# A CORBA-Based Architecture and Synchronization Support for Distributed Multimedia Services

Lassaâd Gannoun and Jacques Labetoulle
Institut Eurecom
2229, route des crêtes, BP. 193
06904 Sophia Antipolis, France
Voice: +33 4 93 00 26 71
Fax:+33 4 93 00 26 27
E-mail: {gannoun, labetoul}@eurecom.fr

**Abstract**

*This paper focuses on distributed multimedia services, their real-time control and the synchronization requirements between all components of these distributed services. We propose in this paper an oriented object model that addresses the requirement of real-time interactions and event based synchronization. The main feature of this model is the use of controller objects for real-time control and synchronization requirements. Moreover this model is supported by a CORBA architecture which helps flexible deployment of such a model. In order to demonstrate the application of the proposed model, we implemented two complementary distributed multimedia services. These distributed multimedia services handle two main media type flows, the X-protocol flows and the HTML document flows and they are used in the context of an asynchronous teleteaching session over the Internet.*

**Keywords**: *CORBA platform, Real-time interaction, Event-based Synchronization, X-protocol flows, HTML flows.*

## 1 Introduction

Multimedia systems provide an integrated environment for the creation, storage and presentation of a variety of media types (e.g. text, graphics, audio, video) [Blai96]. This paper focuses on distributed multimedia applications which operate in an environment of multimedia workstations interconnected by one or more networks. A fundamental characteristic of distributed multimedia applications is the need for real-time synchronization mechanisms. Many research has centered on this requirement [Blai96], [Coul96]. Various styles of real-time synchronization are studied and covered intra- and inter-media synchronization in order to satisfy quality of service parameters. This synchronization task becomes more complex in a distributed environment were each media type is handled by distributed dedicated servers. For example it is necessary to ensure that the presentation of the audio is synchronized with a graphic animation when audio flow and graphic flow are processed on different nodes.

In this paper we focus on distributed multimedia services and we propose an object oriented model that can be a support for distributing multimedia applications. In fact, the present work is influenced by works in the area of object oriented models for real-time distributed multimedia systems described in [Blai96] and [Papa95]. However, these works opted for synchronous languages to support their programming model in order to satisfy the quality of service (QoS) constraints. In this work we opt for Common Object Request Broker Architecture (CORBA) and its programming environment [CORBA96] to support the proposed object model. In fact, along this work we focus on the aspect of distributed objects and their real-time control and the synchronization support between CORBA objects that handle different composite multimedia objects. Even if the CORBA environment does not provide directly a system support for real-time guaranties of QoS, several mechanisms can be implemented [Coul96] on top of CORBA (i.e. special CORBA binding objects) that can satisfy QoS constraints for applications that need QoS guarantees.

This paper is organized as follows, in the next section we expose our context of a distributed application and the required multimedia services that are derived. Next, the basic object model is introduced and its different components are exposed. In this section we give some examples from our context of multimedia services to demonstrate how a CORBA architecture can help the deployment of the proposed object model. Afterwards, we apply the proposed model to implement two distributed multimedia services based on a CORBA architecture. These services are used in the context of a flexible asynchronous interaction. Finally, we give a concluding remarks and outline our future work.

## 2 The context of the multimedia services

The context of the multimedia services consists of a multimedia authoring and presentation systems for asynchronous interaction in a distance learning environment [Gann97]. The authoring system allows a user to record his X-window application session and to add hypertext comments (i.e. HTML documents) along all the session. While the multimedia presentation system provides a user with the capability of replaying the recorded application session and presenting, in an appropriate way, the hypertext comments associated with each portion of the recorded session. A more complex hybrid authoring scenario that combines the replaying and the recording steps, to support complex hypermedia documents, is incorporated in a hole system for a flexible asynchronous interaction.

These systems handle multiple composite multimedia objects and incorporates user interactions. Two main multimedia flows are handled in this context application, the first is X-protocol flows and the second is hypertext flows. In the recording step the authoring system should allow a user to control the different flows of the application, such as control of the stream flows by imposing a particular state to the provided multimedia service (e.g. start, pause, stop, save_revival_point, store_sync_point, etc...) and the

ability to store synchronization points on X-protocol flows. Whereas, in the replaying step (i.e. play-back of the session) which is provided by the multimedia presentation system, the user should, as provided in the recording step, control the different media flows (start, stop, pause, etc...) and also switch to a particular synchronization point in the stored session. In addition this presentation system also offers other forms of user interactions such as fast-forward and rewind.

