REIHE INFORMATIK
14/98

The Recording of Interactive Media Streams Using a Common Framework

Volker Hilt
Universität Mannheim
Praktische Informatik IV
L15, 16
D-68131 Mannheim

The Recording of Interactive Media Streams Using a Common Framework[1]

Volker Hilt
University of Mannheim
hilt@informatik.uni-mannheim.de

Abstract: The development of real-time transport protocols for the Internet has been a focus of research for several years. Meanwhile, the Real-Time Transport Protocol (RTP) is a well accepted standard that is widely deployed for the transmission of video and audio streams. The RTP specification, combined with a companion RTP profile, covers common aspects of real-time transmissions of video and audio in various encodings. This enabled the development of RTP recorders which record and play back video and audio streams regardless of a specific media encoding. *Interactive* media streams with real-time characteristics are now gaining importance rapidly. Examples are the data streams of shared whiteboards, remote Java animations and distributed VRML worlds. In this paper we present a generalized recording service that enables the recording and playback of this new class of media. In analogy to video and audio streams we have defined an RTP profile covering common aspects of the interactive media class. We discuss design principles for this recording service and describe the key mechanisms that allow to randomly access recorded interactive media streams independent of a specific media type and encoding.

Keywords: Recording, Interactive Media, Real-Time Transmission, Multicast.

1 Introduction

The use of real-time applications in the Internet is increasing quickly. For several years researchers from all over the world have conducted meetings using video-conferencing tools. A well-known example is the IETF meeting which is transmitted worldwide. In many universities, teachers are starting to disseminate lectures over the Internet to remote students. For example at the University of Mannheim such tele-lectures have been conducted since 1996. One of the key technologies enabling such transmissions is a transport protocol that meets real-time requirements. In the Internet the *Real-Time Transport Protocol* (RTP) has been developed for this purpose [21]. Now a well accepted standard, it is used by many freely available tools for the transmission of real-time media [16], as well as by commercial products [3].

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 [19]) 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. Examples of interactive applications are: shared whiteboard applications [4], VRML models [24], Java animations [9] and SMIL presentations [25].

Currently most media of these kinds are presented locally to a single user. However, it would be very desirable to be able to use them in a distributed fashion, possibly incorporating this media class into teleconferencing, telepresentation, and telecooperation applications. In addition, several interactive media like shared whiteboards or multi-user virtual reality are inherently distributed. Many existing approaches to define protocols and services for of some interactive media are proprietary [4]. This prevents interoperability as well as sharing of common tools while at the same time requiring re-implementation of similar functionality for each protocol. For this reason, we have developed an RTP profile [14] which covers common aspects of the distribution of interactive media. This RTP profile can be instantiated for a specific interactive media encoding (e.g. collaborative VRML [13] or distributed Java animations [10]).

With the increasing use of real-time applications the need arises to record some of the live transmissions. A num-

[1] This work is sponsored by BMBF (Bundesministerium für Forschung und Technologie) as part of the V^3D^2 initiative.

ber of RTP recorders now exist [6][20] that accomplish recording of video and audio transmissions. These recorders store media streams packetized in RTP which has the major advantage that the mechanisms implemented in the recorder (e.g. media synchronization) are more generic; they build upon RTP and do not depend on a specific media encoding. Recent developments extend RTP recorders to the proprietary protocols of specific applications. Examples are the MASH recorder [17], the mMOD [18] and the dlb recorder for the digital lecture board [5][4]. Unlike the generic RTP video and audio recorders, these implementations are dependent upon the specific media encoding of the application. The RTP profile for interactive media provides a common framework for interactive media which allows the development of a generalized interactive media recorder. In this paper we present the mechanisms required to provide a generalized service for recording and playback of interactive media streams. We show that random access to these media streams can be achieved by these mechanisms without interpreting media-specific data.

