasting and extended HTTP servers. The second version of SmallView added support for user interactions, object behaviour and user representation as proposed by the Dynamic Worlds VRML 2.0 proposal in 1996. Its current third version is based on the VRML 97 specification [6] for the description of virtual worlds. Nevertheless it uses extensions (provided by the extended VRML library) which go beyond the capabilities of this standard for representing users, subdividing large virtual worlds, and to minimize and resolve access conflicts between distributed users. The SmallView browser is developed on SGI workstations. It is based on the three libraries provided by SmallTool and is programmed in C++. It is also available for Sun Solaris and Linux with limited functionality.

2.4. The SmallServ application server

In addition to the SmallView browser, a SmallServ application server was realized. Similar to the SmallView browser this server is an example application built on top of the SmallTool libraries. However, it does not provide any visualization or I/O facilities and is based on the extended VRML library and the DWTP network library only. It is used to dump the current state of the world into a file, or to create events to resynchronize a shared virtual world. The latter is realized by comparing the timestamps of all nodes in the scene with the timestamp of the existing world description. The application server will then create synchronization events for these nodes. Additionally the SmallServ application server is used to realize persistent worlds (see section 3.5).

Given the SmallView VRML browser and the SmallServ application server, users can easily set up their own shared virtual worlds. To do so, one or several SmallServ applications have to be configured and the scene descriptions of the worlds to be shared have to be available as VRML files. Each participant can then connect to such an application server to download the virtual world description and to establish the appropriate network connections by specifying the URL of the application server within the SmallView browser. In order to give a user an embodiment, an avatar file has to be provided by each participant. This requires minimal modifications to a regular VRML 97 description, which can be performed automatically by the browser when importing a file as an avatar. The user is then represented by this avatar. In addition to all changes of the avatars, all changes of the virtual world are distributed to all participants as well.

2.5. Realizing VR applications with SmallTool

In this section we will show how new shared VR applications can be realized based on the libraries provided by the SmallTool toolkit. For a better understanding of this mechanism we will give two small example applications: a virtual store and a simple robot agent.

In our first example, the virtual store will provide its own representation, e.g. a simple building containing shelves with goods. Users visiting the world can enter the store with their avatar and select one or several goods. When the avatar leaves the shop, a virtual cashier will tell the user the price of the selected goods. To pay the user might have to click on a credit card symbol to pop up an appropriate HTML page.

The virtual store application would be based on the DWTP network library to connect to one (or even several) shared virtual worlds. Although it does not have to parse or render any VRML files, it will use the extended VRML library to restore the events transmitted from other participants when interacting with the store objects and to identify these participants. Finally, the application will have an underlying database to handle the transactions. This database will also require an interface accessible from a HTTP server realizing the payment. Additionally, the application has to provide its external representation as a VRML file. This file will be uploaded by the network library when connecting to a shared virtual world.

The second application is a very simple robot agent to welcome new participants in a shared world and to do some chatting with these participants. Similar to the virtual store, the application will be based on the network library for communication and the extended VRML library to interpret the events transmitted (e.g. the location of a particular avatar). Although the robot application does not need to render the VRML scene, it will need a continuously updated copy to realize realistic movements of the robot (e.g. to avoid walking through walls by collision detection). This can be also handled by the VRML library. The application developer only has to add code for moving the robot to the location of new users and create some chat messages based on keywords within the chat messages received from these users. Finally, a VRML file description of the robot is required for distribution to other participants.

3. Networking architecture

This section will discuss the networking architecture as used in SmallTool and provided by the underlying networking library in detail.

