Synchronization in Multimedia Languages for Distributed Systems

A. Guercio[1], A. Bansal[2], T. Arndt[3]

[1]Department of Mathematical Sciences, Kent State University
[2]Department of Computer Science, Kent State University
[3]Department of Computer and Information Science, Cleveland State University

1 Introduction

The rising popularity of multimedia content on the web has led to the development of special-purpose languages for multimedia authoring and presentations. Examples of such languages include SMIL [1], VRML [2], and MPEG4 [3]. These languages support the description of a multimedia presentation containing multiple media sources including both natural and synthetic media as well as media stored in files or streamed live over the network. Some mechanism for specifying the layout of the media on the screen is given as well as a set of primitives for synchronizing the various elements of the presentation. For example, in SMIL we can specify that two video clips be displayed in parallel or that one audio clip be started when another clip finishes. Some of these languages allow for a limited amount of user interactions. A SMIL 2.0 presentation might allow a user to choose a soundtrack in one of several different languages by clicking on a particular area of the presentation. This is accomplished through the incorporation of the events defined in a scripting language such as JavaScript.

While these are well suited for the description of multimedia presentations on the Web, they are of limited use for creating more general distributed multimedia applications since general-purpose programming is only available in the form of scripting languages that have limited power. To support the construction of more large-scale applications approaches such as the use of special multimedia libraries along with a general-purpose language as in the case of Java and JMF [4] or the extension of middleware such as Corba [5] have been proposed. Besides lacking certain essential characteristics for the development of advanced distributed multimedia applications that will be noted below, the use of libraries and/or middleware to achieve synchronization and perform other media related services results in a less well-specified approach than can be achieved by directly extending existing general purpose languages with multimedia constructs with precisely specified semantics. This latter is the approach we follow in our work on multimedia languages that we will describe here.

The language that we want to design should support general-purpose computation; therefore the multimedia constructs whose semantics we will describe should be added to an existing general purpose language such as C, C++ or Java. This is the approach taken by the reactive language Esterel [6]. Reactivity is a very important property for a multimedia language. A reactive system is one which responds to stimuli from the environment [7]. In a multimedia system such stimuli might include user interactions as well as (for example) events generated by the contents of some media stream. The multimedia system must be able to interact with the environment within a short, predefined time period. When used in this context, the difference between reactive systems and interactive systems is that while both may interact with the environment, the latter do not have such a time constraint. The way in which the environment interacts with the multimedia system is through the generation of signals. A signal can be either synchronous (e.g. the reading of some sensing device) or asynchronous (e.g. the recognition of a particular face in a video stream). Our approach to multimedia languages greatly increases the power and flexibility of synchronization by providing synchronization constructs which can be applied not just between media streams, but also between media streams and (synchronous or asynchronous) signals. In fact, in our approach multimedia streams are just a particular type of signal.

The rest of this paper is organized as follows. Section 2 describes the fundamental concepts of signals and streams. Section 3 introduces the various synchronization mechanisms. Section 4 describes the language constructs. Section 5 discusses related research while section 6 describes future research.

2 Signals and Streams

Reactive systems respond to signals generated by the environment. The response must occur within a predefined time period. The signals may have a value. They may also be either periodic or aperiodic. An example of a periodic signal is one which might be generated by a sensor, like a thermometer, which periodically sends the temperature in the form of a signal to the reactive system. An example of an aperiodic signal might be the coordinate values generated by a user using a mouse which are generated only when the user moves the mouse. In order to generalize synchronization of multimedia streams, we define a multimedia stream as a particular type of signal. This allows us to apply the synchronization constructs not just to multimedia streams but to streams and other signals as well.

Reactive multimedia systems often transform or respond to multimedia data which is coming from a

sensor such as a digital video camera. The sensor discretizes the continuous media data, converting it into a periodic stream. In such a stream, the multimedia data is associated with a periodic signal. Other types of interactions can produce an aperiodic stream. An example might be a security camera which transmits an image only when motion is detected along a fence line. We model these streams, both periodic and aperiodic as a pair of data and attributes where the data is a sequence of tuples whose elements can be either (elementary) values or tuples. Since multimedia streams are directly associated with a signal, we use the words "stream" and "signal" interchangeably. Formally, we define three types of streams, periodic, continuous and aperiodic, as follows.

