ODP Channel: an Open Mechanism for Supporting Media Flows in Distributed Environments

R.C.M. Prado*
DCA-FEEC-UNICAMP
State University of Campinas
Campinas, SP, Brazil, 13083-970

E. Cardozo
DCA-FEEC-UNICAMP
State University of Campinas
Campinas, SP, Brazil, 13083-970

L.F. Faina
DEINF - CETEC
Federal University of Uberlândia
Uberlândia, MG, Brazil, 38400-902

L.A. Guedes
DEE - CT
Federal University of Pará
Belém, PA, Brazil, 66000-000

Abstract

Today's distributed applications need to interoperate across different administrative domains, handle multiple types of media, and execute on different computing platforms with different processing and networking capabilities. These demands pose several challenges to the application designer. Issues such as interoperability, real-time communication, security and performance must be properly addressed. This paper addresses the issue of multimedia communication across packet switching networks such as the Internet. The paper describes the design, implementation and evaluation of a generic communication infrastructure based on the Reference Model of Open Distributed Processing (RM-ODP). The infrastructure's components are implemented as distributed objects. The implementation can be easily integrated into cooperative applications such as teleconference and telemedicine; serve as a basis for implementing stream interfaces as proposed by the Object Management Group (OMG); or still be a key component of a Distributed Processing Environment (DPE).

Keywords:

ODP Channel, CORBA, Quality of Service, Distributed Multimedia Systems.

1 Introduction

The idea behind open distributed processing is to have a common framework from which distributed applications are built. Examples of such frameworks are the Reference Model of Open Distributed Processing (RM-ODP) from ISO [1], and the Telecommunication Information Network Architecture (TINA) from the TINA Consortium (TINA-C) [2]. Such frameworks favor interoperability by stating precisely the roles of the application's components, the kind of services they provide, how the components interact, which components are mandatory, and so on.

For instance, RM-ODP states how objects in a ODP-compliant application interact. The model describes an abstraction named *channel* that mediates the interaction between two or more objects. The channel has three main objects at each endpoint: one for presentation services (the stub); one for managing its side of the channel (the binder); and the third for interacting

* {prado,eleri,lffaina,affonfo}@dca.fee.unicamp.br

with the network (the protocol adapter). ODP defines precisely which functionalities are provided by the channel for object interaction as well as where these functionalities are to be placed.

Reference models for open distributed processing do not define precisely the interfaces presented by their components. It is the case of ODP channels where the functionalities of each component are specified but the operations that implement these functionalities are not.

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) [3] 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 OMG streams or TINA-C DPEs. 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 segments are transported outside the ORB through 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 TANGRAN project [4]; 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 [1]. 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 these 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. For media flow one object acts as a producer of media segments (the source) and the remaining as consumers of media segments (the sinks). The channel is responsible for linking source and sinks, establishing a unidirectional, point-to-point or point-to-multipoint flow path. 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.).

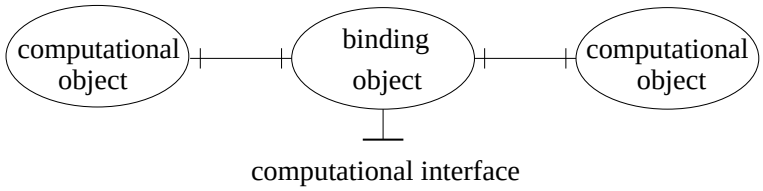

Fig. 1: Computational view of the channel.

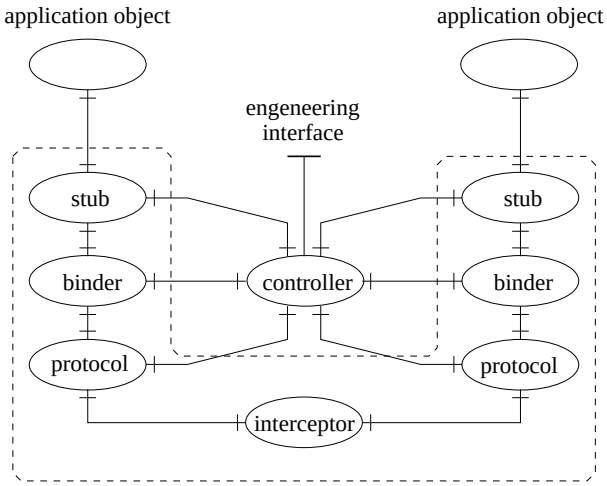

Fig. 2: ODP engineering viewpoint of channel.