3.1. Requirements for networked virtual environments

In contrast to most other application areas, shared virtual environments distributed over a network such as the Internet require a large variety of different data types to be transferred. Additionally there are all kinds of connections from one-to-one, one-to-many to many-to-many. Classical client–server approaches do not fit very well for the needs of general purpose networked virtual environments. After the completion of initial transfers (such as downloads of world descriptions), typically all participants will send and receive data equally. This might include small amounts of data such as events, or even large files (e.g. avatar descriptions). The interactions of users participating at a shared virtual environment will usually not be limited to navigation: users can interact with the objects of the world as well as with other users. Thus, appropriate support for communication as well as for collaboration between users has to be considered. Additionally, users might want to bring new objects with them and leave them in a shared world, hand them to other users, or take objects with them, when leaving the world. To support general purpose virtual environments however, participants should not be limited to users. There might be any kind of participants including (autonomous) agents or applications (see 2.5). Neither of them necessarily have to be defined as part of the scene but may simply connect to one or even several shared virtual worlds, or travel between worlds, offering certain services to other participants.

While most application level network protocols support the transmission of a particular data type only, network support for shared virtual environments requires the support of several different data types. This includes:

- *files* to transmit the scene as well as other object descriptions (including avatars, agents, applications)
- *events* to keep distributed copies of a shared virtual world consistent
- *messages* to organize the participation (join, leave etc) of shared virtual worlds
- *streams* to transmit audio or video data.

3.2. The distributed worlds transfer and communication protocol (DWTP)

We developed DWTP [7] to address the needs of large-scale distributed virtual environments. DWTP is an application layer protocol at the top of existing Internet protocols such as TCP/IP and UDP/IP. Thus, it resides on the same level as HTTP or file transfer protocol (FTP). In contrast to those rather simple protocols the communication structure required to support shared virtual environments of different size and for different purposes is much more complex. For that reason DWTP is not based on a simple client–server approach including a single daemon to provide the required service, but uses a set of different daemons to provide particular services to the participants. DWTP provides the following four different daemons, which will be explained in detail in the following sections.

- *World daemons*, to download static or dynamic files and to keep modifiable worlds persistent.

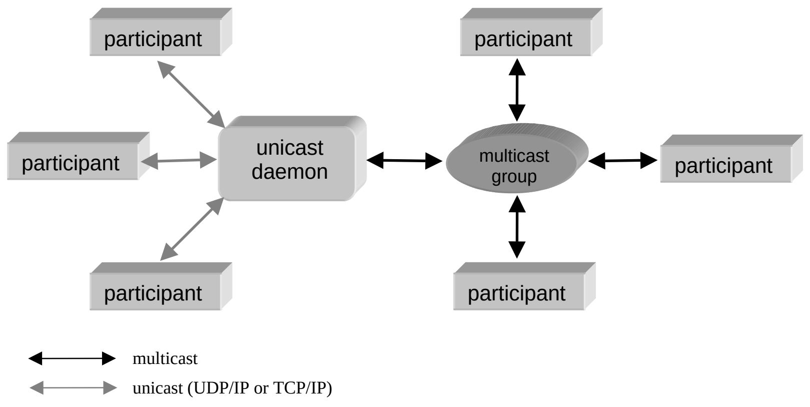


Figure 3. Participating via unicast daemons.

• *Reliability daemons* to provide reliable transfers by unreliable network protocols.

• *Recovery daemons* to receive lost messages.

• *Unicast daemons* to provide access via unicast network connections.

These daemons can either be combined in a single program (optionally combined with an application server) to provide all the services required to realize a single shared virtual environment or they may be split among several hosts.

In addition to these daemons, a client interface is provided by the DWTP library, which enables applications to connect to shared virtual worlds, and to send and receive the different data types used in shared virtual environments (see section 3.1). DWTP is realized as a multithreaded library to support asynchronous network communication independent of the main application.

The basic communication architecture of DWTP is based on IP multicasting groups [8]. The heterogeneous architecture of the approach however also allows non-multicast capable participants to join shared virtual worlds.

