Chapter 5

A Distributed Framework for Media Synchronisation Support

5.1. Introduction

Chapter 4 reviewed the current state of the art in multimedia synchronisation and identified some outstanding issues that need to be addressed to provide end-user media synchronisation. In this chapter, we present a synchronisation formalism and a distributed architecture that tackle some of these issues. The approach is oriented towards stored media and the design attempts to meet the following goals:

- To specify intra-stream and inter-stream synchronisation in an independent way. Temporal relationships for multiple media types have to be described in a common synchronisation formalism that does not depend on a particular media stream type or characteristic.
- To provide end-user media synchronisation for multiple media types coming from independent physical locations.
- To provide a simple, transparent way of controlling media synchronisation in distributed environments so that it is possible to have independent processes handling each media stream. Synchronisation adjustments have to be performed without the application involvement.
- To provide application developers with tools that can be used to control multimedia presentations.

The synchronisation formalism described in Section 5.2 allows temporal relationships to be specified in an independent way, without concern about the format of each media stream. It describes a special file that holds the information related to intra- and inter-stream synchronisation.

The distributed architecture described in Section 5.3 provides a common interface that allows applications to control multimedia presentations in distributed environments. The architecture incorporates a synchronisation mechanism that interprets the temporal relationships and provides transparent media synchronisation to applications.

5.2. Formal temporal relationship

Describing the temporal relationship is necessary so that a synchronisation mechanism can interpret and implement the required synchronisation during presentations [Little,90a][Horn,93]. Synchronisation requirements must be specified using a formalism that does not depend on the type of media stream, or the formalism may become too complex due to the large number of media types and formats. This section describes an object oriented approach and a set of logical rules that can be used to describe temporal relationships independently of the type and format of objects.

Media streams are seen as objects and each object is classified according to its media type and format. For instance, an MPEG video stream is classified as an object of type VIDEO and format MPEG. The type of an object informs the basic characteristics of the object and can be used as a hint when allocating the computational resources for the object. The following table describes the available object types and their associated characteristics:

Object type	Characteristics
VIDEO	Continuous media Tolerance to data errors Tolerance to data lost Tolerance to delays High CPU load Large buffers Medium granularity
AUDIO	Continuous media Tolerance to data errors Small tolerance to data lost Small buffers Small granularity
TEXT	Non-continuous media Medium buffer size
GRAPHIC	Non-continuous media Tolerance to data error Large buffer size
PICTURE	Non-continuous media Large buffer size

Figure 5.1. Object types and characteristics

The object type allows a synchronisation mechanism to best allocate the resources for a particular object without knowing details about the object itself. For example, when dealing with an object of type AUDIO, a synchronisation mechanism may choose to use a buffering scheme that favours a simple and fast allocation/deallocation algorithm for small buffers rather than a more complex algorithm [Clark,89]. Also, the synchronisation mechanism may use a

transport protocol that provides reliable delivery of messages without concern for data errors, avoiding the time consuming task of checksuming all the information [Doeringer,90].

For synchronisation purposes, objects are further divided into *synchronisation units*. A synchronisation unit is the basic entity for retrieval, transmission, processing, and synchronisation of objects. A synchronisation unit is a logical structure that identifies a particular information inside the object and is used to reference that information. The size and format of a synchronisation unit are dependent on the object type and format.

A *synchronisation unit ratio* describes the relationship between the logical synchronisation unit and the physical unit imposed by the object type. For some objects, the logical definition of a synchronisation unit coincides with the internal organisation of the object. These objects have a synchronisation unit ratio of 1:1. Other objects need to have multiple physical units for a single synchronisation unit. For example, VIDEO objects are internally divided into video frames which are suitable to be treated as synchronisation units. Therefore a synchronisation unit ratio of 1:1 is often used. AUDIO objects have to be treated differently because current computer systems can not properly deal with the very small times (e.g.: a few microseconds) required for playback of individual audio samples. Therefore we group together more than one audio sample to create audio synchronisation units, generating a synchronisation unit ratio of 1:xx (where xx is the number of audio samples used to create a synchronisation unit).

To facilitate synchronisation related to the termination of an object, a special synchronisation unit is defined to identify when the end of the object is reached. This synchronisation unit can be used when a temporal relationship depends upon the termination of a previous object playback.

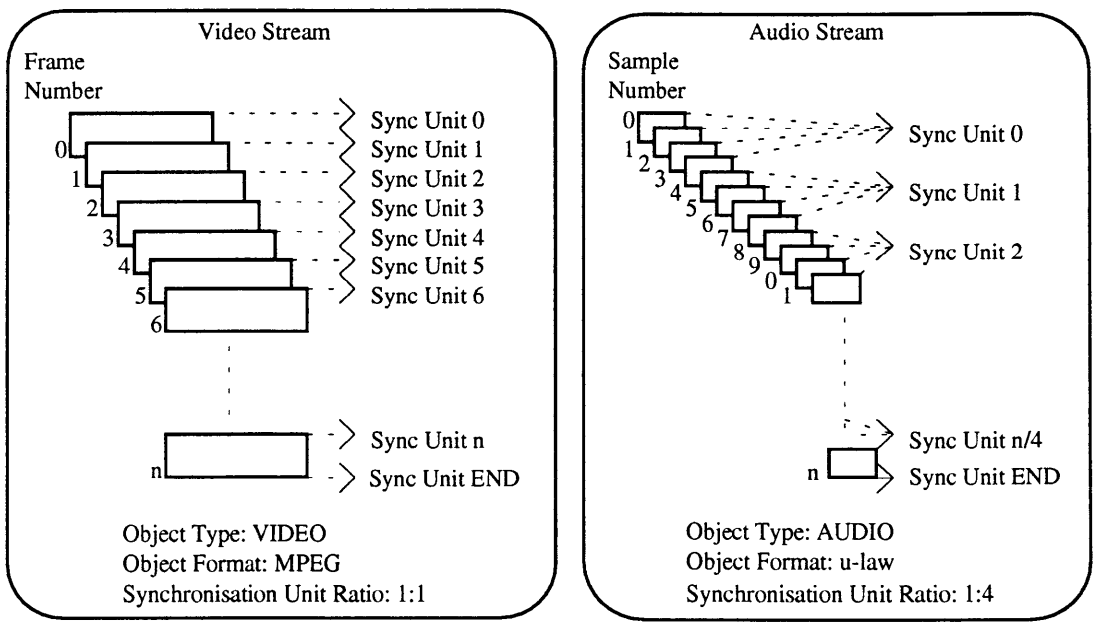


Figure 5.2. Multimedia objects and synchronisation units

The idea behind synchronisation units is that the information in different objects can now be referenced using a common mechanism. The characteristics of each object are hidden inside the synchronisation unit concept.

Media synchronisation can now be expressed using the logical synchronisation units without concern about the physical organisation of objects. By establishing a time relationship between synchronisation units, it is now possible to specify formally the intra- and inter-stream synchronisation in a way that is independent of the object type.

5.2.1. Intra-stream synchronisation

Intra-stream synchronisation (section 5.2.1) is specified by establishing a temporal relationship between consecutive synchronisation units from the same multimedia object. The temporal relationship for a given multimedia object is automatically generated according to the object type or, if the object contains the intra-stream information stored into the data stream, that information can be used. For example, a VIDEO object in MPEG format has a header at the beginning of the data stream that contains the frame rate at which the video stream was recorded. The frame rate describes the temporal relationship between consecutive frames for the object, which is the specification for the intra-stream synchronisation. In contrast, a video object that does not comply with any particular file format is composed of a raw sequence of frames with no header to describe the frame rate for the object. In this case, the frame rate can be implicitly known using the filename extension for the data stream.

Once the intra-stream synchronisation requirements are known, the information is stored together with the description of the objects to be synchronised (see Section 5.2.2.1).

5.2.2. Inter-stream synchronisation

Inter-stream synchronisation requires the participation of multiple objects with different requirements and characteristics. Nonetheless, the synchronisation unit concept allows the inter-stream synchronisation to be described without concern for the different object types. Two things have to be described to achieve inter-stream synchronisation: the objects involved and how they should be synchronised.

5.2.2.1. Describing the objects involved in the synchronisation

The problem here is where the objects involved in the synchronisation should be described. The alternatives are (as described in Section 3.2.2.2): cross references, where each object point to the others involved in the synchronisation; a master stream, where one object drives the synchronisation and contains the references to other objects; or an independent external file that points to the objects. The problem with the first two alternatives is that they involve changing the file format of at least one of the objects to incorporate the pointers to the other objects involved in the synchronisation. The changes required would vary according to the type of the object and the resulting file format would only be understood when used in this

particular synchronisation environment. These disadvantages lead us to the last alternative: an external file that describes the objects for the presentation.

The external file is known as the *synchronisation file* and contains a detailed description of the objects involved in the synchronisation, including the number of objects and the characteristics of each object. These characteristics can not be inferred from the type of the object. They are only relevant when the objects are used in the context of the multimedia synchronisation being described.

Each object and its characteristics are described in an ASCII file format which is then compiled to generated the synchronisation file used during the presentation. The ASCII file is composed of a sequence of statements divided into sections delimited by a token surrounded by left and right brackets. The first section is always a header section (i.e. [syncfileheader]) which describes details about the ASCII file itself. The subsequent sections identify objects and synchronisation points for the presentation.

