A Generic Scheme for the Recording of Interactive Media Streams

Volker Hilt, Martin Mauve, Christoph Kuhmünch, Wolfgang Effelsberg

University of Mannheim, LS PI IV, L 15,16
68131 Mannheim, Germany
{hilt,mauve,kuhmuench,effelsberg}@informatik.uni-mannheim.de

Abstract. Interactive media streams with real-time characteristics, such as those produced by shared whiteboards, distributed Java applets or shared VRML viewers, are rapidly gaining importance. Current solutions to the recording of interactive media streams are limited to one specific application (e.g. one specific shared whiteboard). In this paper we present a generic recording service that enables the recording and playback of this new class of media. To facilitate the generic recording we have defined a profile for the Real-Time Transport Protocol (RTP) that covers common aspects of the interactive media class in analogy to the profile for audio and video. Based on this profile we introduce a generalized recording service that enables the recording and playback of arbitrary interactive media.

1 Introduction

The use of real-time applications in the Internet is increasing quickly. One of the key technologies enabling such transmissions is a transport protocol that meets real-time requirements. The *Real-Time Transport Protocol* (RTP) has been developed for this purpose [16]. The RTP protocol provides a framework covering common aspects of real-time transmission. Each encoding of a specific media type entails tailoring the RTP protocol. This is accomplished by an *RTP profile* which covers common aspects of a media class (e.g. the RTP profile for audio and video [14]) and an *RTP payload* specifying the transmission of a specific type of media encoding (e.g. H.261 video streams).

While the class of audio and video is the most important one and is quite well understood, interactive media streams are used by several applications which are gaining importance rapidly. Interactive applications include shared whiteboard applications [3], multi-user VRML models [9] and distributed Java animations [7]. Many existing protocols for interactive media are proprietary. This prevents interoperability and requires re-implementation of similar functionality for each protocol. For this reason, we have defined an RTP profile [10] that covers common aspects of the distribution of interactive media. It can be instantiated for a specific interactive media encoding.

The RTP profile for audio and video has enabled the development of generic recording services like those described in [4][15]. The RTP audio and video recorders operate independently of a specific video or audio encoding. Instead of decoding

incoming RTP packets and storing video and audio content (e.g. in H.261 or MPEG format), they operate on entire RTP packets. This has the major advantage that the mechanisms implemented in the recorder (e.g. media storage or media synchronization during playback) are available for all video and audio formats.

Recent developments extend these RTP recorders to the proprietary protocols of specific applications. In general, interactive media streams require additional functionality in an RTP recorder since certain information about the semantic of an interactive media stream must be considered. In particular *random access* to an interactive media stream requires mechanisms to provide the receivers with the current media state. For example, if a recorded shared whiteboard stream is accessed at a random position, the contents of the page active at that time must be displayed to the user. Thus, a recorder must provide the receiving shared whiteboards with the page content before the actual playback is started.

Our RTP profile for interactive media provides a common framework that enables the development of a generic services like recording or late join for the class of interactive media. In this paper we discuss the principles of such a generic recording service. We present mechanisms that are required for the recording and playback of interactive media streams, and we show that random access to these media streams can be achieved by these mechanisms without having to interpret media-specific data.

The remainder of this paper is structured as follows: Section Two provides an overview over related work. Section Three introduces a classification of different media types. Section Four provides a short overview of our RTP profile for interactive media on which the presented recording scheme is based. Section Five describes the basic architecture of an RTP recording service. Section Six discusses fundamentals of random access to stored interactive media streams, and Section Seven describes two mechanisms that realize media independent random access to these media streams. Section Eight describes the current state of the implementation. Section Nine concludes the paper with a summary and an outlook.

2 Related Work

Much work has been done on the recording of media streams. The rtptools [15] are command-line tools for recording and playback of single RTP audio and video streams. The Interactive Multimedia Jukebox (IMJ) [1] utilizes these tools to set up a video-on-demand server. Clips from the IMJ can be requested via the Web.

The mMOD [13] system is a Java-based media-on-demand server capable of recording and playing back multiple RTP and UDP data streams. Besides RTP audio and video streams, the mMOD system is capable of handling media streams of applications like mWeb, Internet whiteboard wb [6], mDesk and NetworkTextEditor. While mMOD supports the recording and playback of UDP packets, it does not provide a generalized recording service with support for random access.

