Distributed applications adhering to reference models are implemented above a middleware facility. Such facilities offer a set of services as well as the interfaces to access these services. In our implementation, we adopted the Common Object Request Broker Architecture (CORBA) from the Object Management Group (OMG) [8] as a middleware facility.

This paper presents the design and implementation of ODP channels using CORBA. The implementation is generic enough to be incorporated into other applications or serve as a basis for more elaborated software systems such as OMS streams or TINA-C DPs. The implementation separates the control flow from the media flow. Control actions such as pausing and resuming the flow are performed through the Object Request Broker (ORB) by executing the corresponding method in the proper server object. On the other hand, continuous media flow outside the ORB is in a multimedia transport protocol. Quality of service is also addressed in this paper.

Related work is being conducted at the Fokus Institute, Germany, in the scope of TANGRAM project [7], the ReTINA project funded by the European Union [5], and the ACE project at CSELT, Italy [6]. All of these projects are centered in the TINA-C architecture that adopts ODP as a framework for interoperability.

The paper is organized as follows. Section 2 describes briefly ODP channels and the CORBA architecture. Section 3 presents a proposal for implementing ODP channels using CORBA. Section 4 describes our implementation, evaluating it against other multimedia applications. Section 5 addresses the issue of quality of service (QoS). Finally, Section 6 closes the paper with some concluding remarks.

2 ODP Channels and CORBA

ODP models a distributed system according to five viewpoints [5]. The most important viewpoints are the computational and the engineering viewpoints. The computational viewpoint defines the objects that compose the application, while the engineering viewpoint provides the computing infrastructure for interacting objects to execute and interact.

In the ODP computational viewpoint the channel is an object that mediates the communication among other objects: the binding object (BO), Fig. 1. Interacting objects have complementary roles such as producer/consumer, client/server, and so on. The media flow can be unidirectional, point-to-point or point-to-multipoint.

Still in the computational viewpoint, the BO exports an interface for controlling the communication. Operations for starting, pausing and resuming the media flow are typical for this interface.

In the ODP engineering viewpoint the channel is composed of engineering objects belonging to one of the five classes below:

1. stub: object responsible for media processing immediately after capture or before presentation. A stub exports an interface with operations related to media processing. Operations for setting the encoding format, sample rate and encoding precision can be performed at this interface. In practice, such operations are commonly realized by the device driver assigned to the multimedia object (camera, microphone, etc.).

2. binder: object that assures the integrity of the channel. A binder exports an interface with operations related to the management of the channel. Example of such operations are channel destruction and reconfiguration.

3. protocol adapter: object that interacts with the network for transmitting/receiving media segments between the channel endpoints. Protocol adapters are chosen according to the media being transported, quality of service requirements, etc.

4. interceptor: object located in between the protocol adapters and responsible for adapting the communication for a particular domain. A good example of interceptor is a firewall.

5. channel controller: object responsible for the management of the channel as a whole. The interface of this object matches the binding object's computational interface (Fig. 1).

Figure 2 illustrates the channel in the ODP engineering viewpoint. The CORBA Object Request Broker Architecture (CORBA) [9] is a standard defined by the Object Management Group (OMG), a consortium of companies, universities and research institutes with a common interest in distributed object technology. The CORBA architecture is pictured in Fig. 3. The Object Request Broker (ORB) mediates the interaction between a client and a server. The interaction is usually one-to-one and synchronous in nature. The ORB is responsible for locating servers, transferring the parameters of a call, and return the result back to the client.

A server houses a set of objects whose methods can be invoked by a remote client. These objects have interfaces written in the Interface Definition Language (IDL), a language close to C++ and used solely to declare interfaces. The IDL compiler translates (maps) an IDL interface to a programming language, typically C++, Java or Smalltalk. The mapping includes a server template, the client stub and the server skeleton. The server template contains the methods defined in IDL and mapped to the target language.

Such methods have empty bodies, which are supplied by the application's programmer.

Stubs are presentation facilities employed for converting data (parameters and results) in a remote method invocation. Data is converted from the client's internal representation to a canonical, ORB-wide format. The Skeleton does the inverse at the server side.

The Object Adaptor performs server activation and authentication. A server can be activated persistently by a mechanism external to the ORB, or on demand when a invocation arrives and no server able to handle the invocation is active. In the latter case the Object Adaptor performs the server activation by locating an appropriate server and executing it.

CORBA allows static and dynamic remote method invocation. In the static scheme the client knows the interfaces it wishes to use and links the appropriate stubs for them. Dynamic invocation allows a client to discover interfaces at run time and assemble invocations for methods defined in those interfaces.

The ORB Interface is an application programming interface (API) for accessing some ORB functionalities. Manipulation of repositories, object references and transparency services are examples of operations performed through the ORB Interface.