RTP/I - An Application-Layer Protocol for the Transmission of Interactive Media with Real-Time Characteristics[1]

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

Abstract

In this paper we present RTP/I, an application-layer protocol for the transmission of interactive media with real-time characteristics, i.e. media involving user interaction. By identifying and supporting the common aspects of interactive media RTP/I allows the development of generic services for distributed, collaborative work involving the transmission of interactive media. Derived from the experience gained with audio and video transmission using the Real-Time Transport Protocol (RTP), RTP/I is defined as a new RTP profile for interactive media.

1. Introduction

The development of application-layer protocols for the real-time distribution of audio and video has been a focus of research for several years. Most notable is the success of the Real-Time Transport Protocol (RTP) [12]. RTP is a protocol that must be tailored to the specific needs of different media and media classes. It is therefore accompanied by documents describing the specific encoding of different media types within the RTP framework. The current documents focus mainly on the encoding of various audio and video formats [11]. While these two basic media classes can be considered the most common ones, there exist several others which are gaining importance rapidly.

This paper presents RTP/I, an application-layer protocol for the transmission of interactive media with real-time characteristics, i.e. media involving user interaction. Examples of interactive media are: shared whiteboard applications [1], multiuser VRML models [14] and distributed Java animations [5]. Existing approaches to define application-layer protocols for the distribution of this media class are mostly proprietary [1]. This prevents interoperability as well as sharing of common tools while requiring re-implementation of similar functionality for each protocol.

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

A classification of different media types and important properties of the interactive media class are introduced in Section Two. Section Three covers the common requirements for application-level protocols supporting this media class. The RTP profile - RTP/I - is presented in Section Four. Section Five gives a brief overview of a generic recording service for interactive media. Section Six concludes this paper with a summary and an outlook.

2. Interactive Media

In order to define the scope of our work, we separate media types by means of two criteria. The first criterion distinguishes whether the medium is discrete or continuous. The characteristic of a *discrete medium* is that its state is independent of the passage of time. Examples of discrete media are still images or digital whiteboard presentations. While discrete media may change their state, they do so only in response to external events, such as a user drawing on a digital whiteboard. The state of a *continu-*

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

ous medium, however, depends on the passage of time and can change without the occurrence of external events. Video and animations belong to the class of continuous media. The second criterion distinguishes between interactive and non-interactive media. *Non-interactive media* change their state only in response to the passage of time and do not accept external events. Typical representations of non-interactive media are video, audio and images. *Interactive media* are characterized by the fact that their state can be changed by external events such as user interactions. Whiteboard presentations and interactive animations represent interactive media. Figure 1 depicts how the criteria characterize different media types.

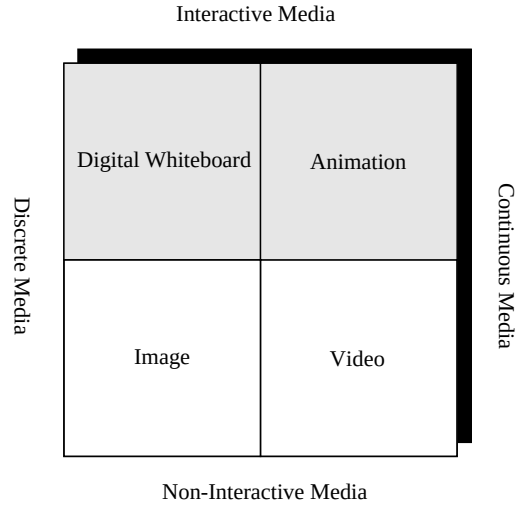

Figure 1. Examples of Media Types

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

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 generalized services for important aspects such as recording or late-join. However, general services like these would be within reach if all interactive media where required to make certain assertions about the protocol they use. It is the aim of RTP/I to define these required assertions, in order to develop a general framework for interactive media.

2.1. 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 passage of time or by *events*. The state of an interactive medium between two successive events is fully deterministic and depends only on the passage of time. Any state change which is not a fully deterministic function of time is caused by an event. A typical example of an event is the interaction of a user with the medium. An example of a state change caused by the passage of time might be the animation of an object moving across the screen. As will be shown later, states and events play an important role in the transmission of interactive media and constitute the main part of an interactive media stream.