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 Common Object Request Broker Architecture (CORBA) [3] 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, upcalling the server, and sending the result back to the client.

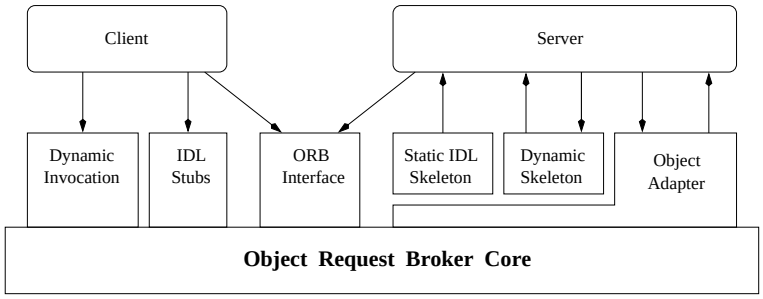

Fig. 3: CORBA architecture.

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 describe interfaces. The IDL compiler translates (maps) an IDL interface to a programming language, typically C++, Java or Smalltalk. The mapping consists of server templates, client stubs and server skeletons. The server template contains the methods defined in IDL and mapped to the target language. Such methods have empty bodies (code will be 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 opposite 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 these 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.

CORBA specifies a large set of distributed services: the CORBAservices and the CORBAfacilities. CORBAservices defines a set of application independent services such are naming, life cycle, security, events and persistence. The CORBAfacilities specifies domain-oriented services such as those supporting distributed documents and information management.

3 The Channel Architecture

In our implementation each channel endpoint is a CORBA server. This server exports IDL interfaces that allow control operations over the channel. These interfaces are assigned with the application object, the binder, and the channel controller. The stub and the protocol adapter do not export IDL interfaces (in other words, these objects are manipulated only within the server). In our current implementation we have no interceptors. Figure 4 shows the channel components that are described in the sequel.

3.1 The Channel Factory

The channel factory is responsible for setting-up a channel endpoint. The factory exports a single operation that receives the configuration parameters for the channel, and, based on these

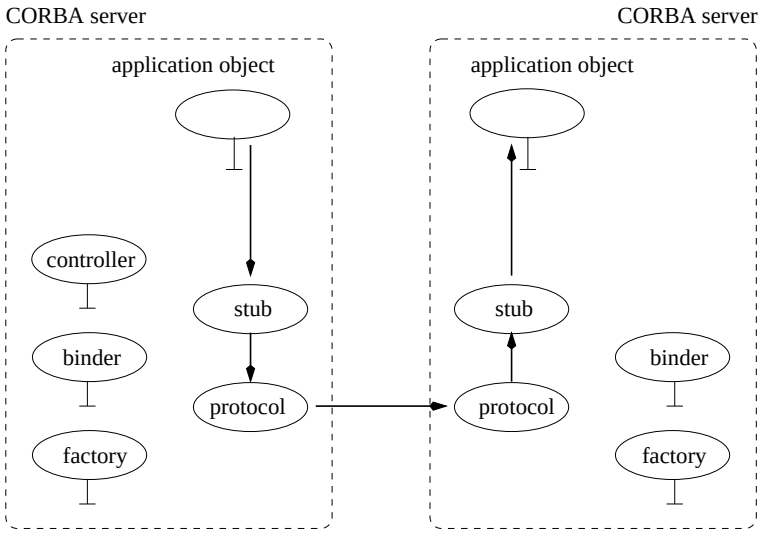

Fig. 4: The proposed channel architecture. Arrows state the direction of the stream flow.

parameters, instantiates the remaining channel components. The parameters passed to the factory are: the channel identifier (a name); the transport protocol (UDP, TCP) and parameters (host and port); the source's transport address (in case of sinks); and the desired level of quality of service (QoS).

