TeCo3D - Sharing Interactive and Dynamic 3D Models

Martin Mauve
Praktische Informatik IV
University of Mannheim
L15, 16
68131 Mannheim
mauve@informatik.uni-mannheim.de

ABSTRACT

In this paper we present a method for sharing interactive and dynamic 3D models that are collaboration-unaware, i.e., models that have not been designed to be used by multiple users at the same time. This functionality is an essential requirement for the inclusion of arbitrary 3D models, as generated by standard CAD or animation software, into teleconferencing sessions. A key aim of this work is to show that a large part of the required functionality can be developed in a way so that it is reusable for other applications such as shared whiteboards or networked computer games. Our method therefore consists of both an application dependent part that handles the specific tasks required for sharing 3D models, and of a number of generic services such as synchronization, scalable support for latecomers, and the ability to record and replay sessions. The generic services are based on an abstract media model and the RTP/I application level protocol for distributed interactive media. Any other application for a medium that shares this model and that uses RTP/I may reuse these generic services. We have implemented a prototype called TeCo3D demonstrating the feasibility of our approach.

1. INTRODUCTION

The tremendous increase in the graphics performance of standard PCs has stimulated the development of many innovative applications for interactive and dynamic 3D graphics. Examples include the usage of 3D models for a variety of purposes: to explain the handling and maintenance of physical products, as virtual prototypes for industrial production, as hands-on demonstration material for teaching purposes, and as virtual products for sales support. While the 3D models for these applications are typically developed for interaction with only a single user, it would often be very desirable to be able to use them in a collaborative environment with multiple users at the same time. For example, an employee in a call-center could demonstrate the usage of a product to a remote customer, a teacher in a teleteaching environment could employ interactive 3D models, or a videoconference could be enriched by including a virtual prototype. Generally multi-user capability cannot be integrated into every 3D model, since this would significantly increase the complexity of developing these models. Neither should one have to re-engineer existing 3D models in order to use them in a distributed fashion.

To solve this problem we have developed a mechanism that allows multiple users to share collaboration-unaware 3D models, i.e., models which were not designed for distributed use. By means of this mechanism many users can share an existing single-user 3D model while the application actually performing the mechanism, rather than the 3D-model itself, will contain the complex tasks that such shared usage entails (e.g., consistency maintenance and support of late-comers). We have designed and implemented such an application to demonstrate the feasibility of our approach. It is called TeCo3D (TeleCooperation 3D). TeCo3D enables the distribution of collaboration-unaware 3D models defined using the Virtual Reality Modeling Language (VRML) [23]. The basic mechanism, however, is not limited to VRML models, but can easily be adapted to other 3D description formats.

TeCo3D has been designed in a way that allows large parts of its functionality to be reused for other distributed interactive media, such as shared whiteboards, networked computer games, and distributed virtual environments. In order to achieve reusable functionality TeCo3D consists of a part that explicitly deals with the sharing of 3D models, and of generic parts which can be reused for other applications. TeCo3D's generic parts are based on an abstract media model and the RTP/I application level protocol for distributed interactive media [12]. Any other application for a medium that shares this model and that uses RTP/I may reuse these generic parts, including important aspects such as synchronization, support for late-comers, and session recording.

In this paper we present the mechanism for sharing collaboration unaware 3D models and we discuss the design of the TeCo3D application. In particular we show how we were able to realize a large part of TeCo3D's functionality in a reusable fashion.

The remainder of this paper is structured as follows: In Section Two we show how interactive and dynamic 3D models can be defined using the Virtual Reality Modeling Language. An overview of different ways to share these 3D models is given in the Section Three. Section Four describes a solution to the two main media dependent problems: how to access the state of, and user interactions with, collaboration-unaware 3D models. In Section Five we describe the foundation for the development of generic services, which consists of an abstract media model and the RTP/I application level protocol for distributed interactive media. Three generic services are summarized in Section Six. These include: consistency, scalable support for latecomers, and session recording. In Section Seven we show how the architecture of the TeCo3D application is composed of medium specific functionality and generic services. The experiences with the development and the usage of the TeCo3D application are described in Section Eight. This paper concludes with a summary in the last section.

2. INTERACTIVE AND DYNAMIC 3D MODELS

For the sake of simplicity we assume that dynamic and interactive 3D objects are described using the Virtual Reality Modeling Language. This does not limit the applicability of our work since the methods presented here can easily be adapted to other 3D description formats, such as VRML's successor X3D [27].

The Virtual Reality Modeling Language (VRML) is a "file format for describing interactive 3D multimedia on the Internet" [23]. The presentation of the VRML content (also called a "world") is realized by means of a VRML browser. A detailed introduction to VRML can be found in [1] and [3].

The basic building blocks of VRML are *nodes* (e.g., a sphere node). Nodes contain *fields* (e.g., the radius field of a sphere node) which can be either elementary data types or nodes themselves. The nodes in a VRML file form a hierarchy called a *scene graph*. The VRML browser displays the visible parts of the scene graph. 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.

Figure 1 shows an example of a dynamic and interactive 3D object that has been described in VRML. It is a red sphere that moves when it is clicked on. Lines 4-18 describe the visible geometry of the model - a red sphere with the radius 2. The model is made interactive by the inclusion of a `TouchSensor` in line 19. This sensor generates an event when the red sphere has been touched (i.e., is clicked on). The event is propagated (lines 30-31) to start a timer (lines 20-22). Upon receiving this event the timer starts generating events periodically for 5 seconds. The events generated by the timer are forwarded (lines 32-33) to a position interpolator (lines 23-26) which converts them into 3D coordinates. These coordinates are used as the new position of the red sphere (lines 34-35 and 5).

While VRML provides all the mechanisms required for simple animations, it does not (and was never intended to) support full programming language functionality. However, such functionality is required for complex 3D content, which involves, for example, state information, non-linear interpolation, or network activity. To allow for the modeling of more complex behavior, VRML provides hook-ups for existing programming languages. Those hook-ups are called *script nodes*. Script nodes define an interface between the VRML world and a script written in a regular programming language such as Java.

```
01:#VRML V2.0 utf8
02:Group {
03:  children [
04:    DEF Trans Transform {
05:      translation 0 10 0
06:      children [
07:        Shape {
08:          appearance Appearance {
09:            material Material {
10:              diffuseColor 1 0 0
11:            }
12:          }
13:          geometry Sphere {
14:            radius 2
15:          }
16:        }
17:      ]
18:    }
19:    DEF TouchMe TouchSensor {}
20:    DEF Timer TimeSensor {
21:     cycleInterval 5
22:    }
23:    DEF Interp PositionInterpolator {
24:      key      [0, 0.5, 1]
25:      keyValue [0 10 0, 10 0 0, 0 -10 0]
26:    }
27:  ]
28:}
29:
30:ROUTE TouchMe.touchTime TO
31:      Timer.startTime
32:ROUTE Timer.fraction_changed TO
33:      Interp.set_fraction
34:ROUTE Interp.value_changed TO
35:      Trans.translation
```

Figure 1: VRML description of a moving sphere

In addition to script nodes, VRML browsers can provide another API for programming language interaction: the *External Authoring Interface (EAI)* [25]. 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 itself does not define any mechanisms for the distributed presentation of and interaction with 3D objects. Instead, script nodes or the EAI must be used to realize multi-user functionality.

