Integrating Support for Collaboration-unaware VRML Models into Cooperative Applications

Werner Geyer, Martin Mauve
University of Mannheim, Germany
{geyer, mauve}@pi4.informatik.uni-mannheim.de

Abstract

This paper presents design and architecture of the distributed virtual reality service (dvr) which allows collaboration-unaware VRML animations and simulations to be integrated into cooperative applications. The dvr service realizes the distribution of VRML content to a group of participants via multicast. User interactions with the VRML model are synchronized in a manner transparent to the VRML content. To demonstrate the feasibility of our novel approach, we have successfully integrated the distributed virtual reality service into the digital lecture board which is a shared whiteboard tailored to the specific needs of collaborative work, for instance, in the area of computer-based distance education.

1. Introduction

One of today's most challenging multimedia applications is computer-based teleconferencing. In addition to audio and video, teleconferencing systems generally come with both a shared whiteboard for joint sketching of ideas, and an application sharing tool. These systems are employed in a variety of fields such as distance education, teleconsulting, telemedicine and telecooperation. In this paper we address the need to integrate the sharing of 3D animations and simulations into existing telecooperation applications. Areas of application for this functionality are the usage of interactive 3D models in teleteaching scenarios, telepresentations involving 3D information, or joint development of engineering objects over the Internet.

The foundation of our work is the digital lecture board (dlb) [3], a novel shared whiteboard supporting a rich functionality ranging from displaying and annotating slides to chatting and voting. The dlb was developed at the University of Mannheim and is tailored to support the specific requirements of telecollaboration. The integration of support for shared 3D animations and simulations is accomplished by providing a distributed virtual reality (dvr) service which can be controlled through the user interface of the dlb. This service makes it possible to share 3D geometry and behavior specified in the virtual reality modeling language (VRML). Due to its design and architecture, the dvr service can be employed by arbitrary cooperative applications.

2. Digital Lecture Board (dlb)

The digital lecture board [3] is a shared whiteboard tool which is being developed in the context of computer-based distance education. The design of the dlb has been mainly influenced by our experience with synchronous teleteaching in the TeleTeaching projects of the University of Mannheim [2].

2.1. Main Features

The current prototype of the dlb supports the following main features. A more detailed description of design issues can be found in [3] and [4].

- *User Interface*. The user interface of the dlb has been designed according to the *look & feel* of common Windows applications (see Fig. 5). The interface is easy-to-use and configurable so as to be able to adapt to different instructional settings. We have implemented a flexible workspace concept with multiple layers where arbitrary media objects can be displayed and modified. Private workspaces are provided for single participants which can be used to prepare material invisible to the rest of the group. Moreover, users can attach private annotations to an online document. The content of the shared workspace can be stored in a structured, SGML-like file format.
- *Media Support*. Since media are very important in modern instruction, we have integrated a variety of media formats (GIF, Postscript etc.) into the dlb, as well as many types of built-in object (rectangles, lines etc.). Similar to word or graphic processing software, the dlb provides editing functions like select, copy, paste, group, raise, lower etc. for the integrated media formats.
  One of the greatest benefits of applying computers in education is the chance to use new media such as simulations and/or animations. Since these media allow for a better presentation of certain complex contents, they can ease students' understanding of complicated subject matter. The graphical presentation and the possibility of exploring a model by changing, for

instance, model parameters represents a strong motivational function. This has motivated us to extend dlb's media capabilities with VRML through the integration of the dvr service presented in this paper.