Definition 1: A *periodic stream* S^P is a sequence of elements associated with periodic signals. S^P_i is the i-th element in the sequence such that the period p does not vary, that is the time between $S^P_{(i+1)}$ and S^P_i is the same as the time between $S^P_{(j+1)}$ and S^P_j, $\forall$ i,j with $i \neq j$.

Definition 2: A *continuous stream* is the data produced continuously by a sensor. A sensor is used to detect information from the environment. A continuous stream can be modeled as a periodic stream with periodicity $0 < p < \varepsilon$ where ε is a small value. The period p represents the rate at which the signal is produced.

Definition 3: An *aperiodic signal* can be generated at any time either by an external stimulus or by an event generated after a computation or through a user interaction (such as voice or mouse movement). An *aperiodic stream* S^A is a sequence (possibly of length one) of aperiodic signals. In an aperiodic stream S^A_i represents the i^{th} aperiodic signal.

A multimedia stream is then defined as follows.

Definition 4: A *multimedia stream S* has two components: *attribute-set* and *data*. Three attributes *periodic* or *aperiodic*, number of data elements per unit time, and type of data (such as *audio* or *video* or *music or audiovisual* etsc.) are essential. Other attributes are specific to the streams, and vary with different types of multimedia streams.

Let us consider an example of a multimedia stream.

Example 1: A quadraphonic audio stream is a continuous periodic multimedia stream, whose components are:

(i) the *data* represented as a sequence of sampled packets;

(ii) the *attribute-set* A={a_0 = periodic, a_1 = audio, a_2 = 44,100 samples per second, a_3 = no. of channels = 4, a_4 = 16 bits per sample, a_5 = media length, …}.

As an example of an aperiodic stream we have the following.

Example 2: Aperiodic signals have data and attributes as well. An example of an aperiodic signal is *mouse movement*. The components of this aperiodic signal are:
(i) the *data* which describes the mouse movement as a sequence of (X-coordinate, Y-coordinate);
(ii) the *attribute-set* A={a_0 = aperiodic, a_1 ="Mouse Movement", …}.

3 Synchronization

Multimedia synchronization represents some logical relationship (temporal, spatial, spatiotemporal, or logical) between two or more multimedia streams or objects [8]. In the context of research in multimedia computing, however, it is customary to use synchronization to describe only the temporal relationship [9]. Synchronization can also mean *intra-media synchronization*, which defines the temporal constraints within a single multimedia stream; however in general synchronization means *inter-media synchronization*.

Synchronization research can be done on a number of issues [8]. Among these are modeling and specification of synchronization requirements, synchronization algorithms and protocols, and fault recovery in the presence of failures. Our research falls in the first category.

In order for multiple streams to be synchronized, they must share a common clock. In centralized systems it is possible for all streams to use the same physical clock, but this is not possible in distributed systems. The use of multiple physical clocks in such systems is problematic since clock drift can cause skew between the clocks. In order to control this, the multimedia data source insert synchronization points into the streams. The synchronization points can be media points or event counters which preserve the partial ordering of events [10]. In general, inserting more sync points allows for a finer degree of synchronization, but with the cost of added overhead. For synchronization purposes, the multimedia streams need to share a common clock (possibly a logical clock). This common clock provides a common time-base used for synchronization purposes [11].

One of the issues we should take into account while synchronizing media streams is the possibility to have synchronization skew. For example, for good lip synchronization the limit of the synch skew is ±80 msec between audio and video. In general, ±200 msec is still acceptable. For a video with speaker and voice the limit is 120 msec if the video precedes the voice and 20 msec if the voice precedes the video [Li04]. The worst sync skew that can occur depends on how far apart the synch points have been chosen. The closer the synch points, the more resources (such as buffer space) are required when the system is running but more precision in synchronization will be obtained. How should the synch points be chosen if we have several streams with different synch points to be synchronized? Of course, a common synch point for all the streams must be found. If do not need a very strict synchronization or we want to save resources, we

explicitly define the synch points to be further apart. For example, if two streams are dependent and one stream has sync points every 2 seconds and the other stream every 3 seconds, a resource saving choice would be to select as the common sync point the least common denominator of the two synch points, i.e. 6 seconds. However, it is highly probable that this choice would not provide great results from the visualization point of view, since it would increase the chance of perceptual distortion. Therefore, this choice is not appropriate in our case where synchronization must be very tight. A more restrictive option is used by selecting the smallest of the values of the synch points of the streams.

