

IMPLEMENTATION OF AN OBJECT-BASED MULTIMEDIA CLIENT-SERVER SYSTEM

Liang Cheng⁺, James Yeung^{*}, Anders Fredriksson[#], and Magda El Zarki[^]
School of Information and Computer Science
UC Irvine, Irvine, CA 92697
USA

{⁺lcheng61, [^]magda}@ics.uci.edu, ^{*}yeungsta@hotmail.com, [#]anders.fredriksson@opentraining.se

ABSTRACT

This paper presents the architecture and implementation of an object-based, multimedia system, which distinguishes itself from traditional audio-video (AV) streaming applications. This work is a continuation of previous research in interactive multimedia systems [1][2][3] by the Video over IP Group at UC Irvine. We focus this paper on the media layer of the system. The architecture, with roots in the MPEG-4 system standard [4], is unique because we treat independent media streams as individual objects, which are delivered in separate channels via the RTP/UCP/IP protocol stack. It allows us not follow all the specifications of the MPEG-4 standard, i.e. the delivery multimedia integration framework (DMIF) and multiplexing. We implement the video and audio encoding/decoding according to the MPEG-4 specifications [5][6]. An extensible process thread framework of the client presentation application allows for the integration of additional media decoders. We explore the problem of multiple-stream media synchronization that arises from an object-based system. In order to maintain the timing and coordination of independent media streams, we implement intra-stream and inter-stream synchronization techniques to achieve a level of media synchronicity.

KEY WORDS

multimedia, object-based, MPEG-4, synchronization

1. Introduction

The paradigm of multi-stream, object-based, audio-visual presentations has been gaining momentum in the area of interactive media applications. With the MPEG-4 standard (ISO/IEC 14496 [4]) approved in 1999 by the Moving Pictures Expert Group, a generic architecture and set of tools have been available. However, the path towards an interactive, streaming, multimedia presentation system is wide open. In this paper, we discuss the design, architecture, and implementation of the client/server of the multimedia system. We attempt to simplify the MPEG-4 system specifications while preserving the advantages of an object-based system. The goal is an interactive system that would allow the end user to manipulate media objects and to create a multimedia

presentation tailored to his or her specific needs [1]. We begin by delineating the architecture of our implementation. Furthermore, we discuss the MPEG-4 video [5] and audio [6] decoders that have been integrated into our system. We conclude by describing the inter/intra-synchronization algorithms utilized to achieve synchronicity during the composition of an audio-visual scene.

2. System Architecture

The MPEG-4 Systems specification, ISO/IEC 14496-1 [4], specifies a system for the communication of interactive audio-visual scenes. In particular, the specification develops the Systems Decoder Model (SDM), which is an abstraction of the behavior of a client-side terminal within an MPEG-4 system in order to present an MPEG-4 scene. While the SDM provides a conceptual framework to assist in buffering and synchronization, much of the functionality is not defined in the realization that they are application dependent. We find redundancy in some of the functionality of the SDM, and we believe that the SDM should be modified to fit the requirements of different applications. Our motivation for re-designing the client-side model is to simplify the SDM model in order to bypass unnecessary complexities while preserving the object-based characteristics of the system. In actual implementation, the new model is abstracted into software modules for functional efficiency and extensibility.

2.1 MPEG-4 Systems Decoder Model

As seen in figure 1, the SDM is composed of several general elements, including DMIF Application Interface (DAI), decoding buffer (DB), decoder, compositor buffer, and compositor. The DAI provides a standardized interface that hides the details of the transport layer from the user-level application. Streaming data received by the DAI is demultiplexed into Synchronization Layer (SL) packetized streams that lead into corresponding decoding buffers. Each SL packet encapsulates a fraction of, or up to one single access unit (AU) along with corresponding timestamps, i.e. the decoding time stamp (DTS) and composition time stamp (CTS). Each AU represents the

smallest data entity to which timing information can be attributed [4] and consumed by a decoder. First, the SL packets are placed in a DB to extrapolate the AU's. AU's are then forwarded to the corresponding decoder at the specified DTS. After the AU's are decoded into composition units (CU's), they are temporally stored in the composition memory. Finally, the compositor takes CU's out of the memory buffer at the desired composition time and either renders the data in a scene or skips them, depending on the composition algorithm or timing model used.