The remainder of this paper is structured as follows: In Section Two an overview over related work is provided. In Section Three a classification of different media types is introduced, and the scope of our work is defined. Section Four provides a short overview of the RTP profile for interactive media on which the presented recording scheme is based. Design issues for an interactive media recorder are pointed out in Section Five. In Section Six fundamentals of random access to stored interactive media streams are discussed, followed by the description of two mechanisms that realize media independent random access to these media streams in Section Seven. Section Eight concludes the paper.

2 Related Work

Much work has been done on the recording of media streams. Many approaches built upon the RTP protocol, which allows the synchronized recording of media streams in a general fashion. The rtptools [20] 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 a Web-based user interface but the playback cannot be controlled any further.

The mMOD [18] system is a Java-based video-on-demand server that is 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 from applications like mWeb, Internet whiteboard wb [8], mDesk and NetworkTextEditor. While mMOD supports the recording and playback of distinct applications, it does not provide a generalized recording service with support for random access and late join.

The MASH infrastructure [15] comprises an RTP recording service called the MASH Archive System [17]. This system is capable of recording RTP video and audio streams as well as media streams produced by the MediaBoard [23]. 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 and video streams from a hardware device and records the interactive media streams produced by one of the two applications AOFwb [11] 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, and recording of other application streams is not possible.

In the Interactive Remote Instruction (IRI) system [12] a recorder was implemented that captures various media streams from different IRI applications. In all cases the recording of media streams is accomplished by 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. For each IRI tool such a specialized recording version must be developed.

There are a number of comerical video-on-demand servers available, one of them 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, SMIL presentations are authored using a specialized authoring tool and consist of multiple predefined media streams.

At the University of Mannheim we have implemented an MBone videorecorder, the MBone VCR [6], which is capable of recording and playing back multiple RTP audio and video streams. During playback, synchronization of media streams is assured, and a Java user interface enables the remote control of the MBone VCR. We have

recently enlarged the MBone VCR by a module [5] that allows the recording and playback of data from a shared whiteboard, the digital lecture board (dlb) [4]. The dlb uses a specific RTP payload format to transmit data on top of our own reliable multicast protocol SMP [4]. The dlb recorder module 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 they change depending on the reliability of transport medium which may change between recording and playback. The SMP protocol must dynamically adapt itself to the current situation and use different SMP headers. Second, the dlb recorder adds the functionality for playback at random positions. The dlb recorder implements a playout mechanism that is specific to the discrete interactive media stream produced by the dlb. The mechanisms described in Section 7 are generalized versions of this mechanism.

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 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.

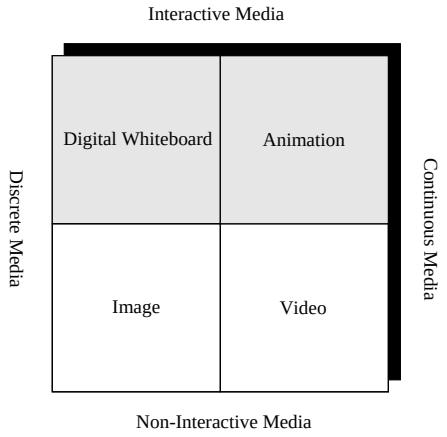

Figure 1: Examples of Media Types

A medium which is neither interactive nor continuous does not have real-time characteristics and is therefore not-discussed any further here. Media types that are non-interactive and continuous have already been investigated to a large extent; digital audio and video streams are the most prominent examples. An RTP profile [19] has been defined, and excellent recording tools exist for these media types. In contrast, proprietary protocols are used by almost all applications for the distribution of, and cooperation with, interactive (continuous, as well as discrete) media. The usage of proprietary protocols prohibits the development of a generalized recording service. For a long time it was impossible to record the whiteboard part of a videoconferencing session. Our first approach to solving this problem was to extend the functionality of an existing recording tool to make it understand the semantics of a shared whiteboard transmission [5]. While this approach enables the recording of one specific shared whiteboard — the digital lecture board [4], it does not provide a generalized solution for all interactive media. However, a recording service can be provided if all interactive media use a general framework for their transmission over the network.