3. SHARING 3D MODELS

The typical approach to share 3D models among a spatially distributed group of users is to develop them specifically for this purpose. The driving forces behind this approach are multi-user 3D worlds [24,18]. These multi-user worlds allow participants to enter a virtual world, enabling them to interact with the representations of other participants as well as with virtual objects. Users are represented by 3D shapes known as *avatars*. Typically, the author of 3D models for multi-user worlds designs them specifically to support distribution. Therefore, we call these 3D models *collaboration-aware*.

However, the vast majority of 3D models is not designed for use in a multi-user world. This is quite natural since multi-user worlds are only one small area where 3D models can be employed. Moreover, making the models collaboration-aware is costly in terms of additional complexity. We call 3D models without support for multiple users *collaboration-unaware*. It is the aim of our work to enable the sharing of collaboration-unaware 3D models.

One approach to support collaboration-unaware 3D models is to share the output of a VRML browser with an application sharing tool. Application sharing allows a spatially distributed group of users to view and interact with an unmodified single-user application. It is typically realized by intercepting the communication between the application and the window system. Doing this, an application-sharing software is able to multiplex the output of a single-user application to multiple users and to forward their actions to the application [16,14].

Unfortunately, application sharing is quite unsuitable for sharing 3D content for the following reasons:

- The required data rate is extremely high. In order to be of acceptable quality 3D content needs to rendered with at least 20 to 30 frames per second. The sharing of 3D content would therefore produce a data rate similar to the transmission of an uncompressed video stream. Even if an application sharing tool used a compression algorithm with high compression rates (which current application sharing tools do not do), the resulting data rate would still be very high.
- The shared VRML browser cannot use the support of 3D accelerators, since the output of the shared application must be transmitted to the other participants of a session. If this output is calculated and stored only by a 3D accelerator, it cannot be accessed for transmission.
- The computer that runs the VRML browser is a single point of failure and a potential bottleneck for the whole system.
- Since there exists only one application, application sharing does not allow different users to view the model from different perspectives.

It is fairly easy to experience the impact of these problems by using freely available application sharing software (e.g., Netmeeting [14]) to share an arbitrary VRML browser. Even with high-end computers and very simple 3D models the results produced by this approach are generally unacceptable.

In order to avoid these problems we propose to use a fully replicated distribution architecture. That is, a separate instance of an application for the sharing of 3D models runs on each participant's computer. This application consists of a VRML browser that is used as a 3D presentation engine, and some additional software controlling the VRML browser via the External Authoring Interface. The user can instruct the additional software to load a collaboration-unaware VRML model. The initial state of the model will then be transmitted to all participants of a session and presented by the VRML browser of each participant. It is the job of the additional software to make sure that the local instances of the 3D model remain synchronized, so that the users get the impression that they all are interacting with the same model. This requires that two fundamental problems be solved:

- **State access**. It is vital to be able to get and set the state of the 3D model. Session participants must be provided with an initial state, latecomers with the current state, and regular participants may need to be re-synchronized with a state transmission if serious problems (e.g., network partitioning) occur. Access to the state of collaboration-unaware VRML models poses a problem since standard VRML browsers do not allow the retrieval of that information, and no standardized format exists for the representation of VRML state.
- **Sharing user interaction**. Provided that all participants received the same initial state for their local instance of the 3D model, these instances will stay synchronized as long as there are no user interactions with the model. This is true since all other state changes of the 3D model are deterministic and depend exclusively on the passage of time (e.g., an animation as described above). However, as soon as a user starts interacting with the local instance, the 3D model is in danger of becoming inconsistent. To prevent this a user action needs to be transmitted to all participants of a session and then be executed at all sites. However, one prerequisite of our approach is that the 3D model is collaboration-unaware. By definition such a model does not contain any mechanisms to capture local user actions or to inject remote user actions into the local model. How to circumvent this limitation is the second key problem that has to be solved.

It should be noted that additional problems need to be addressed in order to enable the sharing of collaboration-unaware 3D models. Examples are how to ensure synchronization among all participants and how to support latecomers in a scalable manner. However, these problems are generic in nature and solutions to them should be reusable for other media, such as shared whiteboards and networked computer games. In the following section we will present solutions to the two fundamental, medium dependent problems before tackling the medium independent problems in Sections Five and Six.

4. MEDIUM SPECIFIC PROBLEMS

4.1 State Access

VRML browsers currently do not offer a standardized way to save and restore the state of arbitrary VRML objects and worlds. This is a severe constraint for applications which use VRML browsers as 3D presentation and execution engines. These applications are not restricted to 3D teleco-operation applications for the sharing of 3D models. Rather they are a large class of applications ranging from 3D authoring tools to 3D presentation applications and multi-user virtual reality systems. None of these applications is currently able to access the state of arbitrary VRML content, even though this is vital to support fundamental functionality such as saving and restoring a certain state or transmitting it to a communication peer.

Given the universal need for such functionality, we developed an extension to the External Authoring Interface (EAI). This extension allows external applications to get and set the state of arbitrary VRML content in a standardized manner. In order to support diverse applications, the proposed methods allow not only the retrieval of a virtual world's full state, but also retrieval of the states of individual 3D objects. Since the results of state access should be independent of browser implementations, we also specified an encoding for state information. Data in this form is either produced or consumed during state access. To encode state information we use an efficient, easy-to-parse binary encoding. A detailed description of the extended EAI and the definition of the state format can be found in [13].

4.2 Sharing User Interaction

In order to solve the problem of sharing user interactions with collaboration-unaware 3D content, the sensors of the original model are automatically replaced with pre-built, specialized script nodes (e.g., a cooperative version of a touch sensor). These cooperative sensors have the same VRML interfaces as the original sensor. They register with the application controlling the VRML browser and allow it to capture events produced by the local user and to inject remote events into the local 3D model. The remaining parts of the 3D model need not be changed in any way. Automatically processing the collaboration-unaware model results in *semi-collaboration-aware* VRML content that can be shared by TeCo3D.

In order to demonstrate how the processing of 3D models works, Figure 2 (a) shows an excerpt from a standard VRML file. This fragment of VRML code contains a `TouchSensor` (line 13 - 14), waiting for the user to touch (click on) some geometry. As the geometry is of no further interest in this example, it is just indicated by three dots. The second element in this example is a `TimeSensor` (lines 15 - 17). As explained in Section 2, the user can activate the `TouchSensor` by clicking on it. In Figure 2 (a) this causes the `TimeSensor` to start ticking, which can then be used to control an animation, as in the moving sphere example.

