Proceedings of the 5th Open Workshop on High-Speed Networks, Paris, March 1996

A CORBA-Based Connection Management Scheme for a Multimedia Platform with Stream Configuration Support

Extended Abstract

Marcus Schmid and Christian Blum
Institut Eurécom
2229, route des Crêtes
F-06904 Sophia Antipolis
{mschmid,blum}@eurecom.fr

1 Multimedia Service Provision

We present a multimedia platform that addresses the problem of how to provide multipoint multimedia services to a large community of users, with these users being for instance private households or the employees of an international enterprise. The design decisions taken for our platform are based on the following three assumptions about future multimedia services:

- there are many services offered in parallel
- most of the services have short life-cycles
- there are many service providers

The amount of services offered on the Internet today gives a hint about the service diversity that tomorrow's multimedia networks will have to support. This does not only concern the retrieval and other man-machine services that will be offered in first generation residential networks, but also the multipoint communication services that will come up as soon as the subscriber lines become symmetric. The number of services will be enormous, and services will have short life-cycles. They will be rapidly developed and deployed, and once deployed they will be constantly improved up to the point where they become obsolete. Service diversity can only be attained in a competitive environment where service provision is open to everybody. The first service providers in residential cable networks will be the network providers themselves, but it can be foreseen that they open their networks to third parties in order to offer a richer variety of services to their customers.

Service provision in a network requires the existence of standard terminal equipment at the user's premises. This is just as true for multimedia service provision as it is for comparatively simple telephony. A crucial problem is the distribution of service intelligence between terminal and network. One might be tempted to think that applications will reside and run on end-systems, and that the only service required from the network is connectivity. The problem with this approach is that it limits service diversity simply because people would need to install on their endsystem about every application they possibly want to use. Another possibility would be to have application servers in the network from which user endsystems can download the executables they need to run a certain application. This approach has the problem that the application, once developed, has to be ported to every possible endsystem architecture, and that the application server would need to store all of these executables. The approach we chose is to distribute application intelligence between the network and the endsystems, as will be explained in the following section.

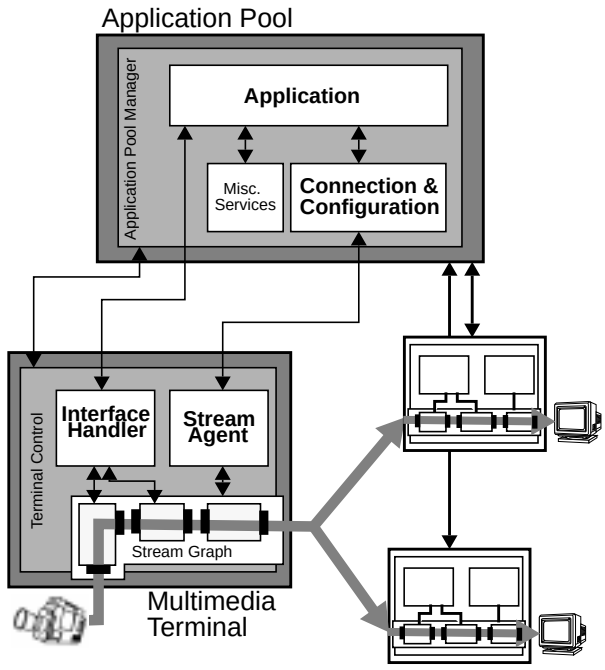

Figure 1. The APMT architecture.

In order to support service diversity, networks must offer a standard framework for the integration of services. Services are implemented on top of high-level programming interfaces that hide network complexities. One such interface would be a connection management support for multi-user applications that typically exhibit complex and dynamically changing connection structures.

The following section will give an introduction into our architecture. The rest of the paper will concentrate on the connection management aspects of this architecture.

2 APMT: Application Pools and Multimedia Terminals

In our architecture [1][2], applications reside on *application pools* inside the network and are accessed by means of *multimedia terminals*. An application will download scripts into the terminals that serve as intelligent sensors for the application and that deal with every issue that is local to the terminal. Connections among the terminals that participate in a multipoint application are established by a central connection manager within the application pool that acts on behalf of the application.

