REIHE INFORMATIK
16/98

A General Framework and Communication Protocol for the Real-Time Transmission of Interactive Media

Martin Mauve, Volker Hilt, Christoph Kuhmünch, Wolfgang Effelsberg
Universität Mannheim
Praktische Informatik IV
L15, 16
D-68131 Mannheim

A General Framework and Communication Protocol for the Real-Time Transmission of Interactive Media[1]

Martin Mauve, Volker Hilt, Christoph Kuhmünch, Wolfgang Effelsberg
{mauve,hilt,cjk,effelsberg}@pi4.informatik.uni-mannheim.de
University of Mannheim, Germany

Abstract. In this paper we present a general framework for the real-time transmission of interactive media, i.e. media involving user interaction. Examples of interactive media are shared whiteboards, Java animations and VRML worlds. By identifying and supporting the common aspects of this media class the framework allows the development of generic services for network sessions involving the transmission of interactive media. Examples are mechanisms for late join and session recording. The proposed framework is based on the Real-Time Transport Protocol (RTP) which is widely used in the Internet for the real-time transmission of audio and video. Using the experience gained through the framework for audio and video, our work consists of three important parts: the definition of a protocol profile, the instantiation of this profile for specific media, and the development of generic services. The profile captures those aspects that are common to the class of interactive media. A single medium must instantiate this profile by providing media-specific information in the form of a payload type definition. Based on the profile, generic services can be developed for all interactive media. In this paper we focus on the description of the profile for the real-time transmission of interactive media. We then present the main ideas behind a generic recording service. Finally we show how multi-user VRML and distributed interactive Java animations can instantiate the profile.

Keywords. Interactive Media, Real-Time Transmission, Multicast, Recording, Java Animations, VRML

1 Introduction

The development of application-layer protocols for the real-time distribution of audio and video has been a focus of research for several years. Most notable is the success of the Real-Time Transport Protocol (RTP) [24]. Today, the results of this work are widely deployed throughout the internet, used for freely available tools [20] and commercial software [2] alike.

RTP is a protocol that must be tailored to the specific needs of different media and media classes. It is therefore accompanied by documents describing the specific encoding of different media types within the RTP framework. The current documents focus mainly on the encoding of various audio and video formats [23] [10]. While these two basic media classes can be considered the most common ones, there exist several others which are gaining importance rapidly.

This paper presents a general framework for the real-time distribution of interactive media, i.e. media involving user interaction. Examples of interactive media are: whiteboard applications [6], VRML models [28], Java animations [15] and SMIL presentations [30]. Currently most media of this kind are presented locally to a single user. Given the increasing importance of interactive media, however, it is very desirable to use them in a distributed fashion, possibly incorporating this media class into teleconferencing, telepresentation, and telecooperation applications. In addition, several interactive media are inherently distributed, like shared whiteboards or multi-user virtual reality. Suitable application layer protocols are thus needed, which support distributed interactive media. Existing approaches to define application-layer protocols for the distribution of some interactive media are mostly proprietary [6]. This prevents interoperability as well as sharing of common tools while requiring re-implementation of similar functionality for each protocol.

[1] This work is sponsored by the Siemens Telecollaboration Center, Saarbrücken, by the DFN (Deutsches Forschungsnetz), and by the BMBF (Bundesministerium für Forschung und Technologie).

In order to establish a common foundation, we propose a general, RTP-based, application-layer protocol profile for the real-time distribution of interactive media. For a certain medium the profile can be instantiated by providing medium-specific information, reusing the infrastructure set up by RTP and the profile. The profile itself captures the common aspects of the interactive media class, enabling the reuse of existing code and tools to a much higher degree than would be achievable by using raw RTP.

A classification of different media types and important properties of the interactive media class are introduced in Section Two. Section Three contains the common requirements for application-level protocols supporting this media class. The RTP profile is presented in Section Four. Section Five proposes a generic mechanism for recording and playback of interactive media. The subsequent two sections demonstrate how the profile can be instantiated for different media, namely distributed Java animations and multi-user VRML. Section Eight concludes this paper with a summary and an outlook.

2 Interactive Media

In order to define the scope of our work, we separate media types by means of two criteria. The first criterion distinguishes whether the medium is discrete or continuous. The characteristic of a *discrete medium* is that its state is independent of the passage of time. Examples of discrete media are still images or digital whiteboard presentations. While discrete media may change their state, they do so only in response to external events, such as a user drawing on a digital whiteboard. The state of a *continuous medium*, however, depends on the passage of time and can change without the occurrence of external events. Video and animations belong to the class of continuous media. The second criterion distinguishes between interactive and non-interactive media. *Non-interactive media* change their state only in response to the passage of time and do not accept external events. Typical representations of non-interactive media are video, audio and images. *Interactive media* are characterized by the fact that their state can be changed by external events such as user interactions. Whiteboard presentations and interactive animations represent interactive media. Figure 1 depicts how the criteria characterize different media types.

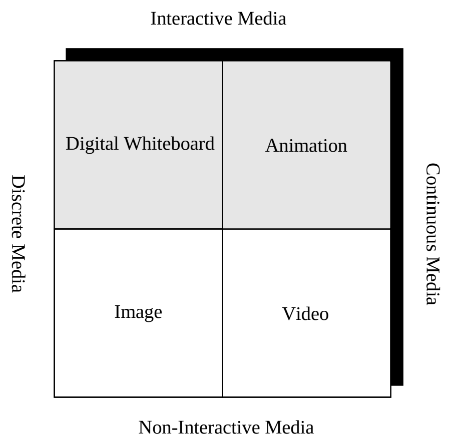


Figure 1: Examples of Media Types

A medium which is neither interactive nor continuous does not require support of a real-time transport protocol and is therefore not further discussed here. Media types that are non-interactive and continuous have already been investigated to a large extent. Several RTP payload types have been defined for different encodings of video and audio. Since these payload types are sufficient for many applications we do not address continuous, non-interactive media.

In contrast, proprietary protocols are used by almost all applications for the distribution of, and cooperation with, interactive (continuous, as well as discrete) media. Examples of those applications are shared whiteboards, remote-controlled Java animations or multi-user VRML worlds. The usage of proprietary protocols prohibits the development of generalized services for important aspects such as recording or late-join. In our research group this problem has become most apparent with the recording of MBone sessions which include a shared whiteboard. For a long time it was impossible to record the whiteboard part of a session. Lately there have been approaches to solving this problem by extending the functionality of the recording tools to understand the semantics of a shared whiteboard transmission [7]. While this approach enables the recording of one specific shared whiteboard — the digital lecture board [6], it does not provide a generalized solution for all interactive media. However, such a solution would be within reach if all interactive media where required to make certain assertions about the protocol they use. It is the aim of this work to define these required assertions, in order to develop a general framework for interactive media.

2.1 Model for Interactive Media

Before discussing protocols for interactive media it is important to establish a common view on this media class. We will therefore describe our model for interactive media and show the difference between displaying traditional and interactive media streams.