The processing needed to transform the original VRML content into a semi-collaboration-aware VRML model is shown in Figure 2 (b). For each sensor listening to user input, an alternative implementation is supplied which is called CoopX, where X is the name of the original sensor. In this example a `CoopTouchSensor` is declared in lines 3 - 10. The declaration indicates that the actual implementation is contained in a file called CoopSensors.wrl.

Since the customized cooperative sensor has the same interface as the sensor it replaces, the remaining VRML code looks very similar to that in Figure 2 (a) except that the `TouchSensor` has been changed into a `CoopTouch-Sensor` (line 13).

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

(a)

```
#VRML V2.0 utf8

EXTERNPROTO CoopTouchSensor [
  field        SFString name
  eventOut     SFTime   touchTime
  ...
]
[
 "../CoopSensors.wrl#CoopTouchSensor"
]
Group {
  children [
    DEF TouchIt CoopTouchSensor {
    }
    DEF Timer TimeSensor {
      cycleInterval 5
    }
    ...
  ]
}
ROUTE TouchIt.touchTime TO
  Timer.startTime
```

(b)

Figure 2: Example of VRML content processing for TeCo3D

It is important to notice that the transformation of the VRML model can be done *automatically* without human intervention. This ensures that the user perceives TeCo3D to provide true support for sharing collaboration-unaware VRML models.

Figure 3 is a code excerpt from the `CoopSensors.wrl` file. It shows how the `CoopTouchSensor` is realized. The sensor consists of two parts: a regular `TouchSensor` and a `Script` node referencing a Java class. An object of the referenced class is generated whenever a `CoopTouchSensor` is used in a VRML model. This object registers with the application for the sharing of 3D content as soon as it is created, establishing the link between VRML content and the application.

```
 1 #VRML V2.0 utf8
 2
 3 PROTO CoopTouchSensor [
 4   eventOut     SFTime   touchTime
 5   ...
 6 ]
 7 {
 8   DEF TheTouchSensor TouchSensor {
 9     ...
10   }
11   DEF TheCoopTS Script {
12     url   "TeCo3D.netsensors.
13           CooperativeTouchSensor.class"
14     field SFString theName IS name
15     ...
16     eventIn  SFTime set_touchTime
17     eventOut SFTime touchTime_changed
18                     IS touchTime
19   }
20   ...
21   ROUTE TheTouchSensor.touchTime TO
22         TheCoopTS.set_touchTime
23 }
```

Figure 3: Implementation of cooperative sensors

The `TouchSensor` (lines 8 - 10) is responsible to provide the functionality of a standard `TouchSensor` for the `CoopTouchSensor`. The output of the `TouchSensor` is routed (line 21) to the Java object of the `Script` node. Whenever an event is generated by the `TouchSensor`, this Java object notifies the application. The application can then transmit the event to its peer instances. Vice versa, the application can inject remote user interactions into the script node, making the 3D model change its state as if the local user had interacted with the model.

5. GENERALIZATION

Having solved the two media dependent problems, it is possible to access the state of arbitrary 3D models and to intercept user interactions with a collaboration-unaware model. While this functionality is certainly most important for the sharing of collaboration-unaware 3D models, other issues still need to be addressed. For example, latecomers need to be supported in a scalable way and a mechanism is needed to make sure that user interactions are executed simultaneously at all sites. It is common that these additional problems are solved in a medium specific fashion for a single application, e.g., a shared whiteboard implementation or a single networked computer game. In fact we have done so, too, in a very early prototype of TeCo3D [10]. However, further consideration reveals this to be a waste of resources. Neither could the solution developed for TeCo3D be reused for other applications nor can existing solutions be reused for TeCo3D.

In order to make the solution to common problems for distributed interactive media reusable, we propose to generalize them. To this end we defined an abstract media model, so that problems and solutions can be discussed independent of any specific medium. Furthermore, we designed an application level protocol called RTP/I, which captures the common aspects of this media class. Generic services, such as synchronization and latecomer support, can then be developed in a reusable fashion based only on the information provided by the media model and the application level protocol rather than on medium specific knowledge.

5.1 Media Model

In order to provide a generic service that is reusable for a whole class of media, it is important to investigate the commonalities of this media class. In the following we give a brief overview of the characteristics of the distributed interactive media class. A more detailed discussion can be found in [13].

States and Events

A *distributed interactive medium* has a *state*. In TeCo3D this is the state of the shared 3D models. In order to perceive the state of a distributed interactive medium, a user needs an *application*, e.g., the TeCo3D application allows a user to view the state of shared 3D models. The application generally maintains a local copy of (parts of) the medium's state. Applications for distributed interactive media are therefore said to have a *replicated distribution architecture*. For all applications participating in a session the local state of the medium should be at least reasonably similar. To guarantee this, it is necessary to synchronize the local copies of the distributed interactive medium's state among all participants, so that the overall state of the medium is *consistent*.

The state of a distributed interactive medium can change for one of two reasons, either by *passage of time* or by *events*. The state of the medium between two successive events is fully deterministic and depends only on the passage of time. Generally, a state change caused by the passage of time does not require the exchange of information between applications, since each user's application can independently calculate the required state changes. An example of a state change caused by the passage of time is the animation of an object (such as the moving red sphere).

Any state change that is not a fully deterministic function of time is caused by an *event*. Generally events are (user) interactions with the medium, e.g., the user clicks on the red sphere to start an animation. Whenever events occur, the state of the medium is in danger of becoming inconsistent. Therefore, an event usually requires the applications to exchange information - either about the event itself or about the updated state once the event has taken place. For the sharing of 3D models the events are the user actions as they are captured by the cooperative sensors.

Partitioning the Medium - Sub-Components

In order to provide for a flexible and scalable handling of state information, it is desirable to partition an interactive medium into several *sub-components*. In addition to breaking down the complete state of an interactive medium into more manageable parts, such partitioning allows the participants of a session to track only the states of those sub-components in which they are actually interested. Examples of sub-components are individual shared 3D objects or the pages of a shared whiteboard.

Discrete vs. Continuous Media

Distributed interactive media can be sub-divided into *discrete* and *continuous* distributed interactive media. While discrete distributed interactive media (e.g., shared whiteboards) change their state only in response to events, continuous distributed interactive media (e.g., shared dynamic and interactive 3D models) may also change their state because of the passage of time. In order to ensure consistency discrete media need to make sure that events are applied to the shared state in the correct order. Continuous media must execute events in the proper order and at the correct point in time. If these conditions are not met the state of the distributed interactive medium may become inconsistent and the shared state may require a repair.

5.2 RTP/I

While the media model provides a first insight into the distributed interactive media class, the design and implementation of generic functionality requires a more formal foundation. The Real Time Application Level Protocol for Distributed Interactive Media (RTP/I) [12] provides such a foundation.