3.1 Groups

A distributed multimedia system receives input data from sources, which can be either local or remote. The input data are either streams of media data or asynchronous signals produced by the interaction of the system with the external environment. Because of the temporality of the input data involved in the model, each multimedia stream S generated from the source has an associated clock $c \in C$ and forms a *multimedia source* Src(S, c). Many streams can be generated at the same instant from different sources. Some of these streams must be treated by the distributed reactive multimedia systems as dependent on one another; other streams must be treated independently. For example, suppose that an audio and a video are produced from two distinct sources and that they must be rendered at the same time. If we later decide to speed up one of the two streams what should happen to the second one? If they are dependent the second should speed up as well, on the other hand if they are independent, the second one should proceed undisturbed. We show the dependence of the streams by defining them to be members of the same group.

Definition 5: A *group* G is defined recursively as:

(i) A single multimedia stream such that the time-base of G is same as the time-base of the stream.

(ii) Two or more multimedia streams sharing a common time base or with their time-bases related through an equation for proper synchronization.

Logically, a group is a tree in which the interior nodes of the tree are groups, and the leaf nodes are multimedia streams. Groups express of dependency of streams and operations performed on the group will influence all of the elements of the group. Operations on a particular multimedia stream of a group (e.g. scaling of the time-base) can either be applied in isolation or to all of the streams in a group to which the stream belongs. We will call the first type of operation *isolated* and the second type *synchronized*.

3.2 The Synchronization Process

The synchronization process consists of a set of spatio-temporal functionalities that enables rendering of multimedia objects in multiple streams to have the same perception as if it happened in real time. *Inter-media synchronization* involving multiple media streams requires the ability to relate the clock of each stream either through a common shared time-base or through an equation relating the two time-bases. If two multimedia streams are grouped together (are dependent) then changing the time-base of one stream similarly affects the time-base of the other stream to maintain synchronization.

Definition 6: Given n streams $S_1, S_2, \ldots, S_n$, in a multimedia group $G = \{S_j, S_{j+1}, \ldots, S_m\}$, and a *synchronization* function f, the application $f(S_i)$ $(1 \le i \le n)$ enforces the application of function f on all the multimedia streams $S_1, S_2, \ldots, S_n$ such that the synchronization constraint is maintained.

Consider the lip synching of an audio and a video stream that are played in a lock step manner to give a realistic perception. The two streams are grouped so that time-scaling on either stream (e.g. time stretching or compressing the video) causes the other stream to be time-scaled to maintain synchronization.

The grouping of streams is dynamic and event-driven. That is, certain events may affect a group in one way while another event may affect the group in another way. This dynamic behavior can be obtained either by having multiple orthogonal groups and associating different events with different groups, or by dynamically ungrouping and regrouping the streams through language grouping constructs.

4 Language Constructs

In this section we will describe those language constructs that support synchronization in reactive multimedia systems. Due to space limitations, we are not able to give all of the language constructs. The exact syntax of the constructs will depend on the host language that the multimedia constructs are embedded in.

4.1 Stream Definition

Declarations of media streams are given in terms of the source of the stream (a URL) and the type of the stream (audio, video, audiovisual, etc.) An asynchronous stream can be defined as well. The granularity of the sync points is given as well. Each different type of multimedia stream has a number of attributes whose values are given as part of the declaration (e.g. audio attributes include the name of the stream, number of samples per second, number of channels, number of bits per sample, etc.).

Example 3: The following code is used to declare and initialize an audio stream coming from a remote source. The file name is "speech.mov" and the origin of the media is indicated by the URI address of the source. The audio received can be rendered in the systems at 44,100 samples/sec with 16 bits/sample over 2 channels and the playback rate should not be slower than half of the normal rate, nor more than twice as fast as the normal rate. The stream should have a sync point every 200 milliseconds. The host language is C.

```
audio_stream mm1={source1, "192.168.2.102",
"speech.mov", 44100, 2, 16, 0.5, 2.0, .2}
```

4.2 Grouping Constructs

Media streams are grouped together when dependency between streams needs to be stated explicitly. In particular, grouping clusters one or more media streams or groups for synchronization. The groups are both *dynamic* and *hierarchical*.