In order to provide for a flexible and scalable handling of state information, it is often desirable to partition an interactive medium into several *sub-components*. In addition to breaking down the complete state of an interactive medium into more manageable parts, such partitioning allows 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. Events have a *target* sub-component which they affect. Other sub-components than the target are not affected by an event.

In cases where a complex sub-component state of an interactive medium is transmitted frequently by an application, it is necessary to be able to send only those parts that have changed since the last state transmission. We call a state which contains only the state changes that have occurred since the last transmitted state a *delta* state. A delta state can only be inter-

preted if the preceding full state and interim delta states are also available (see Figure 2). The main advantages of delta states are their smaller size and that they can be calculated faster than full states.

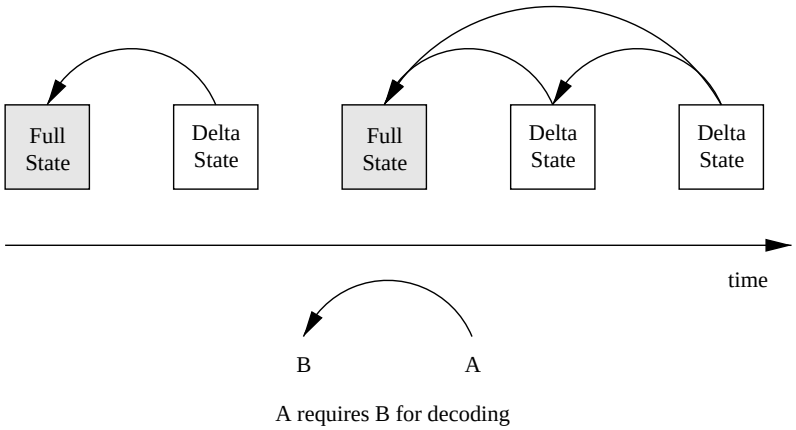

Figure 2. Decoding of Delta States

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

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

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

3. Protocol Requirements

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

RTP is a protocol which was left incomplete on purpose so that it can be tailored to the specific needs of different media. This process of adaptation can be done in two steps. The first step is to define an RTP profile. A profile defines the common aspects of several related media. The second step is to specify in detail how a single medium is transported using the packet types defined by RTP. This specification is called a payload type definition. Following this approach, we define RTP/I as an RTP profile, while each individual interactive medium is specified as a payload type within the profile.

In order to be able to develop meaningful generic service which base solely on the profile and not on specific payload type definitions, RTP/I needs to identify and support the common aspects of interactive media streams which are not already handled by RTP. These aspects can be separated in two groups: information and mechanisms. Information is needed, so that a generic service can analyze the semantics of the application level communication. Mechanisms are needed, so that a generic service can take appropriate actions.

In detail the required information is:

- **Identification of application-layer packet content**. For generic services it is important to be able to identify the type of content transported in an application-layer packet. The profile should specify an identification mechanism for the different packet types common to all interactive media (e.g. event, state and delta-state packets).
- **Identification of sub-component**. A sub-component ID should identify the sub-component of a state or a delta-state. For events, the sub-component ID of the target is needed.
- **Sequence numbers**. Sequence numbers on a per sub-component level are needed, so that packet loss can be attributed not only to a sender, but also to a sub-component.

It is important to notice that this information is needed for each data packet sent by the application. The information should therefore be included in a profile header following the standard RTP header. The mechanisms, however, should be realized as additional RTCP packets. In detail these mechanisms are:

- **Announcement of sub-components**. For many generic services it is important that the sub-components present in a session are known. The profile should therefore provide a standardized way for session participants to announce the sub-components they are currently keeping. Furthermore this mechanism should make it possible to identify those sub-components which are currently needed to display the medium to any one participant. Those sub-components are called *active*. An example for active sub-components are the currently visible pages of a shared whiteboard. All remaining sub-components are *passive* (e.g. those pages which are currently not visible for any user).
- **Requesting state transmission**. There should be a standardized way to request the transmission of the state of a certain sub-component. With this mechanism generic services like late-join and recording can request the transmission of state information without any payload specific information.
- **Mapping aub-component IDs to application-level names**. Typically the sub-component ID will not carry sufficient information for the application to decide whether it is interested in the sub-component or not. Therefore it is necessary to have a mechanism which maps sub-component IDs to application-level names. These names can carry further information like, for example, the page number of a shared whiteboard page, or the spatial position of a 3D object in a shared 3D environment. It is important to note that application level naming is not within the scope of RTP/I. RTP/I should merely provide a mapping mechanism from sub-component IDs to sub-component application level names, while the naming itself is done via other means (e. g. as described in [10]).

4. RTP Profile for Interactive Media

4.1. Data Packet

In order to comply with the requirements in Section 3 most of the data for interactive media is carried in RTP/I data packets. Essentially the RTP/I data packet is an RTP packet with an additional header, it is structured as depicted in Figure 3. The most important fields in these packets are type, sub-component ID, sub-component sequence number, and data. The type field is needed to distinguish between events, states, and delta-states. In state and delta state packets the sub-component ID field is used to hold the sub-component ID of the state included in the data part of the packet. In event packets this field identifies the sub-component in which the “target” of an event is located. A separate sub-component sequence number is present in RTP/I data packets. Based on this information the receivers can ignore packet loss or request the repair of appropriate sub-components. The data field of the packet contains the definition of states, delta-states or events as specified by the payload type.

In addition to the four basic fields, the profile also defines the meaning of three other fields. While RTP/I payloads should try to fit each event, state and delta-state application data unit (ADU) into one transport packet, this might not always be possible. For cases where fragmentation is required, the start (S) bit identifies the start of an application data unit, while the marker (M) bit signals its end. Since setting the state of a sub-component can be costly and might not always be reasonable, a priority (PRI) field is present in state and delta-state packets. This priority can be used by the sender of the state to signal its importance. A packet with high priority should be examined and applied by all communication peers which are interested in the specific sub-component. Situations where high priority is recommended are re-synchronization 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 are late joins where only applications joining the session might be interested in certain state transmissions.

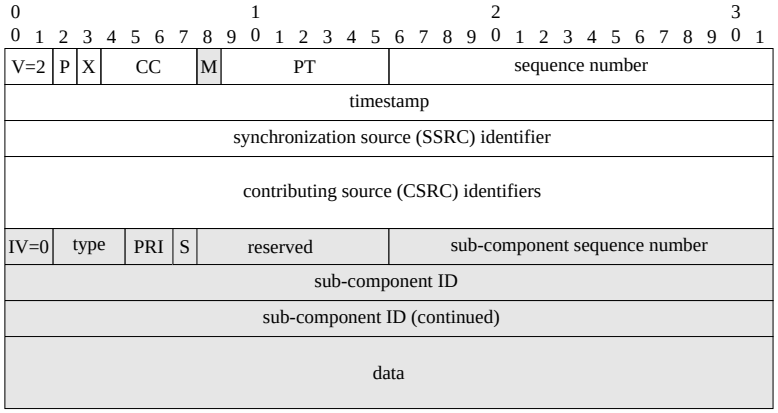

Figure 3. RTP/I Data Packet

4.2. Sub-Components List Packet

As requested in Section 3 the profile provides a standardized way to announce the sub-components of an interactive media session. Similar to the basic RTCP a report interval is calculated based on the overall number of sub-components present in a session. The report interval increases with the number of sub-components, so that a fixed amount of bandwidth is used for sub-component announcements. For each report interval every participant checks whether any sub-component that it keeps has not been reported in the current report interval. All unreported sub-components are then reported by the application. The first check of an participant is made at a random time within the first report interval, so as to avoid that multiple participants send a report at the same time. This algorithm is both scalable - in most cases each sub-component is reported only once, independent of the number of participants - and robust.