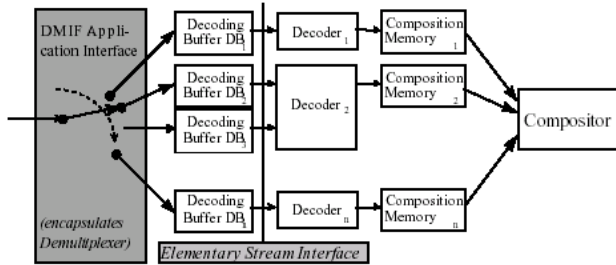


Figure 1. Systems Decoder Model [4]

2.2 VIPClient System Decoder Model (V-SDM)

VIPClient corresponds to the receiving terminal (client) of the VIP Presentation System. V-SDM represents our system decoder model built in the VIPClient. While V-SDM borrows from the concepts of the SDM described in 2.1, the functionalities of the components are modified in a manner that suits our application [2]. Thus, there are sizable differences in our approach of developing an object-based presentation client. The biggest difference is that our system does not utilize the DAI. In fact, our system does not multiplex media streams on the transport level at all; instead, each stream is independently transported from server to client via separate ports. This independence provides our system with additional flexibilities and truly models a unique object-based system [1]. For instance, individual media streams being transported through unique channel ports are directly mapped to the Sync Layer, namely an RTP-to-SL mapping is implemented. Instead of using a separate DB entity that stores AU's for each decoder, the VIPClient allows the SL layer to handle the buffering, i.e. storing the SL packets. The decoder is then responsible for retrieving the AU's with their corresponding time stamps through the SL packet interface. However, in the context of error resilience of video transmissions, SL-packets encapsulating a fraction of an AU are also allowed to be consumed by the decoder; section 3.1 details an error-resilient decoder implemented in the system. After the decoding process, the CU's are stored in a memory buffer available for composition. In the VIPClient, the composition of media comprises of both a rendering module for video media and an audio mixer module for audio media. Figure 2 shows the V-SDM model.

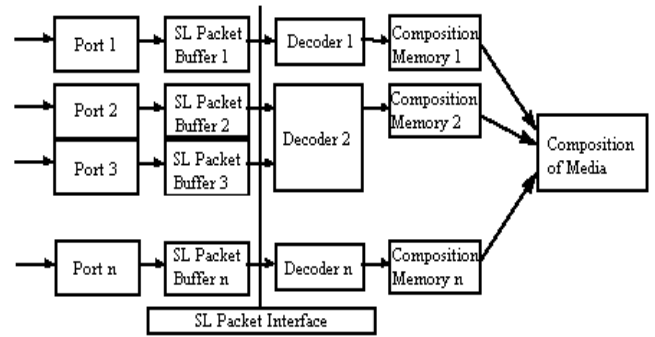


Figure 2. VIPClient Systems Decoder Model (V-SDM)

Another difference lies in the timing model of the V-SDM. Because the MPEG-4 standard does not specify a model for synchronizing decoded media streams, implementations are given flexibility for synchronization. In the VIPClient, decoding time stamps (DTS's) are not used, even though we keep the DTS section in the SL packet header. This is due to the DTS's functional redundancy when considering the CTS. Since the DTS facilitates synchronization immediately before decoding, the timestamp does not benefit our system because RTP transmission-time is not guaranteed and the decoding time of different media is highly variable. Thus in V-SDM, the decoder is assumed to be able to grab AU's/SL-packets as fast as it can from the SL packet interface. In other words, the SL layer should always have SL packets available upon request from a decoder. Otherwise, the decoding pipeline for that media stream is halted until the decoder obtains more data from the SL interface. On the other hand, composition time stamps (CTS's) are used by the VIPClient system in order to achieve synchronization for composition/rendering. In our system, corresponding CTS's of CU's are used by the synchronization modules to determine whether to consume/render the CU or skip it based on the algorithm described in section 4. With the aid of the CTS, our system is able to ensure that independent streams be rendered in a synchronous manner.

2.3 Process Thread Structure

The V-SDM shows the general flow of the media object streams at a receiving terminal, but the actual implementation of the client in software reflects a different view of the architecture. Previously conceived and coded by the Video over IP Group, VIPClient is a multi-threaded program, which relies on lightweight process threads that form a media pipeline. Threads make it possible to efficiently design a complex program with broad functionality such as the VIPClient. A basic decomposition of the functionality of the VIPClient can be achieved by viewing the role of different threads that make up the client. Figure 3 shows a hierarchy of the

VIPClient threads along with the functional pipeline that relates the different processes.