- *Collaborative Services*. Today's video-conferencing solutions suffer from a lack of communication channels compared to the traditional face-to-face situation. Social protocols or rules which control human interaction and the course of group work in a face-to-face situation are not automatically available in a remote situation and are difficult to reproduce. These mechanisms include, for instance, raising hands, giving rights to talk or to write on the blackboard, setting up work groups, or reference pointing. *Collaborative services* provide mechanisms to support the communication of persons through computers and to increase social awareness. The dlb implements basic services such as floor control, session control, voting, online student feedback, chat, and telepointers.
- *Recording*. In contrast to many video conferencing systems, the dlb supports the synchronized recording of sessions, including all media streams, i.e. audio, video, telepointer, whiteboard annotations etc. Students or users who missed a session or who want to review a certain topic are able to retrieve a previously recorded lecture from a server. In order to be able to use existing recording facilities such as the VCRoD service (Video Conference Recording on Demand) [6], we rely on the Internet RTP standard [8] for transmitting data between distributed instances of the dlb.
- *Security*. Many existing teleconferencing applications neglect security issues almost completely, even though security is extremely important to allow for private sessions and billing, specifically in the Internet. The dlb is furnished with a user-oriented cryptographic concept which provides high secure and fast software encryption by integrating state-of-the-art encryption algorithms such as IDEA, CAST, Blowfish etc [4].

2.2. Architectural Issues

In order to provide a high responsiveness, the distribution architecture of the dlb is completely replicated, i.e. an instance of the dlb is running on each participant's computer. Each instance of the dlb holds a complete copy of the session data (on-line document). The dlb instances communicate with each other by using a reliable multicast protocol on top of IP-Multicast.

The software architecture of a single dlb is depicted in Fig. 1. On the application level, the core part of the dlb embeds functional modules such as a postscript interface, a telepointer module etc. Moreover, the dlb makes use of local services such as, for instance, the scalable multicast protocol (SMP) [5] for reliable multicast data delivery. Remote services are currently the VCRoD service for recording, and in the future, a multimedia database for storing teaching and learning materials. The distributed virtual reality service presented in this paper is integrated into the dlb as a local service which is accessed by a functional module (see Fig. 1).

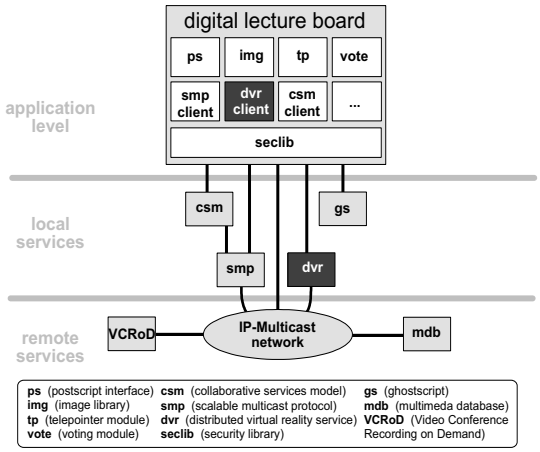

Figure 1. The dlb software architecture

3. Using VRML for Animations and Simulations

The virtual reality modeling language (VRML) is a "file format for describing interactive 3D multimedia on the Internet" [9]. VRML as a description language for 3D models and their behavior is very well suited for the development of simulations and animations. On the one hand, VRML is an international standard and allows the platform-independent definition of animations and simulations. On the other hand it is not limited in the area of deployment as are animation tools for specific application fields. Compared to an implementation in a standard programming language such as Java, the development of animations in VRML offers a higher level of abstraction and is therefore faster and more convenient.

The interaction with VRML content (also called a "world") is realized by using VRML browsers. Similar to HTML, a VRML browser is responsible for the presentation of the content and the handling of user- and application-generated events.

The basic building blocks of VRML are *nodes*. Nodes contain *fields* which can be elementary data types or nodes themselves. The hierarchy formed by the nodes in a VRML file is called a *scene graph*. The visible parts of the scene graph are displayed by the VRML browser. Interaction with VRML worlds is realized by special nodes which are

called *sensors*. A touch-sensor, for example, is able to catch mouse-click events and distribute them to other nodes. These nodes can react to the events, e.g. by starting a timer or modifying the scene graph. A running timer periodically generates specific events which allow for time tracking in a VRML animation. This mechanism enables the modeling of real-time behavior, e.g. moving an object for a period of time.