These multimedia authoring and presentation systems are based on a distributed object model described in the following section. In this model several interacting objects cooperate with each other to provide these multimedia systems. In the next section we describe the proposed object model for a distributed multimedia services and show how it can be supported by a CORBA programming environment.

## 3 The basic object model

The proposed programming model is based on a location independent object model where all the interacting entities are treated uniformly as encapsulated objects [Blai96]. Objects can be composed together to create complex objects such as multimedia documents. These objects are accessed through *operational interfaces* which define named operations. An operation can be either an interrogation (a two-way operation, comprising an invocation, followed later by a termination carrying results or exceptions), or an announcement of a specific event (a one-way operation, comprising an invocation only). Since our architecture is based on a CORBA platform, interfaces are described in an abstract data type language which is the Interface Definition Language (IDL). Using IDL, we describe further the different interfaces of the components of this object model. We emphasize on how these interfaces are used to allow objects to synchronize with each other. The proposed model comprises three different kinds of objects :

- active objects (virtual device objects),
- controller or reactive objects,
- event synchronization objects.

**Active objects** (virtual device objects): are objects that encapsulate a media processing activities. These objects are multithreaded and perform some processing operations on a media stream. They are characterized by the concept of abstract states and state notification. These concepts were originally introduced in [Papa95] when dealing with real-time synchronization and object oriented framework in the context of synchronous languages.

Abstract states consist of states that characterize the behaviour of an active object (e.g. idle, start, stop, pause, resume, empty, full, etc...). These states are also used to constraint the invocation of methods on an active object so that invocations are accepted only when the object is in an appropriate abstract state. Constraints can be defined over these abstract states and hence provide the expressive power to manage concurrency between active objects.

State notification allows active objects to monitor and synchronize with each other when state changes occur. In the proposed model, state notification is supported by two IDL methods, *notify_state* and *get_state*. These methods are inherited by all active objects. In our model, state notification is performed by invoking a method *notify_state* on an active object. A request of a state notification is done by a *get_state* method applied on an active object. The method *notify_state* implements the asynchronous notification state, whereas the method *get_state* represents a synchronous state notification. We give in the following figure, some examples of how defining these methods in an active object interface using IDL language.

```
// IDL
enum STATE {    // basic states for an active object
 PLAY,
 PAUSED,
 STOPPED,
};
interface active_object{     //interface of a virtual device which represents an active object
        oneway void notify_state (in STATE cur_state);
         STATE get_state ();
};
```

Figure 1. Active object interface

To provide the user with a multimedia service, active objects interact and cooperate with each other. These objects are monitored by controller objects that coordinate and control their processing activities.

**Controller or reactive objects**: are real-time controller objects that accept events from the external environment such as user interactions with a multimedia service (e.g. interacting with a recording service or a replaying service). These accepted events may require actions by several different active objects. The controller object provides facilities to coordinate this cooperation among the different objects. The controller object processes the received events in a consistent way according to the controlled active object states.

The verification of the logical consistence rules with an active object is done by a Finite State Machine (FSM) object created within the controller object. This FSM reflects the "successive" states that can be taken regarding the actual states of the monitored objects. This is expressed by a set of rules that govern transitions between states. Actions taken by the controller object may include requesting states of some monitored objects.

Abstract states are defined as internal active object states and hence reflect the internal behavior of each active object. However, in a distributed multimedia environment a collection of objects can cooperate together to provide a final service (e.g. recording X-window session, recording a video/audio session, etc...). The user interacts with the multimedia system in order to control the provided service by imposing some service states such as start, pause, stop, resume and so on. These states represent abstract service states and cannot be handled directly by active objects. In our model, we distinguish between abstract object states and abstract service states.

