Recording and Playing Back Interactive Media Streams

Volker Hilt, Martin Mauve, Jürgen Vogel, and Wolfgang Effelsberg

***Abstract*— Recording systems and media server for networked audio and video streams have become an important part of today's Internet. In contrast to this, only a few recording and playback solutions currently exist for the data streams of interactive media applications (e.g., shared whiteboards and distributed virtual environments). So far these solutions are application-specific: individual algorithms and implementations are required for each application that is to be recorded. In this paper, we are proposing generic algorithms for the recording and playback of interactive media streams. These algorithms are based on a common model for the class of interactive media. They enable full random access to recordings by initializing the replaying applications with the required state information (e.g., the current slide in a recorded presentation). We have implemented these algorithms in the Interactive Media on Demand (IMoD) system. In order to interpret the semantics of an interactive media stream, the IMoD system requires that the RTP/I protocol is used for the framing of the transmitted data. Any application using RTP/I can be recorded directly using the IMoD system without any modification. Interactive media streams not using RTP/I can be recorded using the generic recording algorithms. However, they require an adaptation of the IMoD system so that it is able to extract a minimal set of information from the application-level protocol of these streams. In addition to the generic recording algorithms, we present the architecture and major design considerations of the IMoD system and discuss the experiences gained from recording different interactive media applications.**

***Index Terms*— Interactive media, recording, random access, media server.**

Manuscript received October 24, 2003. This work was supported in part by the Deutsche Forschungsgemeinschaft (DFG).

Volker Hilt was with the University of Mannheim, 68161 Mannheim, Germany. He is now with Bell Labs/Lucent Technologies, Holmdel, NJ 07733, USA (phone: +1 732 332 6432; fax: +1 732 949 7397; e-mail: volkerh@bell-labs.com).

Martin Mauve was with the University of Mannheim, 68161 Mannheim, Germany. He is now with the University of Düsseldorf, 40225 Düsseldorf, Germany (e-mail: mauve@cs.uni-duesseldorf.de).

Jürgen Vogel is with the University of Mannheim, 68161 Mannheim, Germany (e-mail: vogel@informatik.uni-mannheim.de).

Wolfgang Effelsberg is with the University of Mannheim, 68161 Mannheim, Germany (e-mail: effelsberg@ informatik.uni-mannheim.de).

I. Introduction

The class of distributed interactive media, i.e., networked media involving user interaction, has received increasing interest over the past decade. Important representatives of distributed interactive media are shared whiteboards, distributed virtual environments, and networked computer games. While the real-time distribution of interactive media has been investigated to a large extent, the recording and playback of interactive media streams is not very well understood so far. Only a few recording systems for interactive media applications are currently available. All of them are specifically designed for one particular application, e.g., a certain shared whiteboard. The recording of a new application requires the design and implementation of a new recording solution. As a result, similar problems are solved individually for each interactive media application instead of solving them in a generic, application-independent fashion.

Algorithms for the recording of audio and video streams, on the other hand, are available as media independent solutions [1][17]. They typically require that packets are framed by a common protocol (e.g., RTP) in order to understand the basic semantics of the data stream. Based on this information, they employ generic algorithms to record and play back arbitrary audio and video streams, independent of the particular encoding (e.g., H.263 and MPEG). Interactive media streams cannot be recorded and played back by simply applying the techniques developed for audio and video streams. In contrast to the primarily stateless nature of audio and video streams, distributed interactive media applications maintain a shared state, which evolves over the course of a session. Therefore the algorithms for the recording and playback of interactive media streams are fundamentally different from the recording of audio and video streams.

In this paper, we show that the idea of developing generic recording algorithms is also applicable to interactive media applications. The generic recording algorithms presented here are based on an abstract model for distributed interactive media [10] that captures the common aspects of the interactive media class. Since these algorithms do not require any media-specific knowledge (e.g., the ability to calculate the media state), they can operate on all data streams that expose the characteristics of interactive media. We also show that a recording service based on these algorithms can be implemented using the information displayed by a common application-level protocol (RTP/I). This protocol provides a minimal set of information about the semantics of the data stream to the recorder. Applications not using RTP/I may also be recorded by using the generic algorithms presented here. However, the actual implementation of a recording service must be modified to extract a minimal amount of information about the semantics of the data stream from the protocol used by the application.

The key contributions of our work are: the development of generic algorithms for the efficient recording and random-access playback of arbitrary interactive media streams, the design and implementation of a recording service using these algorithms to provide a generic recording solution for applications based on the RTP/I protocol, the validation of this concept by recording three distinct interactive media applications, and the usage of this system to regularly record lectures at the University of Mannheim.

The remainder of this paper is structured as follows: Section Two discusses related work. The model for interactive media and the RTIP/I protocol are briefly described in Section Three. Section Four presents generic algorithms for recording and playback as well as details on random access to recorded media streams. The IMoD system is described in Section Five. Section Six presents experiences we have gained from using the IMoD system with different interactive media applications and Section Seven concludes this paper.

II. Related Work

Recording and playback of networked media streams has been the topic of research for quite some time and a number of generic recording solutions for audio and video streams have been developed [1][7][17].

An approach for the recording of interactive media streams is to convert an interactive media stream into a replayable multimedia document (e.g. a SMIL document). The drawback of this approach is that it requires the recorder to fully understand the semantics of the recorded data and thus makes the recording algorithms highly application-specific. Examples for such systems are the recording capability of Microsoft PowerPoint [12] and the netwoked J-VCR [21] recorder.

Other recording systems for interactive media such as the AOF system [14] are tailored towards the local production of multimedia CDs. AOF recordings must be available in full at the player. The AOF data structure enables fine grained random access but does not support streaming over a network. Applications need to be modified for AOF-recording.

Some distributed applications and shared workspaces provide network-based recording facilities. An example is the MASSIVE-3 [3] system, which provides random access to recorded sessions but it is specialized on MASSIVE-3 data.

The WhereWereWe (W3) [13] system is a local recording system based on an event-replay approach. In this approach, recordings contain a complete history of events that occurred in a session. Random access is realized by replaying the events in a recording from its start to the access position. A drawback of this approach is that the replay of events may contain information irrelevant to the state at the access position, for example, already deleted objects. Another problem is that playing back a long event history is very demanding for the receiving applications and may cause large delays before the access position is finally reached. This approach is also taken by network-based recording systems for interactive media applications such as the mMOD [16] system, the MASH archive system [18] and the Multicast Multimedia Conference Recorder (MMCR) [6]. Here, an additional problem is that the event history needs to be transmitted over a network.

Many database systems [15] and distributed interactive applications [22] provide facilities to recover a past state. A technique frequently used is rollback. Rollback requires applications to store a history of events and to be able to undo the effects of each event. However, reversing events is not supported by all applications and realizing rollback for time-based simulations is very difficult since it requires the simulations to go back in time. In addition, rollback shows similar problems as the event-replay approach.

Our approach is to use generic recording and playback algorithms that are based on an abstract model for interactive media.

