Multimedia Signaling Service

José Rezende and Serge Fdida

Laboratoire MASI-CNRS
4, place Jussieu
75252 Paris Cedex 05 - France

e.mails: {rezende, fdida}@masi.ibp.fr

Abstract

This paper presents an application programming interface (API) for users to select and activate multimedia services in a multivendor environment. This API hides the complexities and details of handling these complex services from the application developers. An abstract model has been defined that accommodates an extensive set of services involving communication and end-systems. The paper defines several abstractions, describes these abstractions by an object-oriented approach and defines a set of primitives for the manipulation of these objects.

1 Introduction

The emergence of distributed multimedia applications on top of multiservice networks has led to the appearance of an important research area that deals with the control, management and coordination of multimedia activities. The *multimedia signaling service* encompasses this category of procedures and describes the interactions with the control plane in the B-ISDN protocol reference model.

The distributed multimedia applications with their stringent requirements in terms of high performance, service flexibility, multiconnection and multiparty communications and quality of service (QoS) configurability lead to further considerable changes in the conventional mechanisms commonly used for this purpose. Thus, any system supporting multimedia applications must provide the users with an efficient interface to select, compose and activate multimedia services. In a distributed environment, this problem is particularly more difficult since different domains are involved.

The main requirement of any interface supporting these applications is the need to offer different facilities with the necessary flexibility to cope with existing and future services. In addition, it must support a level of abstraction where unnecessary details are made transparent to the application developers. Generally, to satisfy these two requirements is not a trivial problem.

An important number of studies has been carried out in communication and operating system support for distributed multimedia applications over high speed networks. Furthermore, not so much work has been done in the design of an efficient interface to support general purpose applications. The main contributions present in the literature that tackle this problem propose solutions with different points of view.

The work presented in [1] supplies an abstract service description of communication systems supporting multimedia multi-user applications. In [2], a signaling protocol for user applications to invoke multimedia services in the context of a broadband ISDN testbed is presented. The preceding works present as drawback the fact that they concentrate merely on the capabilities of the communication systems. In our point of view, efforts must be done towards an integration of all aspects of the required behavior of the entire multimedia system in terms of service specification. Rather than separating the capabilities of the communication systems from those that are related to the end-systems, we argue for a complete service description, and hence, an unique interface used for this purpose.

In [3], extensions to the ISO's emerging standards for Open Distributed Processing (ODP) to support continuous media applications have been carried out. The disadvantages presented by this work are the dependence from a specific distributed system, any flexibility in providing multiconnection and multiparty communications and problems of performance considering the overhead imposed by the layering of different systems (ODP, operating sub-systems, micro-kernel).

In [4], an open platform supporting multimedia multi-user communications has been defined. It provides an application programming interface (API) that makes available network-provided capabilities and core services to multimedia applications. Nevertheless, it lacks for specification of higher layer functions as inter-media synchronization and presentation. It takes into account only those aspects related to the physical resources for the information transport.

In this work, we define an API to be used in a heterogeneous multivendor environment that provides an application developer with the capability of specifying, setting up, modifying and terminating a complete multimedia session (MM-session). That is, a logical communication topology composed of various channels of different media, application processes and end devices.

In the definition of this programmatic interface, our main concerns are to provide abstractions that remove from the user the burden of coping with complex services and to provide users with a maximum flexibility to invoke multimedia services. To reach this goal, we define an abstract model that takes into account a set of services needed to support a distributed multimedia application. It integrates, at the same level of abstraction, services that belong to different domains.

This paper is structured as follows. In the next section we outline our platform with the services that it provides based on the utilization of an abstract model. Section 3 describes the abstract model supplied by our platform and the objects that compose this model. In section 4, we define a set of primitives provided by our API and propose a framework for group management. Finally, section 5 presents our proposals to extend the abstract model, to provide new abstractions for supporting different applications and our conclusions.

2 The Platform

When of the definition of this platform, our main objective is the specification of a proper application programming interface (API) which provides facilities for supporting distributed multimedia applications and their stringent requirements, during the evolution of heterogeneous network technologies and end-terminals.

Through this API, application developers will be able to specify, set up, modify and terminate MM-sessions. A MM-session represents a multiparty and multiconnection association among a set of users. It spans all the necessary services to support a general purpose multimedia application. In addition to communication services, it encompasses end-system services such as device control, synchronization at presentation level, etc.

The use of the platform by applications is based on the use of a set of abstractions and primitives and it consists of three phases:

Specification phase - an important feature for adequately supporting distributed multimedia applications is the need to provide them with a flexible and unambiguous manner to specify their requirements. In our platform, all requirements are specified by means of abstractions, described later, and through the use of primitives provided by the API. So, in this phase, the application states a logical specification of a MM-session that encompasses the communication topology, end-system components of each involved user, set up policies and additional services needed in the MM-session.

