An Execution Architecture for Synchronized Multimedia Presentations

Franck Rousseau[1]* and Andrzej Duda[2]

[1] Open Group Research Institute
[2] LSR-IMAG
Grenoble, France

Abstract. We have defined an execution architecture for playing back synchronized multimedia documents. We suppose that such documents are specified by means of several abstractions including hypertime links, time bases, and dynamic layout. Our architecture is based on three concepts: *synchronization events*, *synchronization managers*, and *synchronizable media objects*. It supports the notion of *elastic* time that adapts to available resources. We have prototyped the architecture using Java and experimented with playback of simple synchronized presentations.

1 Introduction

Integration of different types of media such as text, images, audio and video clips has become a hot topic of research and many advances have been made recently: products such as QuickTime, Director, Shockwave, and many others support time-based digital video presentations and CD-ROM multimedia documents or games provide interactive access to multimedia information. However, we still need new ways for creating synchronized documents from available multimedia contents present in the network. Such documents mix several types of media and have an intrinsic temporal behavior.

Handling multiple media types has become common because of increasing processing power and communication bandwidth, along with advances in software techniques such as compression. The advent of Java, a new platform independent and secure programming language introduced by Sun Microsystems in 1995, has added a new dimension to multimedia. It allows execution of code downloaded form the network and supports operations on basic multimedia data types such as audio and images. Several audio and video conferencing tools can deliver multimedia contents to standard computers even in the currently clogged Internet: vic, vat, Rendez-Vous, and FreePhone. A lot of work has been done to enable transport of continuous data in real time over the network — RTP/RTCP, RSVP.

However, these advances in hardware, software and networks are not followed by real integration of multimedia at the level of documents. There are two reasons for this situation. First, there is no widely accepted standard for specifying

* Franck Rousseau has been supported by Bull SA.

synchronized multimedia presentations that have inherent temporal behavior. Such presentations may include continuous multimedia data (audio and video clips) either stored on network servers or coming from live-data sources. Second, even if such a standard exists, we need a flexible execution architecture to playback synchronized presentations on different platforms (something comparable to Mosaic that made WWW widely known).

The WWW Consortium has initiated an activity to define a new standard for synchronized multimedia documents: SMIL [1]. SMIL adds to HTML some features related to time behavior: components of a SMIL document may be continuous multimedia. However, SMIL has some drawbacks:

– Temporal composition is based on a hybrid approach that mixes two different abstractions: intervals and time-points. A scenario is represented as a tree in which temporal operators *par* and *seq* are nodes and intervals corresponding to multimedia objects are leaves. To make the *par* operator unambiguous, SMIL defines additional attributes that refer to time-points. The attributes specify the exact semantics of the *par* operator. As a result, a given scenario can be expressed in several different ways and specification may become confusing.
– SMIL provides an optional *lipsync* attribute that applies to a group of objects enclosed with the *par* operator. It is not sufficient to specify desired close intermedia synchronization between multimedia objects. Moreover, the exact behavior of the attribute is not defined and left to implementation.
– The layout of a document is based on a *tuner* element that uses a simplified version of CSS positioning. As a consequence, the layout is static — it cannot change in time. The static layout limits authoring possibilities, because temporal objects may require placement that varies in time.

To overcome these problems, we have proposed temporal extensions to HTML based on three concepts [2]:

– *hypertime links* for temporal composition,
– common *time bases* for close lip-sync synchronization between media objects,
– *dynamic layout* that can be seen as an extension of media objects.

A hypertime link is similar to the standard WWW hypertext link, however, it has an explicit temporal semantics: it relates two media samples (e.g. video frames) and assigns time instants to the samples. A time base is a means for specifying close intermedia synchronization. It can be thought of as a virtual time space in which media objects "live". It defines a common time coordinate space for objects that are synchronization dependent. We extend the notion of a document layout: it may be specified as an object having temporal behavior in a similar way to media objects. This feature allows seamless integration of spatial and temporal dimensions of a multimedia document.

