[MESt93] *Thomas Meyer, Wolfgang Effelsberg, Ralf Steinmetz;* **A Taxonomy on Multimedia Synchronization;** 4th IEEE Workshop on Future Trends of Distributed Computing Systems, S.97-103, Lissabon, Portugal, 22.-24. September 1993.

MESt93

A Taxonomy on Multimedia Synchronization

Thomas Meyer, Wolfgang Effelsberg

University of Mannheim
P.O.Box 10 34 62, 68131 Mannheim, Germany

Ralf Steinmetz

IBM European Networking Center
P.O.Box 10 30 68, 69020 Heidelberg, Germany

Abstract

Multimedia systems allow the integration of data streams of different types, including continuous media (audio and video) and discrete media (text, data, still images). The information contained in these data streams is often interrelated, and multimedia systems must guarantee to maintain such relationships as the streams are stored, transmitted and presented to the user. This is usually called "Multimedia Synchronization". Unfortunately different authors use the term with different and often confusing meanings. This article proposes a taxonomy of multimedia synchronization, introducing three layers of abstraction: a Media Layer dealing with intra-stream synchronization, a Stream Layer specifying inter-stream synchronization, and an Object Layer defining the relationships between multimedia objects in the style of an authoring language. An example illustrates the taxonomy throughout this paper.

1: Introduction

Synchronization is essential for the integration of continuous and discrete media in local as well as distributed multimedia systems [36]. In these systems, units of information are coded in various types of data streams (*media*), including continuous media such as audio and video.

In some cases each stream is self-contained and has no temporal restrictions, but in many other cases there are semantic relationships between different streams. For example, a continuous stream generated by a source is only considered to be correct at the sink if the data is reproduced at a specific rate (e.g. in real-time video); the stream must be synchronized with a real-time clock. In other cases two streams have a strong temporal interrelationship (e.g. lip synchronization in audio/video systems). A further example may be where a user wants to "manually" define and edit temporal constraints between multimedia objects presented to the user (e.g. voice annotation or captions for a slide show) [6], [24]. Very often, combinations of these types of synchronization can occur in practice.

In order to understand the various requirements for multimedia synchronization mechanisms and compare solutions for their implementation we propose a classification scheme with three layers of abstraction. We distinguish between the definition of these temporal relationships and the actual provision of such relationships during the presentation of the data streams.

In the current literature, there are a plethora of techniques presented for achieving multimedia synchronization, as well as some approaches for specification at different levels of abstraction. An overall classification was introduced by Little and Ghafoor [25]. They identified a physical, system, and a human level, but without giving a detailed description or classification criteria. Other classification schemes distinguish between intra-stream (fine-grain) synchronization and inter-stream (coarse-grain) synchronization or between live and synthetic synchronization [25], [39]. Live synchronization deals with the presentation of information in the same temporal relationship as it was originally collected. For synthetic synchronization various pieces of information must be synchronized in time. Lip-synchronization deals with the synchronized playback of audio and video and is a special case of inter-stream synchronization.

These classification schemes seem to be orthogonal, and each one of them only captures one specific aspect. In [7] the Firefly system was introduced which provides mechanisms for automatically producing temporal layouts for multimedia documents, according to the spatial layout in document systems (e.g. ODA). The Orchestration Service [8] provides a 'stream-oriented' interface for synchronized playout of continuous media in a distributed environment. These systems deal with multimedia synchronization, but support abstractions from different perspectives. We felt that a unifying classification scheme and taxonomy might be helpful in this confusing world.

This paper is organized as follows. In Section 2 we introduce our taxonomy and define the three layers in detail; a short programming example illustrates each of the layers. Section 3 shows how existing synchronization services can easily be classified using our scheme. Section 4 presents an outlook at future trends, and Section 5 concludes the paper.

2: Classification scheme

We consider multimedia synchronization at three layers of abstraction as shown in Fig. 1. Each layer implements synchronization mechanisms which are provided by an appropriate interface. These interfaces can be used to specify and/or enforce the temporal relationships. Each interface defines an interface language, offering the user a means to define his/her requirements. Each interface can be used by an application directly, or by the next higher layer to implement an interface.