3.3. Supporting a heterogeneous network architecture

While DWTP heavily relies on IP multicasting for scalability, it allows participants to connect to shared virtual worlds by alternate (unicast) protocols such as UDP/IP or TCP/IP. Usually the DWTP client software will try to establish a multicast connection in the first place, followed by UDP and TCP on failure. The application, however, might change this behaviour due to its individual requirements. It can either specify preferred connection types or demand a particular type. Additionally different protocols might be selected for the individual data types (e.g. TCP/IP for files, UDP/IP for messages, IP multicasting for events). However, the selection made by the underlying protocol is usually optimized for the available set of connections and does need to be tailored by the application.

When using UDP/IP or TCP/IP for one-to-many data transfers this is realized via unicast daemons (see figure 3). Unicast daemons forward received data to all participants currently connected. However, there are some basic differences to classical central server approaches: the number of participants at each unicast daemon is restricted to a rather small number (five to ten) and the unicast daemon communicates with all other daemons and participants via multicasting only. This prevents a unicast daemon becoming a bottleneck of the system and allows us to keep the overall approach scalable even with a large number of unicast participants. Thus, unicast daemons allow us to create an application layer backbone for participants which do not have access to the MBONE.

3.4. Realizing reliable transfers

As discussed earlier, at least certain messages distributed in shared virtual environments require a reliable transfer. UDP/IP, however, is a connectionless protocol. Thus packages (datagrams) transmitted by this protocol might get lost, duplicated, or arrive in a different order. IP multicasting currently supports data transmission based on UDP/IP only. A number of approaches have been made to realize reliable data transfers on top of IP multicasting (e.g. SRM [9], LBRM [10], and RAMP [11]). Most of these approaches, however, were focused on a particular application area or network environment. For that reason DWTP uses a new mechanism for realizing reliable data transfers via unreliable protocols. Existing approaches either use acknowledge messages (ACKs) or negative acknowledge messages (NACKs) to realize reliable transfers. When using ACKs, each recipient of a message returns an ACK to the sender. The sender has to be aware of each recipient to identify missing ACKs. It will then send these messages to the particular recipients or all participants again. In a distributed virtual environment where participants join and leave frequently this approach has a number of drawbacks: the large number of ACKs increases the network load dramatically. The sender might have to resend a particular message several times before all other participants have successfully received it. Each participant has to keep track of all other participants currently connected. Leaving a shared world requires special attention, since all messages have to be acknowledged before and the recipient has to make sure that no other participant is still waiting for an ACK. Otherwise the message would be transferred several times unless

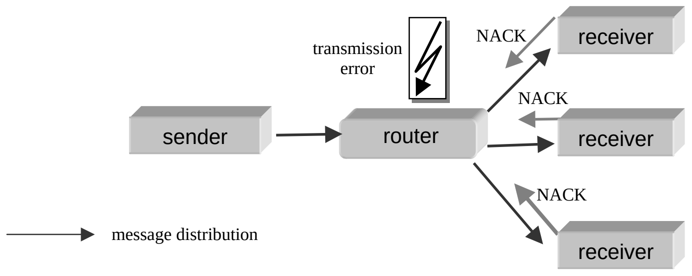


Figure 4. NACK explosion effect.

sending is stopped by a timeout mechanism or restricted by the number of failures.

NACKs, on the other hand, are much easier to handle for the sender of a message. They may, however, lead to the NACK explosion effect (see figure 4). This effect occurs if a message gets lost at a particular host (a router) within the network connection. All participants which should have received the message and which are located *behind* that host will now send a negative ACK message. Since these messages will usually take the same route back, this will even increase the problem. Another problem of NACKs is, that the receiving party needs to know when to send a negative acknowledgment. It is usually not obvious whether receiving no messages means that no messages were sent or all messages got lost. Thus, all sending parties additionally have to send an ‘I’m alive’ message after a certain period. In an environment with a large number of participants this will significantly increase the network load.