While VRML provides all the mechanisms required for simple animations, it does not (and was never intended to) support full programming language functionality. However, this functionality is required for complex animations which involve, for example, state information, non-linear interpolation, or network activity. To allow for the modeling of more complex behavior, VRML provides *script nodes* which define an interface between the VRML world and a script written in a regular programming language such as Java or C++.

In addition to script nodes, VRML browsers can provide another API for programming language interaction: the External Authoring Interface (EAI). The EAI is supported by most state-of-the-art VRML browsers and provides methods by which a VRML browser can be controlled by external applications. The EAI can be employed, for instance, to load and modify VRML content as well as to catch and send events.

VRML per se does not define any mechanisms for the distributed presentation of and interaction with animations. In order to use VRML in cooperative applications such as the dlb, we have developed the *distributed virtual reality service* (dvr). To realize distributed interactive animations, two key problems have to be addressed: first, the VRML content needs to be transmitted efficiently to all participants of a session. Second, mechanisms for the distribution and synchronization of user interaction with the VRML animation are required. The approach presented in this paper solves these problems by using script nodes and the External Authoring Interface.

4. Design Decisions

There are a number of design decisions which influenced the development of the distributed virtual reality service. The most important ones are:

- *Distribution architecture*. Similar to the dlb, the dvr service is based on a replicated distribution architecture so as to avoid the well-known problems of centralized approaches such as *single-point-of-failure, performance bottleneck, low responsiveness,* and *lack of scalability*. Common application-sharing approaches are not viable for 3D animations since the huge amount of data — resulting from sharing the 3D content of the VRML browser window — would consume an unacceptable amount of bandwidth.
- *dlb integration*. A central goal behind the dvr development was a generic software architecture which would allow for a smooth integration of the dvr interface into arbitrary cooperative applications. With respect to the dlb, users should not perceive the service as a stand-alone application. The usage of VRML animations should be as intuitive and simple as the handling of other media (images, graphical objects etc.). To satisfy these requirements, the local dvr service was realized as a client-server software system without a graphical user interface other than the VRML output. On the shared workspace of the dlb, VRML animations are symbolized by an icon. Activating the icon by mouse click starts the animation at all participating sites.
- *VRML interface*. The external authoring interface can be used to control VRML browsers by external applications. Since currently there is no VRML browser, which supports the full VRML specification as well as the EAI and script nodes for UNIX systems, we used the VRML browser development library OpenWorlds [1]. This C++ library supports all required functionality and is available on a variety of platforms such as Solaris, Irix and Windows95. OpenWorlds is a scene graph execution engine which traverses the scene graph, executes VRML nodes, routes events, and displays the resulting 3D graphics.
- *Synchronization*. A replicated architecture implies that a separate instance of a VRML animation is running on each participant's machine. To make sure that the rendered 3D output is the same for all participants, the VRML animations must be synchronized. The synchronization of animations, which are controlled solely by a timer, needs no further consideration since VRML is real-time capable. That is, an animation which is started on two machines at the same time will also end at the same time, no matter how different the processing speed of the two machines is. Visible results of insufficient processing power are artifacts like jerky movements of objects. However, user interaction can easily destroy the synchronization between several instances of the same animation since VRML does not include collaborative sensors for the synchronization of distributed instances of VRML animations. Hence, the dvr service requires specific sensors for the synchronization of user interactions.
- *Collaboration-awareness*. VRML content can be either *collaboration-aware* or *collaboration-unaware*. Collaboration-aware VRML content has been explicitly developed to run in a collaborative environment. An example of collaboration-aware VRML content are multi-user VRML worlds. However, the vast majority of current 3D models were not designed to be used in a multi-user VRML world. This is quite natural since

multi-user worlds are only one small area where VRML can be employed. VRML content without specific mechanisms for the distribution of user interaction is called collaboration-unaware. A major design goal of the dvr service was the usage of arbitrary, collaboration-unaware VRML animations in a cooperative environment such that distribution and synchronization issues would be completely transparent to the developer of VRML content. To achieve this, we provide collaborative sensors whose interface to the VRML content is exactly the same as that of regular VRML sensors. Regular sensors just need to be substituted by collaborative sensors to enable the synchronization of user interactions. This replacement can be done automatically at run-time and is therefore transparent to the original VRML author. We call VRML content which has been transformed in this way *semi-collaboration-unaware* [7].