A synchronized multimedia document requires an execution support for scheduling, synchronizing, coordinating, and controlling different media objects and actions according to a temporal specification. Such a support should be flexible enough to adapt to different platforms, to adjust quality of service to available resources, and to manage scalable content. We have defined a flexible execution architecture based on three concepts: *synchronization events*, *synchronization managers*, and *synchronizable media objects*. Synchronization events support hypertime links: they aim at a target and contain an associated action that must be triggered at a given time in the future. A synchronization manager coordinates all media objects having a common time base. A synchronizable object integrates media and synchronization: it encapsulates the services needed for a media to be played back and for controlling its execution.

The three concepts on which our architecture is based change the notion of time: time becomes *elastic*, which means that it adapts to available resources. In a computing environment that does not have strong real-time support, it provides flexible adaptive synchronization and best effort quality of service. If a real-time support is available, the architecture provides guaranteed quality of service. We have prototyped our architecture using Java and experimented with playback of simple synchronized presentations.

In the remainder of the paper, we summarize the proposed temporal extensions to HTML (Section 2), present the execution architecture that supports the extensions (Section 3), overview its implementation (Section 4), discuss related work (Section 5) and outline conclusions (Section 6).

2 Integrating Time into HTML Documents

The W3C initiated an activity to explore integration of synchronized multimedia into WWW documents. It has defined a working draft of SMIL (*Synchronized Multimedia Integration Language*), a new language that extends HTML with temporal functionalities. It is based on XML and provides some basic functionalities for including continuous multimedia data such as video and audio in WWW documents. However, as stated in the introduction, it has several drawbacks. In a previous paper [2] we have proposed temporal extensions to HTML based on: *hypertime links*, *time bases*, and *dynamic layout*.

2.1 Hypertime Links for Temporal Composition

We propose to use a simple functional paradigm derived from temporal point nets to specify temporal composition: a temporal link between an origin and a target. We call it a *hypertime link* by analogy to its WWW companion. A hypertime link has an explicit temporal semantics: it relates two media samples (e.g. video frames) and assigns the origin's time instant to the target. It can start or skip an object at the specified target position or stop it if the target is the end of the object. Following a hypertime link is automatic in the sense that it does not require any user interaction and consists of skipping in time from the origin media sample to the target. The functional relation has a nice analogy with hypertext links which expresses a relation between an origin place and a target one, and whose activation allows the user to jump (instantaneously, in theory) from one place to another in the space of documents.

We also extend the notion of the origin of a hypertime link to include a possibility of specifying a portion of a temporal media (a range of samples) as an origin, in the same way as a portion of text can be an origin of a hypertext link.

2.2 Common Time Bases for Close Synchronization

Specification of temporal composition is not sufficient to playback a multimedia document. We need some more information about how media objects must be synchronized. This information is particularly useful in a computing environment that does not provide strong real-time support. In such an environment, different media segments started at the same instant may run out of synchronization after some time and require some corrective action (such as dropping samples) to become synchronized again. We want to be able to specify which objects should be kept synchronized, how often, and what is the nature of this close synchronization (in other words, who is the master of the time).

For this purpose, we define the notion of a *time base*. A time base is a virtual time space in which media objects "live". A time base defines a common time coordinate space for all objects that are related by some relations, for example master-slave dependency. A time base can be seen as a perfect time space in which real world phenomena such as jitter or drift do not exist and media objects behave in a perfect manner. Obviously, such a perfect time space does not exist, however, it can be implemented closely enough using real-time support. If such a support is not available, which is the case of many existing systems, a document should indicate how the quality of presentation is to be maintained and what is the nature of synchronization to be enforced.

We define the nature of synchronization between media segments using the notions of *master* and *slave*. A master-slave relationship defines a master controlling a slave according to its needs (Figure 1a). We extend this notion to multiple masters and slaves (Figure 1b) through the common time base: a master can accelerate time or slow it down, hence slaves and other masters must adjust to time. The master-slave relationship allows the user to easily define the behavior of media segments with respect to synchronization.