Our approach tries to overcome the disadvantages of both approaches by shifting the responsibility for the reliable transfer from the sender to some daemons and the recipients of the messages. To achieve this we use a combination of ACKs and NACKs in our approach. ACKs are sent by reliability daemons (see figure 5). This allows the sender of a message to make sure that the message has been transferred. Other participants can use the ACKs sent by the reliability daemon to detect that they missed a message. To detect transfer failures of ACK messages there is a maximum interval between two ACK messages send by the reliability daemon. After this time has passed, the reliability daemon will send an (empty) ACK even if it has not received a message. All ACKs are additionally numbered. Thus, participants can detect missing ACKs either by a timeout mechanism or by missing message numbers.

Missing messages and missing acknowledge messages are not requested from the sender or the reliability daemon but from recovery daemons. Recovery daemons provide a cache of the most recent messages transferred and send these messages to participants upon request via unicast connections (UDP/IP or TCP/IP). Like all other daemons, recovery daemons have to be connected to the multicast group. The reliability daemon itself has to provide recovery services at least to recovery daemons to ensure that all acknowledged messages can be recovered.

3.5. Achieving persistent worlds

When establishing a connection to a shared virtual world, the application will provide a DWTP URL to the DWTP library to connect to a DWTP world daemon. If a shared virtual world is static, i.e. participants may not modify any contents other than themselves, an application (e.g. a browser) may also use a local scene description or download the scene from a HTTP or FTP server. Most shared virtual worlds, however, will incorporate applications or behaviours which continuously alter the contents of the virtual world. Thus, each participant connecting to such a world might find a different world or at least a world in a different state. Additionally participants of such worlds might change or modify their contents. These changes have to be preserved even if no participant is connected to the world. This requires at least one instance of the world being dynamically updated. For that reason the world daemon provides an interface to request dynamically created files or data from another application. Since the requested data highly depends on the application and is actually independent of the network communication, this service cannot be provided by the world daemon itself. The SmallServ application server is an example application on top of DWTP which provides these services (see figure 6). The application server receives all events modifying the virtual world from the DWTP world daemon and updates the virtual world objects accordingly. Similar to the world daemon itself, an application server may handle several shared worlds at the same time. An application server will provide a current description of the virtual world to the world daemon for transferring it to new participants. Additionally it may provide incremental updates by creating a set of events to update an older version of the world to the current state.

3.6. Making worlds scalable

In this section we will present mechanisms which allow us to keep the number of participants, the size of the world, and the number of its objects scalable.

3.6.1. Avoiding bottlenecks. Systems supporting multiple participants which rely on centralized components do not scale very well, since the centralized component becomes the bottleneck of the system. For that reason it is necessary to avoid any centralized components as far as possible and distribute their duties between several sites.

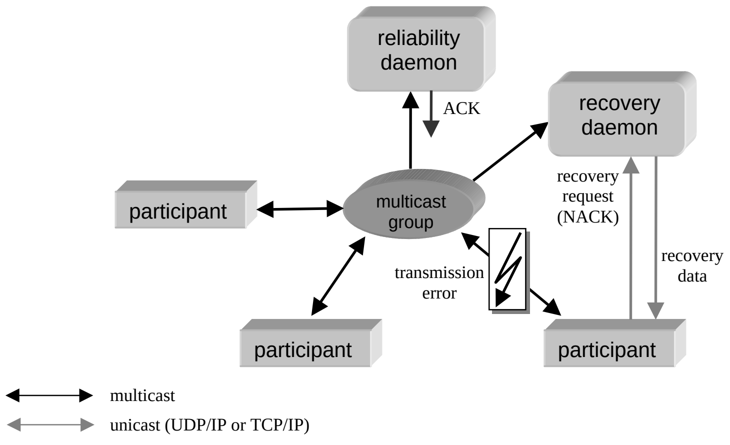


Figure 5. Network architecture for achieving reliability.