A report for a sub-component includes its current status: active or passive. The active sub-components of an 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. It is important to note that declaring a sub-component active does not grant permission to modify anything within that sub-component. It is perfectly reasonable for an application to activate several sub-components just to indicate that these are needed for the local presentation of the medium. However, a sub-component must be activated by a session participant before that participant is allowed to modify (send events into) the sub-component. If a sub-component has been reported as passive by a remote application and that same sub-component is active for the local application, the local application will report that sub-component as active. This could result in two reports being sent for a single sub-component in a single report interval. However, this situation will not last, since the next time the remote application checks for sub-component reports, it will see the active report and will count that sub-component as reported for the report interval. In order to allow for sub-component ID to application-level name mapping, the application-level names of sub-components are included in the report in regular intervals.

Announcement of sub-components is made by means of the sub-components list packet (see Figure 4). The sub-component list packet consists of a one byte RTP/I header with the version number (IV), followed by a list of sub-component stati which in turn is followed by a list of sub-component IDs. For each sub-component the active (A) bit in the list of stati is set if the

sub-component is active for the sender. The description bit (D) is set if the sub-component is followed by its application level name in this sub-components list packet.

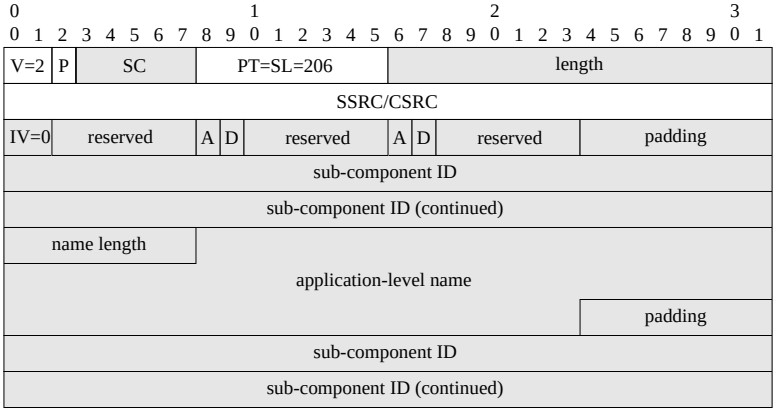

Figure 4. Sub-Components List RTCP Packet

4.3. State Query Packet

In many cases it is reasonable to let the receivers decide when the state of sub-components should be transmitted. For this reason a receiver must be able to request the state from other participants in the session. Within the profile this mechanism is realized by means of a special RTCP packet called *state query*. This packet is depicted in Figure 5. In order to address the requested states, a list of sub-component IDs is present.

As the computation of state information may be costly, the sender must be able to distinguish between different types of requests. Recovery after an error urgently requires information on the sub-component state. These requests will be relatively rare. In contrast to this, a recorder does need the media states to enable random access for the recorded media. It does not urgently need the state but will issue requests frequently. For this reason, the state request mechanism supports different priorities through the priority (PRI) field in the state query packet. Receivers of the state query packet should satisfy requests with high priority (e.g. for late joiners) very quickly. Requests with low priority can be delayed or even ignored, e.g. if the sender currently has no resources to satisfy them. Receivers of a state query packet must be aware that the quality of the service offered by the requesting application will decrease if requests are ignored. In a scenario where multiple participants can receive the same request (e.g. multicast transport), the application needs to make sure that no reply-implosion occurs.