Example 4: Suppose we have a video that shows a opera singer singing and we want to add the audio in lip synch mode. To keep up the impression that the soprano is really singing, if we speed up the video, we expect the audio to speed up as well. In order to provide this type of synchronization, we need to indicate that the audio and the video are somehow dependent on each other. Groups are an elegant and efficient way to specify synchronization on multiple streams. The following code groups the two streams.

```
group soprano = {"videostream_opera",
"audiostream_opera"}
```

Groups are hierarchical since a group member can be a previously declared group. They are dynamic since we have commands `ungroup`, to dissolve an existing group, `add_group`, to add elements to a group, `delete_group` to remove elements from a group, and `regroup` to add elements to a new group. Groups have sync points as well. They can either be given explicitly, or computed implicitly as the smallest value of the elements of the group (chosen so as not to lose precision).

4.3 Event Definition

Our constructs make heavy use of events. An event may be generated based on the characteristics of the multimedia streams. For example, the appearance of a particular face in a video stream or a particular voice in an audio stream might cause an event to be generated. The events in turn can affect the synchronization of multiple streams. Events are defined based on the satisfaction of one or more partial conditions. Partial conditions can involve the presence or absence of some signal, the matching of an attribute value or some other condition. Events have a destination (module or object or synchronization process) that they are sent to, as well start and end times and a priority.

Example 5: A user performs a right click of the mouse every time he or she wants to start a video clip; however the video clip cannot be started until the current video clip is terminated and the video must be terminated within a reasonably small range of time (within 5 seconds) otherwise the request is dropped.

```
partial_condition_signal_presence cond1 = {
// Test for signal presence
      NewVideoClip,
// Name of the signal – it is present when
// the video is playing
      yes }  // Test for its presence
partial_condition_signal_presence cond2 = {
      RightClick,
// Present when user right clicks
      yes,
// Test for presence (not absence)
      0, 5 }
// It was present in last 5 seconds
event start_video = { player1,
// Destination of event is renderer
      cond1,
// The partial conditions of the event
      cond2 }
```

4.4 Synchronization

One basic synchronization construct is the loop. It has a number of elements which are played one or more times in sequence. The number of times the loop is repeated can be specified along with a delay time.

Example 6: In the following example, the second video stream is played three seconds after the first. The pair repeats two times.

```
loop {
      times = 2,
      element = video_stream1,
      delay = 3,
      element = video_stream2
};
```

Another important type of synchronization is when we want to play two or more streams concurrently. We support this type of synchronization with the parloop construct. The syntax is similar to the loop construct however the elements of the parloop are played in parallel rather than sequentially. Loops can also be nested as shown in the following example.

```
parloop {
      times = 4,
      element = audio_stream,
      loop {
            times = 2,
            element = video_stream1,
            element = video_stream2
      }
}
```

We can also specify that we want two or more media streams to play in parallel (in other words start at the same time) and end at the same time as well in a parloop construct. In order for this to occur in general one or more of the streams must be stretched (scaled). In order

for the scaling to occur, the scaling constraints given as part of the Quality of Service requirements in the declaration of the stream must not be violated.

Loops and parloops are the basic constructs used for synchronization of periodic data streams. They are also the mechanism used to start the playback (rendering) of a multimedia data stream. If we wish to play a stream just once, we use a loop construct with a single element and a single loop time. Note also that the presence of a loop embedded in a program does not cause the execution of the rest of the program to wait until the playback finishes – the execution continues concurrently with the rendering of the media.

4.5 Preemption Constructs

Loops, including infinite loops can be ended prematurely in response to an event. For example, consider a situation where a sequence of advertisements are being displayed on a public terminal, and suddenly a weather warning must interrupt or terminate the current show to provide urgent news. The warning is sent as an asynchronous signal, which may generate an alert event and require showing a text stream, which explains the type of emergency. Other multimedia streams such as an audio signal or a video could follow the text. This situation requires the specification of an abortion of a loop based on the presence of an asynchronous signal (for example the pressing of a button). The advertisement is put inside a loop, and the text media is displayed after the asynchronous signal aborts the loop. There are 2 types of abortion: "strong", "weak" which can be specified in the loop or parloop construct:

Strong abortion performs the immediate interruption at the next multimedia sync point without waiting for the completion of the current cycle of the loop as soon as an aperiodic signal is present.