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 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.

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 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 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

This section gives a short overview over the main concepts of our RTP profile for the class of interactive media. An in-depth discussion of the profile can be found in [14].

4.1 State, Delta State, Event

The model presented in Section 3.2 illustrates that two basic elements of an interactive medium must be transmitted in real-time: states and events. But in cases where a complex state is inserted frequently into the media stream, the transmission of states would consume a large bandwidth. Therefore it is necessary to be able to send only those parts of a state that have changed since the last transmission. We call a state which contains only the state changes that have occurred since the last state transmitted 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 the fact that they can be calculated faster than full states.

The state for some media types may get very large so that a transmitted state comprises a huge amount of data. In these cases 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. Examples of sub-components are VRML objects (a house, a car, a room), or the pages of a whiteboard presentation. In the RTP profile the sub-components are used as the level of granularity for state transmissions. To allow applications to discard events for sub-components they are not interested in, all events have to identify the sub-component in which the “target” of the event is located.

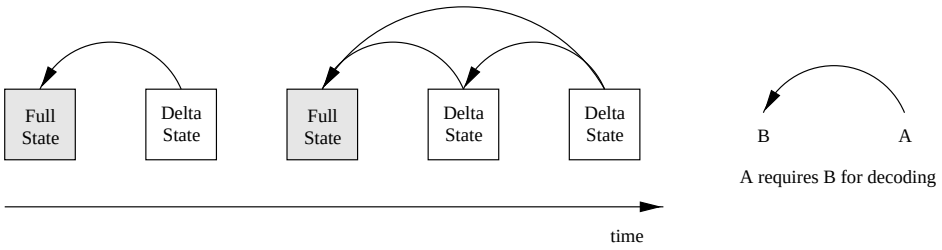

Figure 2: Decoding of Delta States

Most of the data for interactive media are carried in three packet types: state, delta state and event. We define the structure of these packet types as depicted in Figure 3 within our RTP profile; for the general structure of RTP packets see [21]. The most important fields in these packets are type, sub-component ID and data. The type field is needed to distinguish the different packet types 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.

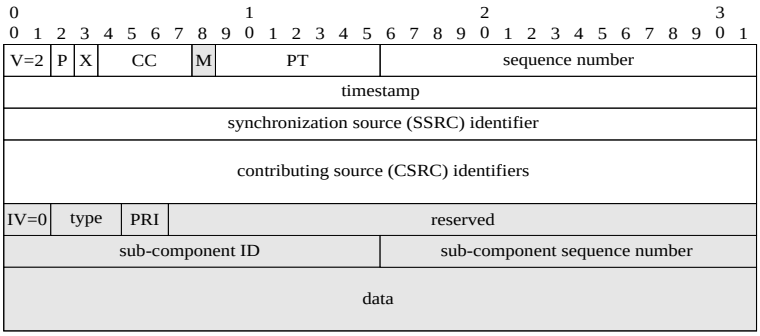

Figure 3: RTP Packet Structure for States, Delta States and Events

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 Active Sub-Components

For the implementation of an efficient recording service it is important that the sub-components necessary to display an interactive medium are known. This allows a recorder to transmit only those sub-components during a playback that are actually visible in the receivers. Our profile provides a standardized way to announce the 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. Declaring a sub-component active does not grant permission to modify anything within that sub-component. It is perfectly reasonable for an application to activate several sub-components just to declare that they are needed for the local presentation of the medium. However, a sub-component must be activated before a session participant is allowed to modify (send events into) the sub-component. It is the responsibility of the application to ensure this behavior, e.g. by using a floor control mechanism.

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

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.

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 recorder 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 Recorder Design

5.1 Recording Scenario

An RTP recorder usually handles two network sessions (see Figure 4). 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 [22] 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. The time when each packet 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 [7].