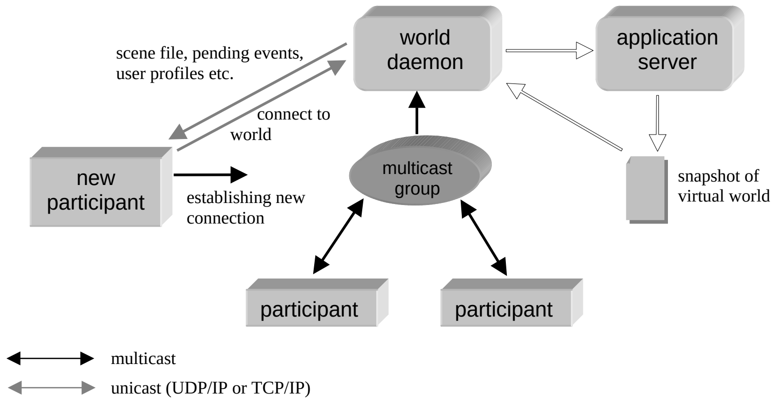

scene file, pending events, user profiles etc.
world daemon
application server
connect to world
new participant
establishing new connection
multicast group
snapshot of virtual world
participant
participant
multicast
unicast (UDP/IP or TCP/IP)

Figure 6. Achieving persistence.

DWTP allows us to distribute the individual daemons on several hosts and by that reduce the load at each site. Additionally most daemons (except the reliability daemon) can be replicated to split the load of a particular daemon between several sites. Several world daemons can be used to provide multiple dial-in points. Similar to HTTP daemons only a single URL is required and other world daemons are addressed via redirection from the primary daemon. As already mentioned above, unicast daemons have to limit their number of participants. Nevertheless any number of unicast participants can be supported by providing several unicast daemons for a particular world. The same applies for recovery daemons. Depending on the amount of recovery requests, additional recovery daemons can be added to a shared virtual world. On the other hand, several daemons can be combined at a single host, if the world and the number of participants are small. It is also possible to use one daemon for an arbitrary number of shared worlds (similar to an HTTP daemon serving several HTML pages). Which solution actually is appropriate, highly depends on the available resources (bandwidth, computing power) as well as on the size of the shared worlds and the number of participants.

3.6.2. Supporting large-scaled virtual worlds. Another mechanism to support large-scaled virtual worlds is to subdivide the world into several regions. Each of them providing its own network communication. The flexible communication approach realized by DWTP to group or distribute files and services between several daemons and hosts supports an easy realization of regions by shared VR applications (see figure 7). The regions can then easily be linked together by VR applications based on the DWTP library. Within the application each region has to be presented by an appropriate DWTP URL. This URL specifies the world daemon to download the region contents and to setup the communication for that region.

3.6.3. Comparing DWTP network load with traditional approaches. In this section we will compare the network load of traditional approaches to the DWTP architecture. For this comparison we assume that the size of one update message is 100 bytes, which is a reasonable value for updates of avatar specific data (i.e. position, orientation, etc). Five updates are sent per second. We will ignore the ACKs created by the reliability daemon, since only a few messages will require reliable transfers. Additionally the number of acknowledges under high-load conditions will not exceed 5% of the messages requiring reliable transfer

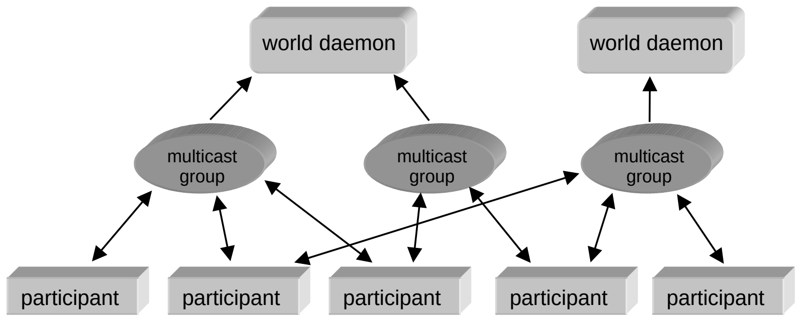


Figure 7. Realizing individual network connections for each region.