Another way to control synchronization between media segments is through *synchronization points*. When close lip-sync synchronization is not necessary, for example between a video clip and an audio comment, we do not need to enforce synchronization for each video frame or audio buffer. We rather specify some time instants that we call *synchronization points* at which synchronization should be enforced. Synchronization points can be specified at some intermittent user-defined instants, for example at the beginning or at the end of a video shot, as well as at the instants when some text must be presented along with a closed-captioned video scene. Synchronization points may also be specified as periodic, for example we can say that two media objects must synchronize at every interval of 1 sec. When no explicit synchronization points are defined, synchronization is

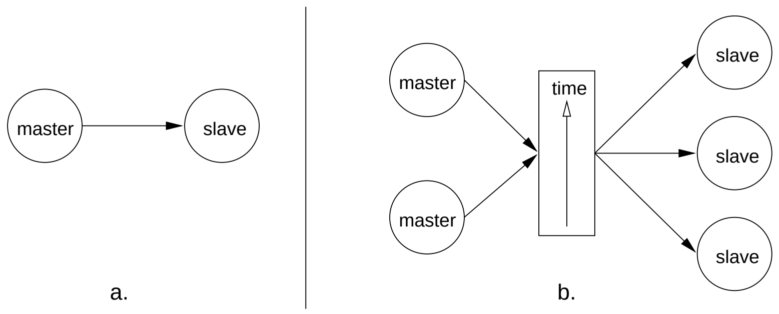

Fig. 1. Masters and slaves in a common time base

enforced at the smallest possible grain, i.e. at each video frame or audio sample buffer.

The two mechanisms for synchronizing media segments: the master-slave relationship in a time base and synchronization points allow authors to express complex synchronization constraints to ensure that the document will be played back as the authors intended it to be, thus preserving the semantics of the document.

2.3 Media Objects and Dynamic Layout

We suppose that a synchronized multimedia document may include a variety of media objects having temporal behavior. A *media object* defines time evolution of media samples of one type. Media samples must be presented at precise time instants defined by the rate of presentation. The rate may be imposed by the author, adapted to match the duration of another object, or adjusted to synchronize with other objects. A media object schedules presentation of samples within a given time base. In this way, objects in the same time base are synchronized. We suppose that traditional media objects such as audio and video can be enriched with temporally scrolled text.

In addition to synchronized presentation of media samples, a media object can be controlled by other objects according to temporal composition. A hypertime link activated by another object can change the current presentation of an object and force it to skip to the target sample or to stop.

We define a *dynamic layout* as a special case of a media object. It defines a temporal behavior of the physical layout. The only difference is that layouts are neither masters nor slaves since they do not contain any media samples to be synchronized. It encapsulates *frames*, a means for defining regions of screen in which media objects are presented. Frames can be mixed with static elements such as traditional HTML text paragraphs. Frames can include other layouts to specify nested layouts that provide nested coordinate spaces. Hypertime links define how the layout changes in time. This approach allows seamless integration of spatial and temporal dimensions into a multimedia document.

The syntax of the temporal extensions, examples and discussion are given in [2].

3 Execution Architecture for Synchronized Multimedia Documents

To support the model presented above, we have defined an execution architecture for playing back documents specified using the proposed temporal extensions to HTML. The extensions are fairly low-level, so all the proposed concepts have their counterparts at the system level.

Figure 2 presents the global functional view of the architecture. *Synchronizable objects* implement media objects with synchronization functionalities. *Synchronization managers* take care of controlling time bases, and *synchronization events* support synchronization points and hypertime links.