The MASH infrastructure [11] comprises an RTP recording service called the MASH Archive System [12]. This system is capable of recording RTP audio and video streams as well as media streams produced by the MediaBoard [18]. The MASH Archive System supports random access to the MediaBoard media stream but does not

provide a recording service generalized for other interactive media streams.

A different approach is taken by the AOF tools [2]. The AOF recording system does not use RTP packets for storage but converts the recorded data into a special storage format. The AOF recorder grabs audio streams from a hardware device and records the interactive media streams produced by one of two applications AOFwb or the Internet whiteboard wb. Random access as well as fast visual scrolling through the recording are supported but the recordings can only be viewed from a local hard disk or CD. The recording of other interactive media streams is not possible.

In the Interactive Remote Instruction (IRI) system [8] a recorder was implemented that captures various media streams from different IRI applications. In all cases a media stream is recorded by means of a specialized version of the IRI application that is used for live transmission. This specific application performs regular protocol action towards the network but stores the received data instead of displaying it to the user. For example, a specialized version of the video transmission tool is used to record the video stream. Such a specialized recording version must be developed for each IRI tool that is to be recorded.

One of a number of commercial video-on-demand servers is the Real G2 server. The Real G2 server is capable of streaming video and audio data as well as SMIL presentations to RealPlayer G2 clients. A SMIL presentation may contain video and audio as well as other supported media types like RealText, RealPix and graphics. In contrast to the recording of interactive applications, a specialized authoring tool is used to author SMIL presentation, which consist of predefined media streams displayed according to a static schedule.

3 Interactive Media

3.1 Classification of Interactive Media

Before discussing the recording of interactive media streams, it is important to establish a common view on this media class. Basically, we separate media types by means of two criteria. The first criterion distinguishes whether a 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.

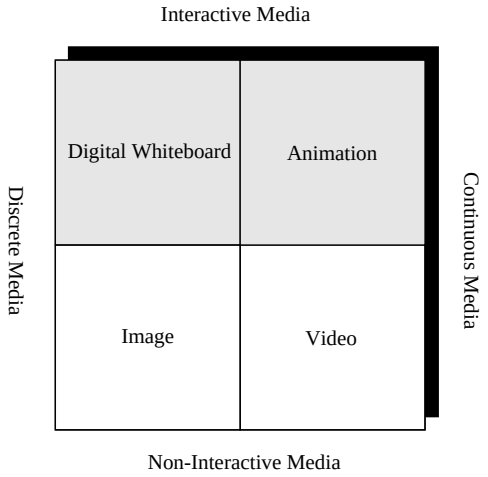

Fig. 1. Examples of Media Types

3.2 Model for Interactive Media

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 the 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 that 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.

In cases where a complex state of an interactive medium is transmitted frequently by an application, 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. The main advantages of delta states are their smaller size and that they can be calculated faster than full states.

In order to provide for a flexible and scalable handling of state information, it is sometimes desirable to partition an interactive medium into several *sub-components*. In addition to breaking down a large media state into more manageable parts, such partitioning allows participants of a session to track only the states of those sub-components they are actually interested in. Examples of sub-components are VRML objects (a house, a car, a room), or the pages of a whiteboard presentation.

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 for 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 a VRML browser. The receiver will also need the VRML world on which the sender acts; otherwise the media stream cannot 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 to the state of the sender before it will be able to interpret the VRML media stream correctly.

Generally speaking, it does not suffice to have a player for an interactive media type. Additionally, the player must be initialized with the *context* of a media stream before that stream can actually 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.

4 RTP Profile for Interactive Media

In order to be able to develop generic services which base solely on our RTP profile for interactive media, common aspects of interactive media streams which are not already handled by RTP must be supported by the profile. These aspects can be separated into two groups: information and mechanisms. Information is needed, so that a generic service can analyze the semantics of the application level communication. The information provided by the RTP profile is: identification of application-layer packet content, identification of sub-components and sequence numbers. Mechanisms are needed by a generic service to take appropriate actions on the medium. The mechanisms provided within the RTP profile are: announcement of sub-components, requesting state transmissions and mapping of sub-component IDs to application-level names.