(approximately 20 messages can be acknowledged within a single ACK). We further assume that all participants are either moving (transmitting changes of their avatar), or interacting with an object within the scene. Since many users often neither move nor interact for some time, the overall number of messages will usually be even less. Finally, we can neglect messages based on animations, since starting or stopping an animation will each require a single event already considered as part of user interactions. The execution of animations is performed locally and does not require the transfer of any message.

In each comparison matrix we will compare the classical client–server approach and direct peer-to-peer connections to DWTP using either a multicasting or unicast connection. In the latter case a maximum of 10 participants for each unicast daemon is assumed. The unicast daemon will forward a message to local participants directly and to all other participants via the multicast group. The load of the daemons (except the unicast daemon) is not shown in the table. However, their load always equals the local network load of each participant. The first comparison matrix (table 1) shows the network load (bits s^{-1}) for 10 participants.

In this example the network load at the unicast daemon is slightly higher than at a traditional server, since the unicast server additionally sends all messages to other daemons and participants via the multicast group.

The second comparison matrix (table 2) shows the network load for 100 concurrent participants.

While the overall load is not a problem for shared virtual worlds with 10 users, these tables show that the traditional client–server approach is not suitable to realize shared virtual worlds with a large number of participants. The reason is that the network load at the server increases by the square of the number of participants. Direct peer-to-peer communication puts a rather high load on the local network and for that reason cannot be used for low bandwidth connections. All other approaches are able to support up to 70 concurrent users on a regular 64 kbit s^{-1} ISDN line, if the update rate is dropped to 1 Hz.

4. Related work

There have been a number of recent developments in the area of protocols for shared virtual environments. However, DIS [12] is still the only existing networking standard for distributed virtual environments. It is currently used in NPSNET [13] as well as in several other shared simulations. It is completely based on IP multicasting and does not support any alternative protocols. Consistency among participants is achieved by sending the current state of each entity (object) frequently to the multicast group. Since all entity types and their potential states are known by all participants in advance, this mechanism can also be used to distribute the contents of the current world. Although this approach is very suitable for the specific application area (military simulations), it does not fit very well to general purpose virtual environments, where the type of objects and their states usually will usually not be known in advance. A subdivision of a DIS environment into several multicast groups similar to that provided by SmallTool exists. The approach, however, is based on a subdivision of the landscape into a set of hexagonal cells. Each cell is represented by an individual multicast group and each participant is connected to his local cell and the neighbour cells only. Again the approach is highly adapted to the application area.

The VR transfer protocol (VRTP) [14] is a new network protocol currently being developed at the Naval Postgraduate School, Monterey, CA to overcome the limitations of DIS. It combines existing client–server technology as used by HTTP with reliable multicasting.

The interactive simulation transfer protocol (ISTP) [15] is a high-level protocol. Similar to DWTP it uses a mixed multicast and client–server model. This enables ISTP to select network connections depending on the data type to be transmitted. ISTP is used within the SPLINE system (scalable platform for large interactive network environments) [16]. Similar to SmallTool, SPLINE supports the subdivision of virtual worlds into several regions, each with its individual network connection. In SPLINE these regions are called *locales*.

Beside these approaches based on heterogeneous network architectures, a number of approaches exist, which entirely rely on IP multicasting. Two of the most advanced systems in this area are MASSIVE and the distributed interactive virtual environment (DIVE).

The MASSIVE VR system was developed at the University of Nottingham. In its second version (MASSIVE-2) [17] it uses third-party objects in addition to the mechanisms introduced by the spatial model to control

Table 1.