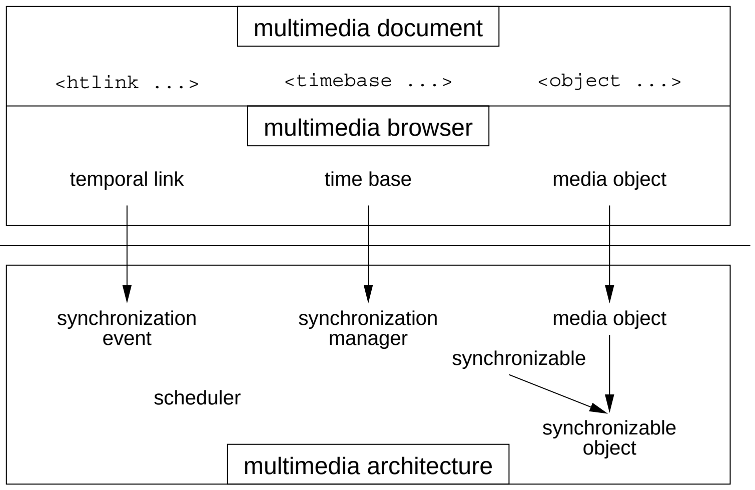

Fig. 2. Functional structure of the execution architecture

3.1 Principles

Our goal is to provide an architecture that takes care of time management in a multimediasystem. We want it flexible and open enough to support our temporal extensions as well as other standards such as SMIL. It should be extensible to allow seamless integration of external components. Active objects provide a nice basis for this goal, because they can guarantee flexibility and extensibility. We have begun with this approach in mind and after considering specific needs of multimedia synchronization, we have came up with an event based model similar to ATOM [3], but much simpler and more specific.

The architecture offers time management support to a collection of heterogeneous objects included in a multimedia document. Its only role is to guarantee a coherent playback of a document according to its temporal specification. Synchronization of media objects raises three kinds of problems that should be solved [4]:

- *Intramedia synchronization* guarantees that a continuous media will be played at a requested rate. This is implicit for most media that have a nominal rate such as audio or video, but it can be defined explicitly for others: for example some text associated with a closed-captioned video.
- *Intermedia synchronization* enforces temporal relations between several media objects played at the same time. It also controls that the media objects are played at a requested relative speed.
- *Media scheduling* is related to the management of media during a playback to make sure that events defined by the temporal composition of a document happen at a right time, for example activation and termination of a media object.

3.2 Architecture Components

There are three main components visible at the external interface of the architecture: synchronizable objects, synchronization events, and synchronization managers. A scheduler, an additional internal component, handles synchronization events. Figure 3 shows the components of the architecture.

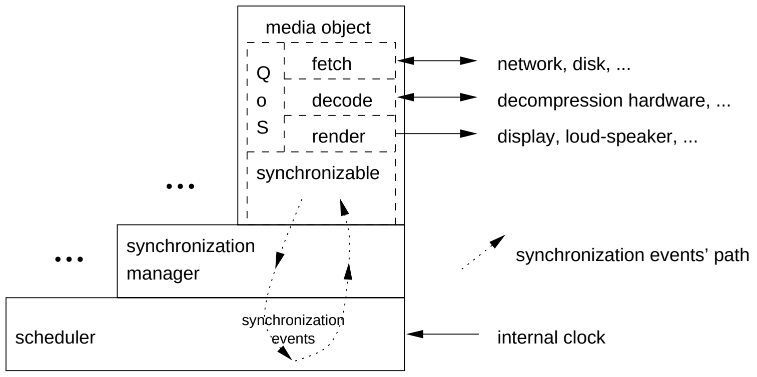

Fig. 3. Synchronization architecture

Synchronizable objects integrate media and synchronization: they encapsulate the services needed for a media to be played back and for controlling its execution. To be able to use external media objects (existing or proprietary software for example) for extending capabilities of multimedia applications supported by our architecture, the functions of media processing and synchronization should be clearly separated. For this purpose, a synchronization objects encapsulates a media object and implements a synchronizable interface. Each object has a synchronization role, master or slave.