5. Distributed Virtual Reality Service (dvr)

5.1. General dvr Architecture

The *distributed virtual reality service* enables the synchronized, distributed execution of VRML animations by using the replicated client-server architecture depicted in Fig. 2. The *dvr server* is responsible for the distribution,

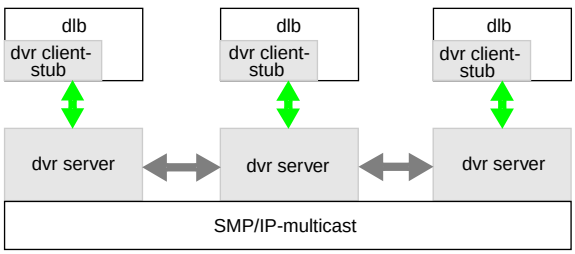

Figure 2. General architecture of the dvr service

presentation and synchronization of a VRML animation. Convenient access to the services offered by the dvr server is provided by a dvr client-stub. The client-stub communicates with the server via local socket connections and is available as a library. The application programming interface (API) provided by the client-stub includes commands for connection management and distribution of VRML content. This content and the information about user interaction are exchanged between distributed instances of the dvr server by means of the reliable multicast protocol SMP [5] on top of IP-Multicast.

In detail, the dvr server offers the following services:

- A VRML animation selected by a user can be transmitted to all other participants in a given session. Since a single animation may consist of multiple files containing either VRML content or other media, the VRML files are parsed and all files belonging to the animation are transmitted via multicast.
- The VRML animation can be loaded and displayed by an integrated VRML browser.
- User interactions at any participating site, which change the state of the animation, are transmitted via multicast to all other participants.

5.2. dvr Server Architecture

Due to the replicated architecture and the need to integrate a VRML browser into the dvr server, we have designed the server as indicated in Fig. 3. The server con-

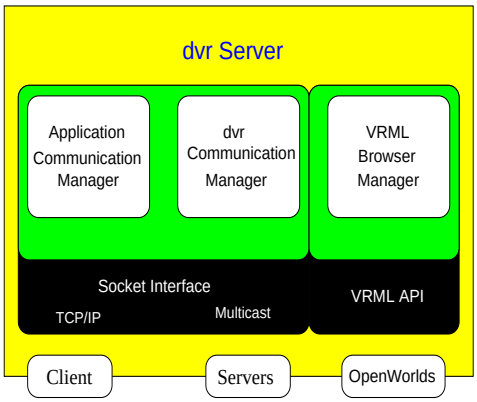

Figure 3. dvr server architecture

sists of the three main components *Application Communication Manager*, *dvr Communication Manager* and *VRML Browser Manager*. The Application Communication Manager controls the local communication between dvr server and dvr client (e.g. the dlb). This communication includes requests from the local client to distribute and display animations. The dvr Communication Manager handles the communication with peer server instances. Main tasks are joining and leaving reliable multicast sessions, transmitting VRML animations, and exchanging information about user actions. The VRML Browser Manager provides an interface to a VRML browser. We rely on the functionality of the VRML External Authoring Interface to instruct the browser to load VRML content and display the content in a separate window.