Translation phase - after collecting all the information provided by the logical specification, the platform will accomplish the physical realization of the requirements specification. To carry out the translation of logical objects into real objects, the platform uses the services provided by the underlying system. It means that connections will be established, resources in all levels of the system will be allocated and application context services will be activated. Then, if all is sucessful, a MM-session is created.

Operation phase - in this phase, the platform will provide the application a way to control multimedia devices by using the API. The applications can initialize devices, initiate devices to transmit, define and resize video windows to be used, etc. Moreover, to permit greater flexibility, the platform allows the application to modify existing MM-sessions by changing their logical specification.

3 The Abstract Model

In order to provide the application with the ability to specify its requirements without concern with details and complexities of the underlying system, we have defined a set of abstractions. These abstractions have been grouped in that we call *abstract model*. It has as main objectives both to integrate, at the same level of abstraction, the different services needed by the applications and to free the application programmer from concern about their physical realization. These services, related below, encompass different functional domains that render their integration difficult.

In the domain of operating system support, services for media generation and presentation must be provided. These services are related to the control of multimedia devices and the manipulation of multimedia data. They include device drivers needed to control multimedia devices, services required for the synchronization of multiple media at presentation level, conversion functions in communication among heterogeneous terminals and specific services such as windowing capabilities for video service.

In the communication support domain, the model must provide flexibility in setting up and dynamically modifying complex multiconnection/multiparty associations. Additionally, end-to-end QoS configurability, dependency relations among logical connections and data presentation compatibility are other important features needed in this model.

In the application context domain, the application must be able to create and activate *multimedia services*. Some application dependent services are needed in most multimedia applications such as the floor control and group management services [5].

To define the abstract model we have used an object-oriented approach in order to obtain modularity and encapsulation in the definition of each service. Another advantage of this approach is the possibility of re-using pre-defined objects which relieves application developers of the burden of creating all objects that characterize its requirements. In this way, they just have to gather pre-defined objects to determine their own requirements. In addition, extensions to the model for supplying new services are straightforward in the object-oriented approach.

Our abstract model is composed of objects that are used as building blocks for specifying the services required by a MM-session. These objects are viewed as logical abstractions and are composed of their attributes, interfaces and relations with others objects. Therefore, the specification of a MM-session is realized by both the objects' instantiation by setting the appropriate parameters and attributes, and the assignment of these objects to indicate the existing relationships among them.

Two types of services are identified in this model: the communication and the end-system services.

3.1 Abstractions of communication services

The four objects described below allow application programmer to specify their needs in terms of communication services by their logical specification. The objects *Link*, *LinkAccess*, *QoSLink* and *Relationship* are described.

3.1.1 Link

Link is an object that defines an unidirectional logical association among a set of user endpoints for a given service. Thus, it is typed by the kind of media that it carries (audio, video, graphic, data, etc.). It is related to the transport capabilities of an association. But, it does not define an end-to-end transport connection by itself, since the access capabilities are not defined by this objet. The choice of unidirectional associations is due to the asymetric requirement of most applications. In order to obtain a bidirectional communication two *Links* are necessary.

3.1.2 LinkAccess

This object defines an access point of an user interface that sends or receives information according a specific encoding format, e.g. NTSC and PAL-M for a video service. It is typed by the user address, the direction of information flow (sink or source) and the type of information that it generates or receives. When assigned to a *Link* object an end-to-end transport object is defined.

The mismatch of the encoding format between the information treated by source and sink *LinkAccess* objects provides evidence for the need of data conversion. Presentation transparency requires that the involved applications should not have to handle this conversion and should not even need to know about the conversion and the location of the converter. Thus, the separation between access capabilities (*LinkAccess*) and transport capabilities (*Link*) allows conversion functions be inferred from the attributes of its incidents edges.

3.1.3 QosLink

Instantiation of *QoSLink* object allows the application to specify the quality of service associated with *Link* object. A vector of QoS parameters is used for this purpose. It is used for QoS negotiation between peer users, network resource allocation and monitoring, depending on the facilities provided by the underlying system.

An example of a *QoSLink* object is depicted by the figure 1. In spite of the fact that it is not so much suitable for an user-level interface, we argue that this is not a responsibility of such an interface. Though we could define an user-level interface where users could specify their requirements in a subjective way which would be translated to a *QoSLink* object.

In the translation phase, the *Link*, *QoSLink* and *LinkAccess* objects will be mapped onto end-to-end transport connections according to the parameters specified in these objects.

```
typedef enum{negotiated,non_negotiated} QoSType;
typedef enum{best_effort,deterministic,statistical} CommitmentType;
class QoSParameter{
  int    compulsory_value;
  int    threshold_value;
  int    maximalquality_value; };
class QoSVector{
  QoSParameter        delay;
  QoSParameter        jitter;
  QoSParameter        error_rate;
  CommitmentType      commitment; };
class QoSLink{
QoSLink(QoSVector,QoSVector,QoSType);
void    assign_to_Link(Link*,...);
void    unassign(Link*, ...); };
```

Figure 1: QoSLink object definition.