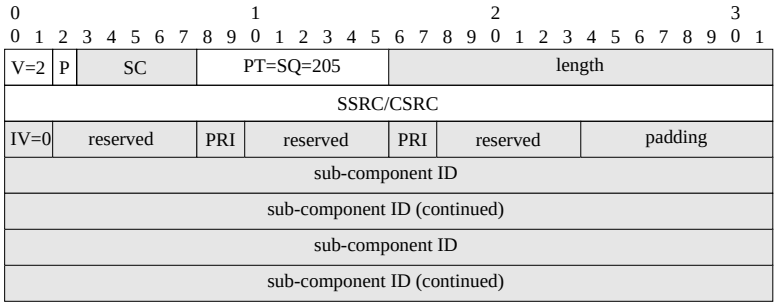

Figure 5. State Query RTCP Packet

5. The Recording Service

With the growing number of real-time transmissions on the Internet, the need arose to record some of the transmissions. Meanwhile a number of RTP recorders exist (e.g. the MBone VCR [4]) accomplishing this task at least for video and audio streams. These recorders store media streams packetized in RTP. The major advantage of this approach is that the mechanisms implemented in a recorder can build upon RTP and do not depend on a specific media encoding. Due to the lack of a common framework for interactive media, recorders for that category exist only for specific applications. Examples of recorders for shared whiteboards are the MASH media board recorder [13] and the recorder for the digital lecture board [2]

[1]. But since there are many types of interactive media, it is advantageous to implement the mechanisms needed for the recording of interactive media streams in a more general fashion. With a general framework for interactive media streams, all transmissions that use RTP/I can be recorded.

5.1. Random Access to Interactive Media Streams

When operating in recording mode, an RTP/I recorder receives RTP/I packets (which basically contain states or events) and writes them to a storage device. The more often the media state is received and stored within the stream, the finer is the granularity with which the stream can be accessed at playback time (see below). Interactive media applications usually send the media state upon request by another application. Thus, the recorder must request the state at periodic intervals. The requests use a low priority because a delayed or missing response reduces only the access granularity of the stream, which can be tolerated to some degree.

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. Then the receiver needs the state of the interactive medium at the access position within the stream.

This state can be recovered from the recorded media stream. Notice that the recorder is not able to interpret the media-specific part of the RTP packets and therefore cannot directly compute the state and send it to the receivers. But the recorder may send RTP packets which are stored within the recorded media stream. Thus, if the recorder can compose a sequence of recorded RTP packets containing states and events that put a receiver into the desired state, the state of an interactive media stream at an access position can be reconstructed. The task a recorder has to accomplish before starting a playback is to determine the sequence of recorded packets that will put the receivers into the state that is required for the access at the desired position. After this initialization, the recorder switches to regular playback mode and sequentially reads the recorded RTP packets from its storage device and sends them to the multicast group.

A detailed description of mechanisms that enable the recovery of the state of an interactive medium out of a recorded stream can be found in [3].

6. Conclusion and Outlook

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

Relying on the profile it is possible to develop media-independent, generic services. As a proof of concept, one of those services - recording and playback of interactive media streams - was briefly summarized within this work. A more detailed version of this paper can be found in [9]. In addition to the recording service we are currently developing a generalized late-join algorithm and sub-component based encryption for RTP/I.

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

- The development of a sample implementation of the proposed profile.
- Based on the sample implementation the payload type-specific functionality for distributed interactive Java animations [6], multi-user VRML [7] [8] and a shared whiteboards [1] will be realized.
- We are working on three generic services for interactive media: late join, recording of sessions and sub-component based encryption. All three services will be tested with the Java animation, the VRML payload and our shared whiteboard.

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

7. References

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

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

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

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

[5] C. Kuhmünch, T. Fuhrmann, and G. Schöppe. Java Teachware - The Java Remote Control Tool and its Applications. In *Proc. of ED-MEDIA/ED'98*, Freiburg, Germany, AACE, 1998. Available on CD-ROM.

[6] C. Kuhmünch. *Collaborative Animations in Java*. Technical Report TR 15-98, University of Mannheim, 1998.

[7] M. Mauve. TeCo3D - A 3D Telecollaboration Application Based on VRML and Java. In: *Proc. of MMCN'99 / SPIE'99*, San Jose, 1999.

[8] M. Mauve. Transparent Access to and Encoding of VRML State Information. In: *Proc. of VRML'99*, Paderborn, 1999.

[9] M. Mauve, V. Hilt, C. Kuhmünch and 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.

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

[11] 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, 1998.

[12] 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, 1998.

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

[14] VRML Consortium. *Information technology -- Computer graphics and image processing -- The Virtual Reality Modeling Language (VRML) -- Part 1: Functional specification and UTF-8 encoding*. ISO/IEC 14772-1:1997 International Standard, 1997.