A following analogy is taken from the three layered classifiaction of programming languages. The lowest level offered to a programmer is the assembler language; only in rare cases the program has to be constructed at this level, where is a direct mapping of assembler state-

ments to machine instructions. At the intermediate level we find imperative programming languages such as C or Pascal. In many cases programmers (and in particular systems programmers) use these languages to solve their problems. These languages are compiled or interpreted to an assembler level. An example of the highest level would be fifth-generation database query languages, such as SQL, or object-oriented languages, such as Smalltalk. The programmer (and in particular the application programmer) can use these languages to specify in a descriptive way what the problem is; the system then translates them into lower-level language statements to be executed by the machine.

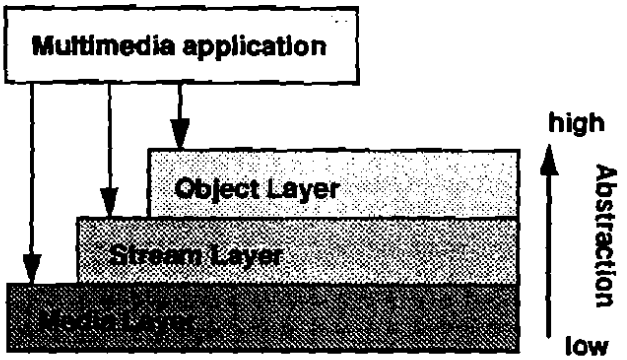

Fig. 1: Classification scheme

For each layer typical objects and operations on these objects are described in the following. The semantics of the objects and the operations are the main criteria for assigning them to one of the layers.

A detailed programming example derived from a real interface provided by a real product, prototype, or standard demonstrates how synchronization can be achieved through this layer. The scenario for the programming example is to display subtitles at predefined times during the playout of a digital movie.

2.1: Media layer

We assume that all media are digital and represented as data in pieces of memory in the system. Each media stream can be segmented into a sequence of *logical data units* (LDUs) depending on the granularity of the medium [39]. The syntax and the encoding of the individual LDUs is determined by the multimedia devices. In general these devices provide interfaces specific to the capabilities of the device.

At the **Media Layer** an application operates on a single continuous media stream, which is treated as a sequence of LDUs. We define the structure of an LDU as follows:

```
struct {
   long time;
   char *buffer;
} LDU;
```

The attribute *buffer* refers to a pre-allocated area of memory where data will be stored. This data for example can be captured from a video device. The processing of the LDUs is limited to media-specific time constraints (e.g. 25 video frames/second with one frame being a LDU). Therefore, the attribute *time* defines the appropriate playout time for this LDU. References to absolute or relative time (e.g. Logical Time System [4]) can be used. For example the frame number of the corresponding frame.

The concept of the LDUs is useful to encapsulate the device-specific implementations of a unified interface that provides operations like *read(devicehandle, LDU)* and *write(devicehandle, LDU)* to process LDUs. Systems like, ActionMedia/II's audio-video kernel [16], or the audio device in the SunSPARC station [40] provide corresponding interfaces.

To set up a continuous media stream an application often executes a process for each stream in a fashion similar to the following example:

```
window = open("Videodevice");
movie = open("File");
while (not eof(movie)) {
   read(movie, &ldu);
   if (ldu.time == 20)
      print("Subtitle 1");
   else if (ldu.time == 26)
      print("Subtitle 2");
   write(window, ldu);}
close(window);
close(movie);
```

The process reads and writes LDUs in a loop for as long as data is available. Synchronous playout of a subtitle is achieved by testing the attribute *time* of the LDUs for the necessary value.

At this layer an application is responsible for the intra-stream synchronization by using flow-control mechanisms between a producing and a consuming device [33]. If multiple streams are running in parallel, then the sharing of resources may effect their real-time requirements. Usually a resource reservation and management scheme allows for the appropriate *quality of service* to guarantee intra-stream synchronization [42]. The operating system(s) schedule(s) the corresponding process in real-time [28], in distributed systems the networking components are taken into account [3],[10]. Another approach introduced by Little calculates a 'playout schedule' for the individual LDUs in respect to the bandwidth limitations [26]. In the special case of lip synchronization the inter-stream synchronization can be provided easily, where simultaneous audio and video frames are interleaved within the same LDU (e.g., ActionMedia-II's audio-video support system [16] and the MPEG data stream [20]). Finally, the synchronous playout of discrete media is a task to be performed by the application.