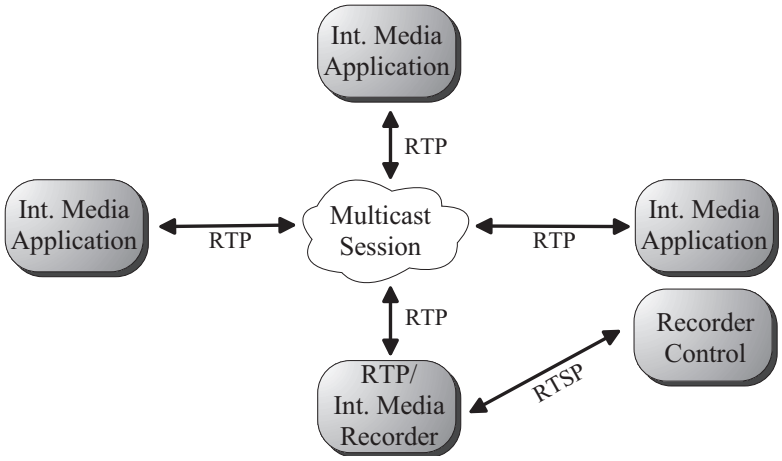

Figure 4: Scenario for the Recording of an RTP Session

5.2 Functionality of a RTP Recorder for Interactive Media

To define the functionality that is required for a recorder for interactive media streams the design metaphor of a VCR is used, since a VCR is a well-known device for dealing with recording and playback of continuous media. 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. A user may have altered this position through operations such as 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 put the recorder into an idle state. Note that the internal clock of a receiver does not stop. The receiver 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 (see Section 7).
  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.

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. 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 desired access position within the recorded stream. Consider a shared whiteboard as an example. If we want to jump to minute 17 of a teleteaching session we must be able to show the contents of the page active at that time, together with the annotations made by the speaker. We would like to go directly to that page, whithout replaying all operations since the beginning to the session.

6.1 Recovering the Media State

The state of an interactive application can be recovered from the recorded media stream. Note that the recorder is not able to interpret the media-specific part of the RTP packets and thus cannot directly compute the state and send it to the receivers. But the recorder may send 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 application the current state is determined by a 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 media 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. Therefore, random access to a continuous media stream will usually result in playback at a position near the desired access position. The more often the state is stored within a stream, the finer is the granularity at which the stream of a continuous medium can be accessed later.

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 only reduces the access granularity of the stream, which can be tolerated in most cases.

6.2 Convergence of Media Contexts

In many cases it is desirable to play the recording of an interactive medium back into an ongoing live session (e.g. a conference). This session incorporates an interactive medium context that has evolved during the course of the session. But when starting the playback of a recording, the medium context of the recording is recovered by the recorder. Thus, the context of the current session must merge with the context of the recording. Basically four policies for merging contexts are possible:

1. The context of the current session is discarded and the context of the recording is loaded.
2. The context of the current session is kept and the context of the recording is loaded. The context of the recording supersedes in the event that parts overlap.
3. The context of the current session is kept and the context from the recording is added. If parts overlap, the current context supersedes.
4. The context of the recording is not loaded at all.

With policies 3 and 4 errors during the playback of the recording are likely to occur because the context of the recording is not restored completely (in policy 4 not at all). With policies 1 and 2, the playback is accurate but the system behavior differs concerning the state of the current session. With policy 1, all the previous work within the session is lost. For example in a session with shared whiteboards, all existing pages will be gone. With policy 2 the context of the session is kept as long as it does not interfere with the context of the recording. In the whiteboard example, all existing pages will remain in the session, and the pages from the recording will be added.

There is no general rule as to which of the above policies is best suited for a specific distributed interactive application. It must be decided for each media type which of the above policies is applicable, for some media types this decision may even be left to the users, who may decide per session. However, a recorder must be able to provide the media context to the participants and thus enable the implementation of all of the above policies in an interactive application.