The remainder of this section discusses basic concepts of the RTP profile for interactive from the view of a generic recording service. A detailed description of the profile can be found in [10].

4.1 Structure of Data Packets

The model presented in Section 3.2 illustrates that states, delta states and events of an interactive medium must be transmitted in real-time. We define the structure of data packets containing theses basic elements of interactive media as depicted in Figure 2 within our RTP profile; for the general structure of RTP packets see [16]. The most important fields of these packets are type, sub-component ID and data. The type field is needed to distinguish the different packet types state, delta state and event defined in

the profile. This is especially important for the recording service, which must be able to identify the type of content transported in an RTP packet without having to interpret the data part of the packet. In state and delta state packets the sub-component ID field holds the sub-component ID of the state included in the data part of the packet. In event packets this field identifies the sub-component containing the “target” of an event. The data field of the packet contains the definition of states, delta states or events specific to the payload type.

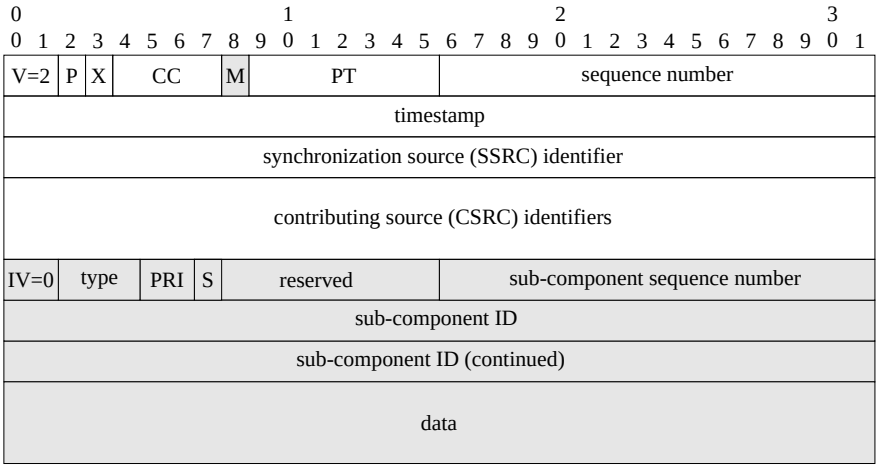

Fig. 2. RTP Packet Structure for States, Delta States and Events

Since setting the state of a sub-component can be costly and might not always be reasonable, state and delta state packets contain a priority (PRI) field. 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. An example of this case is a recorder that periodically requests the media state in order to insert it into the recording.

4.2 Announcement of Sub-Components

For the implementation of an efficient recording service it is important that the sub-components present in a session are known. Furthermore it should be possible to distinguish those sub-components which are currently needed to display the medium. Those sub-components are called *active*. An example for active sub-components are the currently visible pages of a shared whiteboard. All remaining sub-components are

passive (e.g. those shared whiteboard pages which are currently not visible for any user). Declaring a sub-component active does not grant permission to modify anything within that sub-component. However, a sub-component must be activated before a session participant is allowed to modify (send events into) the sub-component. The knowledge about active sub-components in a session allows a recording service to transmit only those sub-components during a playback that are actually visible in the receivers.

The profile provides a standardized way to announce the sub-components of any application participating in an interactive media session and allows to mark sub-components as active. Active and passive sub-components are announced by selected participants in regular intervals within RTCP reports.

4.3 Requesting State Transmissions

In many cases it is reasonable to let the receivers decide when the state of sub-components should be transmitted. Thus, a receiver must be able to request the state from other participants in the session.

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. The state is needed by the receiver to resynchronize with the ongoing transmission. These requests will be relatively rare. In contrast, a recording service needs the media states to enable random access to 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.

5 RTP Recording Service

An RTP recording service such as the MBone VCR on Demand (MVoD) [4] usually handles two network sessions (see Figure 3). In the first, the recorder participates in the multicast transmission of the RTP media data. Depending on its mode of operation (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 [17] 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 loads RTP packets of each media stream and computes the time at which each packet must be sent using the time stamps of the RTP headers. The recorder sends the packets according to the computed schedule. A detailed description of the synchronization mechanism implemented in the MVoD can be found in [5].