An *interactive medium* is a medium that is well defined by its current state at any point in time. For example at a given point in time the medium Java animation is defined by the internal state of the Java program that is implementing the animation. The *state* of an interactive medium can change for two reasons, either by passage of time or by *events*. The state of an interactive medium between two successive events is fully deterministic and depends only on the passage of time. Any state change which is not a fully deterministic function of time is caused by an event. A typical example of an event is the interaction of a user with the medium. An example of a state change caused by the passage of time might be the animation of an object moving across the screen. As will be shown later, states and events play an important role in the transmission of interactive media and constitute the main part of an interactive media stream.

To display a non-interactive media stream like video or audio, a receiver needs to have an adequate *player* for a specific encoding of the medium. If such a player is present in a system, every media stream that employs this encoding can be processed. This is not true of interactive media streams. For example, to process the media stream that is produced by a shared VRML browser, it is not sufficient for a receiver to have an VRML browser. The receiver will also need the VRML world on which the sender acts; otherwise the media stream will not be able to be interpreted by the receiver. But even if the receiver has loaded the correct world into its browser, the VRML world may be in a state completely different from that of the sender. Therefore, the receiver must synchronize the state of the local representation of the interactive medium with the state of the sender before it will be able to interpret the VRML media stream correctly.

Generally speaking, it is not sufficient to have a player for an interactive media type. Additionally the player must be initialized with the *context* of a media stream before the stream actually can be played. The context is comprised of two components: (1) the environment of a medium and (2) the current state of the medium. The *environment* represents the static description of an interactive medium that must initially be loaded into the media player. Examples of environments are VRML worlds or the code of Java animations. The *state* is the dynamic part of the context. The environment within a player must be initialized with the current state of the interactive medium before the stream can be played. During transmission of the stream, both sender and receiver must stay synchronized since each event refers to a well-defined state of the medium and cannot be processed if the medium is in a different state. Synchronization can be realized by various means ranging from reliable transmission with a large reorder buffer to unreliable transmission combined with frequent insertions of state information into the media stream.

It can be derived from our explanation that there are four elements involved in transmitting and playing interactive media streams: the player, the environment, states and events. The first two elements do not need real-time transmission and therefore do not fall within the scope of this work. We merely assume that they are present for all senders and receivers of interactive media streams. Collectively we call these two elements the *application*. Events and states, however, are the basis of the actual interactive media stream. The following section will discuss the requirements for a protocol for transporting this kind of data.

3 Protocol Requirements

A general protocol for interactive media must meet several requirements derived from the nature of the medium, the applications presenting it, and the general services interacting with the media stream. The first and most important requirement is the real-time capability of the protocol. As explained in Section Two, states and events of an interactive medium are generally only valid at a specific point in time. The protocol must therefore be able to convey timing information. Furthermore, receivers of an interactive media stream should be able to inform the sender about the transmission quality of the stream received. This includes loss rates, latency and jitter. The information allows the sender to react to quality changes in the underlying network, e.g. use a varying amount of forward error correction redundancy or change the data rate. Both aspects, real-time capability and quality feedback, are supplied by the Real-Time Transport Protocol (RTP) in combination with the Real-Time Control Protocol (RTCP). RTP is therefore used as the basis for our general interactive media framework.

RTP is a protocol which was left incomplete on purpose so that it can be tailored to the specific needs of different media. This process of adaptation can be done in two steps. The first step is to define an RTP profile. A profile defines the common aspects of several related media. An example of an existing profile is the “RTP Profile for Audio and Video Conferences with Minimal Control” [23]. The second step is to specify in detail how a single medium (e.g.

MPEG1/MPEG2 encoded video [10]) is transported using the packet types defined by RTP. This specification is called a payload type definition. Following this approach, we define a general protocol for interactive media as an RTP profile, while each individual interactive medium is specified as a payload type within the profile.

The proposed RTP profile should identify and support the common aspects of interactive media streams which are not already handled by RTP. In detail these aspects are:

- **Reliable and unreliable communication**. The profile should not limit the choice of whether a medium within the profile needs to be transmitted reliably or unreliably. If reliable communication is required, it should be realized either within RTP or by the transport protocol beneath RTP (e.g. reliable multicast or TCP). An application using reliable transmission must be prepared to take appropriate actions if RTP packets arrive late because of retransmissions. For example the application might maintain a play-out buffer of sufficient size for a small number of retransmissions. If a packet arrives later than the buffer can compensate for, some kind of application-level recovery will have to be done.
- **Identification of RTP packet content**. For general services it is important to be able to identify the type of content transported in an RTP packet. The profile should specify an identification mechanism for the different packet types common to all interactive media (e.g. event and state packets).
- **States and Events.** The profile should specify how the basic elements - states and events - of an interactive media stream are to be transmitted.
- **Sub-components**. To ensure flexible handling of state information it is desirable to partition an interactive medium into several *sub-components*. Such partitioning allows participants of a session to track only the states of those sub-components they are actually interested in. Note that sub-components also allow multiple simultaneous senders for one interactive medium, e.g. each sender could be responsible for the transmission of one specific sub-component. Examples of sub-components are VRML objects (a house, a car, a room), or the pages of a whiteboard presentation. The profile uses sub-components as the level of granularity for state transmissions. Events have to identify the sub-component in which the “target” of the event is located and must include this information in the event packet. This allows applications to discard events for sub-components they are not interested in. Payload types that do not partition the state of the medium are regarded as having only a single sub-component.
- **Delta States**. In cases where a complex state is inserted frequently into the media stream, it is necessary to be able to send only those parts that have changed since the last state transmission. We call a state which contains only the state changes that have occurred since the last transmitted state a *delta* state. A delta state can only be interpreted if the preceding full state and interim delta states are also available (see Figure 2). The main advantages of delta states are their smaller size and that they can be calculated faster than full states. It is important to note that the use of delta states is only reasonable in an environment with low packet loss rates.

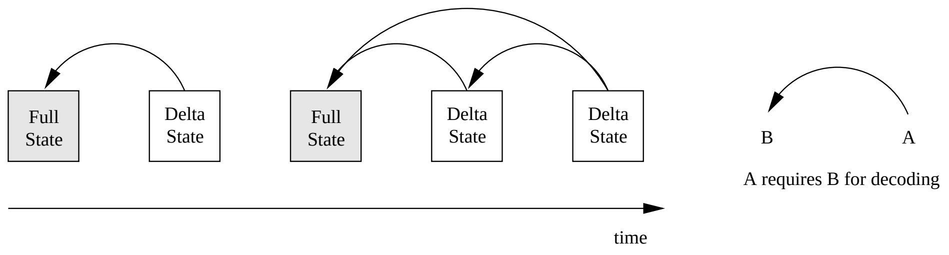


Figure 2: Decoding of Delta States

- **Active sub-components**. For many generic services it is important that the sub-components necessary to display the interactive medium are known. The profile should provide a standardized way for session participants to announce the sub-components they are currently interested in.
- **Requesting information**. On several occasions the receivers of an interactive media stream need to get certain information from the sender. This information might be the state of a specific sub-component or a general description of all sub-components. The mechanisms for requesting this information should be defined within the profile.