2.2: Stream layer

The **Stream Layer** allows the application to operate on continuous media *streams* as well as on *groups* of media streams. In a group all streams are presented in parallel by using mechanisms for inter-stream synchronization. Continuous media is seen as a data flow with implicit time constraints, individual LDUs are not visible. These so called streams are executed in a real-time environment (RTE) where all processing is constrained by

well defined time specifications [12]. On the other hand the applications themselves are executed in a non real-time environment (NRTE) where the processing of events is controlled by the operating system scheduling policies. Typical operations invoked by an application to manage streams and groups from the NRTE are: *start(stream), stop(stream), creategroup(listofstreams), start(group),* and *stop(group).* The interaction with discrete media is performed via the attachment of events to the continuous media streams (e.g. *setcuepoint(stream/group, at, event)*). Such an event is sent to the application whenever the stream reaches the specified point during playback. Once a event passes from the RTE to the NRTE it's time values become meaningless, i.e. discrete media have explicit time constraints. At this layer the application is furthermore in charge of any discrete media processing. This leads to different application interfaces for continuous and discrete media.

The *Sync/Stream Subsystem* of IBM's Multimedia Presentation Manager for OS/2 provides a set of services which can be used to implement data streaming and synchronization. This subsystem, which can be understood as the RTE, is comprised of the *Sync/Stream Manger* (SSM) and several *stream handlers* [17]. Stream handlers are responsible for controlling the continuous data flow in real-time. The SSM provides a resource management and controls the registration and activities of all stream handlers.

The following programming example uses the string command interface provided by the MMPM.

```
open digitalvideo alias ex
load ex video.avs
setcuepoint ex at 20 return 1
setcuepoint ex at 26 return 2
setcuepoint ex on
play ex

switch readevent() {
  case 1: display("Subtitle 1")
  case 2: display("Subtitle 2")
  }
```

In MMPM/2 inter-stream synchronization for synchronized playback of multiple stream within a group is achieved by a master/slave algorithm, where one stream (the *master*) controls the behavior of one or more subordinate streams (the *slaves*). The skip/pause algorithm introduced in [4] gives a detailed discussion in the implementation of such an behavior. This synchronization mechanism in ACME [2] as well as the Orchestration Service [8] (see programming example) support stream layer abstractions for distributed multimedia systems.

```
Orch.request(&sid, listofvcids);
Orch.Prime.request(sid);
Orch.Eventrequest(sid, vcid, event);
   ...
Orch.Stop(sid);
```

The Stream Layer abstraction was derived from the abstraction normally provided by the integration of analog media in the computer system. In the Muse and Pygmalion system of MIT's Project Athena [14] or in the DiME [39] system continuous media was routed over separated channels through the computer. The connected devices could be controlled by sending commands via the RS-232C interface to start and stop the media streams. In such systems live synchronization between various CM stream is directly performed by the dedicated processing devices.

2.3: Object layer

The differences between discrete and continuous media are hidden by a common application interface provided by the **Object layer.** This layer allows for simpler and exact specification of playout sequences, where each media object relates to a time axis and defines a sequence of events. From our understanding the abstractions are similar to the 'object model' presented in [35].