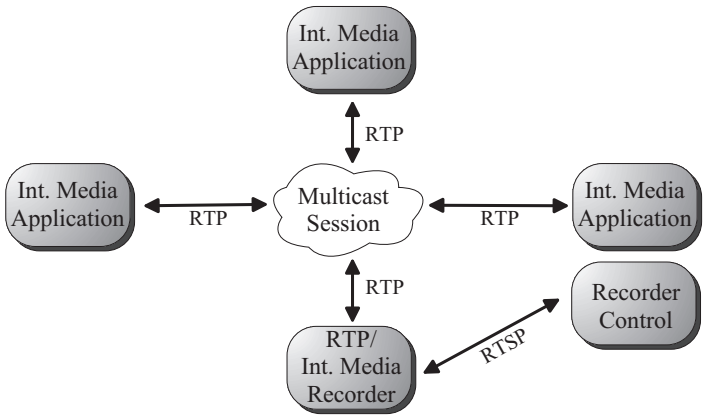

Fig. 3. Scenario for the Recording of an RTP Session

6 Random Access

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. For example, jumping directly to annotations on a whiteboard page only makes sense if the right page is shown on the screen. 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 recording service or by a third party, e.g. an HTTP server. Then the receiver must get the state of the interactive medium at the desired access position within the recorded stream. Let us come back to our whiteboard example. If we want to jump to minute 17 of a recorded teleconferencing session we must be able to show the contents of the page active at that time, together with the annotations made by the speaker. If we did not restore the state of the whiteboard, the page (which might have been loaded originally at minute 12) would not be visible.

6.1 Recovering the Media State

The state of an interactive medium can be recovered from a recorded media stream. Note that the generic recorder is not able to interpret the media-specific part of the RTP packets and thus cannot directly compute the media state and send it to the receivers. But the recorder may re-send existing RTP packets that are stored within the recorded media stream. Thus, it is our goal to compose a sequence of recorded RTP packets containing states and events that put a receiver into the desired state. The task a recorder has to accomplish before starting a playback is to determine the appropriate sequence of recorded packets.

In an interactive media application the current state is determined by an initial state

and a sequence of events applied to that state. In a discrete interactive medium the event sequence is not bound to specific points in time. Thus, the application of an event sequence to an initial state of a discrete interactive medium will always result in the same media state, independent of the speed at which the sequence is applied. In contrast, the event sequence for a continuous interactive medium is bound to specific points in time. A sequence of events that is applied to the state of a continuous interactive medium will leave the system in the correct state only if each event is applied at a well-defined instant in time.

This main difference between discrete and continuous interactive media must be considered when computing the sequence of event and state packets to recover the media state. In the case of a discrete medium, such a sequence can be computed to recover the media state at any point in a recorded stream. In contrast, the media state of a continuous medium can only be recovered at points within a recording where a state is available; events cannot be used for state recovery because they must be played in real-time. Therefore, random access to an interactive continuous media stream will usually result in a position near the desired access point. The more often the state is stored within a stream, the finer is the granularity at which the stream of a continuous interactive medium can be accessed.

Interactive media applications usually send the media state only 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 the access granularity of the stream, which can be tolerated to some degree.

7 Mechanisms for Playback

The mechanisms presented in this section implement the recovery of the media state from recorded media streams. Both mechanisms can be implemented completely in the recorder. The receiving applications need not 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 the RTP profile for interactive media can be recorded, and will be able to receive data from the recorder.

7.1 The Basic Mechanism

This simple mechanism is able to recover the media state from interactive media streams which do not utilize multiple sub-components. When starting playback of such a stream, the best case is if the state is contained in the recorded stream at exactly the position at which the playback is to start. Then playback can begin immediately. But in general, the playback will be requested at a position where no full state is directly available in the stream.

Let us consider, for example, a recorded media stream that consists of the sequence S_0 containing a state, three successive delta (Δ) states and several events (see Figure 4). If a user wants to start playback at position t_p from the recording, the state at t_p must be reconstructed by the recorder. A continuous interactive medium does not allow direct access to t_p because the recorder cannot determine the state at t_p since there is no state available at t_p 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 $\Delta 3$), 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 that may change with the passage of time. Altogether, the recorder will play back sequence S_1 shown in Figure 4.