3.1.4 Relationship

The instantiation of this object allows the application to specify the existing relationships among a set of *Links*. The specification of these relationships is very important to improve both the service provided by the communication infrastructure and the utilization of resources.

We identify three types of relationships: *synchronization*, *static* and *dynamic*. The *synchronization* relationship encompasses *Links* with temporal synchronization requirements. The *static* relationship establishes a relation of existence among the *Link* objects. It relates *Links* whose realization depends on the other *Links*. Finally, the *dynamic* relationship establishes a relation of activation of *Link* objects in the time.

In the *synchronization* relationship, the application may choose within three types: intra-stream synchronization; inter-stream synchronization (e.g. lip-sync); multidestination synchronization.

3.1.5 Example of use

To exemplify the use of the objects defined above and show the flexibility permitted by the model, we provide an example depicted by the figure 2. It shows an audio/video communication performed by the user A to the user B. The encoding format used for audio is G722 and NTSC for video. To each *Link* object we have a QoS specification that depends on the kind of information that it carries; also two *Relationship* objects are specified, one that indicates the need of an inter-stream synchronization (lip-sync) between the two *Links* and another that indicates, for example, that the existence of the audio *Link* is dependent of the existence of the video *Link* and vice-versa.

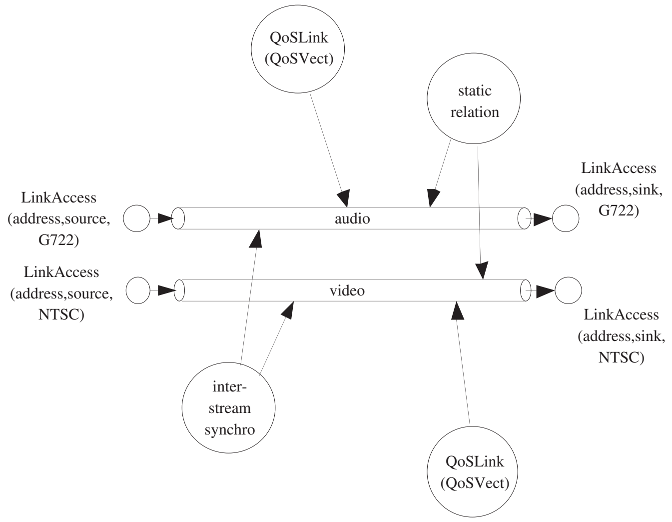
3.2 Abstractions of end-system services

The abstractions provided below are related to the specification of the services performed in the workstations as well as mainframes which have multimedia devices attached to them. They are associated with the processing, storage, generation and presentation of multimedia data. The *MMDevice*, *DeviceAccess*, *SoftHandler* and *ScriptPresentation* objects are described.

3.2.1 MMDevice

This object is an abstraction of physical devices that represent either sources or sinks of multimedia data. To each device is associated a device description that contains all information with respect to the use of the device, including addresses of device drivers and presentation information [6]. By the instantiation of this object, the operations allowed to be executed by the application will be available in the interface of this object after the translation phase. These operations will be mapped, by the device driver, into specific control sequences as required by the attached device.

Thus, the *MMDevice* object allow application programmers both to specify the devices that will be used in a MM-session, and to control these devices during the execution of the MM-session by the APIs provided by them.




```
video = new(Link(video));
audio = new(Link(audio));
v_accessA = new(LinkAccess(userA,source,NTSC));
a_accessA = new(LinkAccess(userA,source,G722));
v_accessB = new(LinkAccess(userB,sink,NTSC));
a_accessB = new(LinkAccess(userB,sink,G722));
NTSC_QoS = new(QoSLink(HiQualNTSC,LoQualNTSC));
G722_QoS = new(QoSLink(HiQualG722,LoQualG722));
LipSync = new(Relationship(inter_stream,
                           ParamSync));
StaticRel = new(Relationship(Static,StaticScript));

v_accessA->assign_to_Link(video);
a_accessA->assign_to_Link(audio);
video->assign_to_LinkAccess(v_accessB);
audio->assign_to_LinkAccess(a_accessB);
NTSC_QoS->assign_to_Link(video);
G722_QoS->assign_to_Link(audio);
LipSync->assign_to_Link(video,audio);
StaticRel->assign_to_Link(video,audio);
```

Figure 2: Example of use of the abstractions for the communication services.

3.2.2 DeviceAccess

This object is an abstraction of the access point to which a multimedia device may be attached. Instantiation of this object provides a way for the application programmer to define a termination point to which will be associated input or output buffers for the storage of multimedia data before its emission or presentation. It is typed by both the direction of flow and the kind of information that it handles.