The application pool must be considered as a center of control, and will rarely be the source or sink of media data. Media acquisition, transmission, processing and presentation is performed by standard hardware and software devices within the terminals. The application controls the devices and receives the events that they generate. A terminal can activate a certain application only if it has the devices that the application requires. The architecture of both the application pool and the multimedia terminal is device independent, i.e., new devices can be introduced without any modification of the major building blocks of the architecture.

The complete architecture is based on CORBA [3], and is thus defined as a set of IDL interfaces. There are roughly two kinds of interfaces: interfaces that define the interaction between terminal and application pool, and interfaces that are internal to application pool or terminal. The internal interfaces of the application pool must be known by application developers, the

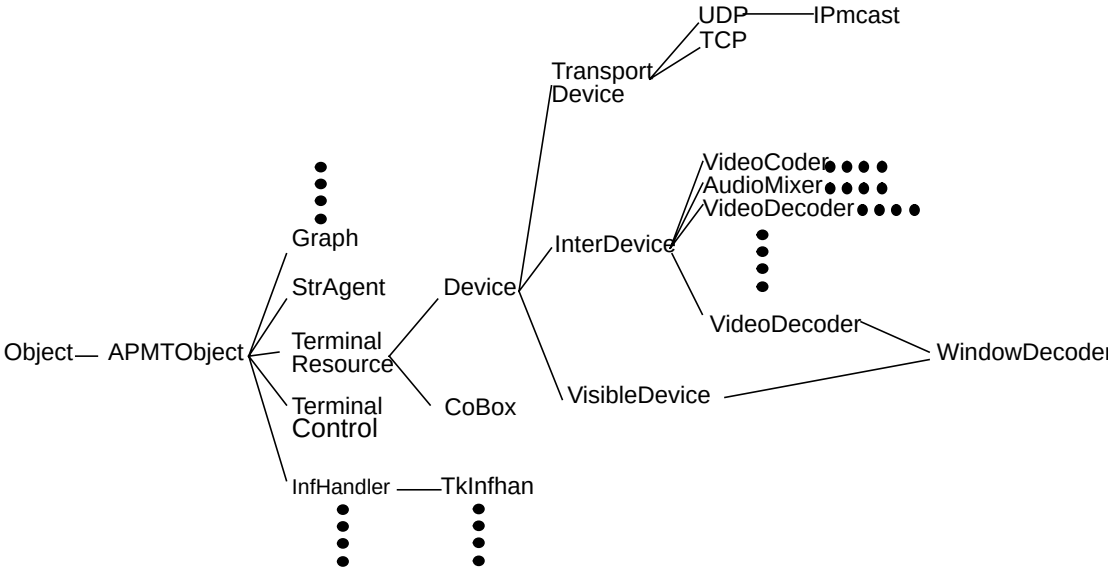

Figure 2. Terminal interface inheritance diagram.

ones in the terminal by device developers, and the interfaces between terminal and application pool by both application and device developers. Internal interfaces may need to reflect hardware and operating system particularities and are therefore system dependent. It is evident that external interfaces must be unique.

Fig. 1 shows the major components of the application pool and multimedia terminal along with control and media flows. For convenience, we will refer to our architecture as the APMT (Application Pool - Multimedia Terminal) architecture.

3 Multimedia Terminal

The multimedia terminal is defined as a set of IDL interfaces. Any computing environment that provides implementations of these interfaces and that is accessible via a CORBA compliant object request broker may act as a multimedia terminal. The interface inheritance tree defining the multimedia terminal is given in Fig. 2. The main interfaces of the interface inheritance tree appear after the general `APMTObject` interface:

Terminal Control The central interface of a multimedia terminal is its terminal control. Its object reference represents the address of the terminal. The terminal control receives invitations from applications and accepts them on behalf of the user. It allows to launch or join applications in the application pool. Every application access to the terminal is supervised by this central control entity.

Stream Agent The stream agent interface provides methods for application-specific configuration of media processing on a terminal. Media processing is done in *graphs.* The nodes of a graph are media processing devices, whereas the edges of a graph represent the data flow in-between these devices. When an application wants to create a graph, it passes a graph description to the stream agent which will instantiate and link the requested devices. The set of available devices reflects the media processing capabilities of a terminal.