Weak abortion performs preemption as soon an aperiodic signal is present but will complete the "current" loop cycle that is playing when the signal occurs.

Suppose that we have declared a group consisting of a video stream – MyAdvertisement, a text stream – MyText, and an audio stream – MyAudio. Further assume that we have declared sync points every two seconds for this group. If we want to stop playing MyAdvertisement when an aperiodic signal named StopAdv is present, we can do this as follows:f

```
loop {
       times = 3,
       abort_when = StopAdv,
       abort_type = strong,
       element = MyAdvertisement
}
```

In this example, even though the abort type is strong, we will wait until the next sync point in the media stream to abort the rendering of the stream in order to maintain synchronization with other members of the group. In the worst case, we will wait 2 seconds from the time the signal is present until the abortion occurs.

Suppose the advertisement is shown in sequence with some text. Suppose further that in the presence of the aperiodic signal StopAdv we want to skip the rest of the sequence of ads and text, and skip to the song which is supposed to follow them. However, we don't want to interrupt an advertisement which has already started. In this case we can use weak abortion as shown below.

```
loop {
       times = 1,
       loop {
              times = 1,
              abort_when = StopAdv,
              abort_type = weak,
              element = MyAdvertisement,
              loop {
                     times = 3,
                     element = MyText
              }
       }
       element = Song
}
```

We can further control the weakness of the abortion and specify other possible synchronization situations by introducing the delayed abort. In this case, the delay value is added to the time required to reach the first synch point after the delay is over. That means that the media stream is rendered for the delay period plus the time to reach the first sync point after the delay is over. Again, let us consider an example.

```
loop {
       times = 1,
       abort_when = StopAdv,
       delay = 3,
       element = MyAdvertisement
}
```

In this example, since MyAdvertisement has sync points every 2 seconds and assuming that strong abortion is the default, the abort will occur between 3 and 5 seconds after presence of the signal StopAdv. If the loop ends before this time, no further delay occurs.

5 Related Research

Athwal [12f] presents a methodology for the synchronization of multimedia streams for engineering and scientific analysis. Since the scientific and engineering phenomena which are recorded and subsequently played back are frequently not well correlated to the human's visual and cognitive timeframe, it is quite possible that the previously captured data must be played back in a different timeframe – perhaps using slow motion or time lapse or some more complex variability in the playback rate. This is termed time elasticity and it has some relationship to our notion of stretching grouped multimedia streams. It should be noted that the methods described in [12] are suitable only for prerecorded multimedia streams and that much of the methodology is concerned with minimizing resource usage during the synchronized play back of such streams.

Besides the limitations as far as application area and type of multimedia streams, the focus of that work is different from our research in that the focus is on implementation details for systems for synchronization rather than on languages to allow for the expression of synchronization. The work also differs in that it lacks the notion of sync points, which can be defined for our groups and loops, which allow a high level of flexibility for synchronizing multimedia streams under many different quality of service requirements. Furthermore, interaction of multimedia streams with aperiodic signals is not even considered. The same points about the difference of focus of our work and this one could be made about most of the other recent research on synchronization.

Cameron [13] proposes a model for reactivity in multimedia systems; however their notion of reactivity is much different than ours. They discuss multimedia systems in which a multimedia artifact (i.e. a multimedia stream) can react to discrete *events*, such as an audio player reaching the end of a track as well as to continuously evolving behaviors such as the volume of an audio track. They make a distinction between a series of discrete events and a continuously evolving behavior. Since behavior is an author-level abstraction, which therefore hides implementation details of the media streams, the approach is more suited for use as a high-level authoring tool for multimedia presentations rather than for construction of distributed multimedia systems.

A number of XML-based multimedia languages have been proposed lately. None of them provide all of the capabilities described in this paper, although some provide complementary capabilities. For example, Gu [14] introduces HQML, an XML-based language to express the quality of service requirements of distributed multimedia systems. Another example is the multi-modal presentation markup language (MPML) [15] which is an easy to use XML-based language enabling authors to script web-based interaction scenarios featuring life-like animated characters.