Authoring systems often use a script language for sequencing and composition of inter-object relationships (Apple's Lingo [9] and IBM's Audio Visual Authoring Language [15]). They have all in common that an author has to specify synchronized actions in relationship to a global time. As the following sample demonstrates, these languages are imperative, which means that the author has to be aware of the synthetic synchronization. The complexity increases dramatically, if more than one continuous media stream is simultaneously played back.

```
DPLAY "movie.avs"
WAIT 20
SAY "Subtitle 1"
WAIT 6
SAY "Subtitle 2"
WAIT 'on video termination'
```

Multimedia documents are declarative, because they support an author in a way where he/she simply describes the animation without knowledge of the synchronization mechanisms. As building blocks, media objects (content) allow for construction of complex multimedia objects (content including presentation attributes). Templates incorporate the definition of fuzzy time constraints between the multimedia objects according to the application and users' needs. The application operates on the composition of presentation objects in terms of assembling playout sequences, the definition of the exact temporal relationship has not to be defined explicitly. Alternative actions and tolerances depending on the quality, time, and space are key features of this environment. Apart from the composition, all elements at the application interface including the operations for *shrinking* and *stretching* of objects in time are visible [7]. These operations are used during a layout process where the templates are formatted according to the applications' requirements and the available resources at the executing system. HyTime [19] and HyperODA [5] allow, to some extent, for such a flexible abstract specification.

For the presentation of a multimedia document an appropriate runtime environment is necessary [41]. As introduced by Markey [27] one solution, to execute a HyTime document, could be to use the forthcoming MHEG[1] standard [20], [30].

1. The acronym MHEG is used for the Multimedia Hypermedia Expert Group.

The scope of the first part of this standard is the coded representation of final form multimedia and hypermedia information objects. The so called 'MHEG objects' deal with the coded representation of the multimedia document and are conceived for the interchange between applications. An MHEG engine parses, encodes, interchanges, decodes, and interprets MHEG objects. In an application the objects are in certain preparation states (*not ready, ready*) or in running states: (*running, not running*). The MHEG engine evaluates these statuses and performs operations (*actions*) like *prepare, run, stop,* or *destroy* on these objects.

MHEG *links* are used to specify spatial temporal and conditional relations between MHEG objects. An MHEG link is directional and connects one *source object* with one or more *linked objects* (Fig.2). The MHEG engine triggers links by the modification of a status evaluation of the source object.

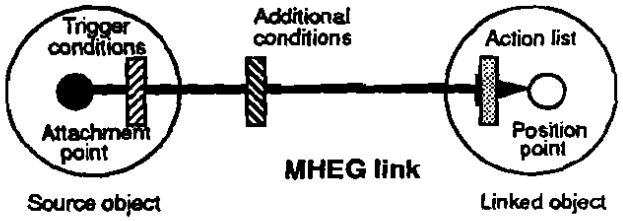

Fig.2: Definition of an MHEG link

In the following we give an rudimentary example of how our scenario might be coded in the upcomming MHEG standard.

```
Composite {
start-up link
viewer start-up
viewer list
    Viewer1: reference to Component1
    Viewer2: reference to Component2
    Viewer3: reference to Component3
Component1
    reference to content "movie.avs"
Component2
    reference to content "Subtitle1"
Component3
    reference to content "Subtitle2"
Link1
    "when timestone status of Viewer1
    becomes 26 then start Viewer2"
Link2
    "when timestone status of Viewer1
    becomes 26 then start Viewer3"
}
```

The *composite* object acts as a 'container' to group a set of MHEG objects. The *viewer* object is a virtual view of a referenced *component* object. Finally, the *timestone* is an identified marker for a temporal position. The identifier is loaded by the MHEG engine into the *timestone status* whenever the presentation of a component meets the marker. *Link1* and *Link2* are triggered by the timestone status to start the viewer as required by the application.

3: Use of the layered classification

The classification scheme has demonstrated that several layers of abstraction can be identified in the existing approaches of multimedia synchronization mechanisms. Fig.3 illustrates how the different kinds of synchronization relate to our classification.

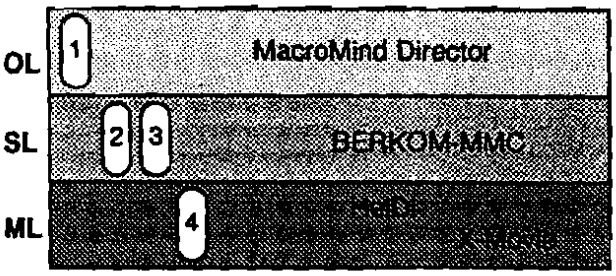

Fig.3 Taxonomy of available abstractions

The intra-stream synchronization at the Media Layer supports basic mechanisms for continuous media streams, which means that applications using this abstractions have to implement many of the synchronization mechanisms themselves (e.g. Heidelberg Distribution Service [13], [39] and the X-Movie system [21]). Synthetic synchronization, which is located in the Object Layer, hides most of the synchronization mechanisms from the application . But, these powerful abstractions are not applicable for all kind of applications.