RTP/I consists of two parts: a data transfer protocol for the transport of event and state information, and a control protocol for meta-information about the medium and the participants of a session:

- The *data transfer protocol* (RTP/I) frames the transmitted states and events of the medium with information that is common to the distributed interactive media class. Given this information generic services can interpret the semantics of events and states without knowing anything about their medium-specific encoding. Typical examples of the information contained in the RTP/I data framing are a timestamp, which indicates at what time an event happened or a state was calculated, an identifier for the affected sub-component, and the type of the data (event vs. state information). In addition to state and event transmission RTP/I is also used to request the state of a sub-component in a standardized manner.
- The *RTP/I control protocol* (RTCP/I) conveys information about the participants of a session. This includes the participants' names and email addresses. This information can be used to establish a light-weight session control. Moreover, RTCP/I provides information about the sub-components that are present in a session. Information about each sub-component is regularly announced. It includes the identifier and the application level name of the sub-component.

RTP/I is closely related to the Real Time Transport Protocol (RTP) [7], which is mainly used for the transmission of audio and video. However, while RTP/I reuses many aspects of RTP, it has been thoroughly adapted to meet the needs of distributed interactive media.

6. GENERIC SERVICES

Based on the media model and RTP/I we are able to provide additional functionality for TeCo3D in the form of generic services that can be reused for other distributed interactive media. We have developed three generic services: a consistency service for continuous distributed interactive media, a late-join service for the scalable support of latecomers, and a recording service for the recording and replay of sessions involving distributed interactive media. While TeCo3D is the first application to use these services there are currently three additional applications that are in the process of adopting them: a shared whiteboard (mlb) [22], a toolkit for remotely controlled Java applets for teleteaching [8], both developed at the University of Mannheim, and the Audio Enabled Multicast VNet project [19] at the Communications Research Center in Ottawa/Canada.

6.1 Consistency

Consistency has long been a research issue in the study of *discrete* distributed interactive media. Generally speaking, consistency for discrete distributed interactive media entails finding a 'correct' sequence of all the events generated by the participants of a session and making sure that, at all participating sites, the state of the medium looks as if all events had been executed successfully in that particular sequential order [20,9]. A number of different approaches exist to ensure consistency for discrete distributed interactive media. These range from a strict single user floor control policy to more sophisticated algorithms such as operational transformation [20].

To understand the challenge behind achieving consistency for *continuous* distributed interactive media, consider the following example: Imagine a shared 3D model of a train that is approaching a switch. The session is attended by two spatially separated users, U_a and U_b. Just before the train arrives at the junction, U_a operates the switch (e.g., by clicking on it). In the copy of U_a's model the event takes place immediately. However, the information about U_a having operated the switch will arrive at the copy of U_b's model at a later point in time. Applying the operation at this point in time to U_b's copy could lead to an inconsistent state, because the train might have already passed the switch in U_b's copy of the model.

As shown by the example, consistency for *continuous* media requires that the result of all events look as if the events had been executed at the correct point in time in addition to having been executed in the proper order. The algorithms for establishing consistency in the discrete domain can therefore be regarded as insufficient for the continuous domain.

The first step to understanding how to realize consistency for continuous distributed interactive media is to specify a consistency correctness criterion. This criterion defines the term consistency for the continuous domain. In a second step we will then show how this criterion can be guaranteed through the use of a generic consistency service.

Consistency Correctness Criterion

A continuous distributed interactive medium is a distributed medium that changes its state in response to (user initiated) events as well as in response to the passage of time. This definition implies that the medium has access to a physical clock that can be used to measure the progress of time. We assume that the physical clocks of all the participants are reasonably synchronized, e.g., using NTP [15] or GPS clocks.

In continuous distributed interactive media a user-initiated event needs to be executed at a specific point in time, denoted by its timestamp. We assume that there exists a total ordering on all events established by using the timestamps of the events and an additional tiebreaker (e.g., the IP-address of the sender) for simultaneous events. This ordering relation is called *complete physical time ordering relation*.

The *consistency criterion for continuous distributed interactive media* can then be specified as follows: A continuous distributed interactive medium is *consistent* if some time after all events have been executed at all sites, the state of the medium at all sites is identical to the state which would have been reached by executing all events in the order given by the complete physical time ordering relation at the physical time denoted by the timestamps of the events.

Ensuring the Consistency Correctness Criterion

We have implemented a generic service that guarantees the consistency criterion for continuous distributed interactive media. The fundamental idea of this service is to delay events that are generated by the local user until it is likely that all other participants will have received the event. That is, the timestamp of an event is greater than the time the event was issued by the user. We call this method local lag [11]. Given local lag the event can be executed simultaneously at all sites. This comes at the cost of a lower responsiveness of the medium, since events are not executed immediately. Preliminary perceptional psychological experiments have shown that a delay value of below 80 ms cannot be noticed by a user independent of the performed action (e.g., click event, drag event, etc.). Furthermore, users adapt to the local lag rather quickly, making delays of up to 200 ms acceptable even if they are noticeable. A value of 100 to 200 ms is generally sufficient, even for intercontinental sessions, to compensate the network induced latency.

While the introduction of local lag makes it likely that an event is executed simultaneously at all sites, it cannot guarantee this. After all, the transmitted event might need multiple retransmissions due to packet loss before it arrives successfully at a given participant. In this case a state repair will be required, since the overall state of the medium is in danger of becoming inconsistent. In general there are multiple algorithms that could be used to repair the problem [11]. In our generic consistency service a participant who receives an event late requests the transmission of the correct current state from another participant. In order to make sure that the participant who replies to a request holds a valid state for the affected sub-component, we use a floor control mechanism. Only the holder of the floor for a specific sub-component is allowed to generate events and to reply to state queries for that sub-component. The floor holder therefore always knows the correct state of the sub-component. The floor control mechanism we use is robust to the failure of a floor holder and it can cope with duplicate floor holders. Moreover, the floor can be passed to another participant in a way that guarantees that the new floor holder has a correct state. More details about the floor control mechanism and consistency in continuous distributed interactive media can be found in [13].

Both the local-lag-based consistency mechanism and the floor control mechanism are realized as generic services that rely exclusively on the media model and RTP/I. Other RTP/I-based applications may reuse them without further modifications.

6.2 Late-Join

When a participant joins an ongoing session it is necessary to provide the application of this participant with the current states of the sub-components present in the session. For example, if several 3D models are shared in a session, a latecomer will need to receive the state of these models before being able to participate in the session.

Since distributed interactive media regularly involve a group of multiple users, it is important that this be done in a scalable and efficient manner. We have developed a generic late-join service with these properties [21]. It can be used for arbitrary distributed interactive media.