4 RTP Profile for Interactive Media

It is the main aim of the RTP profile for interactive media to capture the common aspects of this media class. Therefore all payloads within the profile have to structure RTP and RTCP packets in a similar manner. In addition a number of rules (e.g. when to send what information) for interactive media payloads are specified by the profile. This section gives an overview of the main concepts for the proposed RTP profile.

4.1 State, Delta-State, Event

In order to comply with the requirements in Section 3 most of the data for interactive media is carried in three packet types: state, delta-state and event. These three packet types are structured as depicted in Figure 3. The most important fields in these packets are type, sub-component ID, sub-component sequence number, and data. The type field is needed to distinguish the different packet types defined in the profile. Possible choices for the type field are state, delta state, event, active sub-components list and state description list. This information can not be included in the RTP payload type identifier (PT) since the different packet types exist for each individual medium.

In state and delta state packets the sub-component ID field is used to hold the sub-component ID of the state included in the data part of the packet. In event packets this field identifies the sub-component in which the “target” of an event is located. The RTP SSRC identifier is not able to identify the sub-component, since a single sub-component might change the sender which is not possible for an SSRC. As an example for the usage of sub-component IDs, consider a virtual person (sub-component 1) opening the door of a virtual car (sub-component 2). State transmissions of the virtual person would put a 1 in the sub-component ID field. The open event would have a 2 in this field, since the door (target) is part of the car (sub-component 2).

As the RTP sequence number allows receivers only to determine if they missed packets of a certain sender, there needs to be a way to identify which sub-components are affected by the problem. A separate sub-component sequence number is therefore present in state, delta state and event packets. Based on this information the receivers can ignore packet loss or request the repair of appropriate sub-components. The data field of the packet contains the definition of states, delta-states or events specific to the payload type.

In addition to the four basic fields, the profile also defines the meaning of two other fields. The marker bit (M) is set to one for the last packet describing a single state, delta-state or event, e.g. if a state description needs three packets, the marker bit is set to zero for the first two packets and to one for the last packet. This helps to ascertain whether or not a logical unit has been completely received, and supports recovery after packet loss.

Since setting the state of a sub-component can be costly and might not always be reasonable, a priority (PRI) field is present in state and delta-state packets. This priority can be used by the sender of the state to signal its importance. A packet with high priority should be examined and applied by all communication peers which are interested in the specific sub-component. Situations where high priority is recommended are resynchronization after errors or packet loss. Basically a state transmission with high priority forces every participant to discard its information about the sub-component and requires the adoption of the new state. A state transmitted with low priority can be ignored at will by any participant. This is useful if only a subset of communication partners is interested in the state. Examples of this case are late joins or recording, where only applications joining the session or tools recording it might be interested in certain state transmissions.

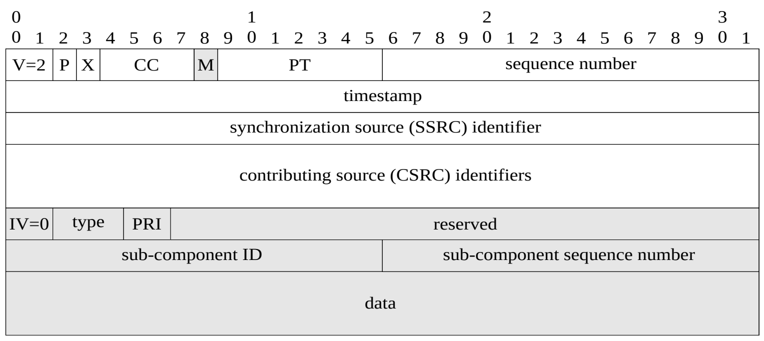


Figure 3: RTP Packet Structure for States, Delta States and Events

4.2 Active Sub-Components

As requested in Section 3 the profile provides a standardized way to announce the active sub-components of any application participating in an interactive media session. The active sub-components of a single application at any point in time are those sub-components which are required by the application to present the interactive medium at that specific time. The active sub-components of a session comprising several participants are the active sub-components of all participants. It is important to note that declaring a sub-component active does not grant permission to modify anything within that sub-component. It is perfectly reasonable for a single application to activate several sub-components just to indicate that these are needed for the local presentation of the medium. However, a sub-component must be activated by a session participant before that participant is allowed to modify (send events into) the sub-component. It is the responsibility of the application to ensure this behavior. In general, interactivity is handled by the application and is not part of the profile. In a cooperative scenario, for example, several participants might want to interact with the same object. It is the responsibility of the application to manage the access to the object (e.g. using floor control). The profile merely provides the transport capability for the interaction. This assumes that for many applications some kind of control packets are defined in addition to those presented here.

Announcement of active sub-components is made by means of the active sub-components list packet type. This packet type is shown in Figure 4 and lists the IDs of the active subcomponents within the participant. The specification of when these packets need to be sent is the responsibility of the application. It is required that this specification ensure that every participant of a session have sufficient information to derive an acceptable approximation of which sub-components are currently active in the session.

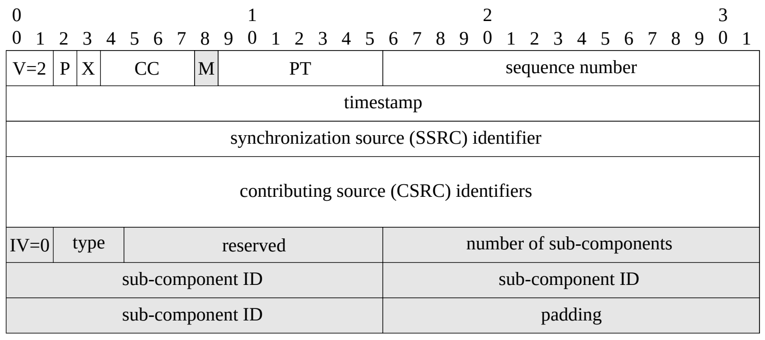


Figure 4: Active Sub-Components List RTP Packet

In order to simplify the handling of sub-component states in the application, the following rule applies: Whenever a sub-component becomes active in a session, the full state of that sub-component must be transmitted. This rule allows local applications to discard any state information that becomes inactive in a session. Only those applications interested in reactivating the sub-component at a later point in time need to remember its state. All other applications can rely on this rule to receive the state should someone else reactivate the sub-component.

4.3 State Query, State List Query, State List

In many cases it is reasonable to let the receivers decide when the state of sub-components should be transmitted. For this reason a receiver must be able to request the state from other participants in the session. Within the profile this mechanism is realized by means of a special RTCP packet called *state query*. This packet is an application-defined RTCP packet; its structure is depicted in Figure 5. The state query packet is identified by the string “IAMP” (InterActive Media Profile) and the value of the subtype field. In order to address the requested state, a sub-component ID field is present.