The assignment of a *MMDevice* object to a *DeviceAccess* object denotes the attachment of a device to a network access channel and allows the system to reserve the corresponding resources. During the operation of a MM-session the programmer may map and unmap the *MMDevice* object to and from the related *DeviceAccess* in order to share the access channels among multiple concurrent MM-sessions.

3.2.3 SoftHandler

This object is an abstraction of a software process that actively manipulates real-time continuous media data in an operating system environment. It may be viewed as an operating system thread acting as component of a flow, and may represent user processes specially elaborated to handle multimedia data. Instantiation of this object provides a clear way to make known an application code that handles data and must be taken into consideration not to jeopardize the end-to-end QoS specified.

3.2.4 ScriptPresentation

This object allows the application programmers to define scenarios to the presentation of multimedia data on the corresponding device taking into account the synchronization problems related to the multiple media presentation. From the specification of these synchronization scenarios the system is responsible for computing a temporal profile for the synchronization scheduler taking into account the information about the characteristics of the network. It is associated with the *DeviceAccess* objects in order to carry out the synchronization at the presentation level of the devices attached to them.

4 Set of Primitives

In order to allow the application programmers to use the platform, we have defined a set of primitives (table 1). It allows the application to specify, set up, modify and terminate MM-sessions. In addition to these primitives, the platform provides primitives to handle multimedia devices. They will be available in the *MMDevice* object interface, after the translation phase.
The *specify_MMsession* primitive allows the user to specify all the components of a MM-session. This specification will be done in a procedural language format. It is composed of three blocks and its structure is shown in the figure 3. The first block specifies each user involved with its components and the communication

primitive	fonction
specify_MMSession	requirements specification of a MM-session
setup_MMSession	set-up a MM-session already specified
modify_MMSession	modifiy an existing MM-session
terminate_MMSession	end a MM-session

Table 1: Set of primitives.

topology that will be used in the MM-session. The setup block specifies the policies to be used in the realization of the specification requirements. The additional services block specifies the extra services needed in the MM-session.

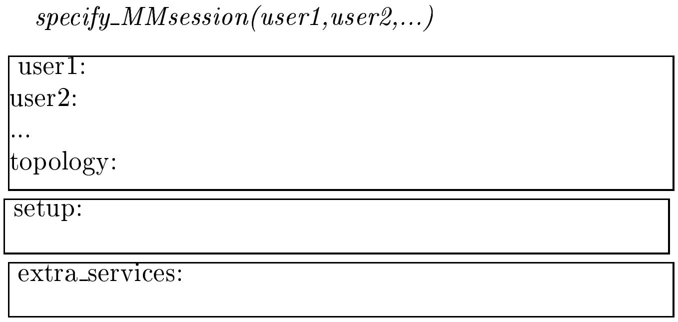


Figure 3: Structure of *specify_MMsession* primitive.

5 Future Work and Concluding Remarks

In this paper we have described an object-oriented abstract model for the specification and operation of multimedia applications. We believe that appropriate APIs are a key issue to provide multimedia services. Emphasis was done on the integration of different services of different domains in a single platform. We have shown that the abstract model defined permits a greater flexibility in specifying a MM-session in terms of both communication and end-system services. The use of an object-oriented approach permits us to achieve the greatest modularity on the specification of each service, and supports the re-use of pre-defined objects which eases this task.

In the future, we will consider the structuration of the defined abstractions to provide higher level abstractions, since we believe that in order to hide all implementation-dependent aspects, the simplicity of the interface is of utmost importance.

References

[1] G. J. Heijenk, X. Hou, and I. Niemegeers, "Communication Systems Supporting Multimedia Multi-user Applications," *IEEE Network Magazine*, vol. 8, pp. 34–44, Jan. 1994.

[2] S. Minzer, "A Signaling Protocol for Complex Multimedia Services," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 1383–1394, Dec. 1991.

[3] G. Coulson, G. S. Blair, N. Davies, and N. Williams, "Extensions to ANSA for multimedia computing," *Computer Networks and ISDN Systems*, vol. 25, pp. 305–323, Apr. 1992.

[4] M. Arango, L. Bahler, P. Bates, M. Conchinwala, and D. Cohrs, "The Touring Machine System," *ACM Computer Communication Review*, vol. 36, pp. 68–77, Jan. 1993.

[5] L. A.Crutcher and A. G. Waters, "Connection Management for an ATM Network," *IEEE Network Magazine*, vol. 6, pp. 42–55, Nov. 1992.

[6] R. Steinmetz, H. Schmutz, B. Schoner, and M. Wasmund, "Generic support for distributed multimedia applications," tech. rep., IBM European Networking Center, Heidelberg, Germany, Nov. 1989.