Objects are described using a **[object]** section header. An object section contains a sequence of statements that describe the object characteristics in addition to the location, type and format of the object. Characteristics of the object include the file format in which the object is stored, the synchronisation unit ratio, and details about the intra-stream synchronisation.

Figure 5.3 shows a synchronisation file description containing two objects that are used in an audio visual presentation. The first object is of type VIDEO and is located on the machine kleene.une.edu.au in the subdirectory /pub/multimedia/video/une.mpg. The second object is of type AUDIO and is located in the machine babbage.une.edu.au on the subdirectory /pub/multimedia/audio/une.au.

```
[syncfileheader]
number_of_objects=2          // Number of objects involved

[object]
objectname=video1            // This object is known as "video1"
objecttype=VIDEO             // Object type
objectformat=MPEG            // Object format
objectlocation=kleene.une.edu.au:/pub/multimedia/video/une.mpg
objectratio=1:1              // One synchronisation unit for every physical video frame
objectrate=30                // 30 synchronisation units per second
objectfileformat=MPEG        // Format of the file in which the object is stored

[object]
objectname=audio1            // This object is known as "audio1"
objecttype=AUDIO             // Object type
objectformat=U_LAW           // Object format
objectlocation=babbage.une.edu.au:/pub/multimedia/audio/une.au
objectratio=1:160            // One synchronisation unit for every 160 sound samples
objectrate=50                // 50 synchronisation units per second
objectfileformat=SUN_AU      // Format of the file in which the object is stored
```

Figure 5.3. Synchronisation File

Once the objects have being described they can be referenced using the object name and the corresponding synchronisation unit. For example, audio1.#0 identifies the first media unit of the object audio1. This construction is used to describe the temporal relationship between objects.

5.2.2.2 Specifying temporal relationship among multiple objects

For describing the temporal relationship between objects, we must provide a way of specifying time in relation to a common synchronisation clock. A synchronisation file has associated with it the relative time at which playback started. This relative time can be used to provide synchronisation relative to the beginning of playback and can be used to synchronise any of the objects described in the synchronisation file.

For inter-stream synchronisation purposes, the synchronisation file contains another type of information: *synchronisation events*. Synchronisation events bind together otherwise independent synchronisation units so that the inter-stream synchronisation can be achieved. A synchronisation event contains a set of logical rules involving synchronisation units that have to be respected at a given time. The time for the synchronisation event is described by a timestamp associated with the event. At the right time the rules are checked out and an exception is raised if the rules are not satisfied.

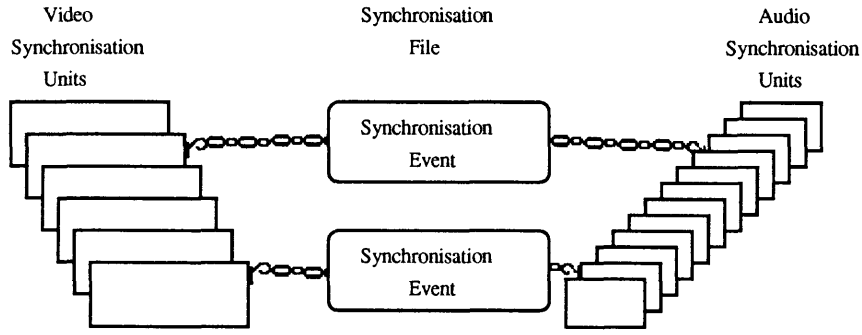


Figure 5.4. Synchronisation events

Logical rules are called *synchronisation rules* and they describe the temporal association between synchronisation units using the logical operators AND and OR. Any logical construction involving synchronisation units and the logical operators are considered a valid synchronisation rule.

Synchronisation events are described by the **[syncevent]** section header and basically contain three types of statements: the event name, the relative time for the event, and synchronisation rules. The event time may be relative to the session time or associated with the occurrence of any other event, including the play time of synchronisation units. Figure 5.5 shows a synchronisation event composed of a single synchronisation rule. The synchronisation event specifies that media unit number 0 from the video1 object has to be played at the same

time as media unit 100 from object audio1 and that the time for playback must be 150 (i.e. relative to beginning of playback).

```
[syncevent]
synceventname=first_event
synceventtime=150
video1.#0 AND audio1.#100
```

Figure 5.5. A synchronisation event section

In the normal case, this synchronisation event instructs a synchronisation mechanism to block the play back of both synchronisation units until the presentation time corresponds to time 150. In case the presentation playback time corresponds to 150 and the rule is not honoured, an exception should be raised.

It is important to note that synchronisation rules can be built using multiple logical operations and multiple synchronisation units from independent objects. Synchronisation rules described on the figure below are considered valid constructions.

```
audio1.#50 OR audio2.#1000 AND video1.#1
(video1.#0 AND audio1.#100) OR (audio1.#50 AND video2.#3)
```

Figure 5.6. Complex synchronisation rules

To demonstrate the flexibility of the synchronisation events approach, Figure 5.7 describes in full a synchronisation file for a multimedia presentation to welcome new students to the university. The presentation starts with a full-motion video welcoming track, then presents a sequence of 5 slides about the university facilities with accompanying audio track, and then concludes with a full-motion video wishing the students luck in their stay at the university. The presentation has six objects: two VIDEO objects, three AUDIO objects, and a PICTURE object. It is assumed that the audio for the slides is stored in a single audio file containing at the synchronisation units 0, 1500, 3250, 4600, and 6300: the starting points for each of the 5 slides.

```
[syncfileheader]
number_of_objects=6          // Number of objects involved

[object]
objectname=welcome_video     // This object is known as "welcome_video"
objecttype=VIDEO             // Object type
objectformat=MPEG            // Object format
objectlocation=neumann.une.edu.au:/pub/multimedia/video/une_welcome.mpg
objectratio=1:1              // One synchronisation unit for every physical video frame
objectrate=30                // 30 synchronisation units per second
objectfileformat=MPEG        // Format of the file in which the object is stored

[object]
```

```
objectname=welcome_audio      // This object is known as "welcome_audio"
objecttype=AUDIO              // Object type
objectformat=U_LAW            // Object format
objectlocation=turing.une.edu.au:/pub/multimedia/audio/une_welcome.au
objectratio=1:160             // One synchronisation unit for every 160 sound samples
objectrate=50                 // 50 synchronisation units per second
objectfileformat=SUN_AU       // Format of the file in which the object is stored

[object]
objectname=slides             // This object is known as "slides"
objecttype=PICTURE            // Object type
objectformat=JPEG             // Object format
objectlocation=metz.une.edu.au:/pub/multimedia/slides/une_slides.sld
objectratio=1:1               // One synchronisation unit for each jpeg picture
objectfileformat=BINARY       // Format of the file in which the object is stored

[object]
objectname=audio_4slides      // This object is known as "audio_4slides"
objecttype=AUDIO              // Object type
objectformat=U_LAW            // Object format
objectlocation=turing.une.edu.au:/pub/multimedia/audio/une_audio_4slides.au
objectratio=1:160             // One synchronisation unit for every 160 sound samples
objectrate=50                 // 50 synchronisation units per second
objectfileformat=SUN_AU       // Format of the file in which the object is stored

[object]
objectname=bye_video          // This object is known as "bye_video"
objecttype=VIDEO              // Object type
objectformat=MPEG             // Object format
objectlocation=neumann.une.edu.au:/pub/multimedia/video/une_bye.mpg
objectratio=1:1               // One synchronisation unit for every physical video frame
objectrate=30                 // 30 synchronisation units per second
objectfileformat=MPEG // Format of the file in which the object is stored

[object]
objectname=bye_audio          // This object is known as "bye_audio"
objecttype=AUDIO              // Object type
objectformat=U_LAW            // Object format
objectlocation=turing.une.edu.au:/pub/multimedia/audio/une_bye.au
objectratio=1:160             // One synchronisation unit for every 160 sound samples
objectrate=50                 // 50 synchronisation units per second
objectfileformat=SUN_AU       // Format of the file in which the object is stored

// Starts the description of synchronisation events

[syncevent]
synceventname=start
synceventtime=0                                  // This must happen at time 0
welcome_video.#0 AND welcome_audio.#0            // Synchronise the welcome tracks

[syncevent]
synceventname=slide1
```

```
synceventtime=welcome_audio.#END          // When the welcome_audio finishes
slides.#0 AND audio_4slides.#0            // Synchronise the slide with the audio track

[syncevent]
synceventname=slide2
slides.#1 AND audio_4slides.#1500         // Synchronise the slide with the audio track

[syncevent]
synceventname=slide3
slides.#2 AND audio_4slides.#3250         // Synchronise the slide with the audio track

[syncevent]
synceventname=slide4
slides.#3 AND audio_4slides.#4600         // Synchronise the slide with the audio track

[syncevent]
synceventname=slide5
slides.#4 AND audio_4slides.#6300         // Synchronise the slide with the audio track

[syncevent]
synceventname=bye
synceventtime=audio_slides.#END           // When the slide presentation finishes
bye_video.#1 AND bye_audio.#1             // Synchronise the video and audio tracks
```

Figure 5.7. The synchronisation file to the welcome multimedia presentation

Note that in Figure 5.7, the synchronisation times for the synchronisation events "slide1" and "bye" are associated with the termination of previous objects. Also, synchronisation events from "slide2" to "slide5" do not have a synchronisation time. The times for these events are implicitly associated with the playback of the audio synchronisation unit described in the synchronisation rule.