Media objects are autonomous entities used as building blocks in multimedia documents. They encapsulate media processing functionalities needed to render their corresponding media. They may contain different modules to fetch data, decode, render, and manage quality of service. Media objects do not need internal synchronization, since it is provided from outside by our architecture. However, they have to implement methods that allow controlling their internal temporal behavior.

A *synchronizable interface* in Java terminology defines a set of methods needed by an underlying multimedia architecture to handle synchronization. When implemented by a media object, it becomes a synchronizable object, which means that it can generate synchronization events and be controlled by synchronization managers.

In this way, we provide a modular and extensible architecture: objects that deal with new media or compression formats can be integrated in a seamless manner.

Synchronization events convey time information between various entities. A synchronization event defines an action aimed at a target and that has to happen at a given moment defined by a deadline. When its deadline is reached, the event is triggered and sent to the target so that it performs the required action. Events are used for intramedia and intermedia synchronization and scheduling as it will be described later.

A hypertime link can be easily implemented using such a synchronization event. The deadline for a hypertime link can be obtained from the relative temporal information associated with the link.

Synchronization managers implement time bases. They manage a pool of synchronizable objects belonging to the same time base. They handle synchronization events on behalf of these objects and enforce synchronization policies defined by time bases through roles and synchronization points. Managers do not handle time themselves—an internal global scheduler is in charge of scheduling events.

The internal *scheduler* processes events posted by managers and triggers them once their deadline is reached. The scheduler runs as a high priority task. It should be carefully optimized so that dispatching events takes a minimum amount of time.

3.3 Principles of Synchronization

We will explain the principles of synchronization: how the components of the architecture cooperate to provide synchronization support. We have seen in Sect. 3.1 that multimedia synchronization raises three kind of problems (from the lowest to the highest level): *intramedia synchronization*, *intermedia synchronization*, and *media scheduling*.

Media scheduling is done using synchronization events. The temporal composition of a document specified by means of hypertime links defines media scheduling: we have seen that hypertime links can be easily implemented with synchronization events. An object can request an action such as an activation of another object just by aiming at it with a start event having a right deadline, so that the desired action will happen at the right instant.

Sequences of events maintain *intramedia synchronization*. Each synchronizable media object implements an event generator for its own needs: it generates events aimed at the object itself so that the object will be notified when needed. In this way, each object can simulate its own clock. For example, a continuous media such as video will generate events to display each frame. It will be notified each time a frame should be displayed. Figure 3 presents the path taken by synchronization events.

According to our temporal composition model, *intermedia synchronization* is done at the time base level using synchronization points. All media objects in the same time base are to be intermedia synchronized at the granularity defined by synchronization points. It is handled by synchronization managers, since they implement time bases. Synchronization events can be marked as being synchronization points at which master and slave should synchronize. If no synchronization points are defined in the document description (see Sect. 2.2), which implies that synchronization will be enforced at the smallest possible grain, all events are marked as synchronization points. To enforce intermedia synchronization, synchronization managers control and modify the relative positioning of events marked as synchronization points according to the role of their targets, master or slave.

3.4 Dynamic Generation of Events

Synchronization events in our architecture are scheduled according to an algorithm based on dynamic generation of events. The algorithm is based on the idea that an object generates an event dynamically and relatively to a previous event. This approach presents several advantages:

– since there is only one event per object, it keeps low the number of events in the system at a given instant. As a result, there is no problem of handling or sorting long lists of elements,
– it copes gracefully with *indeterminism*, because there is no propagation of time correction nor reevaluation of temporal constraints,

– the overhead of event dispatch is kept low, because intramedia and intermedia synchronization and scheduling are centralized in synchronization managers, thus avoiding complex and costly communication between media objects.