As the computation of state information may be costly, the sender must be able to distinguish between different types of requests. Recovery after an error urgently requires information on the sub-component state since the requesting party cannot proceed without it. These requests will be relatively rare. In contrast to this, a recorder does need the media states to enable random access for the recorded media. It does not urgently need the state but will issue requests frequently. For this reason, the state request mechanism supports different priorities through the priority (PRI) field in the state query packet. Senders should satisfy requests with high priority (e.g. for late joiners) very quickly, even if this has a negative impact on the presentation quality for the local user. Requests with low priority can be delayed or even ignored, e.g. if the sender currently has no resources to satisfy them. The sender must be aware that the quality of the service offered by the requesting application will decrease if requests are ignored.

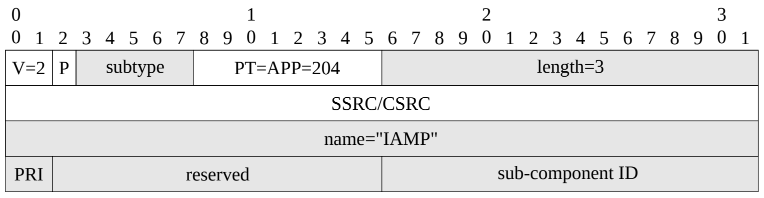


Figure 5: RTCP State and State List Packet

In many situations (e.g. late join) it is desirable for a receiver to have an overview of all sub-components present in a session. The overview should include the IDs of all sub-components (active or inactive) kept by all session participants. Besides the raw IDs, it should also be possible to get application-specific information for each sub-component. Examples of this kind of information are the spatial position of VRML objects or the titles of slides presented on a shared whiteboard.

A session participant can request the overview by sending a *state list query* RTCP packet. This packet is structured like the state query packet, but the sub-component ID field is missing. If the priority of the state list query packet warrants a reply, a selected group of session participants transmits state-list RTP packets containing the information requested. The state list resembles the active sub-components list packet (Figure 4), except that an application-defined description for each sub-component is appended to the packet. It is the responsibility of the application to choose the session participants that will send the state lists.

5 The Recording Service

5.1 Recording RTP Packets

With the growing number of real-time transmissions on the Internet, the need arose to record some of the transmissions. Meanwhile a number of RTP recorders exist (e.g. the MBone VCR [11]) accomplishing this task at least for video and audio streams. These recorders store media streams packetized in RTP. The major advantage is that the mechanisms implemented in a recorder can build upon RTP and do not depend a specific media encoding. For example, a recorder may synchronize audio and video streams independent of the actual media encoding. Interactive media streams have special characteristics (see Section 5.2) which require additional mechanisms in a recorder. Due to the lack of a common framework for interactive media, recorders for that category exist only for specific applications. Examples of recorders for shared whiteboards are the MASH media board recorder [26], the mMOD [21], the AOFwb [17], and the recorder for the digital lecture board [7] [6] (see Section 5.5). But since there are many types of interactive media, it is advantageous to implement the mechanisms needed for the recording of interactive media streams in a more general fashion. With a general framework for interactive media streams, all transmissions that implement the framework can be recorded with tools that combine the well-understood mechanisms for RTP recording with the mechanisms for interactive media recording.

An RTP recorder usually handles two network sessions (see Figure 6). In the first, the recorder participates in the multicast transmission of the RTP media data. Depending on its operation mode (recording or playback), it acts as a receiver or sender towards the other participants of the session. A second network session can be used to control the recorder from a remote client, e.g. using the RTSP [25] protocol. During the recording of an RTP session, the recorder receives RTP data packets and writes them to a storage device. Packets from different media streams are stored separately. When playing back, the recorder successively reads the RTP packets of each media stream from the storage device. For each packet the time when it must be sent is computed using the time stamps contained in the RTP packet. The recorder sends the packets according to the computed schedule. A detailed description of the synchronization mechanism implemented in the MBone VCR can be found in [12].

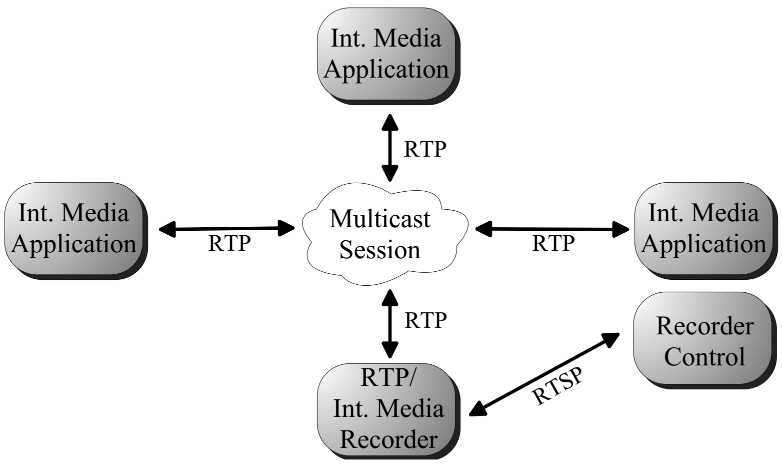


Figure 6: Scenario for the Recording of an RTP Session

5.2 Random Access to Interactive Media Streams

In contrast to the traditional media types where random access to any position within a stream is possible interactive media streams do not allow easy random access without restoring the context of the stream at the desired access position. To restore the context of a recorded stream in a receiver, two operations have to be performed: First, the environment has to be loaded into the receiver. The environment can be provided by the recorder or by a third party, e.g. an HTTP server. Then the receiver needs the state of the interactive medium at the access position within the stream. The state can be recovered from the recorded media stream and must be provided by the recorder. Notice that the recorder is not able to interpret the media-specific part of the RTP packets and therefore cannot directly compute the state and send it to the receivers. But the recorder may send RTP packets which are stored within the recorded media stream. Thus, if the recorder can compose a sequence of recorded RTP packets containing states and events that put a receiver into the desired state, the state of an interactive media stream at an access position can be reconstructed. The task a recorder has to accomplish before starting a playback is to determine the sequence of recorded packets that will put the receivers into the state that is required for the access at the desired position.

In the case of a discrete interactive medium, the sequence of states and events can be computed to recover the media state at any point in a recorded stream. In contrast the media state of a continuous interactive medium can only be recovered at points within a recording where a state is available, events cannot be used for state recovery. Therefore, random access to this type of media stream will usually result in playback near the desired access position. The more often the state is stored within a stream, the finer is the granularity with which the stream can be accessed. Interactive media applications usually send the media state upon request by another application. Thus, the recorder must request the state at periodic intervals. The requests use a low priority because a delayed or missing response reduces only the access granularity of the stream, which can be tolerated to some degree.