5.3. The distributed synchronisation architecture

Once the formal specification for synchronisation is available in a synchronisation file, it is necessary to have a synchronisation mechanism that can interpret the synchronisation requirements and maintain the correct temporal relation during the multimedia presentation [Anderson,91][Rowe,92]. Although a synchronisation mechanism can be implemented in each multimedia application, it is much simpler to have a unified mechanism that provides multimedia applications with a set of Application Programming Interface (API) to control the presentation [Coulson,93]. Such a mechanism can control the pace of the presentation including synchronisation issues as well as the problems that arise from the use of a distributed environment. This approach let application developers concentrate on the implementation of multimedia applications, eliminating the effort necessary to implement the synchronisation mechanism. The *Distributed Synchronisation Architecture* (DSA) attempts to provide such a unified synchronisation mechanism.

The DSA follows the client/server model of programming [Tanenbaum,92][OSF,92]. A multimedia presentation can be seen as a group of client and server processes. There is one client and one server process for every object being presented. A server process retrieves synchronisation units from a file and sends them to the corresponding client process. The client process interprets the content of the synchronisation unit and plays or displays the information.

The main goal of the DSA is to provide media synchronisation transparently to multimedia applications [Bastian,94]. It must account for all possible delays that can be introduced before the data is presented to the user and provide mechanisms to correct these delays or, in the worst case, inform the application that the presentation cannot proceed. This includes delays incurred when retrieving the information, when transmitting the information, and when presenting the information (as described in Section 3.3).

By inserting a new processing layer into the path travelled by each synchronisation unit after it leaves the server process and before it reaches the client process, we are able to control the playback without interfering with clients or servers processes. Also, this way we can easily control buffering requirements and the pace at which each object flows. This new processing layer is known as the *Synchronisation Protocol* (SP).

The SP is placed at the same level as the session layer of the OSI model [Tanenbaum,88] but it is not constrained to services specified by the OSI model. Particularly, intra- and inter-stream synchronisation are not part of OSI specification but they are the main reason for the existence of the SP. Every client and server process interacts directly with the SP using the API provided by the protocol. A server process delivers synchronisation units to the SP in the source machine and a client process retrieves the synchronisation units from the SP in the destination machine.

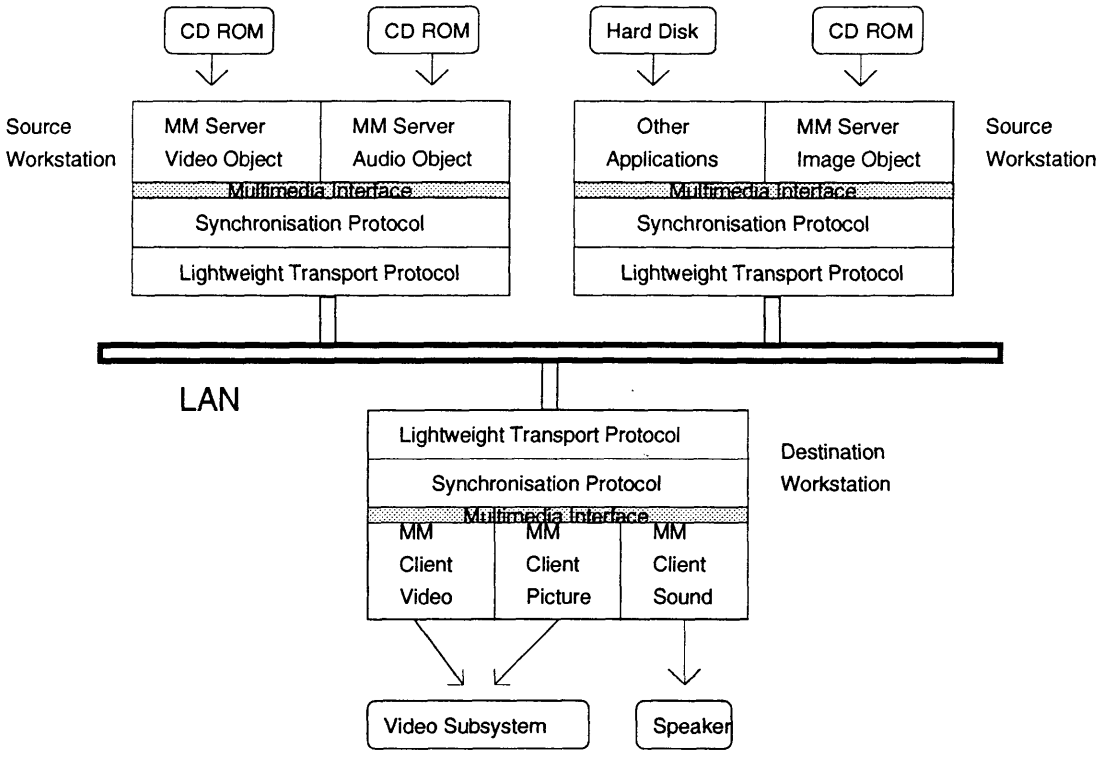


Figure 5.8. The Distributed Synchronisation Architecture (DSA)

5.3.1. The synchronisation abstraction

The DSA provides media synchronisation using an abstraction known as *Synchronised Sessions.* A synchronised session is a run-time instance of the synchronisation file. It could be described as multiple independent, but related data streams, bound together by a synchronisation mechanism that controls the order and time in which the information is presented to the user.

Synchronised sessions are managed by the SP and the information about the objects and their temporal relationships are retrieved from a synchronisation file. Timing information, including the temporal relationship between objects, is handled inside the synchronised session. Applications using the facilities provided by the synchronised session do not know anything about intra- or inter-stream synchronisation. This is controlled transparently and applications are only notified when an error condition occurs.

The SP exports an API that multimedia applications can use to create, control, and terminate synchronised sessions. During the creation of the synchronised session it is necessary to inform the location of the synchronisation file that contains the details about the session. The SP establishes a virtual circuit for every object referenced in the synchronisation file plus a virtual circuit for the transmission of the synchronisation file itself. Each of these virtual circuits is associated with an identification number that is used by multimedia applications to control the behaviour of the presentation.

Section 3.3 described the three sources of delay that interfere with multimedia presentations: retrieval delays, transmission delays, and presentation delays. To avoid the objects in a synchronised session drifting away from each other, the synchronisation mechanism must identify and, if possible, correct these delays. The SP incorporates a mechanism to detect when such delays are introduced. The performance of the synchronised session is monitored in three independent locations: on the server side, when synchronisation units are being retrieved and delivered to the SP; on the client side, when synchronisation units arrive from the network; and when synchronisation units are actually presented to the user.

Delays can be detected either in advance, before they interfere with the presentation, or after the presentation has been affected by the condition. In both cases some action has to be taken to avoid the data streams drifting away from each other. The actions are *prevention* and *correction*. Prevention is used when the detection mechanism foresees a problem. It tries to minimise the impact on the final presentation by changing the QoS of the object. The correction mechanism performs actions that may impact more drastically in the presentation, such as stopping or delaying the presentation until it can be resumed.

Preventing synchronisation drifts very often requires that the synchronisation mechanism discard some information for objects to catch up with the synchronised session. Discarding information is performed in a synchronisation unit basis. When the synchronisation mechanism detects delays in advance, synchronisation units are dropped to keep the correct synchronisation.

To maintain the SP independence from the type of objects, every synchronisation unit is described by a header which contains information pertinent to the synchronisation unit, including the synchronisation unit time and the synchronisation unit priority. The time field indicates the relative time at which the media unit should be presented to the user. This time can be either relative to the beginning of the presentation or to the presentation time of the previous media unit. The priority information indicates how the synchronisation unit should be handled in case we have to adjust the playback by dropping some of the data. It can hold three values: "very important", "standard", or "can be dropped". **Very important** media units are the ones which are essential for a correct playback. The system never drops this type of media unit. **Standard** media units are important for a correct playback but in extreme circumstances they can be discarded to avoid losing synchrony. **Can be dropped** media units are those that are first dropped when the need arises.

During a presentation, it is important that the user has some form of control over the session playback so that he/she can precisely control the playback to suit his/her needs. For example, if the telephone rings during a presentation, the system has to provide a way of pausing the session, so that it can be resumed later without having to start it all over again. Synchronised sessions are seen as object abstractions and they define a set of operations that can be performed on them. The operations allowed are start, stop, pause, restart, and rewind.

Any control operation executed on the session object is reflected in all objects belonging to the session. These operations provide an easy way to control the session playback.

5.3.1.1. Server processes and the synchronisation protocol

A server process reads the object data from a storage device, breaks the information into suitable synchronisation units and then delivers them to the SP. A server process passes the synchronisation unit and the information regarding the synchronisation unit (eg: sequential number, priority, size) to the SP. The SP uses only the information given by the server process, and the synchronisation unit data is passed untouched from the server side to the client side. This way, the SP handles the real-time constraint of intra- and inter-stream synchronisation without concern for the structure of the objects. Compressing/decompressing data and converting data structures from one computer architecture to another are done entirely by client and server processes.

In addition to handling synchronisation units over to the SP, a server process must be able to support the control operations that can be executed on the session object. For example, the rewind operation has to be passed to all server processes in order for the operation to be performed. A server process has to be ready to receive these operations and to carry out the actions required.