In our approach, instead of defining synchronization events using a deadline, we use a creation time and a delay—these two values define a deadline. If e_i is an event, its deadline is:

$$e_i.deadline = e_i.creationTime + e_i.delay \tag{1}$$

Events are generated dynamically in a relative way with respect to a previous event. When event e_i is triggered, then event e_{i+1} is generated with $e_{i+1}.creationTime = e_i.deadline$. This technique has two advantages. Small time variations at the scheduling level have no effect on the sequence of events: if an event is triggered a little too late, the next one will be generated with the correct time instant. Moreover, if an event is modified, for example the delay of an event increases, we do not need to propagate this modification, because relative positioning in time does it transparently.

Figure 4 illustrates this approach. Events are represented on the timeline according to their deadlines. Figure 4a shows events generated when no event is delayed. Obviously, the time between triggering an event and the execution of its associated action depends on system load. If there are no real-time guarantees, this delay is not bounded. For time sensitive media like audio, buffering techniques can be used to limit the impact of such small time variations.

Assume that an event is triggered and for some reason the target cannot execute the required action. There are two solutions. If the target is a slave, it cannot influence the time. Hence, the following events will be discarded until an action can be executed at its correct instant (see Fig. 4b). Figure 4c presents the case of a master. If its action cannot be executed, then it will be delayed and executed later. Hence, the event is observed with a delay Δ after the effective trigger instant and the following events will be automatically adjusted with this additional delay in a transparent way. When such a situation occurs, we also need to keep intermedia synchronization. In the case of a slave, it just discards events to keep going with time. In the case of a master, the problem is more complex: if we want that all other objects managed by the same synchronization managers keep with time, their events must add the additional delay Δ. In this case, events have to be modified and rescheduled in a complex way, because many possible cases exist: they might just have been triggered, queued up in the scheduler, or just generated. Synchronization managers can handle this, because they are the only objects that have the global view of events and their states: as we can see in Fig. 3, all communications between media objects and the scheduler pass through synchronization managers.

3.5 Discussion

The architecture is able to adapt to external conditions imposed by media objects according to a synchronization policy defined in a multimedia document.

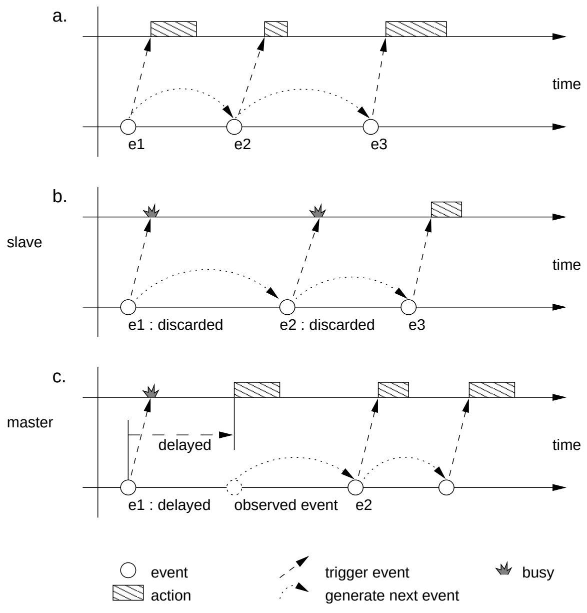

Fig. 4. Event scheduling for synchronization

Dynamic generation of events makes the notion of time *elastic*: the architecture follow synchronization constraints at best. This approach has several advantages. First, it guarantees temporal coherence of a document regardless of any system conditions. The playback will never fall into an incoherent state, which means that the architecture recovers from transient disturbances automatically.

If other services such as media objects or transport protocols implement adaptive quality of service so that they can react to synchronization problems signaled by the synchronization layer, documents will be played back at the best available quality. Finally, if a real-time support is available, the architecture provides guaranteed quality of service, since all time-critical operations will be guaranteed.

4 Implementation

We have begun the implementation of our architecture using Java. Java has many advantages that makes it a prototyping tool of choice: good language, easy to program, full range of libraries for network and graphics support. Its main drawback is performance, but it increases steadily. *Just-in-time compilers* (JITc) already allow a significant gain in performance, and with static compilers like *TurboJ* developed at the Open Group Research Institute, we can expect better performance soon. For critical parts like heavy decoding algorithms, native libraries called from Java code can be used.