III. Media Model

In order to provide generic recording algorithms that are usable for an entire media class, it is important to have a suitable media model for this class. The following media model [8][10] is derived from existing interactive media applications. It has been validated against common representatives of shared whiteboard applications, distributed simulations, and networked virtual environments [8].

A. States and Events

A distributed interactive medium has a well-defined *state*. For example, at any given point in time the state of a shared whiteboard is defined by the content of all pages present in the shared whiteboard. In order to perceive the state of a distributed interactive medium a user needs an application, e.g., a shared whiteboard is required to view a shared whiteboard presentation. This application generally maintains a local copy of (parts of) the medium's state. Applications for distributed interactive media are therefore said to have a replicated distribution architecture. For all applications participating in a session, the local state of the medium should be at least reasonably similar. It is therefore necessary to synchronize the local state copies among all participants, so that the overall state of the medium is consistent.

The state of a distributed interactive medium can change for two reasons, either by *passage of time* or by *events*. The state of the medium between two successive events is fully deterministic and depends only on the passage of time. Generally, a state change caused by the passage of time does not require the exchange of information between applications, since each user's application can independently calculate the required state changes. An example of a state change caused by the passage of time is the simulation of a train traveling along a rail circle.

Any state change, which is not a fully deterministic function of time, is caused by an event. Generally, events are (user) interactions with the medium. An example is the annotation of a shared whiteboard page. Whenever events occur, the state of the medium is in danger of becoming inconsistent. Therefore, an event usually requires that the applications exchange information - either about the event itself or about the updated state once the event has taken place.

B. *Sub-Components and Delta-States*

In order to provide for a flexible and scalable handling of state information, it is desirable that applications can partition an interactive medium into several *sub-components*. In addition to breaking down the complete state of an interactive medium into more manageable parts, such partitioning allows the participants of a session to track only the states of those sub-components, in which they are actually interested in. Examples of sub-components are 3D objects (an avatar, a car or a room) in a distributed virtual environment, or the pages of a shared whiteboard. Sub-components are independent state entities. An application can track the state of a sub-component without being required to track the state of other sub-components as well. Events only affect the target sub-component and must not have side-effects on other sub-components. These side-effects would not be visible to applications, which are only tracking the affected sub-components but not the target sub-component. Therefore, it would essentially prevent applications from tracking only a subset of sub-components.

A second concept that can be used by applications to increase the efficiency of managing state information is to transmit *delta-states* instead of full states. A delta-state contains only those parts of a state that have changed since the last transmission of a full state. Applications need the previous full state in order to be able to decode a delta-state. The delta-state concept is orthogonal to the sub-component concept.

C. *Continuous vs. Discrete Interactive Media*

Interactive media types can be classified into two categories: *continuous* and *discrete* interactive media. A continuous interactive medium allows state changes by the passing of time as well as by events. Examples are distributed simulations, networked computer games and virtual 3D worlds. Discrete interactive media allow state changes only through events. A shared whiteboard is an example that falls into this media class.

Applications can combine interactive media, audio and video streams. For example, a video stream could be displayed within a virtual 3D environment. Interactive video, i.e., video allowing user interaction, could be created by adding an interactive media layer containing clickable objects on top of a video layer. Interactive media remains a separate logical entity even if displayed in conjunction with other media types.

D. *RTP/I: A Protocol Framework for Interactive Media*

The media model describes the common aspects of interactive media and provides a foundation for the design of algorithms that can be used for the class of interactive media. An implementation of these algorithms must be able to extract the common aspects from interactive media streams. For example, it must be able to detect states and events. An implementation therefore needs to be able to parse at least parts of an interactive media stream.

The Real-Time Application-Level Protocol for Distributed Interactive Media (RTP/I) [10] is a protocol framework for interactive media, which defines header fields and mechanisms commonly needed for the transmission of interactive media streams. The framework consists of two parts: the RTP/I data transfer protocol defines a standardized framing for interactive media streams, providing information about the transmitted stream in standardized header fields. Examples for such header fields are a timestamp, a sub-component identifier, and the data type (event, delta-state, or state). In addition, RTP/I can be used to request the state of a sub-component. Media specific information such as the encoding of a state or an event is carried in the payload section of a packet. The second part of the RTP/I framework is the RTP/I control protocol (RTCP/I). RTCP/I implements a light-weight session control and provides meta-information about the sub-components used in a session. The RTP/I framework is closely related to the Real-Time Transport Protocol (RTP) [19], which is mainly used for audio and video. However, RTP/I has been thoroughly adapted to meet the needs of distributed interactive media.

Even though many interactive media applications do not use the RTP/I protocol, they may still employ the algorithms for recording and playback presented in this paper. In order to do so, a recorder must be able to extract a minimal set of information from the application-level protocol used by the specific application. If this is realized by an adaptation layer for an RTP/I-based recorder, then the implementation of the actual algorithms for recording and playback can be reused directly.

IV. Algorithms for Recording and Playback

The algorithms for the recording and playback of interactive media streams must accomplish two major tasks: they need to reconstruct the course of a recorded interactive media session and they need to provide random access to recordings.

A. State and Event-based Linear Playback

The algorithms for recording and playback are based on the elements of the media model and are therefore independent of the media specific encoding used for a given interactive media application. A fundamental principle of these algorithms is that they handle abstract elements (i.e., states and events) instead of media specific objects (e.g., pages and lines strokes on a shared whiteboard). A recorder implementing these algorithms does not try to decode an incoming media stream; instead it takes the states and events in an incoming media stream, processes them and writes them to hard disk. During playback, the recorder reads the stored states and events from disk, computes the instant at which they must be sent and transmits them according to the computed schedule. Users can view the playback by joining the playback session with any application that is capable of handling the recorded data stream, e.g., the application that was used in the original session.

The packet sequence transmitted by a recorder enables the receiver applications to reconstruct the course of the media state exactly as it was in the original session. The example in Figure 1 depicts the recording of a train simulation containing a state with timestamp t_s followed by a “start train” event with timestamp t_{e1} and a “stop train” event with timestamp t_{e2} ($t_s < t_{e1} < t_{e2}$). During playback, the recorder sends the initial state and, in the correct temporal distance, the remaining two events. The receiving applications load the state and execute the “start train” and “stop train” events at the time determined by their timestamps. This causes the train to move over the same path as in the original session and to stop at exactly the same position in the rail circle. The reconstruction of the media state is solely the task of the receiving applications and does not require any actions from the generic recorder besides playing back recorded packets.

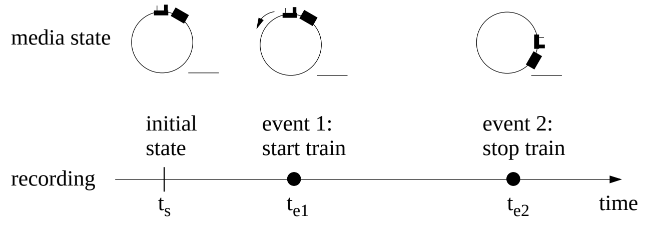


Fig. 1. Example for the playback of a recorded interactive media stream.