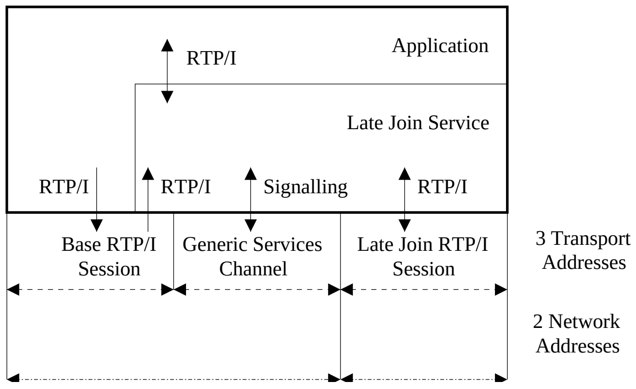

Figure 4: Architecture of the late-join service

The architecture of the generic late-join service is depicted in Figure 4. The late-join service intercepts the data (events, states, and requests for states) that arrives from the base session. Since the data is transmitted using RTP/I the generic late-join service can understand the semantics of this data to a degree sufficient to provide the late-join functionality. Knowledge about the medium-specific encoding is not required. After examining the data the late-join service forwards it to the application.

The application transmits all regular data directly to the base RTP/I session without informing the late-join service.

Late-join information is handed from the application to the late-join service. An example of this type of information is the state of a sub-component that is required by the late-join service to support a remote latecomer. The reason for passing this data to the late-join service instead of transmitting it over the base RTP/I session is as follows: the generic late-join service maintains an additional late-join RTP/I session. This session is used to transmit all late-join oriented data. The late-join service joins and leaves this additional RTP/I session at appropriate times. This ensures that only a small subset of all participants needs to handle late-join data.

Finally there exists a generic services channel. This channel is shared by all generic services. It is used to convey signalling data for the generic services. An application remains a member of the generic services channel for the lifetime of a session.

When joining an ongoing session the late-join service will learn about the sub-components that are present in a session through the RTCP/I reports on sub-components. Whenever a new sub-component is detected the late-join service informs the application and requests information on how to perform the late-join for that sub-component. The application may choose between a set of policies:

- **No late-join**. The application is not interested in the state of the sub-component.
- **Immediate late-join**. The application needs the state of the sub-component as fast as possible.
- **Event-triggered late-join**. The application needs the state of the sub-component only when an event is received that targets this sub-component.
- **Bandwidth-oriented late-join**. The application requires that the state of this sub-component is only requested when the bandwidth consumed by the session falls below a certain limit.

When the condition occurs that was specified by the policy, the state of the sub-component is requested by the late-join service. This is done by transmitting RTP/I state queries to the late-join group using a message implosion avoidance mechanism. This mechanism makes sure that no message implosion occurs if multiple latecomers want to request the state of the same sub-component at the same time. A similar mechanism was first used by SRM [4] to achieve a scalable reliable multicast. In order to avoid a message implosion, the late-join service waits a random length of time before transmitting an RTP/I state query. Any other late-join service that wants to send a state query for the same sub-component suppresses this message when it sees that the message has already been transmitted by someone else. This can effectively prevent a message implosion.

The request is repeated if there is no answer after a certain amount of time. If multiple requests for the sub-component's state go unanswered, it is concluded that there is no application present in the late-join group who can send the state of that sub-component. In this case the late-join service will use the generic services channel to request that a participant who is able to transmit the state of the sub-component joins the late-join session and transmits the state. If this too fails the application is informed.

Once the late-join service receives the desired state information, it passes it on to the application and marks this sub-component as complete. When there are no new sub-components detected for a period of time and all sub-components have been marked as complete, then the late-join is finished. At that time the late-join service may leave the late-join group. However, in order to serve other latecomers an application will stay a member of the late-join group for a certain period of time even when it has completed its late join. This period of time is dynamic and depends on the value the local application has for other late-comers. For example, the more states of sub-components the application can serve to other latecomers, the longer will it stay a member of the late join RTP/I session.

By means of the generic late-join service arbitrary applications that rely on RTP/I can join an ongoing session in a scalable and efficient way. The ability to specify policies allows easy adaptation of the service to the specific needs of a given application.

6.3 Recording

The ability to record a session is one of the most important and universally needed aspects of video-conferencing. So far the generic recording of audio and video streams has been investigated and understood to a large extent [6]. Due to the lack of a common framing protocol, a *generic* recording service for distributed interactive media was long thought to be infeasible.

However, since RTP/I provides the required information, we have been able to develop a generic recording service that can be used to record and replay sessions involving both RTP-based audio and video streams and RTP/I-based distributed interactive media sessions [5]. The recording service allows near-random access to the recording and it is based exclusively on the common aspects of audio/video and distributed interactive media as they are exposed by RTP and RTP/I.

The main idea of the generic recording service for distributed interactive media is to record the RTP/I packets as they are transmitted during a live session. Later on, the stream of packets can be replayed in the proper order and timing as provided by the RTP/I header information. The timestamps of the replayed packets need to be re-calculated so that they match the replay time. The stream produced in this way can be interpreted and presented to a user by an unmodified application, just like the stream of a regular session. It is therefore not necessary to develop special viewers for the recorded sessions.

One challenging aspect of the development of a generic recording service for distributed interactive media is the realization of random access to the recorded stream. The main problem here is that the listening applications need to get the current state of the medium before they are able to follow the recorded stream. Since the recorder is generic, it will not be able to calculate this required state information. Instead it will need to transmit a combination of recorded packets which puts the receivers into the desired state at the access position. If the medium has just one sub-component and if the state of that sub-component is inserted frequently into the live stream, a very simple mechanism for random access would be: Start the reply at a time close to the desired access point, where a state transmission was recorded. In the event that the live stream of a medium does not normally contain frequent state transmissions, the recording service will be able to request this information during the recording by using the appropriate RTP/I mechanisms.

A more sophisticated random access method which supports realistic applications with more than one sub-component or discrete media types with heavy weight states can be found in [5].

7. TECO3D ARCHITECTURE

The TeCo3D architecture (see Figure 5) combines the elements presented so far to form an application which is able to share collaboration-unaware 3D models. All elements have been developed completely in Java. As a 3D presentation engine we use the Java3D VRML browser [26], which we have extended to support the state access methods described above. At the start-up of the application, the VRML browser is loaded with an empty VRML world. When the local user imports a collaboration-unaware 3D object into the shared workspace, this object is processed so that it becomes semi-collaboration-aware. The processed object is then added to the VRML world by means of a regular EAI call. As the object is loaded into the VRML browser, the collaborative sensors register with the application. In this process they are assigned a unique ID so that remote events can be routed to the correct cooperative sensor.

The browser and the cooperative sensors are controlled by TeCo3D specific functionality. This is accomplished by using EAI calls and direct access to the script nodes that implement the cooperative sensor functionality. The TeCo3D specific functionality is supported by the generic functionality for consistency and latecomers. In turn, these generic services rely on RTP/I.

The generic recording service is not part of the TeCo3D architecture. Rather it is a separate application that connects to an existing TeCo3D session, either to record the session or to replay a previously recorded one.

Reliable communication over UDP multicast and UDP unicast is enabled by a reliability service. Currently we use the iBus library for this purpose [17]. In future versions of TeCo3D we will use a reliability service that is tailored to the specific needs of distributed interactive media. For example, such a reliability service would allow us to add redundancy to the transmission of events, so that state repairs due to lost and retransmitted events would become less likely.