7 Mechanisms for Playback

The mechanisms presented in this section implement the recovery of the media state in order to allow random access to interactive media streams. Both mechanisms are implemented completely in our 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 the RTP profile for interactive media can be recorded and will be able to receive data from our recorder.

7.1 The Basic Mechanism

This simple mechanism is able to recover the application state from interactive media streams which do not utilize multiple sub-components. 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 full state is directly available in the stream.

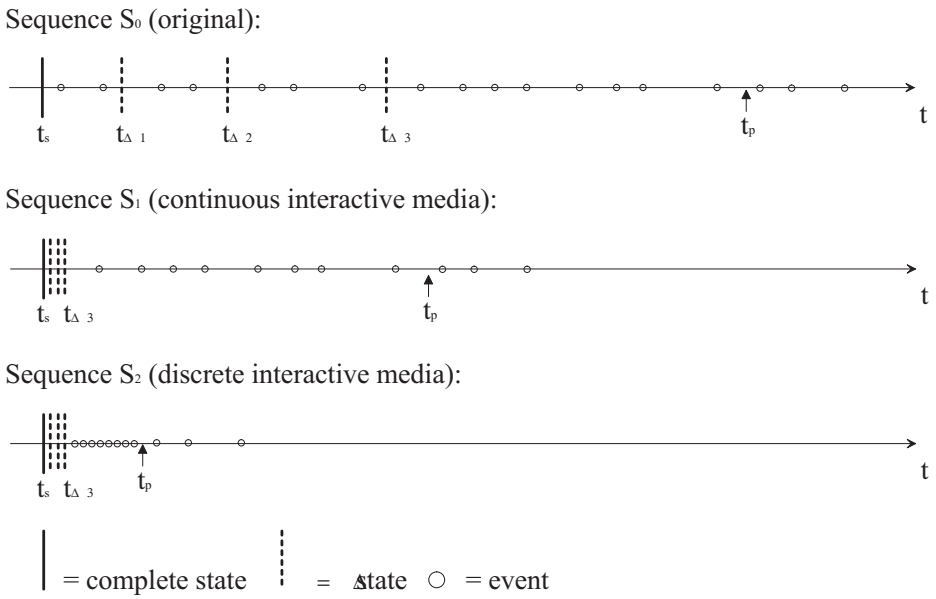

Figure 5: Playback of a Recorded Sequence of States, Delta States and Events

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 5). If a user wants to recover 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 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 that may change with the passage of time. Altogether, the recorder will play back sequence S_1 shown in Figure 5.

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 5.

To implement the basic mechanism, the recorder must collect metadata about the location of states within the stream. For random access to position t_p, it is required to locate the nearest state before t_p, without having to rescan the entire stream. Furthermore all Δ states between that state and t_p must be found. The time needed to locate the states within the stream should be small so as to minimize the initial delay of the playback. To achieve short access times, the locations of states can be stored in an index created during the recording. The index must contain the time stamp of all states as well as the type of each state (full state or Δ state). The recorder can now find the latest complete state by getting the highest time stamp before t_p out of the index and may directly jump to this state within the recording.

7.2 Mechanism with Support for Sub-Components

In a more sophisticated mechanism the existence of sub-components can be exploited to reduce the effort for the recovery by selectively recovering the required sub-components. For example, take the recording of a conference with many speakers where a shared whiteboard was used to distribute slides and annotations. Assume the shared whiteboard has divided its media state into several sub-components where each page corresponds to a sub-component. If a talk within the recording of the conference is accessed by a user, the recorder normally would have to

restore the complete media state at the beginning of the talk. This complete state would comprise all slides from previous speakers in the conference. Using sub-components, the recorder could recover only those slides that are actually relevant for the playback of the talk.