5.3. Distributing User Interaction

The dvr service allows for the distributed and synchronized execution of arbitrary VRML content. Hence, the

```
 1 #VRML V2.0 utf8
 2
 3
 4
 5
 6
 7
 8
 9
10
11 Group {
12   children [
13     DEF TouchIt TouchSensor {
14
15     }
16     DEF Timer TimeSensor {
17       cycleInterval 5
18     }
19     ...
20   ]
21 }
22 ROUTE TouchIt.touchTime TO Timer.startTime

(a)
```

```
#VRML V2.0 utf8

EXTERNPROTO CooperativeTouchSensor [
  field        SFString name
  eventOut     SFTime   touchTime
  ...
]
[
 "../CooperativeSensors.wrl#CoopTouchSensor"
]
Group {
  children [
    DEF TouchIt CooperativeTouchSensor {
      name "mauve:13089:134.155.48.116:1"
    }
    DEF Timer TimeSensor {
      cycleInterval 5
    }
    ...
  ]
}
ROUTE TouchIt.touchTime TO Timer.startTime

(b)
```

Figure 4. Semi-collaboration-unaware VRML content

distribution of user interactions must be transparent. As already mentioned, our concept of distributing user interactions relies on an automated transformation of collaboration-unaware content into a semi-collaboration-unaware content by replacing the standard VRML sensors with customized cooperative sensors.

In order to demonstrate how the processing of VRML data works, Figure 4 (a) shows an excerpt of a standard VRML file [7]. This fragment of VRML code contains a `TouchSensor` (line 13 - 15), waiting for the user to touch (click on) some geometry. Since the geometry is of no further interest in this example, we replaced it by three dots. The second element is a `TimeSensor` (lines 16 - 18) which tracks the passage of time. Initially, the `TimeSensor` is not activated. The user can trigger the `TouchSensor` by a mouse click which makes the `TimeSensor` in Fig. 4 (a) start ticking. The `TimeSensor` then emits events which indicate the passage of time. These events can then be used to control an animation (e.g. the movement of an object).

The processing needed to transform the original VRML content into a semi-collaboration-unaware VRML model is shown in Figure 4 (b). For each sensor listening to user input, we supply an alternative implementation, called CooperativeX, where X is equal to the name of the original sensor. VRML provides a standardized means to integrate these customized nodes into a VRML file by using the EXTERNPROTO statement. In this example the EXTERNPROTO statement (lines 3 - 10) is used to declare a `CooperativeTouchSensor` which is implemented in a file called CooperativeSensors.wrl. One of the key ideas behind the transformation is that a transformed sensor looks like the original sensor to the remainder of the VRML content. The only difference is that an additional `name` attribute has been added. This attribute is needed to identify the cooperative sensor which should receive an incoming event from other participants. The `name` needs to be unique within the session, and can be of the form "UserName:ProcessID:IPAddress:NodeNumber".

It is important to note that the transformation of the VRML model can be done automatically without human intervention. This ensures that the users will be able to share arbitrary VRML models without manual modification of the code.

5.4. Accessing Services

Cooperative applications can access dvr services by using the dvr client-stub API. The client-stub is a C++ library which can be linked to the client. The application programming interface, which was intended to be as simple as possible, offers the following methods to access the dvr service:

- `Connect`. This method allows to establish a connection between client and dvr server. The local port number of the server has to be passed as an argument.
- `Disconnect`. The `Disconnect` method releases the connection to the dvr server.
- `JoinSession`. By using `JoinSession`, the client requests the server to join a multicast session at a specific address supplied by the client.
- `LeaveSession`. The server is instructed to leave a multicast session.
- `LoadVRML`. This method requests the transmission of a VRML animation to all participants (dvr servers) and initiates rendering of the content at all sites. The filename of the root VRML file for the animation needs to be passed as an argument.

6. Integration into the dlb

VRML animations based on the distributed virtual reality service are autonomous items of the shared workspace of the dlb. Since direct rendering of VRML on the shared workspace of the dlb was not feasible, the VRML item is represented by an icon as indicated in Fig. 5. The VRML