The close interaction between server processes and the SP allows the SP to detect if the server is being able to supply the synchronisation units at the rate needed by the object. If the server process is running faster than the rate required, it will eventually overflow the SP buffers. If the server process is running slower than the rate required, some action has to be taken otherwise the client process will eventually be starved of data. The solution for the first problem is to delay the execution of the server process by blocking the server process in every write operation on the SP. The second problem is more difficult to deal with and requires the SP to send a status report back to the server process telling it to accelerate the retrieval of synchronisation units. One way that a server process can accomplish this is by transmitting only the synchronisation units that are considered very important. The other types of synchronisation units can be skipped without the need to retrieve them from the storage device, thus improving the performance of the server.

Upon receiving the synchronisation unit from a server process, the SP inserts a small header containing details about the synchronisation unit and transmits it to the SP in the destination machine. The synchronisation file is a special object, but each synchronisation event is transmitted in the same way as a synchronisation unit to simplify the way the SP handles synchronisation files. The following diagram demonstrates the interaction between servers and the SP.

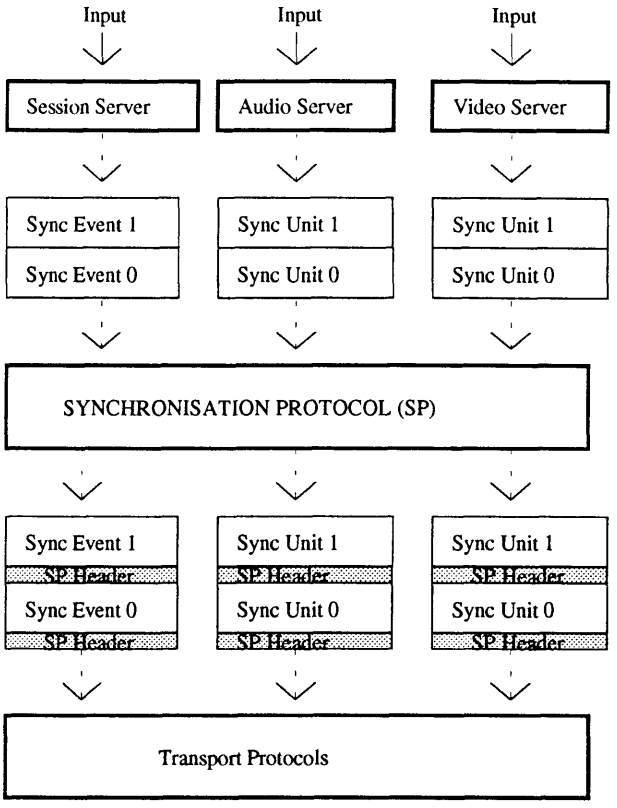


Figure 5.9. Server processes and the SP

5.3.1.2. Interaction between the SP source and the SP at the destination

Interaction between SP entities is performed using a Synchronisation Protocol Data Unit (SPDU). Two types of SPDU exist: control and data SPDUs. Control SPDUs are used to communicate status and control between SP entities. They do not contain any user-defined data. For example, connection establishment and presentation control operations are sent using control SPDUs. Data SPDUs are used to transmit the synchronisation units from source to destination machines. A data SPDU is composed of a synchronisation unit and the SP header described in the previous section. SPDUs are sequentially numbered so that the SP can detect if SPDUs are being lost during transmission.

On arrival of data SPDUs, the SP verifies the time deadlines for the synchronisation unit and sends a feedback to the source machine. This feedback contains the approximate time elapsed since the SPDU was transmitted. The source SP interprets this information and, if necessary, performs any preventive action to avoid starvation at the destination.

5.3.1.3. Client processes and the synchronisation protocol

On the destination machine, every client process reads synchronisation units from the SP. Before the synchronisation units can be presented to the user, they will most likely require some sort of processing. For example, MPEG synchronisation units have to be decompressed before they can be displayed [Patel,93]; text media units may have to be formatted before they can be presented and so on. Depending on the system load, this processing step can take quite a

long time. Decompressing a video object, for example, is a lengthy operation and may take an unacceptable amount of time before the data can be presented. In order to provide end-user media synchronisation down to a few milliseconds, the synchronisation adjustment has to be the last step before the data is rendered to the user, otherwise delays can still be introduced.

In our architecture, the SP does not know anything about the content of each synchronisation unit, therefore it cannot process the synchronisation unit. Synchronisation units have to be delivered to client processes so that the required processing can take place. Therefore a two step synchronisation approach is necessary. The first step provides a coarse grain synchronisation and it takes place when the client process reads the synchronisation unit from the SP. If the SP discovers any anomaly, it can take the appropriate action for that particular object. This action varies depending on the object type and importance. Some of the actions that can be taken are to drop some synchronisation units to catch up with the other objects, or block the execution of the client process by not returning from the read operation. The second step provides fine grain synchronisation (e.g.: less than 30 milliseconds) and takes place after the client has processed the synchronisation unit and it is ready for displaying or playing the content of the synchronisation unit. It requires the client process to tell the SP when it is ready to playback the synchronisation unit. The SP blocks the client process until all synchronisation rules for this particular synchronisation unit have been honoured. As soon as the synchronisation is correct the SP releases the client process to playback the information. After this point, the maximum difference in playback time among synchronised objects will be the longest time used by one of the client processes to display/play the data.

Client processes know little about how synchronisation is achieved. They only need to know two things. Firstly, if a client process can read data from the SP, the object playback time is in accordance with the rules imposed by the synchronised session. Secondly, after processing the synchronisation unit read from the SP and before sending the data to be played back, the client process must execute a synchronisation operation. This operation guarantees that the synchronisation unit is synchronised at the synchronisation unit level with other objects inside the same synchronised session.

Essentially, every client process has three processing steps: read the information from the SP; process the information; and present the information to the user. These three steps are executed for every synchronisation unit. Existing media playback applications use the program structure described in Figure 5.10a and are not concerned with the rate of playback. The playback rate is imposed by the speed at which the CPU can execute the application. It is important that existing playback applications can be ported to our software architecture with a minimum number of alterations. Figure 5.10b shows the client structure for use with the SP.

```
open file                          open object with SP
while (not EOF) {                  while (not EOF) {
   read data from file;               read sync unit from SP;
   process data;                      process sync unit;
   present the information;           synchronise with SP;
}                                     present the information;
                                   }
```

5.10a 5.10b

Figure 5.10. Fine grain synchronisation at the destination

By looking at Figure 5.10b we can see that the client process does not know anything about deadlines for playback. Deadlines are handled internally by the SP during the read and synchronise operations. The next diagram demonstrates the interface between clients and the SP.

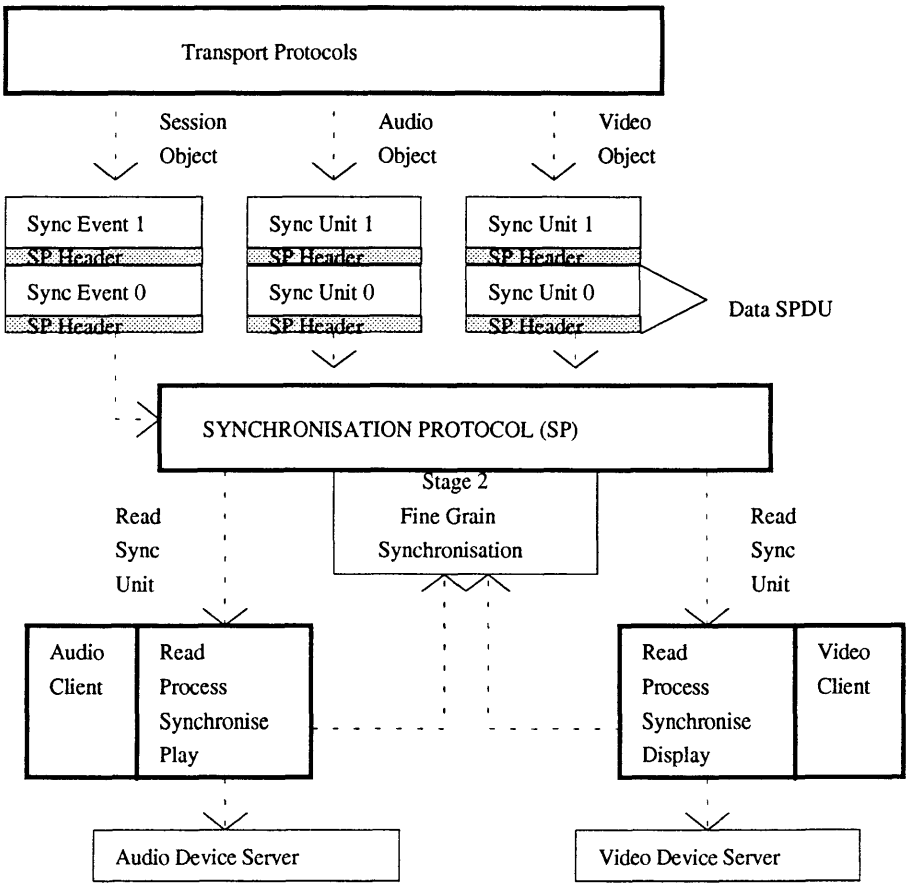


Figure 5.11. Client processes and the SP

Chapter 6

An Experimental Implementation

6.1. Introduction

This chapter presents an experimental implementation of the concepts described in Chapter 5. It describes in more detail the API available to applications, including the read and synchronise operations, the implementation of the SP, client and server processes, and a simple multimedia application to playback multimedia presentations.