B. Random Access

One of the major challenges of playing back interactive media is to provide random access to recorded media streams. Before a receiver is able to decode a playback, it must be initialized with the current state of the interactive medium. For example, in the recording of a shared whiteboard session a slide could have been displayed at minute 32, annotations could have been added at minute 37 and the slide could have been replaced by another slide at minute 40. Would this recording be accessed at minute 35 without any random access mechanism, the playback would not contain the slide visible at the access position since it is located in the recording far before the access position. Thus, the receiving applications would not be able to decode the playback correctly between minute 35 and 40. A recorder for interactive media must therefore provide mechanisms for random access which resolve such dependencies.

1) Recovering the media state: a generic recorder for interactive media cannot directly calculate the media state at an access position. However, it can extract existing packets from the recording and send them to the receivers. The receiver applications have the media-specific knowledge, which is needed to calculate the state out of received packets. Thus, a generic approach for reconstructing the media state during random access is to compose an initialization sequence consisting of recorded packets, which puts the receiving applications into the correct state.

A recorded interactive media stream may contain states, delta-states and events. These elements can be used by a recorder to reconstruct the media state as follows:

- A *state* is self-contained and can directly be loaded by an application. It is a starting point for random access within a recorded stream, similar to an I-frame in an MPEG movie.
- A *delta-state* contains all state changes since the creation of the previous full state. An application must have the preceding full state in order to be able to decode a delta-state (i.e., to compute a full state that can be loaded into an application). In conjunction with the preceding full state, a delta-state also represents a starting point for random access.
- An *event* can only be applied to a certain, well-defined state. The execution of an event on a different state leads to an unpredictable result. In a *discrete* medium, the state is changed only by events and remains constant between two events. The state required to decode an event is therefore available as soon as the previous event has been decoded successfully. Consequently, the state resulting from a sequence of events does not depend on the temporal distance between the events. Merely the order of events is significant. For example, a sequence consisting of a “draw line” and a “move line” event leads to the same final state independent of the temporal distance between the execution of both events.

  The state of a *continuous* medium can change between two events through the passing of time. The state required for the execution of an event is therefore only available exactly at the predefined execution time. A sequence of events in a continuous medium only leads to the desired final state if every event is executed exactly in the correct temporal distance to its predecessor and at the correct point in time relative to the medium's clock. For example, the sequence of a "start train" and a "toggle switch" event in a train simulation only leads to the desired result if both events are executed in the correct temporal distance.

Summing up, an initialization sequence for random access consists of a state, an optional delta-state and a (possibly empty) sequence of events. For a discrete medium, a recorder can send all elements of an initialization sequence right after another without maintaining the original temporal distance between events. The duration of an initialization sequence for discrete media is therefore determined only by implementation constraints such as the time needed to compute the sequence, read the elements from disk, and transmit them over the network. In an idealized scenario this duration is zero and the receiving applications can compute the desired state immediately. This situation is different for a continuous medium where all events must be played back in their original timing. Applications can therefore only load the initial state and a following delta-state immediately and need to apply all successive events in real-time. The initialization sequence for a continuous medium has a duration.

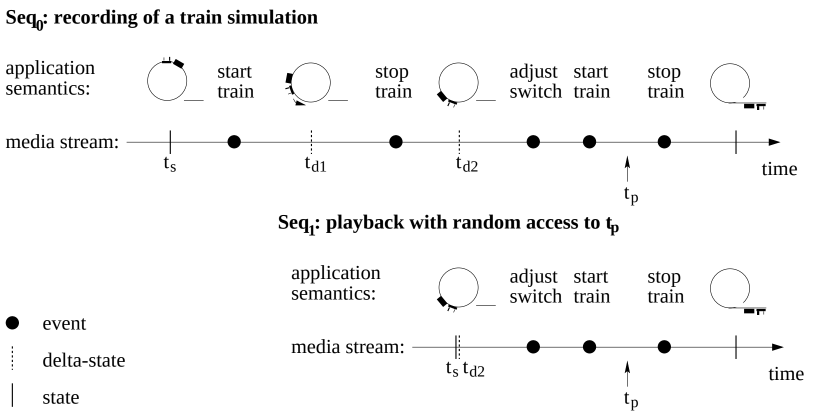


Fig. 2. Example for random access to a continuous interactive media stream.

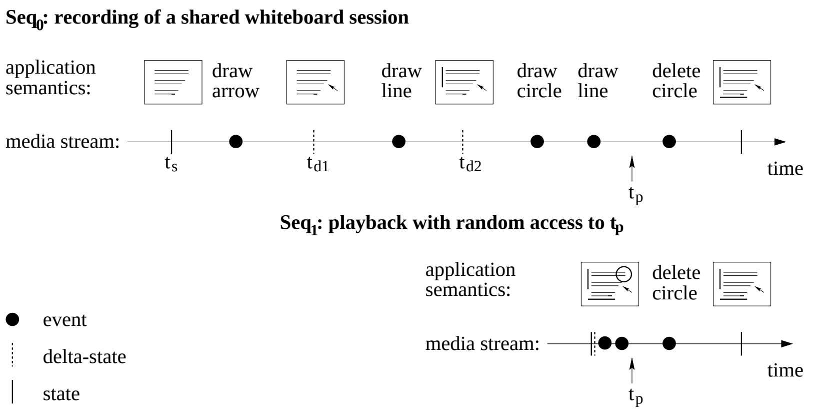


Fig. 3. Example for random access to a discrete interactive media stream.

2) Basic algorithm: the simplified algorithm described in this section realizes the reconstruction of a state, which consists of only one sub-component. It illustrates the basic principle of random access to an interactive media recording.

The example in Figure 2 (sequence Seq_0) depicts the recording of a train simulation which is accessed by a user at position t_p. The first packet needed for the initialization sequence is a state. In our example, the delta-state at t_{d2} is the one that directly precedes t_p. To decode a delta-state, an application needs to hold the last preceding full state, which is located at position t_s in our example. The events and delta-states between t_s and t_{d2} can be left aside since their effects are already contained in the delta-state t_{d2}. Since the train simulation is a continuous medium, all events have to be played back in real-time. For this reason, t_{d2} can be accessed instantaneously whereas all positions after t_{d2} are reached through the successive processing of events by the receiving applications. Consequently, the position t_p is reached by processing the events between t_{d2} and t_p. Sequence Seq_1 (Figure 2) shows the resulting initialization sequence.

Since events can be sent without time constraints in initialization sequences for discrete interactive media, applications of this media type can calculate the state from an initialization sequence at any point in a recording. Figure 3 shows the recording of a shared whiteboard session that is accessed at t_p. After sending the state t_s and the delta-state t_{d2}, the recorder immediately transmits all events between t_{d2} and t_p. Overall, the recorder generates the initialization sequence Seq_1 depicted in Figure 3.

C. Selective Random Access

Applications with a large or complex state often partition their state into multiple sub-components [8]. For a partitioned state, it is typically not desirable to restore the entire state including all sub-components during random access. Instead, it is much more efficient to restore only those sub-components that are actually needed for the requested playback. In the above example, a user wanted to access minute 35 of a shared whiteboard recording. Without the selective reconstruction of sub-components, the recorder would have to recover all slides that are part of the state at minute 35. In a typical shared whiteboard application, this would be all slides that have been presented so far in the session. If we assume that all slides were presented sequentially in the original session, only the slide visible at the minute 35 would be relevant for the remaining playback. All previous slides would not be needed. Their reconstruction would just consume network bandwidth for the transmission and processing power for decoding them in the application. This may introduce significant delays during random access, in particular if the state is large or complex.