5.3 Functionality for an Interactive Media Stream Recorder

A VCR is a well-known device for dealing with recording and playback of continuous media and therefore it is used as a design metaphor for the functionality an interactive media recorder may have. Six basic operations for the recording and playback of interactive media streams can be derived:

- *Record*. The recorder must be able to record interactive media streams in synchronization with other media streams.
- *Play*. The synchronized playback of interactive media streams must start at (or near) the current position of the recorder within the stream (see Section 5.4). A user may have altered this position through the operations random positioning, fast forward, fast rewind or a previous play.
- *Fast Forward*. The fast forward operation allows a user to scan through a recording. Especially in the case of continuous interactive media, fast forward is hard to implement because the internal clock of a medium runs at normal speed whereas the recorder sends recorded packets at high speed. Another problem with fast forward is that it consumes much more bandwidth than does regular playback. Finally, for all media types (like audio and video) to which the interactive media stream is synchronized, a fast forward operation must also be implemented. For these reasons, the fast forward operation is not further discussed here.
- *Fast Rewind*. This is analogous to the fast forward operation, implying the same problems.
- *Stop*. The stop operation will bring the recorder into an idle state. Note that the internal clock of a receiver does not stop. The interactive medium may therefore perform further operations without receiving packets from the recorder. Before an interrupted playback can be resumed, the mechanism for random positioning must be executed.
  When stopping playback, the user regains control over the medium and can freely interact with it. For example, this can be used to guide a user through an animation by playing back a recording while retaining the ability to let the user take over and explore the animation in its current state.
- *Random Positioning*. Random access should be possible within a recording by directly jumping to a desired position.

5.4 The Playback Mechanism

The mechanism for playback at a random position is implemented completely in the recorder. It is not required that the receiving applications recognize the recorder as a specific sender nor does the recorder need to interpret media-specific data. All applications that use a payload based on our RTP profile for the transmission of interactive media can be recorded and will be able to receive data from the recorder.

When a recorded stream is accessed, the playback mechanism must reconstruct the media state before the actual playback can be started. In the best case, the media state will be contained in the recorded stream at exactly the position at which the playback should start. Then playback can begin immediately. But in general the playback will be requested at a position where no state is directly available in the stream.

For example consider a recorded media stream that consists of the sequence S_0, which contains a state, three successive delta (Δ) states and several events (see Figure 7). If this sequence is accessed at some later time at position t_p, the state at t_p must be reconstructed by the recorder. In a continuous interactive medium access to t_p is not possible because the recorder cannot determine the state at t_p. At t_p there is no state available in the recorded stream. However, access to position $t_{\Delta 3}$ within the stream is feasible, because $t_{\Delta 3}$ is the location of a delta state. The complete media state at $t_{\Delta 3}$ can be reconstructed from the state located at position t_s and the subsequent delta states until position $t_{\Delta 3}$, which is the position of the last delta state before t_p. The events between t_s and $t_{\Delta 3}$ can be ignored, because all modifications to the state at t_s are reflected in the delta states. The packets that contain states can be sent at the maximum speed at which the recorder is able to send packets. If required by the medium, the internal media clock is part of the media state. Thus, after applying a state, the media clock of a receiver will reflect the time contained in the state. When the recorder finally reaches $t_{\Delta 3}$ (and has sent Δ3), fast playback must be stopped and playback at regular speed must be started. The start of the regular playback may not be delayed because events must be sent in real-time relative to the last state. This is important since for continuous interactive media the events are only valid for a specific state which may change with the passage of time. Altogether, the recorder will play back sequence S_1 shown in Figure 7.

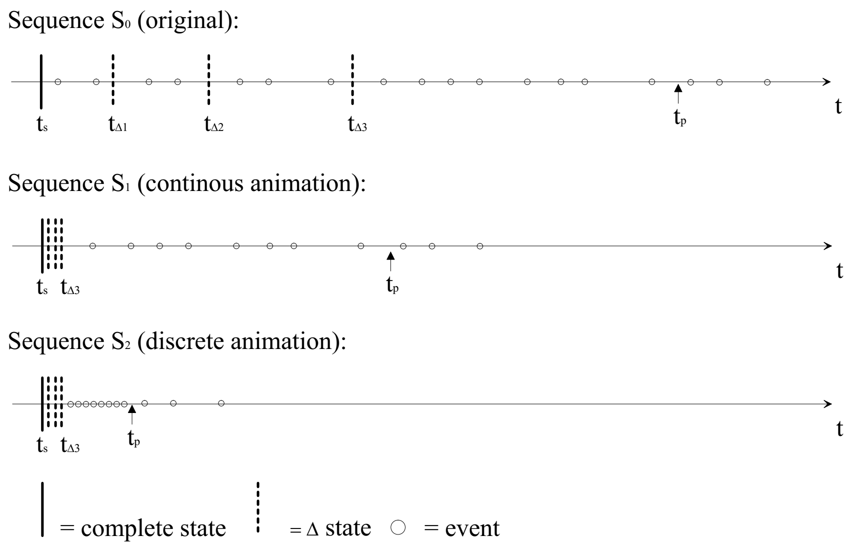


Figure 7: Playback of a Recorded Sequence of States, Delta-States and Events

For discrete interactive media, the fast playback of events is possible. Therefore true random access to position t_p can be achieved by also sending the events between $t_{\Delta 3}$ and t_p at full speed. The resulting sequence S_2 is also shown in Figure 7.

The above mechanism is applicable to interactive media types that do not utilize sub-components. In a more sophisticated mechanism the existence of sub-components can be exploited to reduce the effort for the recovery of the media state. When a recorded stream is accessed, the set of active sub-components at the access position can be computed. The states of these active sub-components must be recovered before starting playback. If the medium is partitioned into reasonable sub-components, this is much cheaper than recovering the state of the complete medium. A detailed description of this mechanism can be found in [9].

5.5 The Implementation of a Shared Whiteboard Recorder

At the University of Mannheim we have implemented a recorder for a shared whiteboard [7], the digital lecture board (dlb) [6]. The digital lecture board uses a specific RTP payload format for the transmission of data on top of the reliable multicast protocol SMP [6]. The recorder for the dlb is implemented based on the MBone VCR [11], which is used to store the plain RTP packets in synchronization with other media streams.

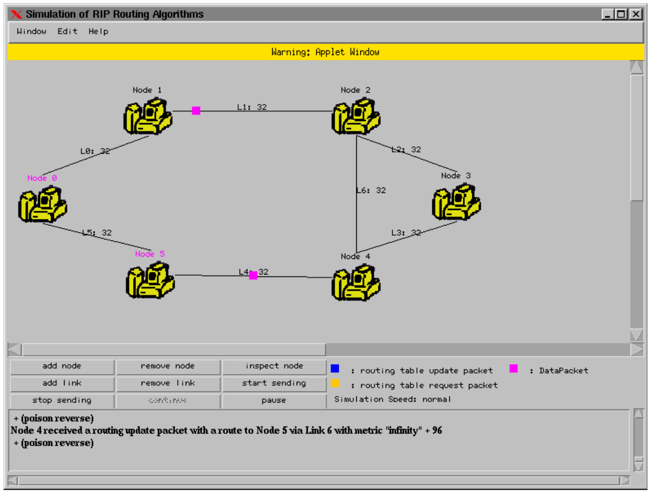
The dlb recorder extends the MBone VCR in two respects: First, it handles reliable multicast towards the shared whiteboard by adding SMP headers during playback and stripping them during recording. The SMP headers are not recorded because the reliability of transport may change between recording and playback and therefore SMP must dynamically adapt itself to the current situation. Second, the dlb recorder adds the functionality for playback at random positions. The dlb recorder implements an playout mechanism which is specific to discrete interactive media streams. The mechanism described in Section 5.4 is a generalized version of this mechanism.