Interface Handler The local representative of an application on a terminal is a script that generates a graphical user interface. An application accesses the interface handler to download its specific script. The script interprets user input and either executes the desired actions locally or forwards the user input to the application.

Terminal Resources Those interfaces whose implementation needs resources for transporting or processing media streams are contained in the subtree starting at the node `TerminalResource`. The basic media processing unit on a terminal is the *device*. A device is a standard abstraction for a hardware component or some processing functionality that is implemented in software. The media stream enters or leaves a device at its *ports. Connectors* link device ports, and linked devices form a graph. Connectors may be unicast or multicast. *Connector Boxes* contain a set of connectors that can be activated or deactivated. This allows to dynamically control the flow of a medium stream into graph branches.

Each terminal resource interface must offer at least the methods `activate()` and `deactivate()`. These two methods control the acquisition of a terminal resource: a deactivated terminal resource exists only as a computational object and does not allocate any media transport or processing resources. As the terminal resource is activated, it tries to acquire the needed hardware device, the CPU time, transport facility etc. Access to resources is granted by a terminal resource manager. An active device processes the data streams entering at its input ports and conveys the results to its output ports. An active connector transports the media stream in-between the ports that it connects.

A graph as a whole offers an interface which is instantiated by the stream agent together with the graph's devices and connector boxes. This interface allows for run-time graph modification and overall resource acquisition control. It is assumed that a graph is operational only if all of its elements are operational. Resource acquisition on graph level is therefore atomic, i.e., the `start()` method of the graph interface either activates all its terminal resources or none of them.

The network to which a terminal is attached is reflected by the available *transport devices.* Together with the respective peer transport device, they perform media stream transmission in-between terminals. To assure interoperability among terminals, a standard exchange format for the transmission is defined for every medium type. Interdevice communication that is not of interest for the application and that does not need a reliable transport service is done by attaching *stream attributes* to a packet of the media stream. An arbitrary number of stream attributes may be associated with a stream. The transport device is also in charge of merging this in-band control information to the media stream at the sender side and extracting it at the receiver side, respectively.

4 Connection and Configuration Module

The connection and configuration module (CCM) is an auxiliary application in the application pool. It was designed to assist multi-user applications in establishing and maintaining complex, dynamic connection structures among terminals. An example for such a multi-user application is the teleconference: a teleconference features a small set of audio and video sender and receiver graphs that are instantiated on every implicated terminal and that are interconnected via numerous network connections that have to be established and reconfigured as participants join or leave the conference. The CCM API aims at shielding this application class from directly dealing with the network and with terminal configuration. In the following we introduce the main abstractions of the CCM API.