In audio/video conferencing systems (e.g. BERKOM Multimedia Collaboration Service (BERKOM-MMC) [1]) the application enables the user to setup connections between one or more participants, intra as well as inter-stream synchronization is needed, because the audio and video data has to be collected in real-time from mircrophones and cameras. Therefore, the live synchronization as provided through the mechanisms at the Stream Layer is the preferred method. These applications normally do not deal with synthetic synchronization.

On the other hand the definition and delivery of complex relationships in time [11], [24] and space is a task for authoring applications. Therefore, authoring and hypermedia systems (e.g. MacroMind Director [9]) use the synthetic synchronization mechanisms in the Object Layer.

Finally, joint viewing and joint-editing applications need additional synchronization mechanisms, which are not in the scope of our understanding of temporal relationships as introduced at section 1 in this paper. These mechanisms have to guarantee the integrity of data structures and may be achieved by ordering events as discussed in [22], [34], and [23]. In this other context, synchronization applies to the notion of a well defined interaction between various users and applications.

4: Future trends

Synchronization has been a key issue in multimedia systems since at least five years. In this time span some major advances in research and development concerning the media relationships have taken place, as also pointed out through this paper all operational systems include component(s) and/or algorithm(s) to assure some kind of temporal relationships. Nevertheless, we still encounter work to be done which we understand as the natural evolution from the already achieved results and actual work in progress towards the required and open issues, we want this to be known as the *future trends* in multimedia synchronization.

4.1: Abstraction

The major issue of this papers relates to the taxonomy of synchronization techniques developed in order to understand different approaches, to identify more easily their mutual relationships and to clarify the notion of abstraction in this area. From the evolutionary point of view, we encounter a first trend related to the increase of abstraction. While intra-stream synchronization between the individual LDUs was the first issue to take into account, afterwards we addressed the key issue of inter-stream synchronization at the Stream Layer. Today, considerable work in the research community is devoted to appropriated class hierarchies and multimedia objects [41], [38]. The future trend covers aspects of platform independency leading to this Object Layer in our taxonomy. It is still a matter of research to find out the most suitable approach(es) at this level. This trend involves the programming of applications making use of the before mentioned abstractions as shown in section 3.

4.2: Real-time requirements

Recent exhaustive human perception experiments revealed a not well studied peculiarity of multimedia synchronization, the detailed notion of defined skew values as quality of service (QoS) parameters [37]. In most of the existing system, synchronization is either achieved by an *as good as possible* or an *be exact in-sync* methapher. As the above mentioned experiments showed, in this context the notion of correctness from the human perception point of view must be redefined. As an example take lip synchronization which applies between audio and video: An artificially introduced 40 ms skew value between audio and video was shown to have the same quality from the human perception point of view as no skew at all. Therefore it turns out, that between the various type of media certain skew values expressed as a QoS parameter can be easily tolerated. The trend goes from hard to weak real-time requirements of the skew. While the QoS parameters have often been *dictated* they will more and more often been negotiated between the involved components.

A similar trend can be observed in the resource reservation area: We observe a move from very telephone call oriented approaches known as *guaranteed* towards the *best effort* solutions. This allows for a higher continuous media workload on the basis of the same resources.

4.3: Media streams

The most often mentioned media streams to be synchronized are audio and video. Early implementation as well as most of the CD-ROM based approaches *multiplex* these data streams, it is also known as *file interleaving* and *interleaved files*. Often audio and video are also transmitted over the same communication channel and treated as a single connection. This approach alleviates the end-systems to take care of any eventually occurring synchronization error. But, it does not allow to save system resources and it does not provide the ultimate flexibility to control each data stream individually. Therefore, individual streams have become the atomic entity to be able to control at the Stream Layer.