The performance of three multimedia presentations is also examined: the first presentation involves a single audio object; the second involves a single video object presented at different frame rates; and the last involves the playback of both video and audio objects simultaneously.

6.2. The distributed synchronisation architecture implementation

6.2.1. The OSF/1 environment

It is generally agreed that the conventional Unix operating system does not provide the services required to support multimedia applications (see Section 2.3.3). Particularly, the scheduling mechanism is not capable of handling time deadlines. The solution is to have an enhanced operating system that addresses some of the real-time characteristics required to implement multimedia applications. The Open Software Foundation (OSF) has designed a Unix-like operating system based on the Mach 3 microkernel architecture [Accetta,86][Tanenbaum,92]. The OSF/1 operating system provides support to multiple threads of control [Birrel,89] and to fast switching time within the kernel.

Our environment consists of DEC Alpha workstations running the DEC OSF/1 operating system. In addition to providing the base OSF/1 services, the DEC implementation provides support to real-time programming through the POSIX 1003.4 draft 11 specification (P1003.4/D11) [DEC,92]. P1003.4/D11 specifies functions to handle real-time programming, including fixed priority scheduling policies, and high-resolution clocks and timers. These facilities are extensively used in the implementation to better predict the performance of time-sensitive threads.

6.2.2. The SP application programming interface

Access to synchronisation services is realised through the SP API. The interface hides the distributed environment and synchronisation details from client and server processes.

The SP API provides applications with an object oriented interface. It models the objects described in Chapter 5 into application objects. Two types of application objects have been described: *stream objects* and *session objects*. Stream objects can be of any type described in Section 5.2 (e.g.: VIDEO, AUDIO, PICTURE, etc). Session objects represent the synchronised sessions described in Section 5.3.1 and are used to control the presentation.

Both stream and session objects share the same base class. This class contains the common control operations available to all objects, plus the operations to read, write, and synchronise synchronisation units. Figure 6.1 lists the operations available in the base class and the correspondent parameters.

Operation	Description	Parameters
start	Starts the object playback.	*object id*
stop	Stops the object playback.	*object id*
pause	Pauses the object playback.	*object id*
continue	Continues a paused object.	*object id*
rewind	Rewinds the object to the beginning.	*object id*
write	Writes a synchronisation unit. The SP transmits the synchronisation unit to the destination machine.	*object id, sync unit, priority, sequential number*
read	Reads a synchronisation unit. The SP drops synchronisation units if the object is running behind the presentation time.	*object id*
sync	Synchronises the presentation of the synchronisation unit. The operation is blocked if the object is running ahead of the presentation time. It returns the delay for the operation.	*object id*

Figure 6.1. Base operations for application objects

During a presentation, a client process handling a stream object continuously executes the operations *read* and *sync* to maintain the object synchronisation. The server process continuously executes the *write* operation to deliver synchronisation units to the client process. The SP transparently delivers synchronisation units from the server to the client process and maintains both intra- and inter-stream synchronisation.

A session object controls the execution of synchronisation files. A session object is identified by the SP as being the controlling entity for the session being presented. Each session may contain up to 256 stream objects. Any of the base operations on the session object reflect

to all stream objects being controlled by it. A session object is created and destroyed using the operations shown in the following table.

Operation	Input parameters	Return
opensession	*location, filename*	*session id, description of stream objects*
closesession	*session id*	*status code*

Figure 6.2. Creating and destroying session objects

When creating a session object, the location and filename must be described to the synchronisation file. The SP locates the synchronisation file and creates a session identification number to identify the synchronised session and identification numbers to identify each stream object. The operation returns the identification number and type of each stream object involved in the presentation. Destroying a session object is just a matter of passing its session id to the SP. The SP automatically destroys all the stream objects associated with the session.

Once a session object has been created, an application can dispatch threads or processes to handle each stream object in the presentation. The number of stream objects and their descriptions are returned to the application when the *opensession* operation is completed. A thread is attached/detached to a stream object using the operations listed in Figure 6.3.

Operation	Input parameters	Return
attachobject	*stream id*	*status code*
detachobject	*stream id*	*status code*

Figure 6.3. Attaching and detaching stream objects

Attaching a thread to a stream object requires the identification number returned upon completion of the *opensession* operation. Once a thread has attached the stream object it can start reading synchronisation units from the SP.

6.2.3. The SP implementation

The synchronisation protocol is implemented as a Unix process. It runs in the user level space and interacts with the OSF/1 kernel using standard system calls. The SP interfaces with client and server processes using the APIs described in the previous section.

The SP is a multi-threaded process that handles incoming requests from multimedia applications as well as network packets from external SP entities. The SP uses the real-time priorities and scheduling policies of OSF/1 to guarantee that deadlines are successfully met. All SP threads run at a higher priority that normal Unix processes, and use a fixed priority scheduling policy (i.e. SCHED_RR) [DEC,92]. The use of a fixed priority scheduling policy guarantees that the SP threads will not have their priorities altered by the operating system,

regardless of how much CPU time each thread is consuming. Threads are created dynamically to handle session objects and stream objects. Each object is handled by two independent threads: one for receiving application requests and another for receiving data from the network.

When a new object is created, the SP establishes two end-to-end network connections with the peer SP. The first connection is used to transmit synchronisation units. It is a one-way connection, through which packets only travel from the source to the destination. The second connection is used to send and receive control information about the object. Having two independent network connections is important because the requirements for transmitting data about the object are completely different from the requirements for transmitting control information [Campbell,92][Shepherd,90]. This approach enables the SP protocol to find the best transmission protocol for both the object data and the object control information.

Packets exchanged between SP entities are called synchronisation protocol data units (SPDU) and each SPDU contains some common fields that describe the type and characteristics of the packet. The common fields of SPDUs are described in Figure 6.4.

Field	Content
sp_dlength	The length of the packet, excluding the header.
sp_flags	Contains various information (e.g. packet type, protocol version, priority, byte order).
sp_seq	The sequential number for this packet.

Figure 6.4. The SPDU fields

The common fields of SPDUs allow the SP to quickly identify incoming packets. The *sp_dlength* field informs how many bytes are still to be received so that the SP can allocate a buffer to receive the information. The *sp_flags* field contains information about how to interpret the rest of the information (e.g.: protocol version, byte order, and packet type) and also about the priority of the synchronisation unit. The priority flag is used to discard synchronisation units when synchronisation is at stake. The *sp_seq* field allows the SP to detect packets lost during network transmission.

Internal timers are kept for each session and stream object so that the SP can accurately maintain end-user synchronisation. For intra-stream synchronisation the SP monitors the flow of each object at three points: at the server, when the server process is delivering synchronisation units to the SP; at the client, when synchronisation units arrive from the network; and again at the client, when the client process executes the operation *sync* for the synchronisation unit. If the SP detects any synchronisation delays, it performs the appropriate preventive action. The preventive action varies from dropping synchronisation units to requesting that the corresponding processing unit (i.e. the client process, the network transmission sub-system, or the server process) performs its task faster.

Inter-stream synchronisation is checked only on the destination machine after the intra-stream synchronisation for the stream object has been performed. The inter-stream synchronisation mechanism delays the presentation of the synchronisation unit until all synchronisation rules are met. If a synchronisation problem is detected, the SP raises an exception to notify the thread that controls the session object.

The SP communicates with client and server processes using shared memory and semaphores to control concurrent access to critical regions. It implements a library of functions that can be used by multimedia process to interact with the SP.

6.2.4. Clients and servers implementation

Two stream objects and types are supported in the initial implementation: one VIDEO object and one AUDIO object. VIDEO objects have to conform with the MPEG compression specification and AUDIO objects have the format u_law and are stored using the SUN file format.

Each object is handled by two independent Unix processes: one server process to read the information from the source machine, and another process to process and present the information to the user. Client and server processes are implemented as stand alone Unix processes. They interface to the SP using the API provided.

6.2.4.1. MPEG video objects

The client to handle MPEG video objects is a port of the Berkeley MPEG decoder available from the University of California at Berkeley [Patel,93]. Porting the MPEG decoder to the distributed multimedia architecture was straightforward. Two basic alterations had to be performed: 1) Read the MPEG stream from the SP rather than from a file; and 2) Inform the SP when the video frame is decompressed and ready to be presented. Both alterations were easily done with the inclusion of a few conditional compilation statements (i.e. #ifdefs).

The main problem that had to be overcome was that to successfully decompress a MPEG stream containing type B frames (i.e. Bi-directional) some read ahead had to be allowed. The normal SP algorithm releases the next synchronisation unit (e.g. a video frame in the case of MPEG) to the application only after the application has informed the playback of the current synchronisation unit. When there are B type frames, the decoder requires that at least 3 frames (e.g.: one I, one P, and one B frame) be read before the first one can be presented, thus requiring the SP to release frames in advance to the application.

Inter-stream synchronisation is automatically maintained by the SP without interfering with the MPEG client process. When the SP protocol detects that the inter-stream synchronisation for the object is lagging behind the session time, the SP drops synchronisation units to catch up with the session time.