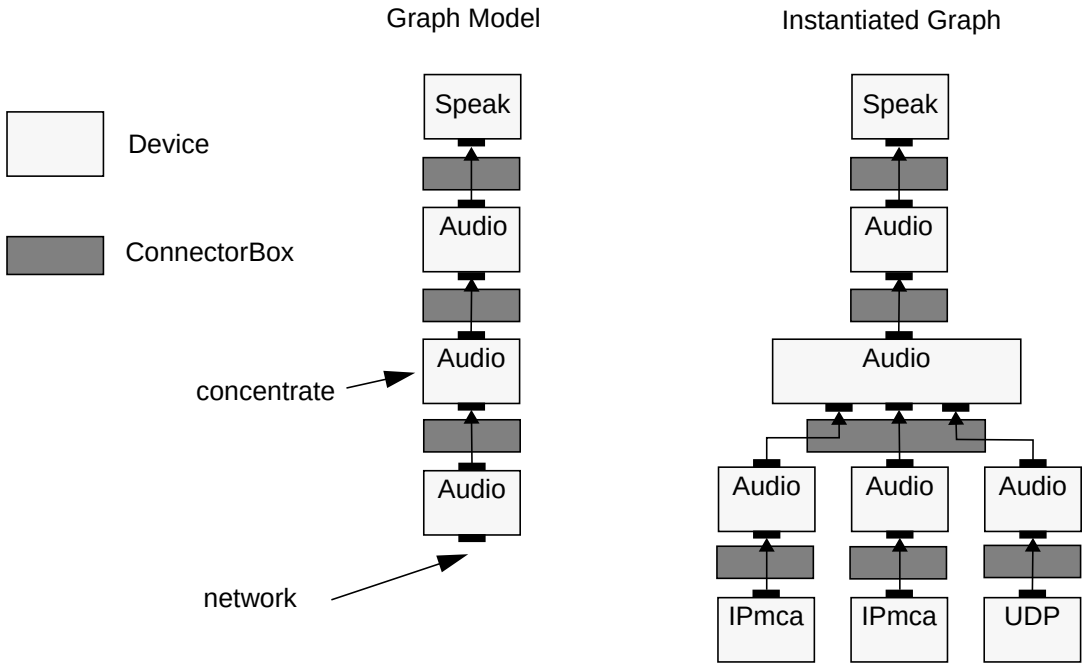

Figure 3. Relationship graph model and instantiated graph

Terminals are grouped into *subsets*. Among the members of a subset, an arbitrary number of *bridges* may be established. Bridges are fundamental connection structures whose vertices are graphs. The CCM supports *simplex*, *duplex*, *multicast*, *all-to-all* and *all-to-one* bridges. A bridge creation request must specify *graph models* for the graphs that are to be created at the vertices of the bridge. Three points have to be considered when transforming the model to a graph:

1. The sink vertices of all-to-all and all-to-one bridges receive media streams from all source vertices and must know how to concentrate them. A receiver graph model must indicate which device performs this concentration. The subgraph that is attached in up-stream direction to the concentrating device represents the device network that copes with one incoming stream. Therefore, the entire graph has as many of these subgraphs attached to the input ports of its concentrating device as there are incoming streams. If a receiver graph model does not indicate a concentrating device, the graph consists of as many copies of the graph model as there are incoming streams.

2. Graph models do not contain transport devices, but they indicate an open device port to which the transport device is to be attached. The choice of the transport device is up to the CCM. The central CCM knows which transport devices are available at each terminal. Furthermore, it is aware of the network structure and of the transport services that this network offers. Based on this knowledge, it attaches appropriate transport devices.

3. Media streams are typed and ports know about the medium type flowing through them. When an application registers a graph model with the CCM, it sets the medium type only at a subset of the ports within the graph. The medium type of the remaining ports has to be determined by the CCM. Of particular interest is the medium type at the network port of a graph model. Since the QoS requirements of a media stream can be deduced from its type, the CCM may reserve network resources for the transmission of a stream.

Fig. 3 shows on the left side a graph model for audio reception and on the right side the graph that will be instantiated when this graph model is a sink vertex in an all-to-all or all-to-one bridge over four terminals.

The CCM API is specified in IDL and accessed via the omnipresent object request broker. Usage of the API will mostly occur in two phases. In the initialization phase, graph models are registered and terminal subsets are formed. During the operational phase, bridges are created, modified and removed. In this phase, the CCM changes connections structures by creating, deactivating and reactivating graphs.

5 Prototype

A prototype of the APMT platform including the CCM has been implemented at Eurécom. The prototype uses Sun Sparc10 workstations as multimedia terminals and application pools. The ORB is Orbix ™from Iona Technologies. Media transport is based on UDP and IP multicast over an ATM-LAN. On the top of the CCM, a videoconference application has been implemented that highlights the most important features of the architecture.

References

[1] C. Blum, R. Molva and E. Ruetsche,"A Terminal-Based Approach to Multimedia Service Provision", in *Proceedings of the 1st International Workshop on Community Networking*, San Francisco, July 1994.

[2] C. Blum and R. Molva,"A Software Platform for Distributed Multimedia Applications", to appear in *Proceedings of the 1st International Workshop on Multimedia Software Development*, Berlin, March 1995.

[3] Object Management Group,"The Common Object Request Broker: Architecture and Specification", John Wiley & Sons, Inc., 1992.

The articles [1] and [2] can be retrieved from our web server: http://www.cica.fr~blum