The implementation of the dlb recorder showed that a more general recording facility of interactive media streams would be desirable and feasible. In our further work we will implement such an interactive media recorder based on the experience gained from the dlb-recorder.

6 Applying the Framework to Distributed Interactive Java Animations

The use of Java for animations and simulations has been a topic of research in many projects; examples include [1], [4], [15]. Its platform independence and integration into Web browsers recommend it as programming language for the development of distributed interactive animations. New Application Programming Interfaces (APIs) such as Java-3D and new concepts like Java Beans provide additional support for the implementation of such animations and simulations. While Java-3D [14] provides methods for 3D visualization, the concept of Java Beans [27] can be used to build re-usable components which can be combined to build new applications through a graphical interface. Thus the rapid development of applications is facilitated.

Java animations can be used to visualize a broad variety of scenarios, e.g. physical phenomena, statistical models or computer algorithms. An example is an animation that simulates and visualizes routing algorithms within computer networks in order to allow a better understanding of the underlying algorithms [15]. The animation allows the interactive construction of a network of nodes and links. Senders and receivers can be chosen, and packets are animated along the network links. If the simulation model is changed - e.g. new links/nodes are added or existing links/nodes are removed - the changes become visible simultaneously for all participants in the session. Figure 8 shows a screen shot of such an educational applet that visualizes the Routing Information Protocol[1] described in [8].



```
 + (poison reverse)
Node 4 received a routing update packet with a route to Node 5 via Link 6 with metric "infinity" + 96
 + (poison reverse)
```

Figure 8: A Java Simulation of the Routing Information Protocol

Distributed interactive Java animations can be defined as a subtype of continuous interactive media. Section Two describes interactive media as well-defined by its current state, which is influenced by the passage of time or by events. The state of a distributed interactive Java animation is defined by a set of objects that represent the animation model. In the routing animation mentioned above the objects might contain the current state of the packets, the network nodes and links. Similar to the definition in Section Two, the state of the animation model is changed by the passage of time, e.g. packets are moved from node to node, or by events, e.g. a node or a link is removed by user interaction. Typical user interactions are:

- Placing objects on the stage by clicking at the desired position
- Movement of the object by drag & drop operations
- Grouping of objects
- Opening of objects with the right mouse button
- Activating objects with a double-click.

In the following the requirements for a payload for distributed interactive Java animations are defined, a concept for distributed animations and simulations that is based on the Model-View-Controller concept is outlined, and the basic ideas behind an RTP Payload for distributed interactive Java animations are discussed.

[1] Url: http://www.informatik.uni-mannheim.de/~cjk/java/animation/routing/routing2/rip/

6.1 Requirements of Distributed Interactive Java Animations

At the beginning of a simulation session each participant needs to adopt the current context of the session. In the case of distributed Java animations, the context consists of the Java Virtual Machine (JVM), the program code of the animation, and the current state of the animation. This model assumes a replicated architecture in which each participant holds the complete animation context. Every participant's JVM maintains its own animation model which is synchronized by means of the transmitted states and events.

Two essential functions are needed to accomplish this synchronization: First, it is necessary to be able to extract a copy of the *current state* of the animation in order to update other participants. Essentially the objects concerned must be serialized [27] and broadcast to other participants. The concept of serialization was introduced into Java 1.1 in order to accomplish object persistence. It allows the current binary state of an object defined by its attributes to be read or written to a stream. During a session, the current state can be transmitted over the network. Generally, if a state is received from the network, the current state will be replaced by the received one.

Due to performance requirements it may be necessary to extract only a subset of these objects, thus deploying sub-components as defined in Section Three. Additionally an animation's performance can be improved by sending only those objects which were changed since the last full state transmission, using the concept of the delta states as proposed in Section 3. In any case a container must maintain references to all objects belonging to the animation model, and all objects of the animation model have to be serializable. The second essential function concerns the *transmission of events* (i.e. the control of remote animations). Typically an event is triggered by user interaction but it is also possible that the event is produced by an incident within the animation program. An example of such an event is the production of a random number that must be shared with the other participants in order to maintain synchronization. The prior condition for the remote control of animations is the ability to intercept local events and transmit them to remote participants. When an event is received through the network it must be triggered within the receiving animation.

The described essential functionalities provide only the basic means for the distribution of Java animations. Many scenarios require additional functionality:

Late Join and Recording: An important feature of distributed animations is the ability to allow participants to join and leave a collaborative session (e.g. a teleteaching lecture) at any time. In particular, participants may join a session after the animation has already been started. In that case the animation at the site of the late comer must be synchronized to the current state of the animation by other participants. In order to allow a member of a collaborative session to join the animation at any time during the session, it is necessary to extract a copy of the current state of the simulation and send it through the network. The receiver must be able to replace his current state with the received one. Similar to the demands of the late join functionality, it is necessary to extract and transmit a copy of the current state regularly in order to support a general recording service as described in Section Five.

Real-time animations: Unfortunately Java applets will run differently on different machines. For example the drawing of a line takes much longer on a PC than on a workstation. Therefore it is not sufficient to share events; rather time synchronization mechanisms are required. Usually animations are implemented via a sequence of “paint” & “sleep” operations. This approach results in run-time differences among the animations depending on the performance of the machine. A better result can be achieved with real-time oriented approaches, e.g.: ”Animate a packet from router A to router B in 10 seconds starting at a certain point t in time”. Now the speed of the packet is the same at all sites whereas the refresh rate of the animation depends on the power of the machines. The smooth and platform-independent animation of objects has to be accomplished for many different types of animation. Thus generic real-time and synchronization mechanisms are necessary.

We conclude that two basic functions form the core of a distributed interactive animation: (1) the ability to extract the objects belonging to the simulation model in order to implement state synchronization and (2) the ability to intercept events in order to control other animations remotely. An approach to realize these two functions is the Model-View-Controller Concept that is briefly outlined in the following section. A more detailed discussion of the requirements of distributed interactive Java animations can be found in [16], [13] and [15].