Applications with a partitioned state usually distinguish between *active* and *passive* sub-components. Active sub-components are currently used by applications to display the medium to the local user whereas passive sub-components are silently tracked but not used for rendering. Examples for active sub-components are the visible pages in a shared whiteboard or the rooms currently inhabited by users in a distributed 3D world. This implies that an application, which holds all active sub-components at a certain point in time, can fully display the current session at that instant. Consequently, if a recorder transmits all active sub-components to the applications during a playback, the applications are able to fully render that playback. Passive sub-components can be omitted since they do not impact the display of the medium at that point in time. The following two conditions must be met when playing back a partitioned medium:

- the state of all sub-components that are active at the access position must be included in the initialization sequence and
- the state of all sub-components that are activated during a playback must be provided to the receivers before or at their activation.

The first condition assures that applications are able to display the recorded medium at the access position. The second condition ensures that applications hold all relevant sub-components during the remaining playback. This is problematic if a passive sub-component is reactivated without the transmission of its state. In the shared whiteboard example, such a situation would occur if the recording contained a change page event at minute 45, which re-displays a slide that has already been presented at minute 10. This slide would not have been recovered during the selective random access at minute 35, since it was passive at that time. This problem can be solved by assuring that sub-components are never activated during a playback without re-sending their state. Reactivating sub-components without a state transmission is legal for an application during a live session. Thus, a recorder must take additional actions to avoid such situations during playback.

In the *reactive strategy*, the recorder constantly monitors the medium during playback. As soon as it detects the reactivation of a sub-component without a state transmission, it interrupts playback and generates an initialization sequence for the missing sub-component using the random access algorithm described above. This initialization sequence is inserted into the playback before the activation. In the example in Figure 4, an initialization sequence for sub-component *S2* consisting of a state and an event is created at the time *S2* is activated. A pre-requisite for this strategy is that a recorder can generate initialization sequences that instantly initialize sub-components at any point in a recording. For this reason, the reactive strategy can only be applied to discrete media types. In the *proactive strategy*, the recorder constantly monitors the medium during the recording. As soon as it detects the re-activation of a sub-component without a corresponding state transmission, the recorder requests the missing state from the applications. The returned state is then recorded just like any other state. This approach works for all interactive media types. However, it requires that the application-level protocol used supports requesting the transmission of a state. A drawback is that requesting and returning the state takes one round trip time and the time needed by an application to calculate the state. This leads to the artefact that activated sub-components are displayed slightly later in the playback than they were in the original session. However, the delay is usually very small and the medium is fully consistent with its original after the state transmission.

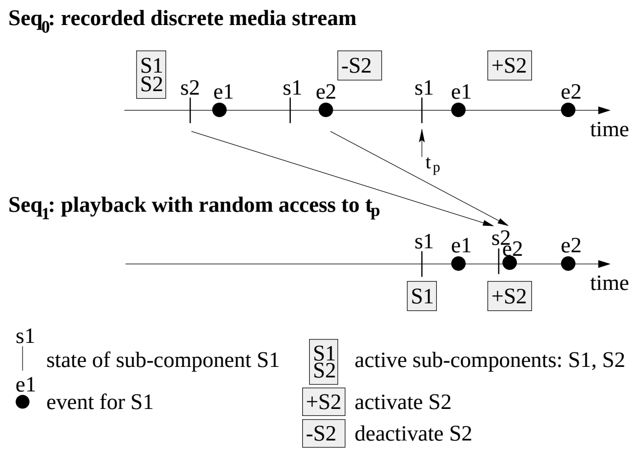

Seq$_0$: recorded discrete media stream
S1 S2
-S2
+S2
s2
e1
s1
e2
s1
e1
e2
time
t_p
Seq$_1$: playback with random access to t_p
s1
e1
s2
e2
e2
time
S1
+S2
s1
state of sub-component S1
S1 S2
active sub-components: S1, S2
e1
event for S1
+S2
activate S2
-S2
deactivate S2

Fig. 4. Example for selective random access using the reactive strategy.

1) Sub-component aware algorithm: during random access, a recorder initializes the applications by successively reconstructing all relevant sub-components. The example in Figure 5 illustrates selective random access to a recorded continuous interactive media stream. At the beginning of the recording the sub-components *S1*, *S2* and *S3* are active. After a while, the sub-component *S2* is deactivated and, later on, sub-component *S4* is activated. To provide random access to this recording at t_p, a recorder must first determine the set of sub-components that are active at that position. In our example, these are the sub-components *S1* and *S3*. For each of those sub-components, the recorder locates the closest state before t_p in the recorded stream. These are the states at t_{s1} and t_{s3}. For simplification, delta-states are not considered in this example. They can be handled as described in the algorithm above. The state of the sub-component *S1* at t_{s1} is located furthest away from the access position (in our example is $t_{s1} < t_{s3} < t_p$) and sent first by the recorder. Since the medium is continuous and there are no other states for *S1* between t_{s1} and t_p, the recorder must play back the remaining initialization sequence in real-time. The event for sub-component *S3* located between t_{s1} and t_{s3} is omitted since *S3* has not been initialized with a state so far. The state and event for sub-component *S2* are also omitted, since *S2* is not part of the set of active sub-components at t_p. As t_{s3} is reached, the recorder sends the state for *S3* and, after that, all events for the sub-component *S3* also pass the filter. When the recorder finally reaches t_p, all sub-components needed to display the medium at t_p have been restored and the regular playback can begin. Shortly after t_p, sub-component *S4* is reactivated. Since the proactive strategy is used, the state of *S4* had been requested from the applications during the recording and is already contained in the recorded stream. Receivers can initialize *S4* with the provided state and process all subsequent events for *S4*.

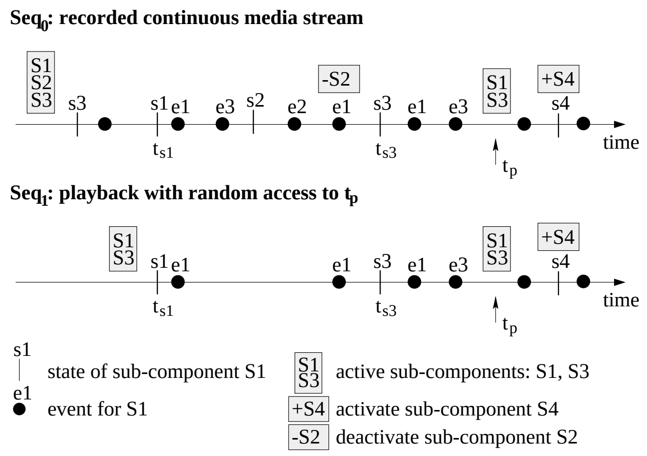

Seq0: recorded continuous media stream
S1 S2 S3
s3
s1 e1
e3
s2
e2
-S2
e1
s3
e1
e3
S1 S3
+S4
s4
ts1
ts3
tp
time
Seq1: playback with random access to tp
S1 S3
s1 e1
e1
s3
e1
e3
S1 S3
+S4
s4
ts1
ts3
tp
time
s1
state of sub-component S1
e1
event for S1
S1 S3
active sub-components: S1, S3
+S4
activate sub-component S4
-S2
deactivate sub-component S2

Fig. 5. Example for selective random access to a recorded continuous medium.