Abstract states defined within active objects are not sufficient to express that these states are internal object states and do not reflect necessarily the provided multimedia service states. These abstract states can not be manipulated directly by a user of this multimedia service. However, abstract service states reflect a general service state that can be provided by a multimedia service. These states are visible from an external client object (e.g. client applet by which the user can interact with the provided service). These abstract states are defined within the controller object whose purpose is to accept the external user events and process them in a consistent way regarding the states of the different active objects that monitors. This is done by an IDL method *change_state* invoked by a controller object on an active object that controls. We extend in figure 2 the interfaces of active objects and we give the basic interface of a controller object. These interfaces are relative to the context of multimedia services described in section 2.

```
// IDL
enum STATE {  // basic state for an active object
  PLAY,
  PAUSED,       // pause do nothing
  STOPPED,      // do not record any application
};
interface active_object { // active object interface
            oneway void notify_state (in STATE current_state); // invoked by other virtual devices
            STATE get_state ();   // invoked by both virtual and controller objects
            void change_state (in STATE to_state);  // invoked only by a controller object
};

enum SERVICE_STATE {
  PLAY,
  PAUSED,
  STOPPED,
  SAVE_REVIVAL_POINT, SAVE_SYNC_POINT,
}
interface controller_object{ //  controller object interface
      oneway void notify_state (in STATE    state   ); // invoked by active objects/virtual devices
      oneway void request_state (in SERVICE_STATE to_state); // invoked by a client object
};
```

**Figure 2. Active and controller object interfaces**

From a client side view, to request a special service state (start, pause, resume, stop, etc...), an operation is *request_state* is invoked by the client on the controller object interface. When this request operation is invoked, the controller object forwardes it as appropriate to its controlled objects. For example, if calling ControllerObject.request_state(pause), causes all of its controlled objects to be paused (by invoking as appropriate *change_state* methods on them). The same holds for states like, stop, play, etc. This allows the client to call a single object method and have this command propagated to each monitored object by the controller object.

Figure 3 shows the computational environment of an active object and its relations between the other active objects and the controller object.

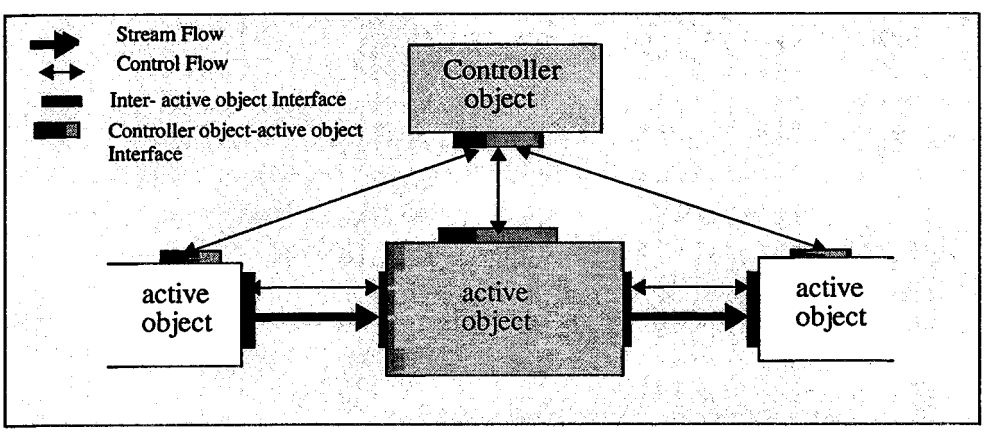


**Figure 3. Computational environment of an active object**

These relations are defined by object interfaces. Two types of flows are distinguished : control flow and stream flow (data flow). The stream flow is processed by only active objects, whereas control flows are both between active objects and also between the controller object and each active object that controls. In the following we expose the event synchronization objects and how they are managed by the controller object.

**Event Synchronization objects**: are objects that represent the synchronization events that are exchanged to synchronize activity between a client (e.g. client of the provided service) and the controller object. These objects have an internal one dimensional coordinate space. This space can be, extended real ($R_\infty$) or extended integer ($Z_\infty$) or extended time ($Time_\infty$) [Premo96]. The semantic meaning of this coordinate space is relative to the application e.g. media objects audio/video may represent time. However, X-request media objects may represent X-request numbers (e.g. frame numbers). This coordinate space is referred to as the progression space of the event synchronization objects.

Reference points are points in the progression space of event synchronization objects where these objects are attached. An event synchronization object instance contains an event type, a reference to an object and a boolean "wait" flag. In contrast to the Premo Model [Herm96] where synchronizable objects are autonomous, in our proposed model Event synchronization objects are managed within the controller object and are used in the following manner. When a reference point is reached, the controller object uses the event synchronization instance to dispatch this event, and possibly suspends itself if the wait flag is set to "True" and waits for an external message request to continue its progress. Hence, through this mechanism the controller object can stop other objects, restart them, suspend them, etc. Afterwards, the controller object will forward this event to one or multiple recipient objects.