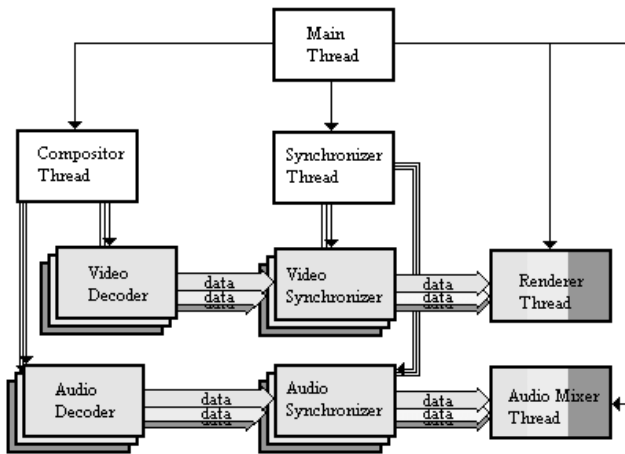


Figure 3. VIPClient Thread Structure

The thread structure shows the main thread as the parent process of the VIPClient. The main thread in turn spawns four threads: compositor, renderer, audio mixer, and synchronizer. The compositor thread's role is to parse a scene description, map it onto a memory tree structure, and initiate proper decoders according to the type of media streams. The decoders are spawned as new threads, and each thread in turn opens a unique RTP session for transport of the media stream. The renderer is responsible for initializing the GUI and handling the final scene assembly on the screen of the client application. It also allows for user interactivity at the GUI level such as moving objects and playing/stopping a scene. The audio mixer thread provides audio-rendering functionality by allowing up to sixteen audio streams to be played simultaneously. The synchronizer thread is responsible for ensuring synchronicity between independent media object streams. The thread spawns a new media synchronizer thread corresponding to each incoming object stream, which utilizes time stamps and the composition memory to determine the instant of consumption/composition.

3. Media Decoders

The current version of the VIPClient model incorporates three decoder modules: an MPEG-4 video decoder, an Advanced Audio Coding (AAC) audio decoder, and a JPEG decoder. The JPEG decoder uses the standard Simple Direct Media Layer (SDL) API library [7]. The video and audio decoders are compliant to the video part [5] and the audio part [6] of the MPEG-4 standard, respectively.

3.1 Video Decoder

MPEG-4 14496-2 is a state-of-the-art, natural-video compression standard. It distinguishes itself from other video standards through efficient rectangular coding, arbitrary-shaped coding, etc. [8]. We made modifications to the MPEG-4 reference code provided by 14496-5 [9] while also implementing real-time, error-resilient decoding. The video frame is segmented into an MPEG-4 video packet [5] of desired size at the macro-block boundary. The channel carrying video is assumed to be error-prone, because the RTP/UDP/IP protocol stack does not guarantee reliable transmission. In addition, the MPEG-4 video standard utilizes both the temporal and spatial redundancies to achieve a higher compression rate. In order to raise the tolerance of the decoder against incoming, incomplete AU's and to guarantee a reliable system, we implement the error-concealment at the video decoder. The decoder predicts the motion vector of the lost packet by using spatially and temporally neighboring MBs. In order to conceal the lost packet, we improve the SL packet format by adding the AU sequence number to every SL, which used to occur only at the first SL-packet of the associating AU. Thus, the smallest data unit that the decoder can consume is an SL-packet instead of an AU in the V-SDM. To facilitate the error concealment at the decoder, we applied error-resilience tools to the compressed video stream at the server side [10]. Unlike audio PCM data, uncompressed video data usually occupies a large amount of memory. For instance, in the quarter-common intermediate format (QCIF) the uncompressed frame size is around 38016 bytes. For common intermediate format (CIF) video, 152064 bytes/frame. As a result, we set the video composition buffer size to only one frame. Because we found that decoding time is trivial compared to rendering time, we buffer just one decoded video frame before composition.

3.2 Audio Decoder

Based on the MPEG-4 audio specifications [6], the AAC perceptual codec is chosen to be integrated in the VIPClient. Figure 4 illustrates the AAC encoding/decoding/rendering process in the VIP Presentation System. The diagram shows how encoded audio frames flow from the server, through the SL layer, the transport layer, the audio decoder, the decoding buffer, and finally ending at the point of audio rendering.