The MPEG server process was implemented from scratch, since a server with the characteristics required was not available. Nevertheless, it was easy to implement the MPEG

server using the interface provided by the SP. The main function of the MPEG server is to break a continuous MPEG video stream into suitable synchronisation units. This can be easily accomplished using the information at the beginning of each video frame. This information also contains the frame type (e.g.: I, P, or B frame), which is subsequently used to inform the priority of the synchronisation unit. Type I frames are usually tagged as "very important" synchronisation units unless the video stream is composed of only I frames. In this case even frames are tagged "very important" while odd frames are tagged as "can be dropped". The priority for type B frames is always "can be dropped".

6.2.4.2. U_law audio objects

The client process for AUDIO objects was ported from the aplay program distributed with the AudioFile package available from DEC Cambridge Research Laboratory [Levergood,93]. The same modifications required for the MPEG client were required for the audio client. Additionally, due to the interface provided by the AudioFile server, which requires the application to describe the time deadline for playback of the information, the audio client has to resynchronise the playback time whenever a synchronisation unit is discarded to maintain the intra-stream synchronisation.

The server process for AUDIO objects also had to be written from scratch. The server process reads the audio stream from a file and breaks the audio stream into synchronisation units. Ideally, synchronisation priorities should be associated according to an analysis of the audio stream, with lower priorities given to parts of the audio stream that contains silence or bandwidth that do not affect the human perception. This was not implemented in the current server process because of the difficulty involved in analysing the audio stream. Alternatively, synchronisation units with priority "can be dropped" are allocated for every two synchronisation units with priority "very important". This pattern does not provide a very accurate way of adjusting the synchronisation for the audio object, but it provides the SP with some information to discard if required. Nonetheless, is has been proven accurate enough to maintain the audio synchronisation and the audible signal when audio synchronisation units are dropped is a simple "tick" coming out of the speaker.

6.2.5. Device servers

It is implicit to the DSA that multiple client processes can share the same physical output device during multimedia presentations. Two physical devices are available in the initial implementation: the video screen and the audio speaker. Multiple accesses to the video screen are controlled by the MIT X11 server [Poundian,89] and access to the speaker is controlled by the Audiofile server [Levergood,93].

The MIT X11 server allows multiple windows to share the same video screen. Positioning, focus, and overlapping windows are controlled by the server.

The Audiofile server controls access to a speaker similarly to the way that the X11 server controls access to the video screen. One feature of the Audiofile server is that it mixes concurrent access to the same physical speaker to create a single audio output.

6.3. The presentation application

In order to test the concepts, a simple presentation application has been designed and implemented. The application can play back any synchronisation file regardless of how many objects the presentation has. Full synchronisation support during the lifetime of the presentation is guaranteed by the SP.

The application provides a graphical user interface to control the presentation. When the application is started, the user has to input the location and the filename of the synchronisation file that describes the presentation. The application uses the *opensession* operation to create the session object and once the operation returns it dispatches the processes that will handle the stream objects for the presentation. Figure 6.5 displays the main screen for the presentation application.

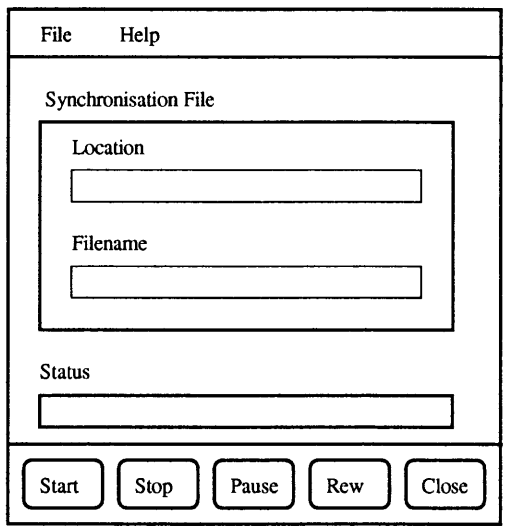


Figure 6.5. The presentation application main screen

6.4. Performance

As a full implementation of the inter-stream synchronisation mechanism was not carried out, it is not possible to evaluate performance of inter-stream synchronisation. However, for audio visual presentations containing audio and video objects, inter-stream synchronisation can be achieved by maintaining the correct intra-stream synchronisation for each object. The results reported in this section evaluate the performance of the intra-stream synchronisation for independent objects as well as related objects.

The tests were carried out on two DEC Alpha 500 workstations interconnected through FDDI. Two data files were used for the test: one audio stream containing about 110 seconds of

audio encoded in the u-law format and one video stream containing 1000 video frames (i.e. 111 seconds at 9 frames/sec) with a resolution of 160x120 pixels/frame. The video stream was stored in MPEG compressed format. During the presentation, the streams had to be transmitted in real-time from the server machine to the destination machine.

6.4.1. Single audio object

This test comprises a single audio object being transmitted across the FDDI network and played in the workstation speaker using the Audiofile server. The following figure shows the synchronisation file for this presentation.

```
[syncfileheader]
number_of_objects=1          // Number of objects involved

[object]
objectname=audio             // This object is known as "audio"
objecttype=AUDIO             // Object type
objectformat=U_LAW           // Object format
objectlocation=babbage.une.edu.au:/pub/multimedia/audio/beth05.au
objectratio=1:160            // One synchronisation unit for every 160 sound samples
objectrate=50                // 50 synchronisation units per second
objectfileformat=SUN_AU      // Format of the file in which the object is stored

[syncevent]
syncventname=start
synceventtime=0              // This must happen at time 0
audio.#0                     // Synchronise the audio object to start at time 0
```

Figure 6.6. The synchronisation file for the audio presentation

Each audio synchronisation unit represents 20 milliseconds worth of audio. To maintain the correct intra-stream synchronisation, the SP has to maintain the correct timing between them. The results shown in Figure 6.7 represent the average times for three executions of the presentation. For all three tests, the performance of the playback was almost always 100% correct. In one of the tests, the SP had to drop a synchronisation unit to maintain the correct synchronisation. This demonstrates that the sum of resources required to retrieve, transmit, and present the audio object are well below the maximum capacity of the environment.

	Server Process	SP (Source)	SP (Destination)	Client Process	Audio Server
system	1.09	1.17	2.30	4.54	2.60
user	0.32	0.30	0.60	1.78	2.80
total	1.41	1.47	2.90	6.32	5.40
% CPU	1%	1%	2%	5%	5%

Figure 6.7. Single audio object processing times

The considerable time spent in system level is due to the high number of synchronisation units generated each second. It reflects the overhead imposed on the system to allow audio synchronisation in 20 milliseconds steps.

6.4.2. Single video object

This test comprises a single video object being presented at different frame rates to verify the performance of the architecture. The same video stream was presented at the rates of 9, 18, and 27 frames per second. This is a good test to verify the effectiveness of the priority field when dropping synchronisation units.

The synchronisation file for the presentation at 9 frames per second is shown in Figure 6.8. Changing the frame rate for the presentation is just a matter of changing the *objectrate* statement to the desired frame rate.

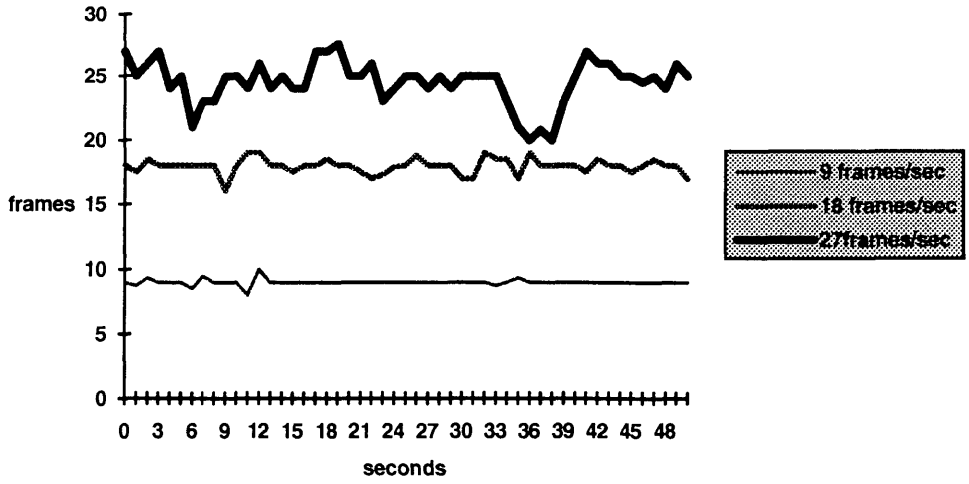
```
[syncfileheader]
number_of_objects=1          // Number of objects involved

[object]
objectname=video1            // This object is known as "video1"
objecttype=VIDEO             // Object type
objectformat=MPEG            // Object format
objectlocation=church.une.edu.au:/pub/multimedia/video/flight.mpg
objectratio=1:1              // One synchronisation unit for every physical video frame
objectrate=9                 // 9 synchronisation units per second
objectfileformat=MPEG        // Format of the file in which the object is stored

[syncevent]
synceventname=start
synceventtime=0              // This must happen at time 0
video1.#0                    // Synchronise the video with time 0
```

Figure 6.8. The synchronisation file for the video presentation

The graph in Figure 6.9 shows the performance of the DSA according to the number of frames played during 1 second intervals. In the ideal case, where the frame rate is kept constant, a flat line should be displayed. However, due to the non-deterministic characteristics of computer resources, variations in the frame rate can occur. The SP deals with such variations by increasing the frame rate for successive seconds or discarding frames if the delay variations exceeds a maximum threshold. In the table that follows, timers are only represented by a single value because the timings for the presentation of all three streams remain constant. It is the amount of CPU required during the presentation that varies when the frame rate is altered.