This is done by two different IDL methods which are *notify_event* and *report_event* applied on a recipient object (controller object). The *report_event* method allows the invoker to report an event to a controller object that simply transforms it to an event synchronization object and stores it. This mechanism allows later a special synchronization (event based synchronization) between controller objects when this stored event object is reached, and hence a derived event from this object is notified to a remote controller object. This is done by invoking the *notify_event* method on a remote controller object which has the synchronization event as argument.

We give in the following figure 4 an example of the definition of these methods in the controller object interface using IDL. This example is relative to the multimedia services of the context of multimedia services exposed in the second section.

```
// IDL
enum EventType {text, video, audio}; // This can be defined with the semantic of the application
struct EventSync {   // synchronization event
      long  sync_id;              // identifier of the synchronization event
      EventType  sync_type;  //  type of the synchronization  event
};

interface controller_object{ //  recative object
        ......                                                          // invoked by active objects (devices)
        oneway void notify_event (in EventSync event ;                  // invoked by a client object
        oneway void report_event (in EventSync event);                  // for a client object invocations
        oneway void request_state (in SERVICE_STATE to_state); // invoked by a client object
};
```

**Figure 4. Synchronization event methods and controller object interface**

The progression space of event synchronization objects can be of two dimensional coordinate space to solve some conflicts that may occur when we have two different synchronization objects with the same first coordinate space. In our example, when the first coordinate space represents the number of media objects the second can be relative to the time elapsed from the starting point of the application session. Hence, the controller object can distinguish between different events having the same first coordinate space (i.e. having the same number of media objects) when notifying them to remote recipient objects (e.g. client/Applet objects).

Figure 5 shows the computational environment of a controller object.

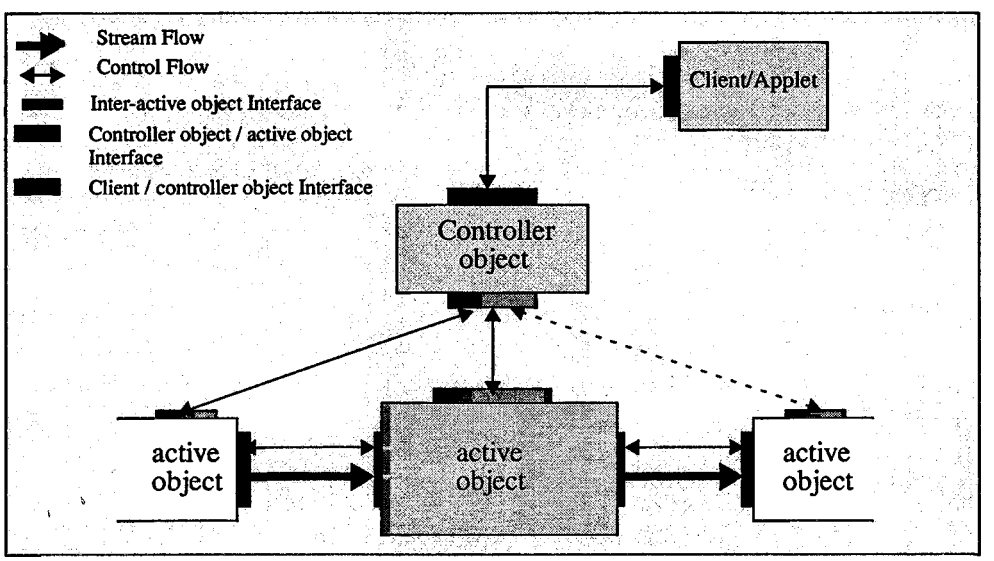


**Figure 5. Computational environment of a controller object**

We identify in this computational environment an additional type of interface between the controller object and the client/applet object. This interface allows to control the provided multimedia service by imposing a particular service state (start, pause, save_revival_point, etc...). In addition this interface

offers the capability of reporting and notifying synchronization events between the client/applet and the controller object. In the next section we apply the proposed model to implement two multimedia services incorporating CORBA objects and that are relative to our context of application.

## 4 Applying the Model: Asynchronous Interaction for a teleteaching session

The proposed object programming model is applied to construct two main distributed multimedia services : the multimedia authoring service and the multimedia presentation service. These services are complementary and provide to user an efficient asynchronous interaction service that handles composite multimedia object flows. These objects consist of two main flows which are X-protocol object flows and hypertext object flows. In the following we present the authoring multimedia service.