In the actual system, the raw Pulse Code Modulation (PCM) audio stream is encoded before hand and stored as an AAC-encoded file at the server. Each AAC frame, or access unit (AU), is then encapsulated at the SL layer along with a corresponding time stamp used for composition/rendering. The transport layer uses the real-time transport protocol (RTP) to encapsulate each SL packet and to ensure prompt delivery to the VIPClient. At the client, the de-capsulations of RTP and SL packets

result in the availability of the original AAC frame and composition time stamp to the AAC decoder. After the AAC frame is decoded back into the PCM format, it is stored in a memory buffer along with its time stamp. This decoder buffer is available to the compositor to access decoded data ready to be rendered. In VIPClient, the synchronizer thread is responsible for accessing the audio decoder buffer and deciding when to render.

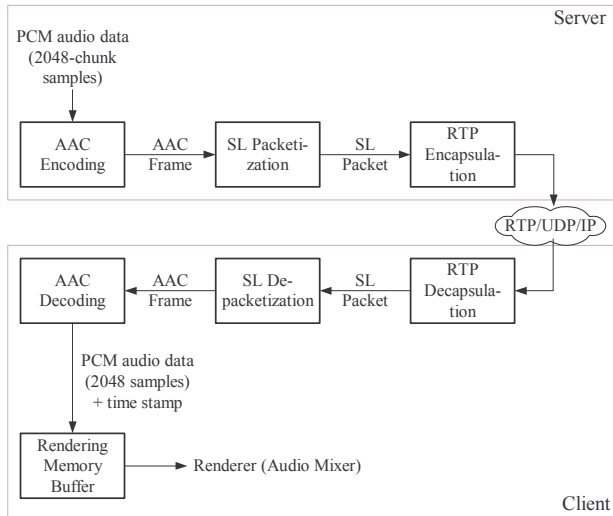


Figure 4. AAC Audio in VIP System

4. Synchronization of Media Object Streams

Crucial to the functionality of a multi-stream, multimedia system is the synchronization of media streams during presentation. Simultaneous handling of plural media streams related in time is the challenge that faces multimedia systems. The multi-stream approach of media transmission is used by the VIP Presentation System in order to achieve flexibility in handling different media and also to decouple the inter-dependence between unique media streams. The traditional single-stream approach involves multiplexing individual media streams in order to create a single stream. Both multi-stream and single-stream approaches have often been compared; Tasaka et al. concludes that the multi-stream approach is slightly superior in synchronization performance [11]. However, the challenge in applying synchronization in a multi-stream environment is still an open topic.

4.1 Synchronization Model

As explained in the literature [12], a media synchronization specification should be comprised of intra-object synchronization and inter-object synchronization. For our system, an object represents a media stream (audio, visual, etc.). In [13], a synchronization model consisting of both *intra-stream* and *inter-stream* synchronization mechanisms is proposed. Whereas *intra-stream* refers to the temporal preservation among media units of the same stream, *inter-stream* involves

synchronizing multiple, individual streams. While each stream utilizes *intra-stream* synchronization, only *slave* streams will require *inter-stream* synchronization mechanisms. This holds because *slave* streams are synchronized to a *master* stream, which acts as a temporal guide for the associated *slave* streams. Normally, a stream with the most sensitivity to synchronization error and a higher sampling rate should be chosen as a master. Because human perception of audio jitter is acutely sensitive [14], audio streams are generally master streams while the corresponding video streams use *inter-stream* synchronization to follow the audio. In [15], the authors extend this model and introduce an adaptive control scheme, which is able to maintain both *intra-stream* and *inter-stream* synchronization while adaptively regulating the equalization delay to compensate for the network delay variation. For *intra-stream* synchronization, the target output time, which represents the instant that a media unit should be rendered, can be manipulated to effectively shrink or extend the rendering interval. *Inter-stream* synchronization constraints are applied among corresponding media units from different streams. In addition, skipping and pausing media units are utilized to maintain synchronicity.

4.2 VIP System Synchronization

Many factors contribute to the asynchrony shown in the final presentation in a realistic networking multimedia system. These factors include variable network delay, fluctuating computing power allocated to the process, etc.. Hence, synchronization control should take place as close as possible to the final presentation point in order to take all potential sources of jitters into account. Since the VIPClient's goal is a cohesive and synchronized presentation of audio/video objects to the user, we choose to implement synchronization algorithms after media decoding but immediately before media rendering. Synchronizing at this point allows for complete flexibility and independence between media streams from transport, reception, decoding, up until the final point of rendering. At that point, *inter-stream* synchronization mechanisms take over to ensure a level of temporal preservation between streams. In VIPClient, each media stream performs *intra-stream* synchronization. A *virtual-time* is created by shrinking or extending the time of rendering a media unit. Skipping and pausing media units are typically the mechanisms to enforce the shrinking or extending of virtual time. However, instead of manipulating a target output time, the VIP system measures the interval of time it takes to render the *previous* media unit, and applies a media skipping/pausing algorithm accordingly. Figure 5 shows the algorithm:

```

Loop:
if (current_time - previous_time) <= (2 *
interval)
    if [interval - (current_time -
previous_time)] > 0
        Pause for [interval - (current_time -
previous_time)].
        previous_time set to current_time.
        Render audio frame.
    else
        previous_time set to current_time.
        Skip rendering audio frame.