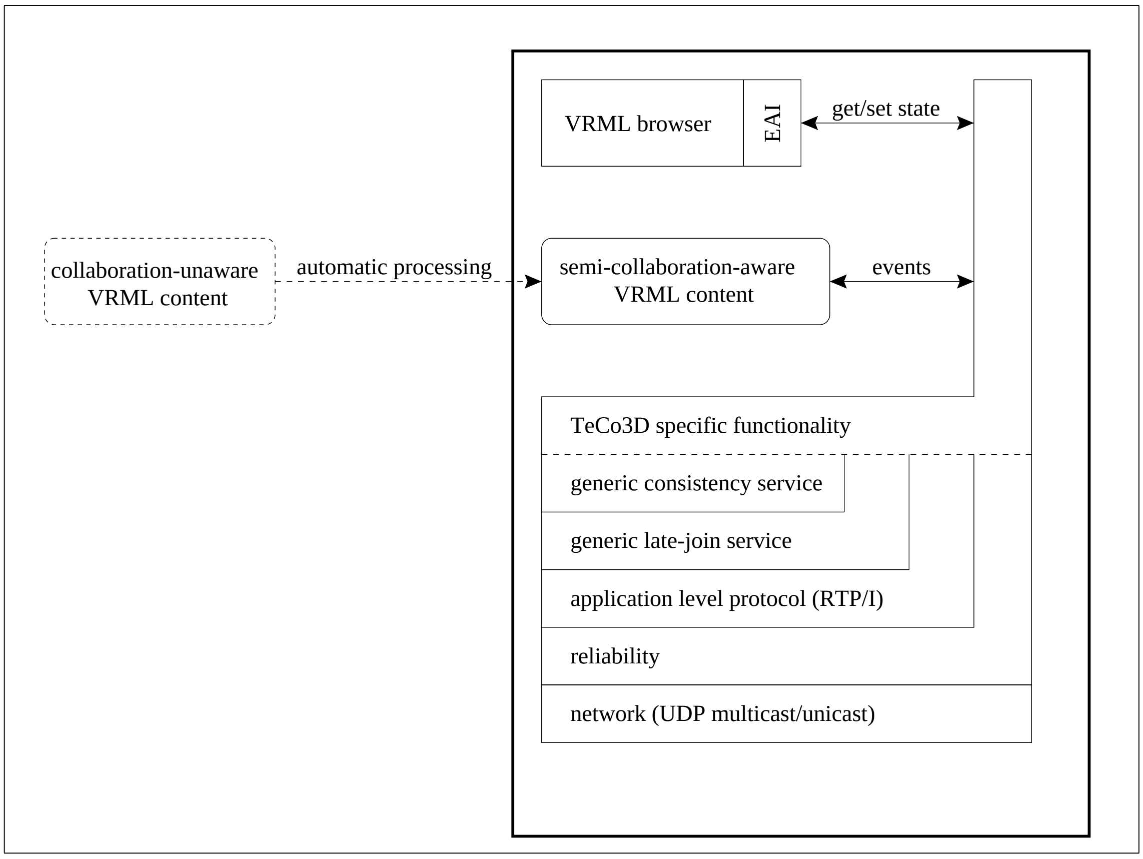
VRML browser
EAI
get/set state
collaboration-unaware VRML content
automatic processing
semi-collaboration-aware VRML content
events
TeCo3D specific functionality
generic consistency service
generic late-join service
application level protocol (RTP/I)
reliability
network (UDP multicast/unicast)

Figure 5: TeCo3D architecture

Once a 3D model has been loaded into the shared workspace of a single participant two important things might happen. First, if there is more than one participant present in the session, the initial state of the 3D model needs to be extracted from the VRML browser and transmitted to the other participants. Once a remote application has received this state information, it can load the 3D model into the shared workspace. Besides the initial transmission of a new sub-component's state, the ability to extract, transmit, and set the state of sub-components is also needed to support the synchronization and the late-join services.

The second thing that could happen is that the user starts interacting with the 3D model. The actions of the local user are captured by the cooperative sensors that have been inserted into the model during the pre-processing. When informed about a local event, the TeCo3D specific functionality takes two actions. First, it hands the event to the generic synchronization service, which is responsible to ensure that the event is delayed by the proper amount of local lag before it is applied to the local 3D model. Second, it is handed to RTP/I for framing and transmission. After the time specified for the local lag has elapsed, the synchronization service hands the event back to the TeCo3D specific functionality, which in turn injects the local event into the 3D model.

When a remote event is received, it is handed directly to the synchronization service. There it is checked whether the event was received in time. If this is the case, then the event will be buffered for the time that this event arrived early. After this buffering it is treated exactly like a local event (i.e., it is forwarded to the TeCo3D specific functionality and then injected into the 3D model). If the event has not been received in time (e.g., because of packet loss and retransmission), the synchronization service triggers the repair of the damaged state.

The TeCo3D architecture shows that only a minimal part of the application needs to be developed in a media-specific way. The generic services, the handling of the application level protocol, and the reliability mechanisms are all realized in a reusable form.

8. EXPERIENCES AND PERFORMANCE

The telecollaboration with collaboration-unaware VRML content using TeCo3D has been demonstrated with different VRML models on several occasions. The simple example displayed in Figure 6 shows how a piece of furniture can be assembled from a number of parts. It is a single-user model that has been taken from a web page. With the help of TeCo3D this single-user model can also be used in a cooperative fashion. SIEMENS plans to use the TeCo3D application as a starting point for the development of a 3D telecollaboration product for call-center and help-desk solutions.

Breaking down the application into several generic components has proven to be a good idea. The reusability of components is very important for distributed interactive media, especially since they tend to be complex and tedious to implement.

In addition to the reusability argument, the subdivision into components also improved the overall architecture of TeCo3D. In this context it is noteworthy that the late-join service has been designed and implemented completely separate from the remainder of the TeCo3D application. Nevertheless, the integration of the service into the TeCo3D application took only a couple of hours and worked right away. This shows that the concept of generic and reusable services based on RTP/I is valid and provides a significant improvement over a situation in which services are application-dependent.

While we were well satisfied with the overall performance of the TeCo3D prototype in almost all tests we have conducted, one issue needs to be addressed before it can be brought to product-level quality:

With the Java3D VRML browser that we used for our prototype the state access is rather slow. Extracting and encoding state information can be done within an acceptable length of time, in the range of 10ms to 200ms on a computer with a Intel PII 400 processor. However, the decoding of state information and the creation of the 3D model from this information is very slow for complex VRML models (e.g., the well known Floops cartoon). It can range between 50ms and several seconds. Values of more than 200ms to 400ms do lead to a noticeable delay and distract the user. We attribute this performance problem to the Java implementation of the state decoding functionality.

Currently all major VRML browsers have become available in sourcecode format. We would therefore recommend to use a C/C++ based professional VRML browser implementation as the 3D presentation engine for a TeCo3D product. A prime candidate would be the blaxxun Contact VRML browser [5]. This should alleviate the performance problem of decoding VRML state.