The multimedia authoring service allows a user to record his X-window application session and to add comments on it. The recording of an X-window session is done by a recording system which the user can interact through a WWW-based Java applet. Figure 6 shows the CORBA architecture of the authoring system.

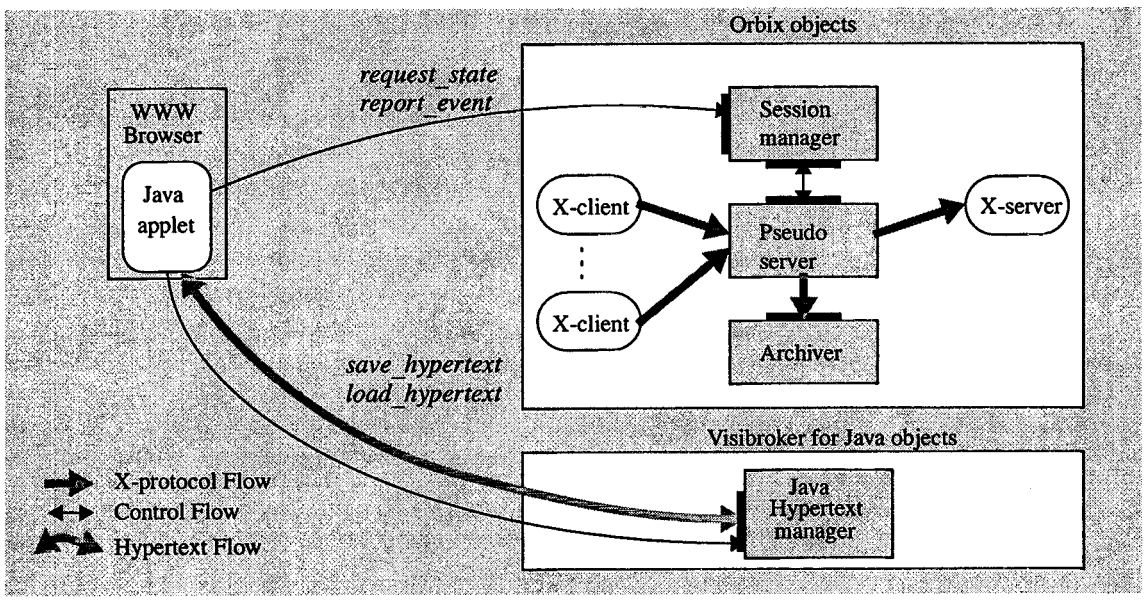


**Figure 6. CORBA architecture of the distributed multimedia authoring system**

The authoring system consists of a distributed cooperating objects and comprises three major components: The X-protocol recording system, the hypertext manager server and the Java applet user interface component. The X-protocol recording system consists of three CORBA objects implemented in the orbix programming environment (Orbix for C++ from Iona Technologies) [Iona96] and they are the session manager object, the pseudo server object and the archiver object. The session manager object is a controller object that accepts user real-time interactions and processes them in a consistent way regarding the pseudo server state. User interactions are processed as CORBA requests and consist of requesting states. The session manager provides an interface for event synchronization reporting that allows a synchronization between X-protocol flows and hypertext flows. The pseudo server object role is to capture X-protocol flows and passe them to the archiver to filter and to store them. The main functionalities of the X-protocol recording system are (thoroughly) detailed in [Gann97].

In order to comment the recorded X-window session, a Java multimedia object server implemented with the Visibroker for Java programming environment (from the Visigenic Software, Inc) [Visi97], provides the applet with a special interface (e.g. *save_hypertext*, *load_hypertext* methods, etc...) to store and update hypertext comments. These comments consist of HTML documents. They should be associated with a special point of a stored X-window session (i.e. stored X-protocol flows). This is achieved, by reporting special synchronization events and hence invoking a *report_event* method on the interface of the session manager object. Therefore, we can later (e.g. in the presentation step) synchronize between the stored X-window flows and the hypertext flows that are both handled by different servers.

The multimedia presentation service is supported by a distributed presentation system that provides to the user with the capability of presenting in a synchronized manner the recorded X-window session and the hypertext comments associated with the different points of this recorded session. Figure 7 shows the CORBA architecture of the distributed presentation system.