In general, when a recorded stream is accessed, the set of active sub-components at the access position can be determined. It is sufficient to recover the states of these active sub-components before starting playback because active sub-components are those sub-components which are necessary to display an interactive medium (see Section 4.2). If the medium is partitioned into reasonable sub-components, this is much cheaper than recovering the full state of the application.

To determine the sub-components active at an access point the recorder must compute a list of active sub-components for each point in time during the time of recording. At the beginning of a playback, the recorder recovers only the sub-components that are active at the access point. These sub-components are sufficient for a receiver to interpret all data recorded subsequenty. If the set of active sub-components is enlarged later on in the recording, the state of each added sub-component is also contained in the recorded media stream. If an active sub-component is added to a recorded session the application adding the sub-component must send its state. Otherwise the rule in Section 4.2 would be violated. Thus, a recorder recording this session also captures the state of all added sub-components.

The use of sub-components facilitates the coordination of multiple simultaneous senders as each sender may act on a distinct sub-component of the medium. For this reason we have considered multiple senders in the description of the following mechanism. Basically, multiple simultaneous senders may also transmit events targeting the same sub-component although the consistency of the sub-component state will then be hard to maintain for an application.

Let us consider the example shown in Figure 6. If a recorded stream of a continuous interactive medium is accessed at position t_p, the recorder has to compute the list of sub-components s1, ... ,sn, that are active at t_p. 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}, ... , $t_{sn} \leq t_p$ (for the sake of simplicity, we will only consider states and sub-component states; support for Δ states can be achieved similar to the basic mechanism.) Assume s1 is the sub-component of which the state is located farest from t_p ($t_{s1} < t_{s2}$, ... , t_{sn}). Then the recorder would have to start playback at t_{s1} because events referring to s1 may be located between t_{s1} and t_p. Remember that we are considering a continuous interactive medium where events must be played in real time. Unfortunately, we cannot play back all events that are contained in the stream between t_{s1} and t_p. The set of active sub-components at the position t_{s1} will probably comprise more sub-components than s1 (e.g. s3). Therefore events may occur that refer to sub-components other than s1, but s1 is the only sub-component for which we have sent a state so far. As a result, events that refer to other sub-component states must be filtered out. Additionally, there may be state data of sub-components in the stream that are not in the set of active sub-components at t_p and are therefore not needed (e.g. s4). The states of these sub-components must also be filtered out. When starting playback at t_{s1}, the recorder will send the state of sub-component s1 and all events referring to s1. The next required state (e.g. s2) will be sent as soon as it shows up and, after that, all subsequent events referring to s2 will also pass the filter. Finally, once the recorder has reached position t_p, all sub-components that are active at t_p will have been recovered and regular playback without any filtering may start.

In Figure 6 an example session with Sender 1 and Sender 2 is given. Sender 1 is responsible for sub-components