	Participant sending	Participant receiving	Participant local network	Server (or unicast daemon)
Client–server approach	4 kbit s^{-1}	36 kbit s^{-1}	40 kbit s^{-1}	400 kbit s^{-1}
Peer-to-peer connections	36 kbit s^{-1}	36 kbit s^{-1}	72 kbit s^{-1}	0
DWTP using IP multicasting	4 kbit s^{-1}	36 kbit s^{-1}	40 kbit s^{-1}	0
DWTP using unicast connections	4 kbit s^{-1}	36 kbit s^{-1}	40 kbit s^{-1}	440 kbit s^{-1}

Table 2.

	Participant sending	Participant receiving	Participant local network	Server (or unicast daemon)
Client–server approach	4 kbit s^{-1}	396 kbit s^{-1}	400 kbit s^{-1}	40 Mbit s^{-1}
Peer-to-peer connections	396 kbit s^{-1}	396 kbit s^{-1}	792 kbit s^{-1}	0
DWTP using IP multicasting	4 kbit s^{-1}	396 kbit s^{-1}	400 kbit s^{-1}	0
DWTP using unicast connections	4 kbit s^{-1}	396 kbit s^{-1}	400 kbit s^{-1}	4.4 Mbit s^{-1}

the interaction of users and the presentation of information. In addition to the approach realized by SmallTool, MASSIVE's third-party objects provide mechanisms to manage the communication via several multicast groups dynamically, i.e. add and remove participants depending on their number, the current load, etc.

The DIVE [18] system has been developed by SICS. It does not guarantee the persistence of virtual worlds. Nevertheless the basic synchronization mechanism is very similar to the one used by the SmallTool. The system, however, does not take care of package loss on multicast connections, but assumes that most differences will be fixed over time by later transmissions.

Beside this, a large number of networked VR systems, many of them based on VRML, exist. These systems are usually based on central servers. While this architecture is rather simple and provides easy realization of reliable message transfers, the number of concurrent participants is either restricted to a small number (typically some 10 users), or the number of object changes distributed to other participants is reduced significantly. The latter usually implies that avatar movements are only transmitted at very low rates and consistency is limited to certain objects of a shared virtual world. Examples for such systems include Sony's Community Place [19] or Blaxxun's CC3D/CCPro [20].

5. Future directions

Our research can be subdivided into two major areas. On the one hand, we try to provide better (technical) mechanisms to enable users to participate at shared virtual worlds. On the other hand we try to improve the quality of the representation of the users, their interaction with the worlds' contents and collaboration with other users.

The first area of interest includes research on how to enable participants connected via low-bandwidth connections without making them lose important information on the current state of the world. It further attempts to increase scalability by reducing network traffic. Additional work has to be done to make the scalability mechanisms more dynamic. So far several daemons might be either combined on a single host or distributed or even replicated among several hosts. While this approach is already very scalable, it is nevertheless static. Thus, it cannot deal very well with a suddenly rising number of participants. The same applies for the subdivision mechanism. Some effort is required to make these approaches dynamically adjustable to the current load of the system by launching additional (optional) daemons or subdividing the world on the fly.

The second area includes research on the appropriate representation of users in shared virtual worlds and examines the impact of various representations on the mutual awareness of other users' interactions. This includes experimenting with avatars which are completely non-human, human-shaped, human-shaped and animated up to representations using embedded video. Additionally, the representation of human gestures and body language plays an important part in this area.

6. Conclusions

In this paper we presented SmallTool, a toolkit for realizing shared virtual environment on the Internet. SmallTool provides support for the ISO standard VRML as a basic description language for 3D objects and for the representation of users by avatars through its extended VRML library. It also simplifies the use of arbitrary input devices by its device library. Combined with its DWTP networking library, SmallTool provides us with a platform for further investigating the area of collaborative virtual environments and multi-user applications. We demonstrated the use of the toolkit by our two example applications SmallView and SmallServ, which enable users to setup and participate in shared VRML worlds.

We further discussed the realization of the DWTP library in detail. We showed how DWTP can be used to realize networked virtual environments which are consistent, persistent and scalable. The architecture we presented allows us to support a large number of participants distributed worldwide on the Internet. These participants may use individual network connections to share a virtual world. We finally showed how the concepts realized within DWTP keep the overall network architecture scalable.