For discrete interactive media, fast playback of events is possible. Therefore, 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 4.

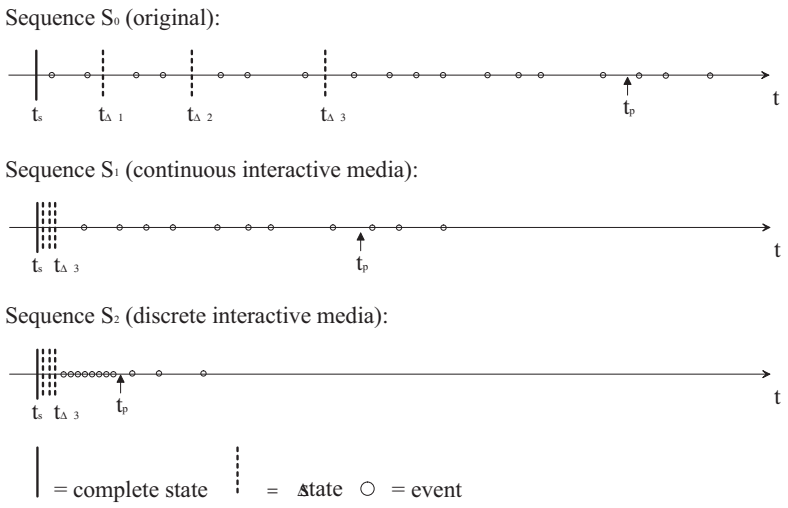

Fig. 4. Playback of a Recorded Sequence of States, Delta States and Events

7.2 Mechanism with Support for Sub-Components

In a more sophisticated mechanism the existence of sub-components can be exploited to reduce the amount of data required for the recovery of the media state. Using sub-components, the state of an interactive medium can be recovered selectively by considering only those sub-components which are actually required to display the medium in its current state.

Let us take a closer look at the shared whiteboard example of Section 6 where we

wanted to access minute 17 of the recording of a teleteaching session. Without the use of sub-components, the recorder would have to recover the complete media state valid at minute 17, which comprises all pages displayed so far. But if the shared whiteboard has divided its media state into several sub-components (e.g. a whiteboard page per sub-component) the recorder is able to determine the sub-components that are active at minute 17 and may recover them selectively.

In general, when a recorded stream is accessed, the set of active sub-components at the access position can be determined and their state can be recovered. This is sufficient to display an interactive medium at the access position. However, it must be assured, that a receiver is enabled to display all subsequent data in the recorded stream. If the subsequent data contains the re-activation of a passive sub-component (e.g. a jump to a previous page in the recording of a shared whiteboard session), a receiver would not hold the state for this sub-component as passive sub-components were not recovered initially. Consequently, the receivers would not be able to decode data referring to that sub-component. Thus, the recorder must assure that the state of a sub-component is present in the recorded stream at any position where a passive sub-component is re-activated. This can be accomplished at the time of recording if the recorder requests a state transmission for each sub-component that gets activated and inserts the retrieved state into the recording. For discrete media streams this scheme can be optimized by not requesting the state of a sub-component if the recorder can reconstruct it from previous data within the recorded stream.