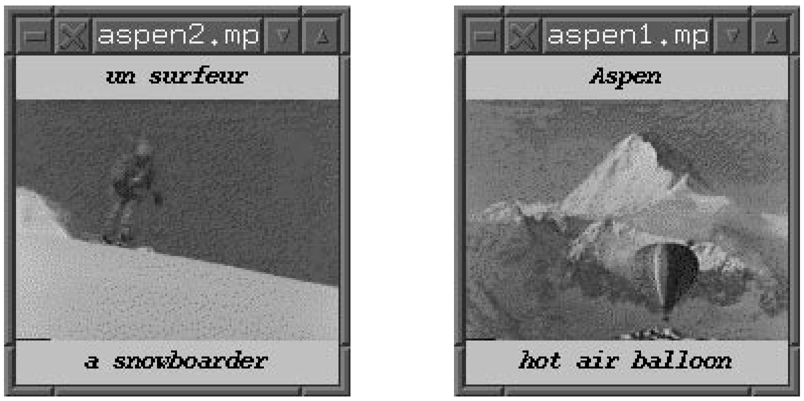

Fig. 5. Closed-captioned video clips synchronized with text

4.1 Prototyping

We have prototyped several concepts of our multimedia architecture. The prototype can handle MPEG-1 and MPEG-2 video, text, audio, images and text. A MPEG-2 decoder has been written in Java: it is a port of `mpeg2decode` (version 1.2 of July 19, 1996) available at the MPEG Software Simulation Group[3] that has been encapsulated and slightly modified to allow synchronization control. It is backward compatible with MPEG-1.

The prototype provides multimedia support including synchronization managers, synchronization events, and media objects as described above. A compiler allows to automatically generate presentations from descriptions in the language we have introduced. Experiments with presentations containing closed-captioned video synchronized with audio and text in two languages have been made. Figure 5 shows snapshots of video clips. Each of them is synchronized at scene changes with two text sequences.

The main problem with our prototype is performance, however using Sun's JITc on UltraSparc already allows us to obtain 15 fps for small MPEG-1 videos (160×120). We expect to perform even better with TurboJ, because it can be configured to remove the code for array bound checking while compiling. Our first tests with the MPEG decoder show performance gains around 25% when these checks are off.

5 Related Work

This paper builds upon previous work done in the domain of multimedia synchronization, time representation, and temporal composition.

Many proposed solutions are based on active objects such as ATOM [3]. We have not adopted this model, because it seems to us that handling time critical

[3] `http://www.mpeg.org/MSSG/`

tasks with fine grain synchronization would be too costly. However, during the design of our architecture, we have noticed that the global concepts of ATOM correspond to our ideas, however they differ with respect to synchronization managers. To implement the concept of time bases, we have chosen to centralize time management in synchronization managers to reduce information exchange between objects. Compared to ATOM, it is as if managers of media objects belonging to a common time base have been merged into a single external entity. Event management is then much more simple and accurate, since it is performed by one entity. Communication delays between objects are shorter and we do not need associated locking mechanisms.

A synchronization mechanism based on global events has been already proposed [5]. In this approach, events are global: all the streams to be synchronized contain the same events, so that they can be matched to enforce synchronization. In our model, events are provided by objects and synchronization is maintained by tight control of the distance between these events. A distributed service for orchestration of multimedia has been proposed by Gutfreund et al. [6]. Flinn has proposed a mechanism for sound effect control [7]. In his solution, events are scheduled in a similar way to our architecture. A scheduler dispatches events or sequences of events as requested by independent applications. Such a framework is very flexible and is used to provide graceful degradation. However, events are handled in a static way in the form of collections and the solutions proposed are more oriented toward rhythmic patterns and phase synchronization. Other work in this domain concerns systems supporting musical applications [8] and distributed multimedia systems [9].