Apart from the multiplexing issue, most of the systems focused on either synchronized audio and video, or combinations of continuous media data like audio with some discrete media. This latter is usually performed by issuing some user defined events at a certain point in time with respect to the related continuous media, i.e., the continuous media stream serves as master. In more complex scenarios more kinds of media streams exist, and they have to be supported by the provided mechanisms. The evolution goes from the synchronization of some media data (often just two kinds of media) towards any kind and amount of media data.

4.4: Media data

As another fundamental issue we must consider the trend towards an integration of synchronization into distributed multimedia systems. While in the very beginning each application had by itself to take care of synchronization, this burden has been alleviated by a system based support which is often part of some so called *extensions*. This evolution took place because very dissimilar applications had to perform similar synchronization tasks and therefore must include a similar processing. At the time being still sophisticated synchronization processing remains at the application level. We encounter a well defined trend towards the incorporation of all these feature into the system rather than being part of the applications.

Today's multimedia systems and system environments like Apple QuickTime [9], Microsoft Multimedia Extensions [29], and IBM Multimedia Presentation Manager/2 [17] contain synchronization mechanisms for being applied in the local domain. First networked systems like the IBM Ultimedia Server cover some synchronization issues in a distributed environment. Research project more often deal with networked than with local solutions only. The direction shows from local towards the distributed approaches.

Synchronization issues covering various computers have most often been introduced in the context communication systems or sometimes together with database systems. This is too specialized because the same algorithms and features are used in the local as well as in the networked environment: The notion of a *stream* with the source and the sinks applies, e.g., to a local as well as a distributed scenario. We encompass the trend from the

specialized networked or database driven approach to a generic system support as extension to the operating system or even as integral part of it.

5: Conclusion

We have introduced a layered classification scheme for multimedia synchronization. The three layers are the Media Layer for intra-stream synchronization, the Stream Layer for inter-stream synchronization and the Object Layer for authoring complex multi-stream multimedia applications. At each layer objects and operations are defined. Each layer can be accessed directly by the application or indirectly through higher layers.

The criteria for our classification were derived from the analysis of present work in standards (e.g., MHEG [20], HyTime [19]), prototypes (e.g. ACME [4], Orchestration Service [8]), and products (e.g. Multimedia Extensions for Windows [29], MMPM/2 [17]). We gave examples how existing multimedia systems and their interfaces can be easily classified using our scheme, and at what level of abstraction the upcoming standards can be placed.

We are now in a position to extent our work to propose precise application interfaces with generic features at each layer, and to define the mapping of objects and operations at layer n to the interface of layer n-1.

References

[1] Michael Altenhofer, Jürgen Dittrich, Rainer Hammerschmidt, Ralf G. Herrtwich, Thomas Käppner, Carsten Kruschel, Ansgar Kueckes, Thomas Steinig: *The BERKOM Multimedia Collaboration Service*, Proceedings of the 1st ACM International Conference on Multimedia, Anaheim, USA (CA), 1-6 Aug., 1993.

[2] David Anderson, Pamela Chan: *Toolkit Support for Multiuser Audio/Video Applications*, Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Nov. 18-19, 1991, Lecture Notes in Computer Science, Springer Verlag, vol.614, 1992, pp.230-241.

[3] David P. Anderson, R. G. Herrtwich, C. Schaefer (1990) SRP: A Resource Reservation Protocol for Guaranteed-Performance Communication in the Internet. Report No. TR-90-006, International Computer Science Institute, Berkeley, February 1990.

[4] David P. Anderson, George Homsy: *Synchronization Policies and Mechanisms in a Continuous Media I/O Server*, International Computer Science Institute, Technical Report no. 91-003, Berkeley, 1991.

[5] Wolfgang Appelt: *HyperODA*, ISO/IEC/JTC1/SC18/WG3.

[6] Gerold Blakowski, Huebel, Langrehr: *Tools for Specifying and Executing Synchronized Multimedia Presentations*, 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Nov. 18-19, 1991.

[7] M. Cecelia Buchanan, Polle T. Zellweger: *Automatic Temporal Layout Mechanisms*, Proceedings of the 1st ACM International Conference on Multimedia, Anaheim, USA (CA), 1-6 Aug., 1993.