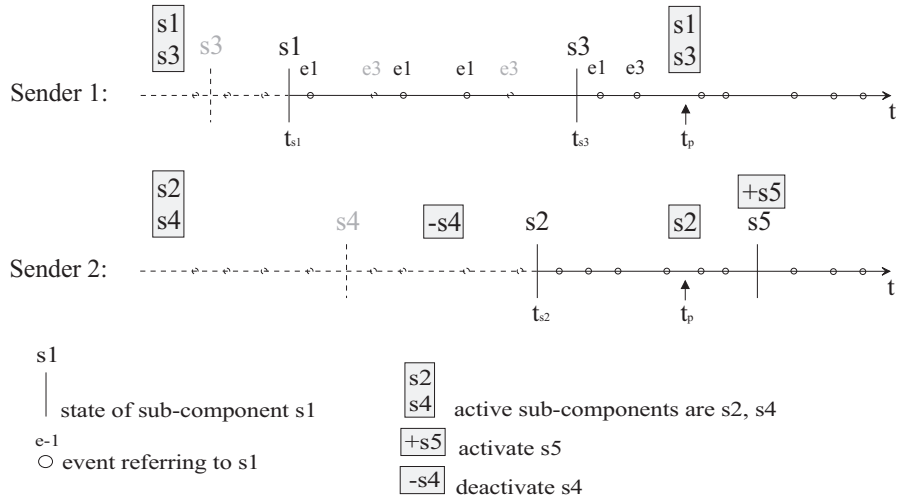

Fig. 5. Playback of a recording containing sub-components. Greyed states and events of the recorded streams are filtered during the recovery of the state at t_p

The example shown in Figure 5 depicts recorded streams of a continuous interactive medium with two senders. Sender 1 operated on sub-components 1 and 3, whereas

the recorded stream of sender 2 contains packets for sub-component 2 and 4 and later for 2 and 5. If these recorded streams are accessed at position t_p, the recorder has to compute the list of sub-components which are active at t_p. In our case these are s1, s2 and s3. For each of these sub-components, the position of the most recent sub-component state before t_p must be located in the recorded stream. As a result, the recorder gets the positions of sub-component states t_{s1}, t_{s2}, t_{s3}. (For the sake of simplicity, we have only considered states; support for Δ states can be achieved similar to the basic mechanism.) s1 is the sub-component whose state is farthest from t_p (here $t_{s1} < t_{s2} < t_{s3}$). Thus the recorder has to start playback at position t_{s1} and recovers the state s1. The recorder must continue with the playback because events referring to s1 are located between t_{s1} and t_p. Notice that we are considering a continuous interactive medium where all events must be played in real time. During the playback of the stream between t_{s1} and t_p two problems may occur: At first, events may be located in the stream which refer to states that have not yet been sent. The sending of these events must be suppressed because a receiver can not interpret them correctly. In our example, events concerning s3 and s4 are filtered out. Secondly, there may be sub-component states in the stream that are not in the set of active sub-components at t_p (s4 in our example) and thus are not needed for playback at t_p. Therefore the state of s4 (and all events referring to s4) must also be filtered out.

Summing up, the recorder will start playback at position t_{s1}, sending the state of sub-component s1 and events referring to s1. All other states and events will be filtered out. The next required state is s2, which will be sent as soon as it shows up and, after that, all subsequent events referring to s2 will also pass the filter. The same holds true for s3. Finally, once the recorder has reached position t_p, the sub-components s1, s2 and s3 will have been recovered, and regular playback without any filtering may start. After the start of the regular playback, the set of active sub-components is enlarged by s5. As the state of a newly activated sub-component has been inserted into the stream during the recording, the state of s5 can be sent by the recorder. Thus, all receivers are enabled to interpret upcoming events referring to s5.

8 Status of the Implementation

Our work on the recording of interactive media streams initially started with the implementation of a recorder for a shared whiteboard, the digital lecture board (dlb) [3]. The dlb recorder is based on the MBone VCR on Demand (MVoD) [4] service which is capable of recording and playing back multiple RTP audio and video streams. The MVoD server assures the synchronization of multiple media streams during playback, and a Java user interface enables the remote control of the recorder. The shared whiteboard dlb uses a specific RTP payload format to transmit the dlb media. The dlb recorder module basically extends the MVoD by implementing the functionality for random access to recorded dlb media streams. To achieve random access, the dlb recorder uses a mechanism that is specific to the dlb media stream.

Based on the experiences from the implementation of the dlb recorder, we are currently implementing the recording service for interactive media described in this paper and we are now finishing a very early alpha version. Like the dlb recorder, the interac-

tive media recorder is realized as an extension to the MVoD. Thus, the existing algorithms for the synchronization of multiple media streams as well as the recording facilities for audio and video streams can be reused. The implementation of the mechanisms for random access allows the presence of discrete and continuous interactive media streams as well as audio and video streams within the same recording.

9 Conclusion

We have presented a generic recording service for the class of interactive media with real-time characteristics. Examples of such media are shared whiteboards, multi-user VRML worlds and distributed Java applications. In analogy to RTP video and audio recorders, we have developed a generic recording service that is based on an RTP profile for the interactive media class. The profile covers the common aspects of this media class. We have presented the basic ideas of the RTP profile, pointing out the features that enable the recording and playback of interactive media regardless of a specific media encoding.