6 Conclusion and Future Work

We have defined an execution architecture for playback of synchronized multimedia documents. We suppose that such documents are specified by means of several abstractions including hypertime links, time bases, and dynamic layout. Their playback requires some support for scheduling, synchronizing, coordinating, and controlling different media objects and actions according to a temporal specification. In addition to that, an architecture supporting documents should be flexible enough to adapt to different platforms, to adjust quality of service to available resources, and to manage scalable content.

We have defined a flexible execution architecture based on three concepts: *synchronization events*, *synchronization managers*, and *synchronizable media objects*. Synchronization events support hypertime links: they aim at a target and contain an associated action that must be triggered at a given time in the future. A synchronization manager coordinates all media objects having a common time base. A synchronizable object integrates media and synchronization: it encapsulates the services needed for a media to be played back and for controlling its execution.

We have prototyped the architecture using Java and experimented with a playback of simple synchronized presentations. We perform tests to increase performance, which would allow us to experiment with more complex documents.

After performance tunning, we would like to investigate adaptive quality of service at the level of media processing and transport protocols. We will consider solutions such as software feedback [10, 11], receiver-transmitter control [12], client-server negotiation to see how they can be integrated with our architecture.

References

1. SMIL: Synchronized Multimedia Integration Language. WD-smil-971109, World Wide Web Consortium (W3C), November 1997. Latest version available at `http://www.w3.org/TR/WD-smil`.
2. Rousseau, F., Duda, A.: Synchronized Multimedia for the WWW. 7th Intl. World Wide Web Conf. (WWW7), Brisbane, Australia, April 14–18, 1998.
3. Papathomas, M.: ATOM: An Active Object Model for Enhancing Reuse in the Development of Concurrent Software. RR 963-I-LSR-2, LSR - IMAG, Grenoble, France, November 1996.
4. Lu, G.: Communication and Computing for Distributed Multimedia Systems. Artech House, 1996.
5. Manohar, N.R., Prakash, A.: Dealing with Synchronization and Timing Variability in the Playback of Session Recordings. Proc. of ACM Multimedia'95, San Francisco, CA, pages 45–56, November 5–9, 1995.
6. Gutfreund, Y.S., Diaz-Gonzalez, J., Sasnett, R., Phuah, V.: CircusTalk: An Orchestration Service for Distributed Multimedia. Proc. of ACM Multimedia'93, Anaheim, CA, pages 351–358, August 1–6, 1993.
7. Flinn, S.: Coordinating Heterogeneous Time-Based Media Between Independent Applications. Proc. of ACM Multimedia'95, San Francisco, CA, pages 435–444, November 5–9, 1995.
8. Orlarey, Y. and Lequay, H.: MidiShare: A Real Time Multi-tasks Software Module for MIDI Applications. Proc. of the Int. Computer Music Conf., Computer Music Association, San Francisco, CA, pages 234–237, 1989.
9. Steinmetz, R.: Synchronization Properties in Multimedia Systems. JSAC, 8(3)401–412, April 1990.
10. Cen, S., Pu, C., Staehli, R., Cowan, C., Walpole, J.: Demonstrating the Effect of Software Feedback on a Distributed Real-Time MPEG Video Audio Player. Proc. of ACM Multimedia'95, San Francisco, CA, pages 239–240, November 5–9, 1995.
11. Cen, S., Pu, C., Staehli, R., Cowan, C., Walpole, J.: A Distributed Real-Time MPEG Video Audio Player. Proc. of the 5th Int. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'95), volume 1018 of Lecture Notes in Computer Science, pages 151–162, Durham, NH, April 18–21, 1995.
12. Correia, M., Pinto, P.: Low-Level Multimedia Synchronization Algorithms on Broadband Networks. Proc. of ACM Multimedia'95, San Francisco, CA, pages 423–434, November 5–9, 1995.