6 Conclusions and Future Research

A model for distributed multimedia systems which incorporates the synchronization constructs discussed in this paper along with an active repository which allows for the constant testing of the multimedia data for deterministic and non-deterministic events is given in [16]. The behavioral semantics of the language constructs have been developed as well in order to provide a formalism for verification, compilation, and validation. The semantics incorporates the temporality and the communication aspects of the system and uses a variation of the π-calculus [17] for modeling distributed reactive multimedia systems. The π-calculus has its roots in the CCS (Calculus of Communicating Systems) [18, 19, 20], which is able to describe interactive concurrent systems as well as traditional computation. The π-calculus adds to the CCS, mobility of the participating processes and uses the transmission of processes as values, and the representation of data structures as processes. This research will be presented in a future paper.

References

1. *Synchronized Multimedia Integration Language 2.0 Specification*, http://www.w3.org/TR/smil20/ Aug. 2001.
2. *Virtual Reality Modeling Language*, ISO/IEC14772, 1997 http://www.web3d.org/x3d/specifications/vrml/ISO_IEC_14772-All/part1/concepts.html.
3. H. Kalva, L. Cheok, A. Eleftheriadis, "MPEG-4 Systems and Applications", *Proc. of the 7th ACM Intl. Conf. on Multimedia (Part 2)*, Orlando, Florida, pp. 192-192, October 1999.
4. R. Gordon, S. Talley, "*Essential JMF – Java Media Framework*", Prentice Hall, 1999.
5. Object Management Group, "*Control and management of A/V streams specification*", OMG Doc. telecom/97-05-07, 1997.
6. G. Berry, G. Gonthier, "The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation," *Sci. of Comp. Progr.* vol. 19, no. 2, pp. 87-152, Nov. 1992.
7. G. Berry, "The Foundations of Esterel", in *Proof, Language and Interaction: Essays in Honour of Robin Milner*, G. Plotkin, C. Stirling and M. Tofte ed., MIT Press, pp. 425-454, June 2000.
8. N.D. Georganas, R. Steinmetz, T. Nakagawa, "Guest Editorial on Synchronization Issues in Multimedia Communication", *IEEE J. on Sel. Areas in Comm.* vol. 14, no. 1, pp. 1-4, 1996.
9. L. Li, A. Karmouch, N.D. Georganas, "Multimedia Teleorchestra with Independent Sources: Part 1&2 – Temporal Modeling of Collaborative Multimedia Scenarios", *ACM/Springer Multimedia Sys.* vol. 1, no. 4, pp. 143-153, 1994.
10. L. Lamport, "Time, Clocks and the Ordering of Events in a Distributed System", *Comm. of the ACM*, vol. 21, no. 7, pp. 558-564, 1978.
11. R. Gordon, S. Talley, "*Essential JMF – Java Media Framework*", Prentice Hall, 1999.
12. C.S. Athwal, J. Robinson, "Synchronized Multimedia for Engineering and Scientific Analysis", *Multimedia Systems,* vol. 9, pp. 365-377, 2003.
13. H. Cameron, P. King, S. Thompson, "Modeling Reactive Multimedia: Events and Behaviors", *Multimedia Tools and Applications*, vol. 19, pp. 53-77, 2003.
14. X. Gu, K. Nahrstedt, W. Yuan, D. Wichadakul, "An XML-based Quality of Service Enabling Language for the Web", *J. of Visual Lang. and Comp.* vol.13, no. 1, pp. 61-95, 2002.
15. H. Prendinger, S. Descamps, M. Ishizuka, "MPML: a Markup Language for Controlling the Behavior of Life-like Characters", *J. of Visual Lang. and Comp.* vol. 15, pp. 183-203, 2004.
16. A. Guercio, A.K. Bansal, "TANDEM - Transmitting Asynchronous Non Deterministic and Deterministic Events in Multimedia Systems over the Internet", *Proc, of DMS 2004*, pp. 57-62.
17. R. Milner, "*Communicating and Mobile Systems; the π-calculus"*, Cambridge University Press, 1999.
18. R. Milner, "A Calculus of Communicating Systems", *LNCS*, vol. 92, Springer-Verlag, 1980.
19. R. Milner, "*Communication and Concurrency*", Prentice Hall, 1989.
20. Y. Wang, "CSS + Time = an Interleaving Model for Real Time", *LNCS* 510, pp. 217-228, 1991.