6.2 Model - View - Controller Concept

The Model-View-Controller concept distinguishes between three separate components within an application. The *model* component contains only objects belonging to the animated scenario, e.g. if the Java application simulates routing algorithms, the model component contains classes which represent routers and network links. A typical attribute of a router would be the list of connected links, and a typical attribute of a network link would be its bandwidth. The *view* component, on the other hand, contains only objects concerned with the graphical representation of the model. In a routing simulation the view component might hold classes concerned with the graphical representation of routers and links (typical attributes are the position, color and image of a router, or the line width of a link). The *controller* component controls the interaction between model and view. In distributed interactive animations the Controller is the central component: Its task is to control external influences, e.g. the system clock and user interaction. Additionally the controller must handle the communication with remote animations. Thus, the remote control functionality (event sharing) and the session control have to be implemented within this component.

Figure 9 depicts the asynchronous interaction between the three components within the Model-View-Controller concept. The communication is based on Java's event handling model. Each component produces its own class of events. The model component triggers *state change events* if the state of the model has been changed significantly and an update of the view is necessary. These events can be caught by the view (via C2) as well as by the controller (via C1). The events generated by the view should only be caught by the controller (via C3) for several reasons: First of all, a view event is generally triggered by user interaction (e.g. when a button is pressed) and therefore its handling is clearly a task of the controller. Furthermore, the controller carries out the event sharing: Events triggered by user input have to be transmitted over the network to remote applications, and events received from a remote application have to be processed and forwarded to the model (via C1) and the view components (via C3).

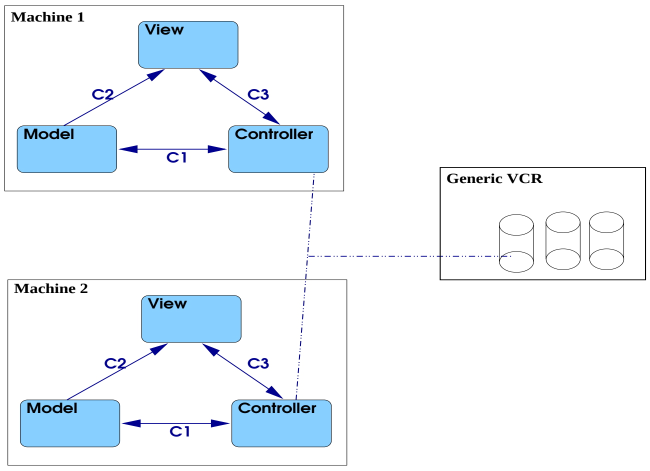


Figure 9: The Model-View-Controller Concept

A more detailed discussion of the outlined concept for distributed interactive Java animations can be found in [16].

6.3 Basic Ideas of an RTP Payload for Interactive Distributed Java Animations

The RTP profile discussed in Section 4 defines three different packet formats which are used for the payload of Java animations. In a payload document it must be specified how to packetize state information and events. The prior condition for the packetization of state information is the ability to extract a binary copy of the state, which can be solved in the case of Java by serializing the objects that hold state information. The Model-View-Controller concept allows a rigid separation of objects belonging to the animation model and thus enables us to extract an animation's state. Additionally a payload document must specify how to produce IDs for sub-components. In particular IDs for the whole state and for sub-components must be regulated in some way. Since the state of Java animations is determined by a set of objects, such a set can be used to form a sub-component. Thus the ID of a sub-component can be defined by the hash-code of a distinguished object within the set. The encoding of events can be done similarly to the encoding of state information by serializing the event object. Details about the Java-specific payload for interactive media will be described in more detail in a separate payload type definition.

7 Applying the Framework to VRML

VRML is the "file format for describing interactive 3D multimedia on the Internet" [28]. A VRML file contains the platform-independent description of 3D objects and their behavior; this description is called a 'world'. In order to interact with the content of a VRML file, the user needs a VRML browser - just as a web browser is required to view HTML files. In fact, VRML browsers are usually plug-ins into the most common HTML browsers. VRML does provide powerful functionality by allowing static descriptions of 3D objects, user-interactions with the objects, animations, and the usage of programming languages (e.g. Java) to define object behavior.

Clearly, VRML is an interactive medium as specified in Section 2. The state of the medium VRML is well defined by the state of the geometry and the state of the VRML browser at any point in time. State changes are either time-dependent and deterministic (e.g. animation) or are caused by an event (e.g. user interaction). It is therefore possible to specify a VRML payload within the proposed RTP profile. The VRML payload can then be used for applications which employ VRML in a distributed way, like multi-user worlds [29] or 3D telecollaboration tools [18].

Before presenting the main ideas of the VRML payload, we shall prove that the proposed RTP profile indeed captures the most important aspects of a protocol for distributed VRML. Such a protocol should satisfy certain requirements. It should:

- allow the communication of VRML states and events,
- provide real-time information,
- allow the partitioning of the medium into different sub-components (e.g. VRML objects),
- enable multiple senders within one session (e.g. for multi-user worlds),
- support late join / provide a repair facility if problems occur,
- allow the recording of a session.

It can be seen that all but the last two requirements mentioned here are a subset of the requirements for the general protocol (Section 3). The support of late join and recording are generalized services made possible by the proposed framework. It is therefore valid to define a VRML RTP payload for the interactive media profile.

The VRML payload uses the packets defined in the profile for interactive media. The main challenge for the payload type definition is the specification of a format for VRML states and events. This format is not part of the current VRML specification. To solve this problem we have developed a binary encoding which is able to capture the state of arbitrary VRML content. Details about this format can be found in [19]. The top-level grammar for the binary encoding of VRML state information is shown in Figure 10. This figure shows that the encoding of an VRML state starts with the meta information: `HEADER` and `TYPE`. The `HEADER` of a VRML state is the string “#VRMLSTATE 1.0\n” while the TYPE contains information about the encoding (e.g. whether or not it is a delta state). Following the meta information is the encoding of the browser state. This includes the current viewpoint of the user as well as the URL of the displayed world. The actual state of the geometry is encoded in the `SCENEGRAPH` part of a VRML state.

```
VRMLSTATE ::=
  HEADER                      // #VRMLSTATE 1.0\n
  TYPE                        // Information about the encoded content
  BROWSER                     // State of the VRML browser
  SCENEGRAPH;                 // State of the scene graph
```

Figure 10: Top-Level Grammar for the Encoding of VRML State Information

The binary encoding can be used for both full VRML worlds and single VRML objects contained in a world. In addition it allows to retrieve and set full states as well as delta states. The encoding of events can be done similarly to the encoding of VRML state information and will be specified in more detail in a special VRML RTP payload definition.

8 Conclusion and Outlook

In this paper we presented a first survey of our general framework for the real-time transmission of *interactive media*. The most important aspect of this RTP-based framework is the introduction of a protocol profile which captures the common attributes of the distributed interactive media class. In particular we identified common requirements of interactive media, e.g. the necessity for the transmission of state information and events. Based on these common requirements we defined the structure of the profile so that it fulfills the specific needs of interactive media.