```

Figure 5. Intra-stream Synchronization

If the interval of time measured to render the previous media unit is much more than the interval (2x) specified in the time stamp of the unit, the current media unit would not be rendered. Effectively, this skipping “shrinks” the virtual-time scale by not allowing the rendering of the current unit to add to the virtual-time. Otherwise, the system “pauses” for the difference in time between the time stamp interval and actual measured interval (if there is any) before rendering the frame. This delay effectively “extends” the virtual-time scale by allowing an interval of time to pass. This intra-stream synchronization allows for a level of temporal preservation among media units in an environment of jitter. In implementation, exact preservation of time within a stream may not be possible, but small synchronization errors are generally tolerable by the human ear [16].

Inter-stream synchronization is necessary in the VIPClient for lip-synchronization. Similar to other synchronization algorithms, we employ a *master-slave* system for inter-stream synchronization. In our application, a slave stream is a corresponding audio track of a video stream, e.g. a person talking. Within each *slave* stream, inter-stream synchronization follows its intra-stream synchronization. Each media stream, whether it is a slave or master, has a time-base (clock) that is incremented by intervals as each media unit is rendered. In order to achieve a level of inter-stream synchronization, each slave stream periodically compares its own time-base to its master’s time-base. If the time-base of the video slave stream is more than an interval *greater* (2x) than the audio master stream’s time-base, this indicates that the video stream is asynchronously faster than the audio stream. In this case, the slave stream should “pause” for at least the duration of one video frame interval in order for the master stream to catch up. Likewise, if the time-base of the video slave stream is more than an interval *less* than the audio master stream’s time-base, this indicates that the video stream is asynchronously slower than the audio stream. In this case, the slave stream should skip the current media unit in order to catch up to the master stream. Figure 6 shows the algorithm:

```

Loop:
if (slave_tb - master_tb) > (2 * slave_interval)
    Pause for [(slave_tb - master_tb) -
slave_interval].
    Continue with rendering media unit.
else if (master_tb - slave_tb) > (2 *
slave_interval)
    Skip rendering media unit.
else
    Continue with rendering media unit.

```

Figure 6. Inter-stream Synchronization

A benefit of this inter-stream synchronization mechanism is that the video slave stream will always follow the time-base of its audio master stream. In the case where the audio stream is halted or slowed, the video slave will adaptively decrease the rate of media rendering by the guidance of the master time-base. On the other hand, if the video stream is halted or slowed, the audio master will not be affected because there is no inter-stream dependency in a master stream. This behavior is desirable because audio typically has a higher priority in an audio/visual object (e.g. a person speaking), in terms of a human multimedia experience.

Integrating the synchronization algorithms into the VIPClient involves the addition of the synchronizer thread into the client architecture. Figure 3 shows the main synchronizer thread and its individual media synchronizers that are spawned for each media stream. While the main synchronizer thread is used primarily for spawning and destroying media synchronizer threads, it is the individual media synchronizer threads that actually execute intra-stream and inter-stream synchronization control.

5. VIP System Demonstration

In our experimental system, we test the results of streaming up to four media streams. The raw media streams are captured, pre-processed (e.g., video is pre-segmented), and compressed offline. Once the *start* command from the client is received via the control plane [1], the server begins to packetize each stream and incorporates the time stamps, which are calculated based on the specified attributes of that stream (i.e., frame rate, packet length, bit rate, etc). The server then schedules the packets and pumps them out in the appropriate order. The server and the client reside at geographically different locations in terms of IP addresses. The specified port number distinguishes the RTP channels, through which different media streams will be transmitted. Our test run involves two video objects with their corresponding voice audio tracks. When the VIPClient is played, the audio/video objects are rendered in synchronicity,

although there are minor, tolerable skews. We also test the performance of inter-stream synchronization by deliberately congesting one of the audio master streams during transmission. As a result the video slave stream slows down to a halt, and picks back up in synchronicity with the master once the audio stream begins to flow smoothly again. Figure 7 shows the client-rendering window captured from our system in real time. One background JPEG picture is displayed locally. Two arbitrary-shaped video objects are being streamed, decoded and rendered. Simultaneously, the lip-synced voice of the left person is being played. The scene also includes the background music that plays continuously.