[8] Geoff Coulson, Francisco García, Andrew Campbell, David Hutchison; Orchestration Services for Distributed Multimedia Synchronisation; Proceedings of the 4th IFIP International Conference on High Performance Networking (hpn), Liège, Belgium, 14-18 Dec. 1992.

[9] David L. Drucker, Michael D. Murie: *QuickTime Handbook*, published by HAYDEN, ISBN: 0-672-48533-0, 1992.

[10] Domenico Ferrari: *Design and Application of a Delay Jitter Control Scheme for Packet-Switching Internetworks*, Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Nov. 18-19, 1991, Lecture Notes in Computer Science, Springer Verlag, vol.614, 1992, pp.72-83.

[11] C. L.Hamblin: *Instants and Intervals*, Proceedings of the 1st Conference of the International Society for the Study of Time, Editor J.T.Fraser et.al., Springer Verlag, 1972, pp.324-331.

[12] Ralf. G. Herrtwich: *An Architecture for Multimedia Data Stream Handling and Its Implication for Multimedia Transport Service Interfaces*, 3rd IEEE Workshop on Future Trends of Distributed Computing Systems, Taipei, April 1992.

[13] Ralf G. Herrtwich: *The HeiProjects: Support for Distributed Multimedia Applications*, IBM Technical Report 43.9206, IBM European Networking Center, Heidelberg, Germany, 1992.

[14] Matthew E. Hodges, Russel M. Sasnett, Mark S. Ackermann: *A Construction Set for Multimedia Applications*, IEEE Software Magazine, Jan. 89, pp.37-43.

[15] IBM Cooperation: *Audio Visual Connection User's Guide and Authoring Language Reference*, Version 1.05; IBM Form S15F-7134-02, August 1990.

[16] IBM Cooperation: *ActionMedia/II - Technical Reference*, Version 1.0, 1992.

[17] IBM Cooperation: *IBM Multimedia Presentation Manager Programming Reference and Programming Guide 1.0*, IBM Form: S41G-2919-00 and S41G-2920-00, Mar. 1992.

[18] ISO/IEC JTC1/SC29: *Information Technology - Coding of Moving Pictures and Associated Audio for Digital Storage Media up to about 1.5 Mbit/s*, Draft International Standard ISO/IEC DIS 11172, 1992.

[19] ISO/IEC JTC1/SC18; International Standard: Information Technology - Hypermedia/Time-based Structuring Language (HyTime); ISO/IEC CD 10744, April 1991.

[20] ISO/IEC JTC1/SC29/WG12: *Information Technology - Coded Representation of Multimedia and Hypermedia Information Objects - Part 1: ASN.1 notations*, MHEG Working Document, Version 7, 19. Oct. 1992.

[21] Bernd Lamparter, Wolfgang Effelsberg: *X-Movie: Transmission and Presentation of Digital Movie under X*, Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Nov. 18-19, 1991, Lecture Notes in Computer Science, Springer Verlag, vol.614, 1992, pp.72-83.

[22] Lesli Lamport: *Time, Clocks, and the Ordering of Events in Distributed Systems*, Communications of the ACM, vol.21, July 1978.

[23] L. Li, L. Lamont, A. Karmouch, N. Georganas: *A Distributed Synchronization Control Scheme in A Group-oriented Conferencing System*, Proceedings of the 2nd International COnference on Broadband Islands, Athens, Greece, 15-16 Jun., 1993, pp.115-124.

[24] Thomas D.C. Little, A. Ghafoor: *Synchronization and Storage Models for Multimedia Objects*, IEEE Journal on Selected Areas in Communication, vol.8, no.3, Apr. 1990, pp. 413-427.

[25] Thomas D.C. Little, Arif Ghafoor: *Network Considerations for Distributed Multimedia Objects Composition and Communication*, IEEE Network Magazine, vol.4 no.6, Nov. 1990, pp. 32-49.

[26] Thomas D.C. Little: *Protocols for Bandwidth-Constrained Multimedia-Traffic*, Proceedings of the 4th IEEE ComSoc International Workshop on Multimedia Communications, Monterey (CA), USA, pp.150-159, 1-4 April 1992.