Besides experimenting with the TeCo3D prototype we also evaluated the performance of the generic consistency and late join services. The performance of the recording service was not evaluated since it does not provide time critical functionality.

At the start of this prototype's development we anticipated problems with the consistency service for two reasons:

- Java might be too slow to provide for accurate synchronization, and
- NTP might be too inaccurate, causing frequent short-term inconsistencies even with high values for local lag.

Experimenting with our prototype, we were surprised to learn that neither problem occurs in reality. A whole passage of an event from the generation inside the VRML browser on one machine to the delivery of the event to the VRML browser of another machine took less than 6ms in a local network with two 400MHz Intel PII PCs. This includes all protocol overhead and the processing in the synchronization service. Furthermore, NTP proved to be very accurate. Even when NTP servers from different countries were used, the time offset rarely exceeded 20 to 30ms. These values are very acceptable for local-lag-based synchronization.

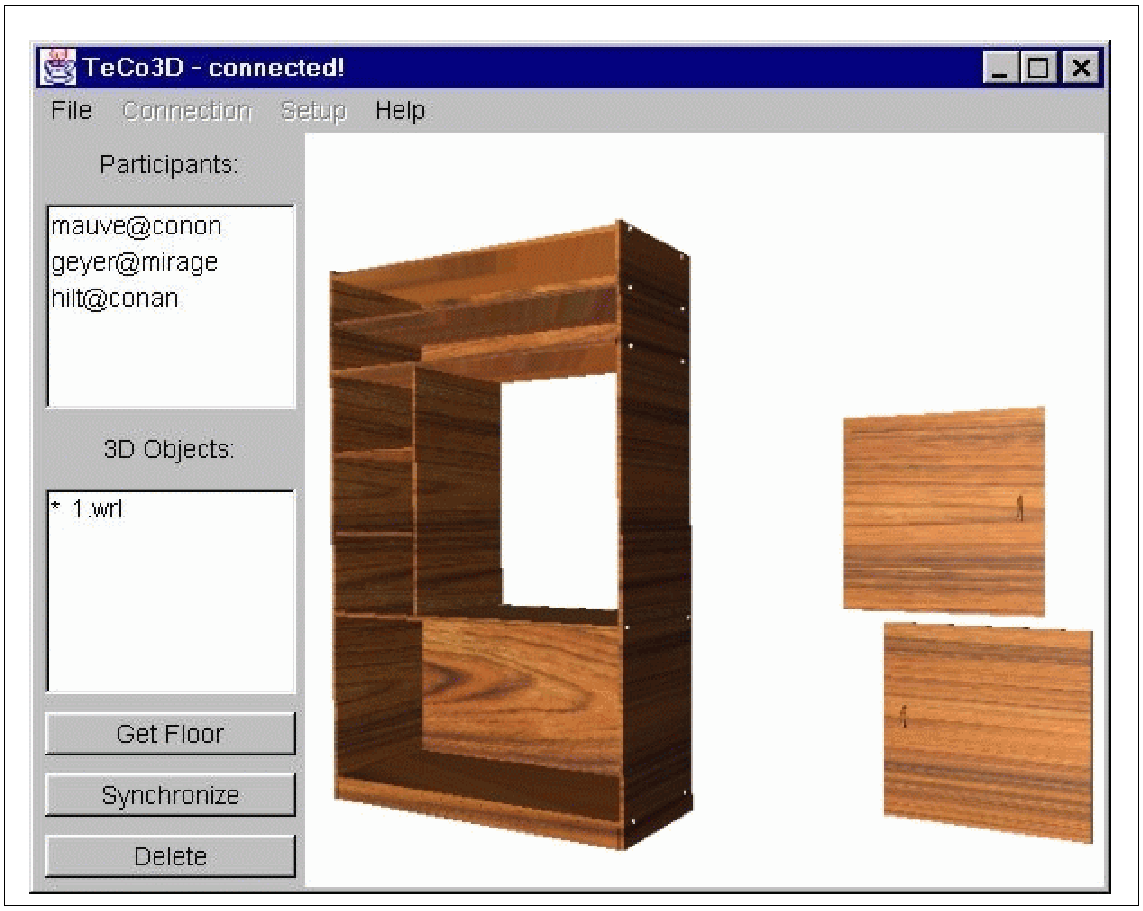

Figure 6: Screen Shot of TeCo3D

To further examine the performance of the consistency service we calculated the average amount of time required to repair an inconsistent state that could not be prevented by local-lag. In general the amount of time required for this task consists of three main components: (1) the network latency, (2) the time needed by the floorholder to extract and transmit the state for the affected sub-component and (3) the time required by the application to decode and apply that state. Given average network delays of 50ms to 100ms for a session within one continent, and the time required to encode and decode the state of 3D models as discussed above, we expected that TeCo3D will correct an inconsistent state within a fraction of a second for low to medium complexity models while complex models such as a full Floops cartoon might take multiple seconds - dominated by the slow state decoding in the prototype.

Preliminary experiments validated this expectation. The repair of an inconsistent moving sphere needed less that 50ms on a LAN, while the repair of an inconsistent Floops cartoon required several seconds. This suggests that the performance of the consistency service is acceptable while the state encoding in the VRML browser needs further improvement.

The performance of the late join service can be split into two parts. The first part is the amount of time required until a late-comer knows about the existence of all sub-components in a session. The second part is how long it takes to request and receive the state of an individual sub-component. As long as there is an application present in the late join group which can answer the state request immediately, the second part is very similar to the situation described for repairing an inconsistent state. We therefore concentrate on examining how long it takes for a late-comer to receive information about all sub-components.

The RTP/I control protocol transmits information about the sub-components present in a session in regular intervals. These intervals are determined so that the control protocol does not exceed a given bandwidth. The time required for a late-comer to learn about all sub-components in a session is therefore proportional to the number of sub-components and the average amount of data transmitted for each sub-component. The amount of data transmitted for each sub-component, in turn, is dominated by the length of the application level name chosen for the sub-component. For example, in a session with 100 sub-components, average application level names of 20 bytes, and a maximum RTCP/I bandwidth of 4 kbit/s each sub-component is announced once every 5 to 6 seconds. Summarizing we expected that as long as there is no packet loss, a late-comer will know about all sub-components within a small amount of time on the order of a few seconds. This expectation was met by experiments with the TeCo3D prototype.

9. CONCLUSION

We have presented a method for sharing interactive and dynamic 3D models that are collaboration-unaware. The TeCo3D application discussed here requires that these 3D models be defined in VRML. However, our solution can easily be adapted to other 3D description formats such as X3D, the successor of VRML.

We proposed to use a distributed approach for TeCo3D since centralized architectures like application sharing have severe drawbacks for the sharing of 3D content (e.g., bandwidth usage, inability to use 3D accelerators, single point of failure). The problems that had to be solved to enable a distributed architecture were divided into two classes: those that are specific to the sharing of 3D models, and general problems that also occur for other distributed interactive media.