The selective reconstruction of discrete interactive media follows the same scheme but all elements of the initialization sequence can now be sent without considering their timing. Thus, the entire initialization sequence can be sent directly at the access position t_p and immediately installs the desired state within the applications.

V. Interactive Media on Demand System

The algorithms for the recording and playback of interactive media streams have been implemented in the *Interactive Media on Demand (IMoD)* system. The implementation of the IMoD system is based on the RTP/I protocol framework and handles discrete as well as continuous interactive media streams. In addition, it can handle RTP-based audio and video streams. The IMoD system is fully operational and available in versions for Linux and Solaris.

A. Overall Architecture

The architecture of the IMoD system is depicted in Figure 6. The *IMoD server* is implemented in C++ and handles the recording and playback of media streams. The IMoD server is controlled by the *IMoD client* which is implemented in Java and realizes all user interface related aspects.

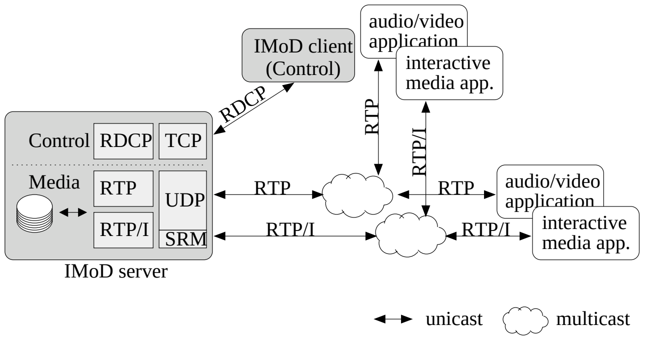


Fig. 6. Architecture of the IMoD system.

The IMoD server communicates with interactive media applications using the application-level protocols RTP and RTP/I. These protocols are decoupled from the underlying transport protocol, enabling the IMoD system to support multiple (reliable and unreliable) transport protocols. Currently, RTP over UDP, RTP/I over UDP and RTP/I over the reliable SMP protocol [2] are possible combinations. UDP and SMP can both run over IP unicast and IP multicast. Other transport protocols may be integrated into the IMoD server as they are needed. The IMoD server implements all recording, playback, and random access algorithms, manages the recordings on hard disk, and provides simple authentication and user management mechanisms.

The IMoD client implements a user interface (see right hand side of Figure 7) to the IMoD server. It communicates with the IMoD server through the text-based control protocol RDCP, which is similar to RTSP [20].

B. Design Considerations

The following section discusses design considerations relevant for an implementation of the recording and playback algorithms.

1) Multimedia sessions: all media streams belonging to a session (e.g., audio, video, and shared whiteboard) are recorded and played back by the IMoD system in synchronization. Synchronizing multiple streams during random access requires that the state of these streams is reconstructed in parallel maintaining the correct temporal relation between streams. Since the initialization sequences of non-interactive media and discrete interactive media do not have a particular duration, they do not require special treatment. In contrast, the real-time playback of continuous interactive media streams must start at the same point for all streams of a session. Initialization sequences for multiple streams can be computed by extending the scheme for partitioned interactive media. The IMoD system merges all sub-components of the entire session into a single set and runs the access algorithm on this set to determine the overall initialization sequence. This initialization sequence is then divided into initialization sequences for the individual streams. All sequences now share a common start time for real-time playback.

2) Recording quality: the quality of a recording is mainly determined by two factors: the performance of random access and the overall fidelity of the playback.

In the ideal case, random access is executed without a perceptible delay and provides real-time playback starting exactly at the access position. The degree to which random access can achieve this goal, is determined by the internal structure of the recorded media stream. Each state in the recording of an interactive media stream represents a potential starting point for an initialization sequence. The more often such a state is present in a recording, the smaller is the average time span between a random access position and the preceding state. A high frequency of state transmissions reduces the time needed to compute an initialization sequence since the portion of the recording that has to be searched for states and events is smaller. It also decreases the number of elements that need to be included in the initialization sequence. This usually leads to a smaller amount of bandwidth required for its transmission and to a quicker rendering of the state in the receiver. For continuous interactive media streams, which require real-time initialization sequences, a high frequency of state transmissions also reduces the average time by which real-time playback is shifted ahead of the access position.

A recorder can periodically request state transmissions from applications to ensure that a recording is accessible with an acceptable quality. However, extracting the state is typically a costly operation for an application and, in addition, each state transmission increases the bandwidth used in the recorded session. A recorder should therefore only request the transmission of a state if the increase in recording quality outweighs the burdens of creating and transmitting the additional state. Since this trade-off highly depends on the medium and the application, the state request rate can be configured individually for each media type in the IMoD system (possible state request rates for an example application are discussed in Section VI). Furthermore, the IMoD system only includes active sub-components in the periodic state requests since passive sub-components are not relevant for random-access. Periodic state requests are assigned a low priority so that they can be ignored by the applications in certain cases (e.g., if the current processor load does not permit the calculation of a state without disturbing the display of the medium). This enables applications to temporarily bail out of the periodic state transmission. Ignoring a periodic state requests decreases the quality of a recording but doesn't endanger its integrity.

The second major aspect of recording quality is that a recording has to be self-contained and should be identical to the original session. A recording gets in danger of not being self contained whenever data that has been transmitted in the live session before the start of the recording is used by an active sub-component. In these situations a state transmission is not necessary in the live session but is required for a later playback of the recording. A recorder therefore needs to request a state transmission in such situations. A typical example is the beginning of a recording where the state of all active sub-components has to be requested. Since the IMoD system uses the proactive random access strategy, state requests are also sent whenever a passive sub-component is reactivated without a state transmission. The IMoD system requests these states with a high priority since they are necessary in order to preserve the integrity of a recording.

Like any other networked application, a recorder is subject to transmission errors. Losses caused by transmission errors must be compensated since they degrade the playback quality and may impede the playback at all. If the recorder receives an interactive media stream over a reliable transmission protocol, error compensation is taken care of by that protocol. If an unreliable transmission protocol is used for an interactive media stream, the recorder must compensate missing data on the application-level. The IMoD system can detect packet losses in interactive media streams based on the RTP/I sequence numbers. Since RTP/I has no built-in reliability mechanisms, the IMoD system can't request the retransmission of a lost packet. However, the IMoD system can request a state transmission for the sub-component that is affected by the loss. The state sent in response to such a request will have a later timestamp than the lost packet and therefore contain the state changes that were introduced by the missing packet. It repairs the loss of the original packet. The drawback of this mechanism is that transmitting a state for each loss may increase the required bandwidth. If losses are caused by congestion, this can aggravate congestion and may lead to further losses. Another way to compensate packet losses when using an unreliable transmission protocol is the use of repair caches as described in [5][18].