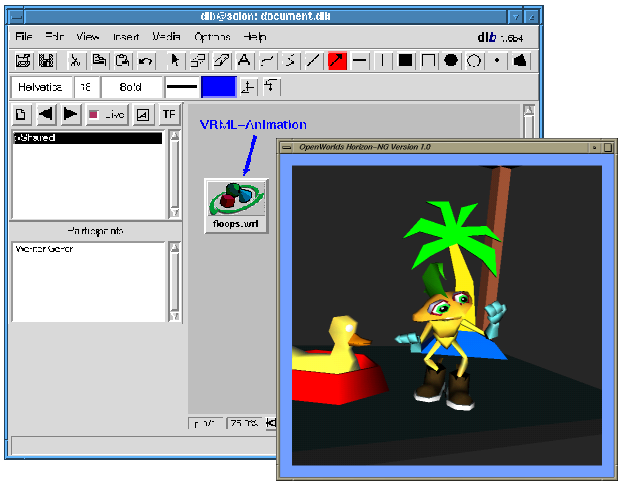

Figure 5. A VRML animation integrated into the dlb

item realizes the interface to the dvr service. It stores the parameters required by the dvr client-stub such as, for instance, multicast IP address, port number, or SMP parameters. After having selected a VRML file, parameters are requested through a dialog. Then a VRML item is created and transmitted to the distributed dlb instances which locally create a corresponding item. The VRML animation is started via mouse click on the VRML icon. Upon this event the dvr service distributes the VRML file(s) to all involved dvr instances and launches a local VRML window for rendering the VRML content (see Fig. 5).

7. Conclusion and Future Work

This paper addressed design issues and architecture of the distributed virtual reality service. The dvr service is based on a replicated client-server architecture on top of IP-Multicast. Since rendering of VRML content is performed locally at each participating site, a high performance and responsiveness is achieved. We have demonstrated the feasibility of our ideas by implementing a prototype of the dvr service. This prototype has been integrated into the digital lecture board, which is a cooperative application for computer-based distance education.

Some open issues still need to be considered in the future such as, for instance, the support for late comers by providing an adequate late join algorithm, the synchronization of viewpoints so as to provide guided tours, or the handling of concurrent user interaction by means of floor control mechanisms.

8. Acknowledgements

This work is funded by the Siemens Telecooperation Center, Saarbrücken, Germany and IBM's European Networking Center, Heidelberg, Germany. Moreover, we would like to thank Tino von Roden for his excellent contribution to the implementation of this work.

9. References

[1] Diefenbach, P., Mahesh P., Hunt D.: "Building OpenWorlds (TM)". In: *Proc. of VRML'98*, February 1998.

[2] Eckert, A., Geyer, W., Effelsberg, W.: "A Distance Learning System for Higher Education Based on Telecommunications and Multimedia - A Compound Organizational, Pedagogical, and Technical Approach". In: *Proc. of ED-MEDIA'97*, Calgary, June 1997.

[3] Geyer, W., Effelsberg, W.: "The Digital Lecture Board - A Teaching and Learning Tool for Remote Instruction in Higher Education". In: *Proc. of ED-MEDIA'98*, Freiburg, June 1998.

[4] Geyer, W., Weis, R.: "A Secure, Accountable, and Collaborative Whiteboard". In: *Proc. IDMS'98*, Springer, LNCS 1483, 1998, pp. 3-14.

[5] Grumann, M.: *Entwurf und Implementierung eines zuverlässigen Multicast-Protokolls zur Unterstützung sicherer Gruppenkommunikation in einer Teleteaching-Umgebung*. Master's Thesis (in German), Lehrstuhl Praktische Informatik IV, University of Mannheim, 1997.

[6] Holfelder, W.: "Interactive Remote Recording and Playback of Multicast Videoconferences". In: *Proc. IDMS'97*, Springer, LNCS 1309, 1997, p. 450-463.

[7] Mauve, M.: "TeCo3D - A 3D Telecollaboration Application Based on VRML and Java". Accepted at *Multimedia Computing and Networking 1999 (MMCN99) / SPIE99*, San Jose, February 1999.

[8] Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V.: *RTP: A Transport Protocol for Real-Time Applications*. Internet RFC 1889, IETF, 1996.

[9] VRML Consortium: Information technology - Computer graphics and image processing - The Virtual Reality Modeling Language (VRML) - Part 1: Functional specification and UTF-8 encoding. ISO/IEC 14772-1:1997 International Standard, URL: *http://www.vrml.org*, December 1997.