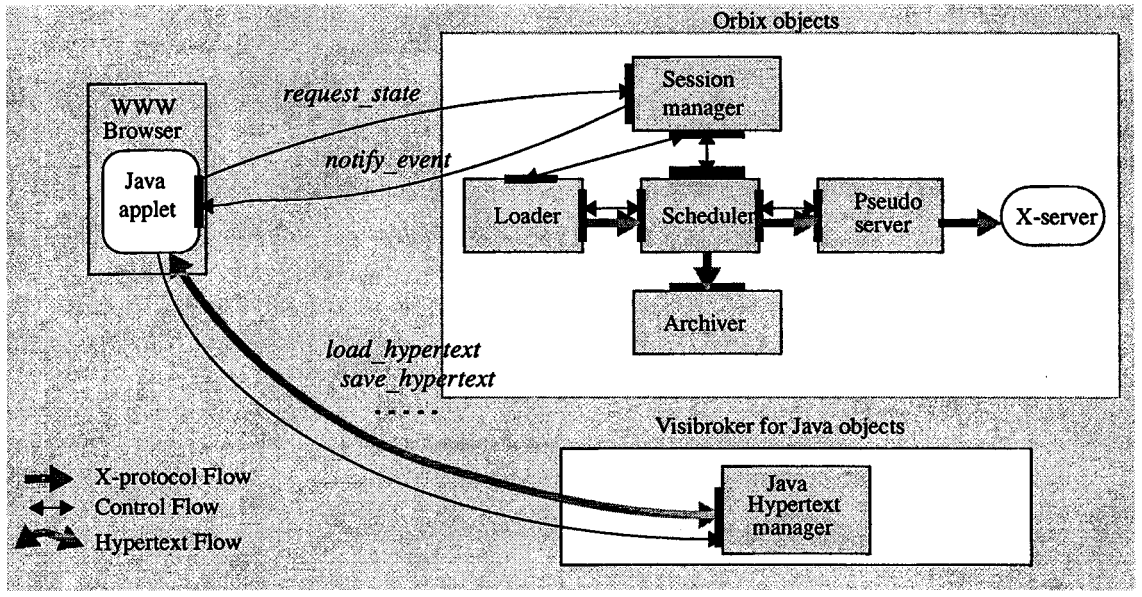


**Figure 7. CORBA architecture of the distributed multimedia presentation system**

The presentation system consists of three major components which are the X-protocol presentation component, the Java hypertext server manager and the Java applet user interface. The X-protocol presentation component consists of several Orbix objects that cooperate to present in a consistent way the stored X-protocol flows. The session manager object plays the role of a controller object that accepts user interactions (e.g. start, pause, etc...) and if necessary forward them through its control interfaces to the loader and the scheduler objects. The loader loads stored X-protocol flows and passes them to the scheduler that schedules them according to their timestamps. The manager object also notifies the Java applet with synchronization events if a synchronization point is reached (event based synchronization). This notification is supported by a notification object (i.e. Visibroker for Java object) created with the Java/applet and that provides an interface with a *notify_event* method. This allows the applet to display hypertext comments relative to a synchronization point reached in the Orbix manager object. HTML comments are managed by the Java hypertext manager which is a Visibroker for Java object that offers an interface for loading, saving and updating these HTML comments. Finally, the Java/applet plays both

the role of a user control interface for the multimedia presentation service and that of an HTML presentation tool.

The hybrid authoring and presentation system allows a user to switch from a replaying step to a recording step and so on. Then the user can replay a portion of a commented X-session and switch to record another portion of an other X-window application session and then continue replaying the first session. This finally produces a composite multimedia document that consists of different successive portions of both X-protocol flows and hypertext flows. This hybrid authoring system needs a communication between the two manager objects (i.e. controller objects) of the authoring and presentation components of X-protocol flows. This communication allows to add a special synchronization points associated with each switching point. An extended interface of the manager object can be added to support such synchronization between multiple X-protocol flows. This will be done in the future work to support complex multimedia authoring documents.

## 5 Conclusion

In this paper we proposed an object model for distributed multimedia services. Our proposed object model addresses the requirement of an event based synchronization supported by event synchronization objects. The proposed object model also distinguishes between the abstract service states that are visible to the user of the multimedia service and abstract object states specified as internal object behavior states and hidden from the user. The main difference between our model and the previous proposed models is that supported by a CORBA architecture. CORBA objects are then derived from a specified model and can be placed at a special node on a distributed environment.