3) Timestamps and identifiers: interactive media applications typically implement consistency control mechanisms, for example, local lag and time warp [11]. These mechanisms enable applications to reach a common view on the distributed state, even if they receive packets at different times or in a different order. The IMoD system does not maintain a copy of the distributed state and therefore does not need to implement consistency control algorithms. However, it must ensure that applications are able to reach a common view on the state when they are decoding a playback. This requires in particular that the media streams played back contain valid timestamps, sequence numbers, and identifiers.

During the playback of a recorded interactive media stream, the IMoD system must adapt the header fields of the recorded packets that are used during playback to the current reality. Each RTP/I packet contains a timestamp that reflects the wall clock time at which the content of a packet must be executed and shown to the user. During the recording, the

IMoD system changes this timestamp from wall clock time to a timestamp relative to the beginning of the recording. During playback, a valid timestamp is generated by adding the wall clock time, at which real-time playback started, to this relative timestamp. If the real-time playback started in the middle of the recording, the respective offset needs to be subtracted. The order of packets that receive the same timestamp (e.g. in an initialization sequence) is generally determined by the sequence numbers of the packets.

The second timestamp that needs to be determined by the recorder, is the point in time at which a packet needs to be sent. Applications usually send packets in response to user interactions and set the RTP/I timestamp to an instant short after the interaction occurred to allow for some transmission time. Since the recorder does not create packets based on user interaction and knows all packets beforehand, it can send them out earlier and thereby increase the time between the transmission of a packet and its RTP/I timestamp. This helps minimizing the use of consistency control procedures in applications. The receiving applications then buffer these packets and render them according to the RTP/I timestamp.

An RTP/I packet carries two identifiers in its header fields: the identifier of the sub-component the packet is targeted at and the identifier of the sender. These identifiers require some additional attention during playback. They have to be unique within a session. Since a media stream can be played back into an existing session, it can not be guaranteed that a recorded identifier does not collide with an identifier that already exists in the ongoing session. This will happen for certain if a part of a session is recorded and the recording is played back immediately into the same session. For this reason, the IMoD system replaces each identifier with a newly generated identifier that is unique in the playback session.

VI. Experiences

The algorithms implemented in the IMoD system were evaluated with three different interactive media applications: the distributed VRML browser TeCo3D [9], a networked space combat game, and the shared whiteboard mlb [23]. The first two applications belong to the category of continuous interactive media, the last one is a discrete interactive medium.

A. *Recording TeCo3D*

TeCo3D is an RTP/I-based application which enables participants to share a VRML world. Each participant can interact with the 3D objects in the shared world. TeCo3D is implemented in Java on a code base separate from the IMoD system. Each instance of the TeCo3D application maintains a local copy of the shared VRML world. The distributed state copies are synchronized by exchanging RTP/I state and event packets. TeCo3D has an unpartitioned state. The TeCo3D application supports the transmission of media streams over TCP and UDP. Packet losses are compensated on the application-level by requesting the transmission of a state.

In our experiments with TeCo3D, the IMoD system was running on a SUN Sparc 10 and the TeCo3D applications were executed on Pentium class PCs ranging from 400 to 700 MHz. All machines were connected through a LAN. The TeCo3D media streams were transmitted using UDP over IP multicast during the recording and UDP over IP unicast during playback. Most of the TeCo3D sessions that were recorded and played back in our experiments consisted of one TeCo3D and one H.261 video stream. Some sessions contained additional TeCo3D streams, streams generated by the shared whiteboard mlb or audio streams. In all of the test runs, the IMoD system could record and play back the TeCo3D streams correctly. Figure 7 illustrates the playback of a session containing a TeCo3D stream (upper left corner) and a video stream (lower left corner). The IMoD client is shown on the right hand side.

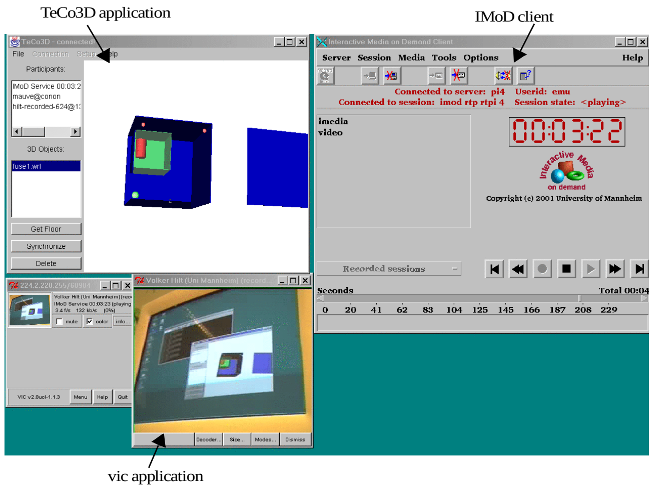

TeCo3D application
IMoD client
vic application
Server Session Media Tools Options
Help
Connected to server: pi4 Userid: emu
Connected to session: imod rtp rtpl 4 Session state: <playing>
imedia
video
00:03:22
interactive Media on demand
Copyright (c) 2001 University of Mannheim
Recorded sessions
Seconds
Total 00:04
0 20 41 62 83 104 125 145 166 187 208 229
Participants:
3D Objects:
Get Floor
Synchronize
Delete
Decoder Size Modes Dismiss
Menu Help Quit

Fig. 7. Playback of a recording containing a TeCo3D and a video stream.

Only a very small number of packet losses occurred during our experiments. Those that did occur were detected by the IMoD system and compensated by requesting a state transmission from TeCo3D and marking the received state as high priority in the recording. During the playback of these recordings, it was noticeable that the TeCo3D application discarded the current VRML world and loaded the new high-priority state. However, this artefact only lasted for brief moment and did not endanger the rest of the playback.

During random access to recorded TeCo3D streams, the TeCo3D application was initialized with the correct state. As expected for a continuous interactive medium, the real-time playback usually began short before the actual access position. During the recording of the test sessions, the IMoD system was configured to issue state requests every five seconds. Responding to these requests had no visible impact on the performance of the TeCo3D applications. An analysis of the packet traces revealed that the applications responded immediately to the state requests in almost all cases and the largest distance between two states was seven seconds. Thus, the real-time playback for this recording starts seven seconds ahead of the access position in the worst case and two to three seconds in the average case.