	Server Process	SP (Source)	SP (Destination)	Client Process	Video Server
system	0.48	0.36	0.67	1.92	1.17
user	1.13	0.24	0.38	32.07	4.25
total	1.62	0.98	2.68	33.99	5.42
% CPU	1% 9 fps 2% 18 fps 3% 27 fps	0% 9 fps 1% 18 fps 2% 27 fps	2% 9 fps 4% 18 fps 6% 27 fps	30% 9 fps 57% 18 fps 73% 27 fps	5% 9 fps 8% 18 fps 12% 27 fps

Figure 6.9. The performance and timings for the video objects

Figure 6.9 shows that the performance for video objects is directly related to the speed at which video frames can be decompressed. The overhead incurred by the other DSA components is almost negligible when compared with the decompression process.

The timings for decompressing video objects reflect the software decompression performed by the client process. It has been shown in [Patel,93] that the primary limitation in performance is the memory bandwidth required to manipulate the video stream.

For all three tests, delays only occurred when decompressing the video object. Retrieval and transmission speeds were fast enough to keep up with the traffic.

The overall quality when presenting 9 or 18 frames per second can be considered very good. Because of the small variations in playback, delays are hardly noticed. However, when presenting 27 frames per second, the presentation quality suffers because of the large number of frames being dropped. The intra-stream synchronisation mechanism does a good job of maintaining the correct synchronisation but at the cost of the quality of the presentation.

6.4.3. Audio and video objects together

This test comprises the presentation of an audio object together with a video object. The audio and video objects are the same as the ones used in Sections 6.4.1 and 6.4.2 respectively. The frame rate used for the video object was 9 frames per second. The synchronisation file for the presentation is presented in Figure 6.10.

```
[syncfileheader]
number_of_objects=2              // Number of objects involved

[object]
objectname=video1                // This object is known as "video1"
objecttype=VIDEO                 // Object type
objectformat=MPEG                // Object format
objectlocation=church.une.edu.au:/pub/multimedia/video/flight.mpg
objectratio=1:1                  // One synchronisation unit for every physical video frame
objectrate=9                     // 9 synchronisation units per second
objectfileformat=MPEG            // Format of the file in which the object is stored

[object]
objectname=audio1                // This object is known as "audio1"
objecttype=AUDIO                 // Object type
objectformat=U_LAW               // Object format
objectlocation=babbage.une.edu.au:/pub/multimedia/audio/beth05.au
objectratio=1:160                // One synchronisation unit for every 160 sound samples
objectrate=50                    // 50 synchronisation units per second
objectfileformat=SUN_AU          // Format of the file in which the object is stored

[syncevent]
synceventname=start
synceventtime=0                  // This must happen at time 0
video1.#0 AND audio1.#0          // Synchronise the video and audio at the start (time 0)
```

Figure 6.10. The synchronisation file for the video presentation

The presentation performance for both audio and video objects was similar to the performance described in Sections 6.4.1 and 6.4.2 respectively. The presentation maintains the inter-stream synchronisation in the sense that the two objects are started precisely at the same time and their intra-stream synchronisation is correctly maintained during the presentation lifetime. Figure 6.11 shows the timing for the DSA components.

	Server Process	SP (Source)	SP (Destination)	Client Process
system	1.24 Audio 0.50 Video	2.35	4.00	4.14 Audio 1.72 Video
user	0.34 Audio 1.16 Video	1.00	1.41	1.05 Audio 32.32 Video
total	1.58 Audio 1.66 Video	3.35	5.41	5.19 Audio 34.04 Video
% CPU	1% Audio 1% Video	2%	5%	5% Audio 30% Video

Figure 6.11. The timings for the audio visual presentation

6.5. Assessment

The major goal of the DSA of providing transparent media synchronisation to independent processes was achieved. In the three previously described performance tests, intra-stream synchronisation was successfully maintained without involving the application. Inter-stream synchronisation was only available to synchronise the starting time of multiple objects, but there is reason to believe that more complex synchronisation events can be easily implemented.

The API available to applications together with the DSA processing environment has proved that porting existing applications to the architecture is straightforward. Two generally

available applications have been successfully ported to the DSA with little effort. The major benefit is that applications that do not take media synchronisation into consideration can be quickly ported to the DSA, and full media synchronisation support is integrated into the application.

Finally, the initial results have shown that the load imposed by the synchronisation layer is very small compared to other parts of the system. This is encouraging because there is a general concern about the performance hit caused by externally controlling the synchronisation. It is important to note that the times shown in the previous figures represent the times for the SP with full debugging support. Additionally, there are many areas in which the SP performance can be improved just by optimising sections of the code.

Chapter 7

Conclusion

Media synchronisation will play an important role in the future of distributed multimedia applications. Future applications can benefit from a software architecture that overcomes the limitations of current platforms and provides support for media synchronisation.

This thesis has described the design of a distributed framework to implement media synchronisation transparently to multimedia applications. Chapter 1 identified the need for media synchronisation and the lack of proper support from the underlying software and hardware. It was argued that multimedia applications require a synchronisation environment that takes into consideration the need for media synchronisation and provides a formalism to describe temporal relationships in multimedia presentations.

Chapter 2 presented the characteristics and requirements of continuous media streams. A comprehensive analysis of the new characteristics and the impact on the current generation of computer systems was presented. Chapter 3 concentrated on the different types of media synchronisation, the sources of delays in distributed systems, and the techniques available to maintain the correct synchronisation when delays interfere with the presentation.

Chapter 4 outlined the current state of the art in support for media synchronisation. It pointed out that although various research efforts are tackling the problem of media synchronisation, little has been achieved towards end-user media synchronisation.

Chapter 5 presented the contribution of this thesis towards a better environment to support media synchronisation. A distributed framework composed of a Distributed Synchronisation Architecture that supports a formal way of specifying temporal relations was described. The advantage of such a platform is that the temporal relations for multimedia presentations can be expressed in a common formalism independent of the type and characteristics of the media streams involved in the presentation. Also, the platform performs the necessary adjustments to maintain the correct synchronisation without application involvement, releasing application developers to concentrate on the design of the application rather than on maintaining synchronisation.

Chapter 6 continued the description of the platform and presented some initial results of an experimental implementation. The initial results reported that the new software layer required

to transparently maintain synchronisation does not incur much overhead on the overall presentation.

This research has shown that it is possible to implement a synchronisation mechanism that hides the synchronisation issues as well as the distributed environment issues from multimedia applications. The formal temporal relationship can be specified using the synchronisation file, the synchronisation event, and synchronisation unit concepts. Such concepts allow temporal relations to be expressed without knowledge of characteristics of the media streams. The synchronisation formalism based on the synchronisation file hides the distributed environment from applications. It allows multiple media objects located in different places and with different characteristics to be handled as if they were of the same type. The benefit to multimedia developers is that applications can be built without concern for media synchronisation and the different characteristics of each media stream. Also, application developers can take advantage of the transparent synchronisation and use independent processes or threads to handle each media stream during a multimedia presentation.

This research can be extended in many ways, particularly the actions to be taken when a synchronisation error occurs requires further work. These actions must be related to the synchronisation event that fails and not to the global presentation. Having a per synchronisation event action to be performed when an error occurs gives the presentation greater flexibility when dealing with temporal errors.

References

[Accetta,86] M. Accetta, R. Baron, D. Golub, R. Rashid, A. Tevanian, M. Young, Mach: A New Kernel Foundation for Unix Development, Technical report, Department of Computer Science, Carnegie Mellon University, August 1986

[Anderson,90] D. P. Anderson, R. G. Herrtwich, and C. Schaefer, SRP: A Resource Reservation Protocol for Guaranteed-Performance Communication in the Internet, Technical Report, Computer Science Division, University of California at Berkeley, 1990

[Anderson,91] D. P. Anderson and G. Homsy, A Continuous Media I/O Server and Its Synchronization Mechanism, IEEE Computer, October 1991

[Bach,86] M. Bach, The Design of the Unix Operating System, Prentice-Hall Inc, ISBN 0-13-201799-7, 1986

[Bastian, 94] F. Bastian and P. Lenders, Media Synchronisation on Distributed Multimedia Systems, Proceedings of the 1994 Intl. Conference on Multimedia Computing and Systems, Poster session, Boston, USA, May 1994

[Birrel,89] A. D. Birrel, An Introduction to Programming with Threads, DEC System Research Center, Palo Alto, California, January 1989

[Bulterman,91] D. C. A. Bulterman, G. Van Rossun, and D. Winter, Multimedia Synchronization and UNIX, EurOpen Autumm 1991 Conference, 1991

[Campbell,92] A. Campbell, G. Coulson, F. Garcia, and D. Hutchison, A Continuous Media Transport and Orchestration Service, SIGCOMM'92, Maryland, USA, 1992

[Clark,89] D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen, An Analysis of TCP Processing Overhead, IEEE Communications Magazine, pages 23-29, June 1989

[Correll,92] K. Correll, T. Fitzpatrick, T. Hellman, R. Stalzer, JV2 Hardware Functional Specification, DEC Multimedia Engineering, 1992