Relying on the profile it is possible to develop media-independent, generic services. As a proof of concept, one of those services - recording and playback of interactive media streams - was discussed within this work. In addition the instantiation of the profile for distributed interactive Java animations and multi-user VRML was outlined.

There are a number of items we will be working on in the near future:

- The integration of general naming approaches for application level framing into our framework. The work of Raman/McCanne [22] and Fuchs et al. [5] could be used to complement our sub-component IDs.
- Crowcroft et al. propose a Reliable Multicast Framing Protocol (RMFP) [3]. Since we anticipate that limited semantic reliability might be desirable for several payload types using our profile, we think about merging RMFP into our work.
- The development of a sample implementation of the proposed profile. The sample implementation will be made available as a library in order to simplify the integration of the general framework into applications using distributed interactive media.

- Based on the sample library the payload type-specific functionality for distributed interactive Java animations and multi-user VRML will be realized. Both payloads will be tested and validated by using them in appropriate applications.
- We are working on two generic services for interactive media: late join and recording of sessions. Both will be tested with the Java animation and the VRML payload.

During the implementation and testing of the sample library, the payload types and the generic services we expect to get enough feedback for a full specification of the profile and the payload types. We intend to publish those specifications as Internet drafts.

9 References

[1] C. Burger, K. Rothermel, and R. Mecklenburg. Interactive Protocol Simulation Applets for Distance Education. In *Proc. of Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS'98), Oslo, Norway,* page 29 - 41. LNCS 1483, Springer Verlag, Berlin, Heidelberg, New York, September 1998.

[2] Cisco. Cisco IP/TV. On-line: http://www.cisco.com/warp/public/732/net_enabled/iptv/.

[3] J. Crowcroft, L. Vicisano, Z. Wang, A. Gosh, M. Fuchs, C. Diot, T. Turletti. RMFP: A Reliable Multicast Framing Protocol. IETF, Network Working Group, Internet-Draft, March 1998.

[4] A. El-Saddik, A. Steinacker, S. Fischer, and R. Steinmetz. Component-based framework for effective visualization of educational algorithms. Submitted to *Kommunikation in Verteilten Systemen (KIVS'98),* 1998.

[5] M. Fuchs, C. Diot, T. Turletti, M. Hofman. A Naming Approach for ALF Design. *Third International Workshop on High Performance Protocol Architectures (HIPPARCH '98),* London, England, June 1998.

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

[7] O. Graß. *Realisierung eines Whiteboard-Recorder Moduls*. Master's Thesis (in German), LS Praktische Informatik IV, University of Mannheim, Germany, September 1998.

[8] C. Hedrick. *Routing Information Protocol*. Internet Request For Comments, IETF, RFC-1058. June 1988.

[9] V. Hilt. *The Recording of Interactive Media Streams Using a General Framework*. Technical Report TR 14-98, University of Mannheim, Germany, 1998.

[10] D. Hoffman, G. Fernando, V. Goyal, M. Civanlar. *RTP Payload Format for MPEG1/MPEG2 Video*. Request for Comments (Proposed Standard) RFC 2250, Network Working Group, IETF, January 1998.

[11] W. Holfelder. Interactive Remote Recording and Playback of Multicast Videoconferences. In: *Proc. of Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS'97),* Darmstadt, pp. 450-463. Springer Verlag, Berlin, September 1997.

[12] W. Holfelder. *Aufzeichnung und Wiedergabe von Internet-Videokonferenzen*. PhD Thesis (in German), LS Praktische Informatik IV, University of Mannheim, Shaker-Verlag, Aachen, Germany, 1998.

[13] E. Hövel. Synchronisation von Java-basierten Animationen im Rahmen des TeleTeaching. Master's Thesis (in German), LS Praktische Informatik IV, University of Mannheim, December 1997.

[14] Javasoft. Java 3D API Documentation. Javasoft. 1998. On-line: http://www.javasoft.com/ products/java-media/3D/index.html

[15] C. Kuhmünch, T. Fuhrmann, and G. Schöppe. Java Teachware - The Java Remote Control Tool and its Applications. In *Proc. ED-MEDIA/ED-TELECOM'98*. AACE Association for the Advancement of Computing in Education, 1998. Available on CD-ROM.

[16] C. Kuhmünch. Collaborative Animations in Java. Technical Report TR 15-98, University of Mannheim, September 1998. On-line: http://www.informatik.uni-mannheim.de/~cjk/publications/cola-api.ps.gz.

[17] J. Lienhard, G. Maas. AOFwb - a new Alternative for the MBone Whiteboard wb. In: *Proc. of ED-MEDIA'98*, Freiburg, Germany, June 1998. Available on CD-ROM.

[18] M. Mauve. TeCo3D - A 3D Telecollaboration Application Based on VRML and Java. Accepted at *Multimedia Computing and Networking 1999 (MMCN'99) / SPIE'99*, San Jose, February 1999.

[19] M. Mauve. *Access to and Encoding of VRML State Information*. Technical Report TR 13-98, University of Mannheim, Germany, 1998.

[20] S. McCanne and V. Jacobson. vic: A flexible Framework for Packet Video. In *MultiMedia '95 (San Francisco)*, New York, page 511 - 523. ACM. November 1995.

[21] P. Parnes, K. Synnes, D. Schefström. *mMOD: the multicast Media-on-Demand system*. 1997. On-line: http://mates.cdt.luth.se/software/mMOD/paper/mMOD.ps, 1997.

[22] S. Raman, S. McCanne. Scalable Data Naming for Application Level Framing in Reliable Multicast. In: *Proc. of ACM Multimedia '98*, Bristol, September 1998.

[23] H. Schulzrinne. *RTP Profile for Audio and Video Conferences with Minimal Control*, Internet Draft, Audio/Video Transport Working Group, IETF, draft-ietf-avt-profile-new-03.txt, August 1998.

[24] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. Internet Draft, Audio/Video Transport Working Group, IETF, draft-ietf-avt-rtp-new-01.txt, August, 1998.

[25] H. Schulzrinne, A. Rao, R. Lanphier. *Real Time Streaming Protocol (RTSP)*. Request for Comments 2326, Multiparty Multimedia Session Control Working Group, IETF, April1998.

[26] T. Tung. *MediaBoard: A Shared Whiteboard Application for the MBone*. Master's Thesis, Computer Science Division (EECS), University of California, Berkeley, 1998.

[27] M. Watson. *Creating Java Beans: Components for Distributed Applications*. Morgan Kaufmann Publishers, Inc, 1998.

[28] 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, December 1997. On-line: http://www.vrml.org/Specifications/.

[29] VRML Consortium. *Living Worlds - Making VRML 2.0 Applications Interpersonal and Interoperable - Draft 2*. Living Worlds Working Group, April 1997. On-line: www.vrml.org/WorkingGroups/living-worlds/.

[30] World Wide Web Consortium. *Synchronized Multimedia Integration Language (SMIL) 1.0 Specification*. W3C Recommendation, REC-smil-19980615, June, 1998. On-line: http://www.w3.org/TR/REC-smil/.