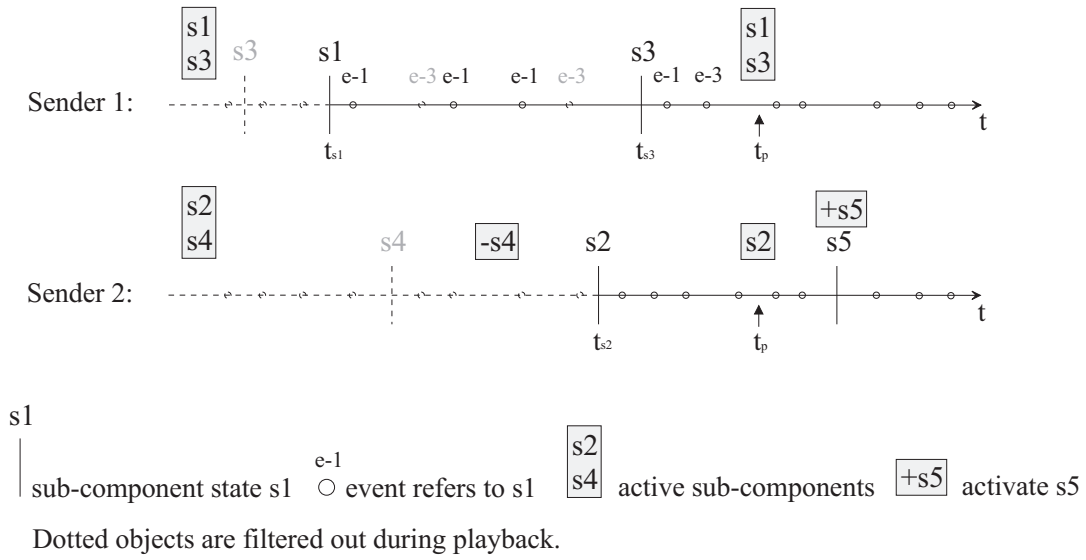

Figure 6: Playback of a recording containing sub-components

s1 and s3, Sender 2 updates s2 and s4. If the recording is accessed at position t_p, the playback must be started at t_{s1} and all dotted objects will be filtered out until t_p is reached. Then the recorder shifts into regular playback mode.

8 Conclusion

We have presented a model for interactive media with real-time characteristics. We have defined a new RTP profile for the transmission of interactive media over RTP. An important aspect of our profile is that it captures common aspects of the interactive media class. Relying on this profile it is possible to develop a media-independent, generic recording service as presented in the second part of our paper. This recording service provides generic random access to all recorded media streams. This is not a trivial task because the context of a medium must be recovered from a recording before starting playback at the desired access position without interpretation of media-specific data. We have presented two mechanisms that accomplish this task in a media-independent manner, relying only on our RTP profile.

Currently we are working on the implementation of the generic recording service by integrating the proposed mechanisms into the MBone VCR. By doing so we are developing a sample implementation of our RTP profile which we will make available as a library in order to simplify the integration of the profile into other applications using distributed interactive media. In parallel two applications which will implement an RTP payload based on our RTP profile are currently under development: a cooperative VRML browser, and remotely controlled Java animations. As a testbed for our generic recording service we are planning to record data from both applications.

During the implementation and testing of the library, the recording service and the interactive media applications, we are expecting to get enough feedback to fully specifiy the profile and the payload types which we intend to publish as Internet drafts.

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

[4] 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/.

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

[6] 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.

[7] 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.

[8] V. Jacobson. *A Portable, Public Domain Network ‘Whiteboard’*, Xerox PARC, Viewgraps, April, 1992.

[9] 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/.

[10] 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.

[11] J. Lienhard, G. Maas. *AOFwb - a new Alternative for the MBone Whiteboard wb.* In: Proc. of ED-MEDIA ’98, Freiburg, Germany, AACE, June 1998. Available on CD-ROM, contact: http://www.aace.org/pubs/.

[12] 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/.

[13] M. Mauve. *Transparent Access to and Encoding of VRML State Information*. To appear in: Proc. of VRML ’99, Paderborn, Germany, 1999.

[14] M. Mauve, V. Hilt, C. Kuhmünch, W. Effelsberg. *A General Framework and Communication Protocol for the Real-Time Transmission of Interactive Media*, Technical Report TR 16-98, University of Mannheim, Germany, 1998. On-line: http://www.informatik.uni-mannheim.de/~hilt/publications/RTPPayloadTR.ps.gz.

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

[16] S. McCanne, V. Jacobson. *vic: A flexible Framework for Packet Video*. In ACM MultiMedia ‘95, San Francisco, pp. 511 - 523, ACM press, November 1995.

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

[18] 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.

[19] 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.

[20] H. Schulzrinne. *RTP Tools*. Software available on-line, http://www.cs.columbia.edu/~hgs/rtp/rtptools/, 1998.

[21] 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.

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

[23] 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

[24] 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/.

[25] 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/.