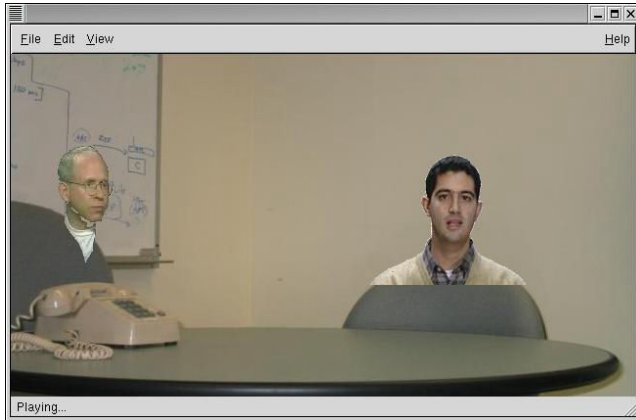


Figure 7. VIPClient Rendering Window

6. Conclusion

In this paper we presented the VIP Presentation System, the implementation of an MPEG-4 based, multi-object, synchronous multimedia presentation system. In order to harness the advantages of an object-based system, we highlight the modifications made to the standard MPEG-4 systems decoder model. We then discuss the architecture of our extensible MPEG-4 client implementation (*VIPClient*) and the incorporated media decoders. We also present the synchronization algorithms utilized to achieve multi-stream synchronicity. The resulting streaming demonstration shows the potential for an interactive distance-learning application [3].

7. Acknowledgements

We are grateful to Haining Liu and Xiaoping Wei for their collaboration on this project.

8. References

[1] M. El Zarki, L. Cheng, H. Liu and X. Wei, "An interactive object based multimedia system for IP networks," *WORDS*, 2003, Mexico.

[2] H. Liu, X. Wei and M. El Zarki, "A transport infrastructure supporting real-time interactive MPEG-4 client-server applications over IP networks," *Proc. IWDC'01*, Italy.

[3] X. Wei, H. Liu and M. El Zarki, "Enabling active engagement in E-tutelage using interactive multimedia systems," *ITCC*, 2003, USA.

[4] Information technology – Coding of audio-visual objects – Part 1: System, ISO/IEC 14496-1: Second Edition, 2001.

[5] Information technology – Coding of audio-visual objects – Part 2: Visual, ISO/IEC 14496-2: 1999/FDAM 1: 1999(E).

[6] Information technology – Coding of audio-visual objects – Part 3: Audio, ISO/IEC 14496-3: 1999/FDAM 1: 1999(E).

[7] Simple DirectMedia Layer. <http://www.libsdl.org/>

[8] L. Cheng, M. El Zarki, "The analysis of MPEG-4 core profile and its system design," *MTAC'01*, U.S.A, 2001.

[9] Information technology – Coding of audio-visual objects – Part 5, Video Reference Software, ISO/IEC 14496-2, Version: *Microsoft-FPDAM1-1.0-000403*.

[10] L. Cheng and M. El Zarki, "An adaptive error resilient video encoder," *Visual Comm. and Image Processing, 2003*, Switzerland, to appear.

[11] S. Tasaka and Y. Ishibashi, "Single-stream versus multi-stream for live media synchronization," *IEEE ICC'98*, pp. 470-176, June 1998.

[12] G. Blakowski and R. Steinmetz, "A media synchronization survey: Reference model, specification, and case studies," *IEEE J. Sel. Areas in Commun.*, vol. 14, no.1, Jan. 1996.

[13] Y. Ishibashi and S. Tasaka, "A synchronization mechanism for continuous media in multimedia communications," *Proc. IEEE INFOCOM'95*, April 1995.

[14] R. Steinmetz, "Human perception of jitter and media synchronization," *IEEE J. Sel. Areas in Commun.*, vol. 14, no.1, Jan. 1996.

[15] H. Liu and M. El Zarki, "A synchronization control scheme for real-time streaming multimedia applications," *Proc. of Packet Video 2003*, France.

[16] Y. Ishibashi, S. Tasaka, and Y. Tachibana, "Media synchronization and causality control for distributed multimedia applications," *IEICE Trans. Commun.*, vol. E84-B, no.3, March 2001.