[27] Brian D. Markey: *Emerging Hypermedia Standards: Hypermedia Marketplace Prepares for HyTime and MHEG*, (U.S.) Assistant Secretary of Defense (Production and Logistics), Washington, DC, PB92-120328, 1991.

[28] Andreas Mauthe, Werner Schulz, Ralf Steinmetz: Inside the Heidelberg Multimedia Operating System Support: Real-Time Processing of Continuous Media in OS/2, IBM Deutschland GmbH, Technical Report no. 43.9214, 1992.

[29] Microsoft Corporation: *Microsoft Windows Multimedia Authoring and Tools Guide*, Microsoft Windows, Programmer's Reference Library, 1991.

[30] Roger Price: *MHEG: An Introduction to the future International Standard for Hypermedia Interchange*, Proceedings of the 1st ACM International Conference on Multimedia, Anaheim, USA (CA), 1-6 Aug., 1993.

[31] Kaliappa Ravindran: *Real-time Synchronisation of Multimedia Data Streams in High Speed Packet Switching Networks*, Proceedings of the Worshop on Multimedia Information Systems (MMIS'92), IEEE ComSoc, Tempe (AZ), USA, February 1992.

[32] P. Venkat Rangan, Srinivas Ramanathan, Harrick M. Vin, Thomas Käppner: *Media Synchronization in Distributed Multimedia File Systems*, Proceedings of the 4th IEEE ComSoc International Workshop on Multimedia Communications, Monterey (CA), USA, 1-4 April 1992, pp.315-324.

[33] Srinivas Ramanathan, Venkat Rangan: *Feedback techniques for intra-media continuity and inter-media synchronization in distributed multimedia systems*, Computer Journal, 1993, vol.36, no.1, pp.19-31.

[34] Kurt Rothermel, Gabriel Dermler: *Synchronization in Joint-Viewing Environments*, Proceedings of the 3rd International Workshop on Network and Operating System Support for Digital Audio and Video, San Diego (CA), USA, Nov. 12-13, 1992, pp.97-109.

[35] Ralf Steinmetz: *Synchronization Properties in Multimedia Systems*, IEEE Journal on Selected Areas in Communication, vol. 8, no. 3, April 1990, pp. 401-412.

[36] Ralf Steinmetz, *Multimedia-Technology: Fundamentals*, in German, Springer-Verlag, September 1993.

[37] Ralf Steinmetz, Clemens Engler: *Human Perception of Media Synchronization*, IBM Technical Report 43.9310, IBM European Networking Center, Heidelberg, Germany, 1993.

[38] Ralf Steinmetz, Christian Fritzsche: *Abstractions for Continuous-Media Programming*, Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Nov. 18-19, 1991, Lecture Notes in Computer Science, Springer Verlag, vol.614, 1992, pp.283-296; and in Computer Communications, vol. 15, no. 4, July/August 1992.

[39] Ralf Steinmetz, Thomas Meyer: *Multimedia Synchronization Techniques: Experiences Based on Different System Structures*, Proceedings of the 4th IEEE ComSoc International Workshop on Multimedia Communications, Monterey (CA), USA, 1-4 April 1992, pp.306-314.

[40] Robert Terek, Joseph Pasquale: *Experiences with Audio Conferencing Using the X Window System, UNIX, and TCP/IP*, Proceedings of the USENIX-Conference about Multimedia - For Now and The Future, Berkley, USA (CA), 10-14 Jun., 1991, pp.405-418.

[41] Dennis Tsichritzis, Simon Gibbs, Laurant Dami: *Active Media*, In Dennis Tsichritzis (Ed.), Object Composition, Université de Genève, Centre Universitaire d'Informatique, Genève, Switzerland, June, 1991, pp.115-132.

[42] Carsten Vogt, Ralf Herrtwich, Ramseh Nagarajan: HeiRAT: *The Heidelberg Resource Administration Technique Design Philosophie and Goals*, IBM Technical Report 43.9307, IBM European Networking Center, Heidelberg, Germany, 1992.