[Coulson,93] G. Coulson, Multimedia Application Support in Open Distributed Systems, PhD Thesis, Computing Department, Lancaster University, April 1993

[DEC,92] Digital Equipment Corporation, Guide to Realtime Programming, Maynard, Massachusetts, November 1992

[Doeringer,90] W. A. Doeringer, D. Dykeman, M. Kaiserswerth, B. Meister, H. Rudin, and R. Williamson, A Survey of Light-Weight Transport Protocols for High-Speed Networks, IEEE Transactions on Communications, 38(11), pages 2025-2039, November 1990

[Escobar,92] J. Escobar, D. Deutsch, C. Partridge, Flow Synchronisation Protocol, Proceedings of the IEEE Global Communications Conference, December 1992

[Ferrari,92] D. Ferrari, A. Banerja, and H. Zhang, Network Support for Multimedia - A Discussion of the Tenet Aproach, Technical Report #TR-92-072, The Tenet Group, Computer Science Division, University of California, Berkeley, 1992

[Ferrari,93] D. Ferrari, J. Ramaekers, and G. Ventre, Client-Network Interactions in Quality of Service Communication Environments, Technical Report, The Tenet Group, Computer Science Division, University of California, Berkeley, 1993

[Fry,93] M. Fry, A. Richards, and A. Seneviratne, Framework for Implementing the Next Generation of Communication Protocols, 4th Intl. Workshop on Networks and Operating System Support for Digital Audio and Video, Lancaster, England, 1993

[Gall,91] D. Le Gall, Moving Picture Experts Group, Communications of the ACM, 34(4), pages 47-58, April 1991

[Gemmel,92] J. Gemmell and S. Christodoulakis, Principles of Delay-Sensitive Multimedia Data Storage and Retrieval, ACM Transaction on Information Systems, 10(1), pages 51-90, January 1992

[Govindan,91] R. Govindan and D. P. Anderson, Scheduling and IPC Mechanisms for Continuous Media, 13th ACM Symposium on Operating Systems Principles, 1991

[Gusella,89] R. Gusella and S. Zatti, The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley Unix 4.3BSD, IEEE Transaction on Software Engineering, 15(7), July 1989

[Horn,93] F. Horn and J. B. Stefani, On Programming and Supporting Multimedia Object Synchronisation, The Computer Journal, 36(1), 1993

[Jeffay,92] K. Jeffay, On Kernel Support for Real-Time Multimedia Applications, Proceedings of the 3rd IEEE Workshop on Workstation Operating Systems, April 1992

[Jurgen,92] R. K. Jurgen, Digital Video, IEEE Spectrum, pages 24-30, March 1992

[Knister,90] M. J. Knister and A. Prakash, Distedit: A Distributed Toolkit for Supporting Multiple Group Editors, ACM CSCW 90 Proceedings, pages 343-354, October 1990

[La Porta,91] F. La Porta and M. Schwartz, Architectures, Features, and Implementation of High-Speed Transport Protocols, IEEE Network Magazine, pages 14-22, May 1991

[Leffler,89] S. J. Leffer, M. K. McKusick, M. J. Karels, J. S. Quaterman, The Design and Implementation of the 4.3 BSD Unix Operating Systems, Addison-Wesley Pub. Company, 1989

[Levergood,93] T. M. Levergood, A. C. Payne, J. Gettys, G. W. Treese, and L. C. Stewart, AudioFile: A Network-Transparent System for Distributed Audio Applications, Digital Equipment Corporation, Cambridge Research Laboratory, 1993

[Liebhold,91] M. Liebhold and E. M. Hoffert, Future Standards, Communications of the ACM, 34(4), pages 104-112, April 1991

[Liou,91] M. L. Liou, CCITT Recommendation H.261, Communications of the ACM, 34(4), pages 60-63, April 1991

[Little,90a] T. D. C. Little and A. Ghafoor, Synchronisation and Storage Models for Multimedia Objects, IEEE Journal on Selected Areas in Communications, 8(3), pages 413-427, April 1990

[Little,90b] T. D. C. Little and A. Ghafoor, Network Considerations for Distributed Multimedia Object Composition and Communication, IEEE Network, November 1990

[Little,91a] T. D. C. Little, A. Ghafoor, C. Y. R. Chen, C. S. Chang, and P. B. Berra, Multimedia Synchronisation, IEEE Data Engineering Bulletin, 14(3), pages 26-35, September 1991

[Little,91b] T. D. C. Little and A. Ghafoor, Multimedia Synchronisation Protocols for Broadband Integrated Services, IEEE Journal on Selected Areas in Communications, 9(9), December 1991

[Little,92] T. D. C. Little and F. Kao, An Intermedia Skew Control System for Multimedia Data Representation, Proceedings of the 3rd Intl. Workshop on Network and Operating System Support for Digital Audio and Video, San Diego, November 1992

[Little,93] T. D. C Little, A Framework for Synchronous Delivery of Time-Dependent Multimedia Data, Multimedia Systems, 1(2), pages 87-94, 1993

[**Luther,91**] A. C. Luther, Digital Video in the PC Environment, Second edition, McGraw-Hill Book Company, 1991

[**Mercer,92**] C. W. Mercer and H. Tokuda, Preemptibility in Real-Time Operating Systems, Technical report, School of Computer Science, Carnegie Mellon University, 1992

[**Mills,91**] D. L. Mills, Internet Time Synchronization: The Network Time Protocol, IEEE Transactions on Communications, 39(10), October 1991

[**Nahrstedt,92**] K. Nahrstedt and J. M. Smith, The Integrated Media Approach to Networked Multimedia Systems, Technical Report, Distributed Systems Laboratory, University of Pennsylvania, February 1992

[**Nicolau,90**] C. Nicolau, An Architecture for Real-Time Multimedia Communication Systems, IEEE Jornal on Selected Areas in Communications, 8(3), pages 391-400, April 1990

[**OSF,92**] Open Software Foundation, Introduction to OSF Distributed Computing Environment, PTR Prentice-Hall Inc, ISBN 0-13-490624-1, 1992

[**Patel,93**] K. Patel, B. C. Smith, and L. A. Rowe, Performance of a Software MPEG Video Decoder, Submitted to the ACM Multimedia 93 Conference, 1993

[**Pehrson,92**] B. Pehrson, P. Gunningberg, and S. Pink, Distributed Multimedia Applications on Gigabit Networks, IEEE Network Magazine, January 1992

[**Postel,81**] J. Postel, DARPA Transmission Control Protocol, RFC 793, 1981

[**Poundian,89**] D. Poundian, The X Window System, Byte Magazine, 4(1), pages 353-363, January 1989

[**Ramanathan,93**] S. Ramanathan and P. V. Rangan, Feedback Techniques for Intra-Media Continuity and Inter-Media Synchronization in Distributed Multimedia Systems, The Computer Jornal, March 1993

[**Rowe,92**] L. A. Rowe and B. C. Smith, A Continuous Media Player, Proceedings of the 3rd International Workshop on Network and OS Support for Digital Audio and Video, November 1992

[**Sammartino,91**] F. Sammartino and D. Blackketter, Desktop Multimedia Communications - Breaking the Chains, IEEE International Conference on Communications, 1991

[Shepherd,90] D. Shepherd and M. Salmony, Extending OSI to support synchronization required by multimedia applications, Computer Communications, 13(7), pages 399-406, September 1990

[Silberschatz,88] A. Silberschatz and J. L. Peterson, Operating System Concepts, Addison-Wesley Pub. Company, ISBN 0-201-18760-4, 1988

[Steinmetz,90] R. Steinmetz, Synchronization Properties in Multimedia Systems, IEEE Journal on Selected Areas in Communications, 8(3), pages 401-412, April 1990

[Tanenbaum,88] A. S. Tanenbaum, Computer Networks, second edition, Prentice-Hall International, ISBN 0-13-166836-6, 1988

[Tanenbaum,92] A. S. Tanenbaum, Modern Operating Systems, Prentice-Hall International, ISBN 0-13-595752-4, 1992

[Wallace,91] G. K. Wallace, The JPEG Still Picture Compression Standard, IEEE Transactions on Consumer Electronics, December 1991

[Watabe,90] K. Watabe, S. Sakata, K. Maeno, H. Fukuoka, and T. Ohmori, Distributed Multiparty Desktop Conferencing System: Mermaid, ACM CSCW 90 Proceedings, pages 27-38, October 1990

[Williams,91] N. Williams, G. S. Blair, and N. Davies, Distributed Multimedia Computing: An Assessment of the State of the Art, Information Services and Use, 11, pages 265-281, 1991

[Witana,93] V. Witana and A. Seneviratne, Operating System Support for Multimedia Applications, Proceedings of the 3rd Australian Multi-Media Communications Applications and Technology Workshop, July 1993

[Wolfinger,92] B. Wolfinger and M. Moran, A Continuous Media Data Transport Service and Protocols for Real Time Communication in High Speed Networks, Technical report, The Tenet Group, Computer Science Division, University of California, Berkeley, 1992

[Zahn,90] L. Zahn, Network Computing Architecture, Prentice-Hall Inc, ISBN 0-13-611674-4, 1990

[Zhang,93] H. Zhang and T. Fischer, Preliminary Measurement of the RMTP/IP, Technical Report, Computer Science Division, University of California at Berkeley, 1993