The first media specific problem was how to access the state of arbitrary 3D models. We solved it by extending the External Authoring Interface of an existing VRML browser with the capability to transparently get and set the state of arbitrary VRML content. The second problem was to intercept the user interaction with the collaboration-unaware 3D model. The solution was to automatically replace the sensors that are responsible for the user interaction with customized collaborative sensors.

We solved the generic problems of consistency, scalable support for latecomers, and session recording such that these solutions can be reused for other distributed interactive media. In order to make this possible we used the RTP/I application level protocol for distributed interactive media as a foundation of our application. All other applications that are based on RTP/I will be able to apply our solutions to the generic problems. There are currently three applications under development which will do so: a shared whiteboard (mlb) [22], and a toolkit for remotely controlled Java applets for teleteaching [8], both developed at the University of Mannheim, and the Audio Enabled Multicast VNet project [19] at the Communications Research Center in Ottawa/Canada.

The implementation of TeCo3D has been done completely in Java. The Java3D browser was extended by the functionality for state access and used as a 3D presentation engine. SIEMENS plans to use the TeCo3D application as a starting point for the development of call-center and help-desk solutions.

ACKNOWLEDGMENTS

This work was supported by a Ph. D. grant from SIEMENS. Several important requirements and areas of application have been identified in discussions with Dr. Jean Schweitzer (Transferzentrum Global Working TGW, DFKI, Saarbrücken) and with Dr. Ralf Buschmann (ZT IK 2, SIEMENS, Munich). Without their support the TeCo3D project would not have been possible. The comments of the anonymous reviewers have helped to improve the paper significantly.

REFERENCES

1. A. L. Ames, D. R. Nadeau and J. L. Moreland: *VRML 2.0 Sourcebook*, Second edition, John Wiley & Sons, New York, 1997.
2. Blaxxun. baxxun Contact. http://www.blaxxun.com/products/contact/index.html
3. D. P. Brutzman. The Virtual Reality Modeling Language and Java, *Communications of the ACM*, Vol. 41, No. 6, June 1998, pp. 57 - 64.
4. S. Floyd, V. Jacobson, C. Liu, S. McCanne and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. In: *IEEE/ACM Transactions on Networking*, Vol. 5, No. 6, 1997, pp. 784 - 803.
5. V. Hilt, M. Mauve, C. Kuhmünch and W. Effelsberg. A Generic Scheme for the Recording of Interactive Media Streams. In: *Proc. of the International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS) '99*, Toulouse, France, M. Diaz et. al. (Eds.), LNCS 1718, Springer Verlag, Berlin, Germany, 1999, pp. 291 - 304.
6. W. Holfelder. Interactive Remote Recording and Playback of Multicast Videoconferences. In: *Proc. of the International Workshop on Interactive Distributed Multimedia Systems and Telecommunications Services (IDMS) '97*, Darmstadt, Germany, R. Steinmetz and L. Wolf (Eds.), LNCS 1309, Springer Verlag, Berlin, Germany, 1997, pp. 450 - 463.
7. V. Jacobson, S. Casner, R. Frederick and H. Schulzrinne. *RTP: A Transport Protocol for Real-Time Applications*, Internet Draft, Audio/Video Transport Working Group, IETF, draft-ietf-avt-rtp-new-04.txt, 1999. Work in progress.
8. C. Kuhmünch, T. Fuhrmann and G. Schöppe. Java Teachware - The Java Remote Control Tool and its Applications. In: *Proc. of ED-MEDIA/ED-TELECOM'98*, Freiburg, Germany, 1998, available on CD-ROM.
9. L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, Volume 21, Number 7, 1978, pp. 558 - 565.
10. M. Mauve. TeCo3D: a 3D telecooperation application based on VRML and Java. In: *Proc. of SPIE Multimedia Computing and Networking (MMCN) '99*, San Jose, CA, USA, published by SPIE, Bellingham, Washington, USA, January 1999, pp. 240 - 251.
11. M. Mauve. *Consistency in Continuous Distributed Interactive Media*. In: *Proc. of ACM CSCW2000*, Philadelphia, USA, 2000, pp. 181 - 190.
12. M. Mauve, V. Hilt, C. Kuhmünch, J. Vogel, W. Geyer and W. Effelsberg. *RTP/I: An Application Level Real-Time Protocol for Distributed Interactive Media.* Internet Draft: draft-mauve-rtpi-00.txt, 2000. Work in progress.

13. M. Mauve. *Distributed Interactive Media.* Ph.D. Thesis, University of Mannheim, Lehrstuhl für Praktische Informatik IV, Mannheim, Germany, 2000.

14. Microsoft. Netmeeting Product Page, March 2000. http://www.microsoft.com/windows/netmeeting

15. D. L. Mills. *Network Time Protocol (Version 3) specification, implementation and analysis*. DARPA Network Working Group Report RFC-1305, University of Delaware, 1992.

16. W. Minenko. *Advanced Design of Efficient Application Sharing Systems under X Window.* Ph.D. Thesis, Department of Computer Science, University of Ulm, 1996.

17. SoftWired. *iBus web pages*. http://www.softwired.ch/ibus

18. J. Sonstein. *VNet Web Page*. 1999. http://ariadne.iz.net/~jeffs/vnet/

19. J. Robinson, J. Stewart, and I. Labbe. WVIP - Audio Enabled Multicast VNet. In: *Proc. of the VRML2000 Symposium on the Virtual Reality Modeling Language (VRML)*, Monterey, California, USA, published by ACM SIGGRAPH, New York, USA, February 2000.

20. C. Sun and C. Ellis. Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achievements. In: *Proceedings of the ACM 1998 conference on Computer Supported Cooperative Work (CSCW'98)*, Seattle, Washinton, USA, 1998, pp. 59 - 68.

21. J. Vogel, M. Mauve, W. Geyer, V. Hilt, and C. Kuhmünch. A Generic Late-Join Service for Distributed Interactive Media. To appear in: *Proc. of ACM Multimedia 2000*, Los Angeles, CA, USA, 2000.

22. J. Vogel. multimedia lecture board homepage, 2001. On-line: http://www.informatik.uni-mannheim.de/informatik/pi4/projects/ANETTE/anetteProject2.html

23. 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, 1997. http://www.vrml.org/Specifications/

24. VRML Consortium. *Living Worlds - Making VRML 2.0 Applications Interpersonal and Interoperable - Draft 2*, April 1997. On-line: http://www.vrml.org/WorkingGroups/living-worlds/draft_2/index.htm

25. VRML Consortium. *Information technology -- Computer graphics and image processing -- The Virtual Reality Modeling Language (VRML) -- Part 2: External authoring interface*. ISO/IEC 14772-2:1999 Committee Draft, 1999. http://www.vrml.org/WorkingGroups/vrml-eai/Specification/

26. VRML Consortium. The VRML - Java3D Working Group, 1999. On-line: http://www.vrml.org/WorkingGroups/vrml-java3d/

27. Web3D Consortium. X3D Web page. http://www.web3d.org/x3d.html