We conducted an initial experiment to learn about the granularity of random access that is acceptable to users. In this experiment, test recordings were generated, in which the IMoD system requested the state from TeCo3D applications every 120, 60, 30, 10, and 5 seconds, respectively. We asked 15 users, who had not used interactive media recorders before and did not know about the algorithms of the IMoD system, to browse through these recordings and search for the beginning of a given sequence. The setup on the user's screen was similar to Figure 7 except that no video was available to them. The users started playback at a random position in the recording and watched the playback for a few seconds to determine if it is within the sequence they had to find. Then they used the time slider on the bottom of the IMoD-Client to access the recording at a different position and briefly watched the playback again before performing another random access until they had found the sequence. The random access characteristics of recordings with a state frequency of 120 and 60 seconds were generally found to be not acceptable. Real-time playback usually started too far ahead of the access position. Recordings with a state frequency of 30 seconds had better rankings but users still felt uncomfortable with them. With a state frequency of 10 seconds the relocation of the real-time playback start during random access was usually noticed but it was considered to be acceptable. Finally, with a state frequency of 5 seconds the relocation was usually not noticed by users. Although these experiments are not representative and specific to the TeCo3D system, they provide first hints about the state request rate necessary for an acceptable random access behavior. Several users suggested that a visual feedback about the relocation of the real-time playback start would be very desirable. This might increase the acceptance of continuous media streams recorded with lower state transmission frequencies.

B. *Recording a Space Combat Game*

The second continuous interactive media application we evaluated the IMoD system with was an RTP/I-based space combat game. This simple game allows multiple players to maneuver a spacecraft across the shared space and shoot at other spacecrafts with a laser beam. Each game instance maintains a local copy of the state and communicates with other game instances using RTP/I states and events over UDP and IP multicast. The game was developed by a student who was not aware of the IMoD system. The media streams generated by the game could be recorded and played back by the IMoD system right at the first attempt. Random access was also possible and the state at the access position could be restored correctly. This demonstrates that the recording and playback algorithms work for interactive media applications that are based on the above media model even if recording was not specifically considered in the application design.

C. *Recording the Shared Whiteboard mlb*

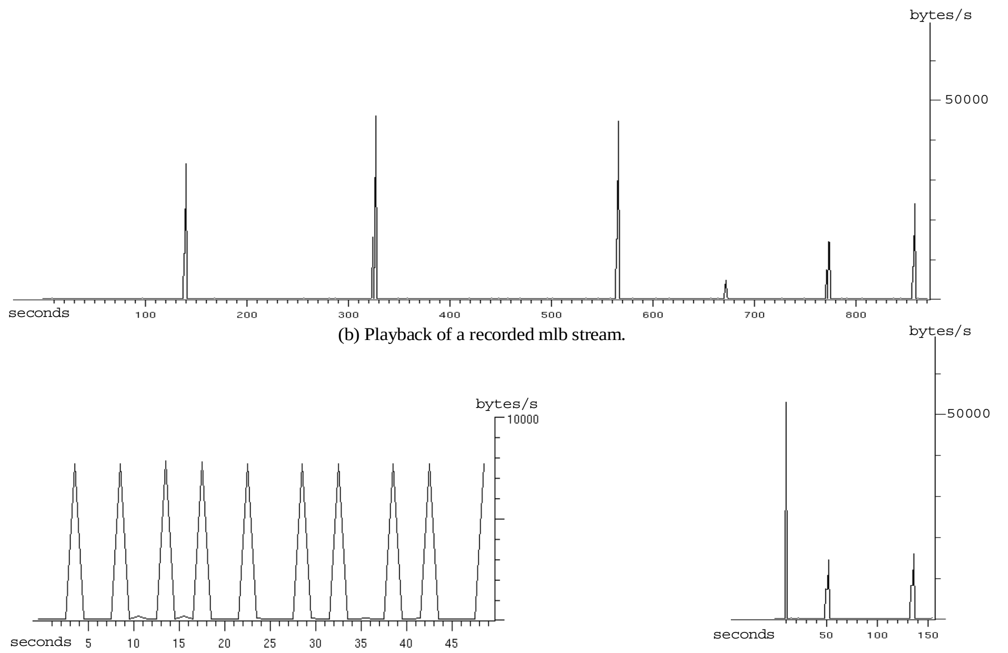
The third application the IMoD system was evaluated with is the discrete interactive medium mlb [23]. The mlb is a RTP/I-based shared whiteboard, which enables users to collaboratively view and annotate slides. The mlb is implemented in C++ and shares the RTP/I protocol library with the IMoD system. The distributed state of the mlb is partitioned into multiple sub-components. Each slide and each annotation on a slide represents a separate sub-component. The state copies are synchronized by exchanging RTP/I states and events over the reliable transport protocol SMP.

Our experiments with the mlb were conducted in the same environment as the TeCo3D experiments. This time, the RTP/I streams were transmitted over the reliable SMP protocol. The IMoD system could successfully record and play back all of our test sessions. Recording mlb streams differs from recording TeCo3D streams mainly in two respects: First, the mlb is a discrete medium and, second, it has a partitioned state. The first characteristic enables the IMoD system to send out the entire initialization sequence at once and allows real-time playback to be started at any position within a mlb recording, which worked fine in all of our experiments. The second characteristic enables the selective reconstruction of the media state. During random access, the IMoD system only initialized the slide and annotations, which were actually visible at the access position. The time needed to initialize the mlb with the necessary sub-components ranged from less than one up to five seconds. This delay is in the range of current video streaming solutions.

The SMP protocol library ensures the reliable transmission of mlb streams. The library delivers retransmitted packets to the application layer (i.e., IMoD) as they arrive, which might be out-of-order. For this reason, the IMoD system implements a re-order buffer to restore the original packet sequence of recorded streams before writing them to disk. In this regard, the IMoD system is able to improve the quality of recorded streams compared to the original transmission.

D. *Performance*

Interactive media streams generally have low bandwidth requirements, which depend on the application and on the content used (e.g., the VRML world or the slide deck). The TeCo3D streams used in our experiments typically consumed 14 kbit/s on average and had peaks of 75 kbit/s for state transmissions. Figure 8 (a) depicts a recorded TeCo3D stream, which contains a state transmission every five seconds. Our simple space shooter game used 3 kbit/s on average with peaks of 17 kbit/s. The mlb streams typically consumed less than 10 kbit/s on average and had very short peaks of up to 500 kbit/s during the transmission of slides (see Figure 8 (b)). Figure 8 (c) shows the playback of the stream depicted in (b) with random access to second 730.



(b) Playback of a recorded mlb stream.

(a) Playback of a recorded TeCo3D stream.

(c) Random access to second 730 of stream (b).

Fig. 8. Bandwidth (bytes/s) requirements of recorded mlb and TeCo3D streams.

Since only the current slide is initialized, the peak at the beginning of the playback is similar to the transmission of a slide.

For interactive media servers that handle a large number of streams, the peaks in one stream are evened out by silence in other streams (assuming an uncorrelated playback of streams). Therefore, a server needs to allocate the respective average data rate to each stream. Playing back an interactive media stream requires the same amount of resources in the IMoD server as the playback of a video stream with the same data rate. Since the data rate of interactive media streams is typically much lower, interactive media servers are expected to be able to play back a larger number of streams simultaneously than a comparable video server. However, random access to an interactive media stream is more expensive than to a video stream. This is mostly caused by the search for the state and the events needed to create the initialization sequence. Thus, video servers can handle more random access operations than comparable interactive media servers. Depending on the access pattern and the structure of the recorded interactive media streams, an interactive media server can be expected to scale similarly to a video server over the number of streams it handles in parallel.