We have described the key concepts of the generic recording service. An important aspect of this recording service is that it enables random access to recorded streams. The media context of a recording is restored before playback is started. We have showed that the context of a medium can be recovered relying only on the RTP profile and we have presented two recovery mechanisms. We are currently finishing the implementation of a first prototype of the described interactive media recording service.

In future work we will implement the RTP payload-type specific functionality for distributed Java animations and multi-user VRML. Our recording service will then be tested and validated with those media types. Furthermore, we are working on a second generic service that will implement a late join algorithm. During the implementation and testing of the RTP profile, 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.

Acknowledgments. This work is partially supported by the BMBF (Bundesministerium für Forschung und Technologie) with the “V3D2 Digital Library Initiative” and by the Siemens Telecollaboration Center, Saarbrücken.

References

[1] K. Almeroth, M. Ammar. *The Interactive Multimedia Jukebox (IMJ): A New Paradigm for the On-Demand Delivery of Audio/Video*. In: Proc. Seventh International World Wide Web Conference, Brisbane, Australia, April 1998.

[2] C. Bacher, R. Müller, T. Ottmann, M. Will. *Authoring on the Fly. A new way of integrating telepresentation and courseware production*. In: Proc. ICCE ‘97, Kuching, Sarawak, Malaysia, 1997.

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

Freiburg, Germany, AACE, June 1998. Available on CD-ROM, contact: http://www.aace.org/pubs/.

[4] W. Holfelder. *Interactive Remote Recording and Playback of Multicast Videoconferences*. In: Proc. IDMS '97, Darmstadt, pp. 450-463, LNCS 1309, Springer Verlag, Berlin, September 1997.

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

[6] V. Jacobson. *A Portable, Public Domain Network 'Whiteboard'*, Xerox PARC, Viewgraps, April, 1992.

[7] C. Kuhmünch, T. Fuhrmann, and G. Schöppe. *Java Teachware - The Java Remote Control Tool and its Applications*. In: Proc. of ED-MEDIA '98, Freiburg, Germany, AACE, June 1998. Available on CD-ROM, contact: http://www.aace.org/pubs/.

[8] K. Maly, C. M. Overstreet, A. González, M. Denbar, R. Cutaran, N. Karunaratne. *Automated Content Synthesis for Interactive Remote Instruction*, In: Proc. of ED-MEDIA '98, Freiburg, Germany, AACE, June 1998. Available on CD-ROM, contact: http://www.aace.org/pubs/.

[9] M. Mauve. *Transparent Access to and Encoding of VRML State Information*. In: Proc. of VRML '99, Paderborn, Germany, pp. 29-38, 1999.

[10] M. Mauve, V. Hilt, C. Kuhmünch, W. Effelsberg. *A General Framework and Communication Protocol for the Transmission of Interactive Media with Real-Time Characteristics*, In: Proc. of IEEE ICMCS'99, Florence, Italy, 1999.

[11] S. McCanne, et. al. *Toward a Common Infrastructure for Multimedia-Networking Middleware*, In: Proc. of NOSSDAV '97, St. Louis, Missouri, 1997.

[12] S. McCanne, R. Katz, E. Brewer et. al. *MASH Archive System*. On-line: http://mash.CS.Berkeley.edu/mash/overview.html, 1998.

[13] 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.

[14] 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-05.txt, March 1999.

[15] H. Schulzrinne. *RTP Tools*. Software available on-line, http://www.cs.columbia.edu/~hgs/rtp/rtptools/, 1996.

[16] 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-03.txt, March, 1999.

[17] H. Schulzrinne, A. Rao, R. Lanphier. *Real Time Streaming Protocol (RTSP)*. Request for Comments 2326, Multiparty Multimedia Session Control Working Group, IETF, April 1998.

[18] T. Tung. *MediaBoard: A Shared Whiteboard Application for the MBone*. Master's Thesis, Computer Science Division (EECS), University of California, Berkeley, 1998. On-line: http://www-mash.cs.berkeley.edu/dist/mash/papers/tecklee-masters.ps.