References

[1] Ames L A, Nadeau D R and Moreland J L 1997 *The VRML 2.0 Sourcebook* (New York: Wiley)

[2] Broll W 1997 Populating the Internet: supporting multiple users and shared applications with VRML *Proc. VRML'97 Symp.* pp 33–40

[3] Benford S and Fahlén L E 1993 A spatial model of interaction in large virtual environments *Proc. 3rd Eur. Conf. on Computer Supported Cooperative Work (ECSCW'93) (Milan, Italy)* (Dordrecht: Kluwer) pp 109–24

[4] 1997 *Living Worlds—Making VRML 2.0 Applications Interpersonal and Interoperable* Draft 2 http://www.vrml.org/Workingroups/living-worlds/

[5] Marrin C and Couch J 1998 VRML external authoring interface (EAI) reference *Proposal for a VRML 2.0 Informative Annex* http://www.vrml.org/WorkingGroups/vrml-eai/

[6] 1997 *VRML'97, the Virtual Reality Modeling Language ISO/IEC International Standard 14772-1:1997* http://www.vrml.org/Specifications/VRML97/

[7] Broll W 1998 DWTP—An Internet protocol for shared virtual environments *Proc. VRML'98 Symp. (February 16–19, Monterey, CA)* to appear

[8] Kumar V 1995 *MBone: Interactive Multimedia on the Internet* (Indianapolis, IN: New Riders)

[9] Floyd S, Jacobsen V, Liu C, McCanne S and Zhang L 1995 A reliable multicast framework for light-weight sessions and application level framing, scalable reliable multicast (SRM) *ACM SIGCOMM 95*

[10] Hobrook H W, Singhal S K and Cheriton D R 1995 Lob-based receiver reliable multicast for DIS *ACM SIGCOMM 95* pp 328–41

[11] Koifman A and Zabele S 1996 RAMP: A reliable adaptive multicast protocol *Proc. IEEE INFOCOM '96 (San Francisco, CA)* http://www.tasc.com:80/simweb/papers/RAMP/ramp.htm

[12] Locke J An introduction to the internet networking environment and SIMNET/DIS http://www-npsnet.cs.nps.navy.mil/npsnet/publications/DISIntro.ps.Z

[13] Macedonia M R *et al* 1995 Exploiting reality with multicast groups: a network architecture for large-scale virtual environments *Proc. IEEE VRAIS'95* (Los Alamitos, CA: IEEE Computer Society Press) pp 2–10

[14] Brutzman D, Zyda M, Watsen K and Macedonia M 1997 Virtual reality transfer protocol (VRTP) design rationale *Proc. 6th IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (June 18–20) (Cambridge, MA: MIT)* (Los Alamitos, CA: IEEE Computer Society Press) pp 179–86

[15] Waters R C, Anderson D B and Schwenke D L 1997 Design of the interactive sharing transfer protocol *Proc. 6th IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (June 18–20, Cambridge, MA: MIT)* (Los Alamitos, CA: IEEE Computer Society) pp 140–7

[16] Barrus J W, Waters R C and Anderson D B 1996 Locales: Supporting large multiuser virtual environments *IEEE Comput. Graph. Appl.* **16** 50–7

[17] Benford S and Greenhalgh C 1997 Introducing third party objects into the spatial model of interaction *Proc. 5th Eur. Conf. on Computer Supported Cooperative Work (ECSCW'97)* ed J A Hughes *et al* (Dordrecht: Kluwer Academic)

[18] Hagsand O 1996 Interactive multi-user VEs in the DIVE system *IEEE Multimedia* **3**

[19] Lea R, Honda Y, Matsuda K and Matsuda S 1997 Community place: architecture and performance *Proc. VRML'97 Symp.* pp 41–9

[20] Blaxxun Interactive http://www.blaxxun.com