E. *Usage in Distance Education*

Since spring semester 2002, the IMoD system is used on a regular basis to record and play back courses in Computer Science at the University of Mannheim. These courses are usually transmitted via IP multicast to one to four remote locations. The sessions consist of an MP3/RTP audio stream, a H.261/RTP video stream, and a mlb/RTP/I shared whiteboard stream. The audio and video streams usually carry the voice and image of the lecturer, whereas the mlb stream contains the slides and their annotations. All recordings are stored on an IMoD server and can be accessed through a Web-based computer-based training unit. Students can access the recording of a single slide or listen to the entire lecture. During playback, the media streams are send via unicast to a student's PC. Our log files indicate that no more than 15 students were viewing lectures at the same time, which generated a load that could easily be handled by the IMoD server.

VII. Conclusion

In this paper, we have discussed the recording and playback of interactive media streams, i.e. data streams generated by distributed interactive media applications. Examples for such applications are shared whiteboards, distributed virtual environments, and networked computer games.

We have proposed generic algorithms for the recording of interactive media streams that are based on a common model for the class of interactive media. These algorithms only operate on the abstractions provided by the media model and can be used with many different interactive media applications. One of the major challenges of the generic recording and playback of interactive media streams is to provide random access to recordings. The receiving applications must be initialized with the current state of the medium before the actual playback is started. This problem is solved by creating initialization sequences based on recorded states and events. These initialization sequences are efficient, since they do not require applications to process the entire history of a recording and are able to recover only those parts of the application state that are actually needed for a playback.

We have implemented the presented algorithms in the Interactive Media on Demand (IMoD) system using a common application-level protocol framework for interactive media (RTP/I). This framework provides the recorder with a basic set of information about the semantics of a media stream. The IMoD system can record all interactive media applications that are based on the RTP/I protocol without modifications. Other interactive media applications can also be recorded using the proposed algorithms. However, a recorder for these applications has to be enabled to extract the required semantical information from their application-level protocol.

We have validated our concepts by using the IMoD system with three different interactive media applications: the distributed VRML-browser TeCo3D, a space combat game, and the shared whiteboard mlb. We have discussed our experiences and the lessons we have learned from designing and using a recording system for interactive media.

REFERENCES

[1] K. Almeroth, and M. Ammar, “The Interactive Multimedia Jukebox (IMJ): A New Paradigm for the On-Demand Delivery of Audio/Video”, in: *Proc. 7th International WWW Conference*, Brisbane, Australia, 1998.

[2] W. Geyer and W. Effelsberg, “The Digital Lecture Board - A Teaching and Learning Tool for Remote Instruction in Higher Education”, in: Proc. ED-MEDIA 1998, Freiburg, Germany , 1998.

[3] C. Greenhalgh, J. Purbrick, S. Benford, M. Craven, A. Drozd, and I. Taylor, “Temporal Links: Recording and Replaying Virtual Environments”, in: *Proc. ACM Multimedia*, 2000, pp. 67 - 74.

[4] M. Handley, V. Jacobson, and C. Perkins, “SDP: Session Description Protocol”, Internet-Draft, IETF, draft-ietf-mmusic-sdp-new, 2003. Work in progress.

[5] V. Hilt, M. Mauve, and W. Effelsberg, “A Light-Weight Repair Protocol for the Loss-Free Recording of MBone Sessions”, in: *Proc. IEEE Distributed Computing Systems*, Phoenix, USA, pp. 63 - 68, 2001.

[6] L. Lambrinos, P. Kirstein, and V. Hardmann, “The Multicast Multimedia Conference Recorder”, in: *Proc. IEEE IC3N*, 1998, Lafayette, USA, pp. 208-213, 1998.

[7] D. Makofske and K. Almeroth, "From Broadcast Television to Internet Audio/Video: Techniques and Tools for VCR-Style Interactivity", Journal of Software and Experience, vol. 31, no. 8, pp. 781-801, 2001.

[8] M. Mauve. “Distributed Interactive Media”, Ph.D. Thesis, Universität Mannheim, DISDBIS No. 71, Infix-Verlag, St. Augustin, Germany, 2000.

[9] M. Mauve. “TeCo3D - Sharing Interactive and Dynamic 3D Models”, *Kluwer Multimedia Tools and Applications*, vol. 20, no 3, pp. 283-304, 2003.

[10] M. Mauve, V. Hilt, C. Kuhmünch, and W. Effelsberg, “RTP/I - Towards a Common Application-Level Protocol for Distributed Interactive Media”, *IEEE Transactions on Multimedia*, vol. 3, no. 1, pp. 152 - 161, Mar. 2001.

[11] M. Mauve, J. Vogel, V. Hilt, W. Effelsberg. Local-lag and Timewarp: Providing Consistency for Replicated Continuous Applications. *IEEE Transactions on Multimedia*, vol. 6, no. 1, pp. 47-57, Feb. 2004.

[12] Microsoft, “Microsoft Office PowerPoint”, [Online]. Available: http://www.microsoft.com/office/powerpoint/

[13] S. Minneman, S. Harrison, B. Janssen, T.P. Moran, G. Kurtenbach, and I. Smith, “A Confederation of Tools for Capturing and Accessing Collaborative Activity”, *Proc. ACM Multimedia*, San Francisco, CA, USA, pp. 523-534, Nov. 1995.

[14] R. Müller, and T. Ottmann, “The "Authoring on the Fly" System for Automated Recording and Replay of (Tele)presentations”, *ACM/Springer Multimedia Systems Journal*, vol. 8, no. 3, May 2000.

[15] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write Ahead Logging", *ACM Transactions on Database Systems*, vol. 17, no. 1, pp. 94-162, 1992.

[16] P. Parnes. “An IP-Multicast based Framework for Designing Synchronous Distributed Multi-User Applications on the Internet”, Ph.D. Thesis, Department of Computer Science and Electrical Engineering, Luleå University of Technology, Luleå, Sweden, Nov. 1999.

[17] RealNetworks. “RealNetwork.com Products & Services”, [Online]. Available: http://www.realnetworks.com/products/

[18] A. Schuett, R. Katz, and S. McCanne, “A Distributed Recording System for High Quality MBone Archives”, *Proc. Networked Group Communications*, Pisa, Italy, 1999.

[19] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications”, RFC 3550, IETF, July 2003. Work in progress.

[20] H. Schulzrinne, A. Rao, R. Lanphier, M. Westerlund, and A. Narasimhan, “Real Time Streaming Protocol (RTSP)”, Internet-Draft, IETF, draft-ietf-mmusic-rfc2326bis, 2004. Work in progress.

[21] S. Shirmohammadi, L. Ding, and N. Georganas, “An Approach for Recording Multimedia Collaborative Sessions: Design and Implementation”, *Multimedia Tools and Applications*, vol. 19, no. 2, pp. 135-154, 2003.

[22] T. Tung, “MediaBoard: A Shared Whiteboard Application for the MBone”, Master's Thesis, Computer Science Divison (EECS), University of California, Berkeley, USA, 1998.

[23] J. Vogel and M. Mauve, “Consistency Control for Distributed Interactive Media”, *Proc. ACM Multimedia*, pp. 221-230, Ottawa, Canada, October 2001.