Finally, we implemented two complementary distributed multimedia services that demonstrate how to use the proposed object model. These distributed services support user interactions and composite multimedia documents that handle two main media flows : X-protocol media flow and HTML document flow. The implementation of these services is done in a CORBA/JAVA programming environment. Our future work, will focus on constructing the hybrid authoring and presentation system based on the proposed model. We will extend this model if required to incorporate a special synchronization objects that can help for a flexible deployement of this hybrid authoring system.

## References


[AW92] H. M. Abdel-Wahab and K. Jeffay. Issues problems and solutions in sharing X-clients on multiple displays. Technical Report TR92-042, University of North Carolina, University of North Carolina at Chapell Hill, 1992.

[Blai96] G.S. Blair, G. Coulson, M. Papathomas, Ph. Robin, J. B. Stefani, F. Horn, and L. Hazard. A programming model and system infrastructure for real-time synchronization in distributed multimedia systems. *IEEE Journal on Selected Areas in Communications*, 14(1):249–263, 1996.

[Blum97] C. Blum and R. Molva. A CORBA-based platform for distributed multimedia applications. In *Proceedings of the Multimedia Computing and Networking, MMCN*, 1997. available at http:// www.eurecom.fr/ blum/.

[Coul96] G. Coulson and D.G. Waddington. A CORBA compliant real-time multimedia platform for broadband networks. In Lecture Notes in Computer Science, editor, *Proceedings of the International Workshop on Trends in Distributed Systems (TreDS)*, volume 1116, Aachen, Germany, October 1996.

[CORBA96] Object Management Group. *CORBA 2.0 Specification*. PTC/96-03-04, available at http:// www.omg.org/, 1996.

[Derm93] G. Dermler, T. Gutekunst, B. Plattner, E. Ostrowski, F. Ruge, and M. Weber. Constructing a distributed multimedia joint viewing and tele-operation service for heterogeneous workstation environments. In *Proceedings of the fourth IEEE Workshop on Future Trends of Distributed Computing Systems*, pages 8–15, 1993.

[Evan97] E. Evans and D. Rogers. Using JAVA applets and CORBA for multi-user distributed applications. *IEEE Internet Computing*, 1(3), May-June 1997. available at: http://www.computer.org/internet/abst.htm.

[Gann97] L. Gannoun, Ph. Dubois, and J. Labetoulle. Asynchronous interaction method for a remote teleteaching session. *International Journal of Educational Telecommunications*, 3(1):41 – 59, 1997.

[Gibb94] S. J. Gibbs and Tsichritzis. Multimedia programming objects, environments and frameworks. *ACP Press, ADDISON-WESLEY*, 1994.

[Herm96] I. Herman, J. Graham, and J. Van Loo. Premo: an emerging standard for multimedia presentation. *IEEE Multimedia*, 3(3):83–89, 1996.

[IMA96] *Interactive Multimedia Association's Multimedia System Services*, 1996. available at: http://www.ima.org/forums/imf/mss/.

[Litt95] T.D.C. Little and D. Venkatesh. The use of multimedia technology in distance learning. In *Proceedings of the IEEE Multimedia Networking, MmNet*, pages 3–17, Aizu, Japan, 1995. IEEE Computer Society Press.

[Mueh95] M. Muehlhaeuser. Cooperative Computer-Aided Authoring and Learning: A system Approach. Kluwer Academic Publishers, 1995.

[Mueh96] M. Muehlhaeuser and J. Gecsei. Services, frameworks, and paradigms fo distributed multimedia applications. *IEEE Multimedia*, 3(3):48–61, 1996.

[Premo96] Information Processing Systems-Computer Graphics International Organization for Standardization. Presentation environment for multimedia objects, part2: Foundation component, ISO/IEC 14478-2:199x(e). May 1996.

[Papa95] M. Papathomas, G. S. Blair, G. Coulson, and Ph. Robin. Adressing the real-time synchronization requirements of multimedia in an object-oriented framework. In *Proceedings of the IS &T/SPIE High Speed Networks & Multimedia Computing 95*, 1995. available at http://www.comp.lancs.ac.uk/computing/research/mpg.

[Iona96] Iona Technologies. The Orbix homepage at Iona Technologies. http://www-usa.iona.com/www/Orbix/index.html, 1996.

[Visi97] Visigenic Software, Inc. The Visigenic homepage at Visigenic Software Inc. http://www.visigenic.com/prod/, 1997.