3.2 The Application Object

The application object does not belong to the channel, but resides in the same address space of the channel components for reasons related to efficiency. We implemented the basic multimedia objects: microphone, speaker, camera and display.

Typical IDL operations exported by these objects are those related to device control (start, pause, play) and device setup (sampling rate, encoding formats, etc.). Application objects are threads that read/write media segments from/into the stub.

In our implementation, a library helps application objects to assemble RTP (Real Time Protocol) packets encoding media segments [7]. RTP packets are passed to the protocol adapter via stub for transmission to the other channel endpoint(s).

3.3 The Stub

The stub is a passive object instantiated at each side of the channel. It serves as a buffer for data exchanging between the application object and the protocol adapter in a producer-consumer scheme. Worth of mention is the fact that the data stream doesn't cross the binder object, also for reasons related to efficiency.

The stub implements a chain of buffers guarded by semaphores. Application object and protocol adapter access these buffers via two methods: getBufferToWrite, called by the producer and getBufferToRead called by the consumer. These methods enforce a maximum offset in terms of number of buffers between the producer and the consumer. Producer or consumer block if this offset becomes out of range.

The stub exports no interface to the ORB, meaning that its functionalities are available only inside its address space. Moreover, this implementation supposes that all the media processing such as encoding and compression are performed by the multimedia object or by its device driver.

3.4 The Binder

Binders export operations related to channel deactivation, reconfiguration and status reporting. Operations related to flow control (start, stop, resume) are also defined in the binder's interface. These operations are available at this interface because many application objects have no flow control capabilities (start, pause and play operations).

The deactivation operation destroys only the endpoint where the binder is located. Exception to this rule is the destruction of the source endpoint, where all the corresponding sinks are also destroyed. The reconfiguration operations allow a sink endpoint to connect to or disconnect from a source. Status reporting returns the latest estimates of delay, jitter, bandwidth and package error rate at the binder's side.

The binder is implemented as a thread dedicated to the processing of its functions.

3.5 The Protocol Adapter

At the source side, the protocol adapter transfers data read from the stub to its counterparts at the other side(s) of the channel. On the other hand, at the sink side, the protocol adapter reads data from the network and writes into the stub, making such data available to the application object. We implemented an Internet-style protocol adapter that employs UDP to transfer RTP packets transporting media segments. The protocol adapter is identified by a port number and a Internet address (IP number). It can employ both multicast and unicast addresses, being multicast addresses preferable for point-multipoint channels in order to minimize packet duplication in the network.

The protocol adapter is implemented as a thread. The flow control in the channel is performed by creating, suspending and resuming the execution of this thread. Since the channel is asymmetric, the behavior of the protocol adapter depends on the side of the channel (source or sink). Same as the stub, the protocol adapter exports no IDL interface.

3.6 The Channel Controller

The channel controller offers a single point of control for channels. The channel controller keeps a list of all binders in the channel in order to propagate control operations to all the channel endpoints (a changing in the encoding format, for instance).

The channel controller is also a thread inside the channel endpoint (server process). This object exports interfaces for getting the references for the binder objects in the channel; controlling the media flow; and reporting the status of the overall channel.

4 Implementation Details

The ODP channel was implemented in a network of Sun workstations under the Solaris 2.5 operating system. Orbix-MT[1] from Iona Technologies was the chosen CORBA platform [8]. Four classes of application objects were developed: Microphone, Speaker, Camera and Display. These objects define IDL interfaces for flow control (start, pause, play), and for device setup (sampling rate, precision, etc.). For video, the Xil library from Sun Soft [9] is used for capturing and displaying video frames. Cell-B and MJPEG are the two video formats allowed in our Xil version. For audio, the native device driver is employed (/dev/audio). Solaris provides a set of system calls (ioctl) to control this device. Such control allows to set the sampling rate (8 to 44.1 KHz), precision (8 or 16 bits), coding format (linear PCM, u-law and a-law), gain (0 to 255), among other parameters.

The whole application was implemented in approximately 4,900 lines of C++ code, divided as follow: 2,300 lines for the ODP channel, 700 lines for the audio objects (microphone and speaker), 1,600 lines for the video objects (display and camera) and 300 lines for the user-side application.

[1] Multi-threaded Orbix.

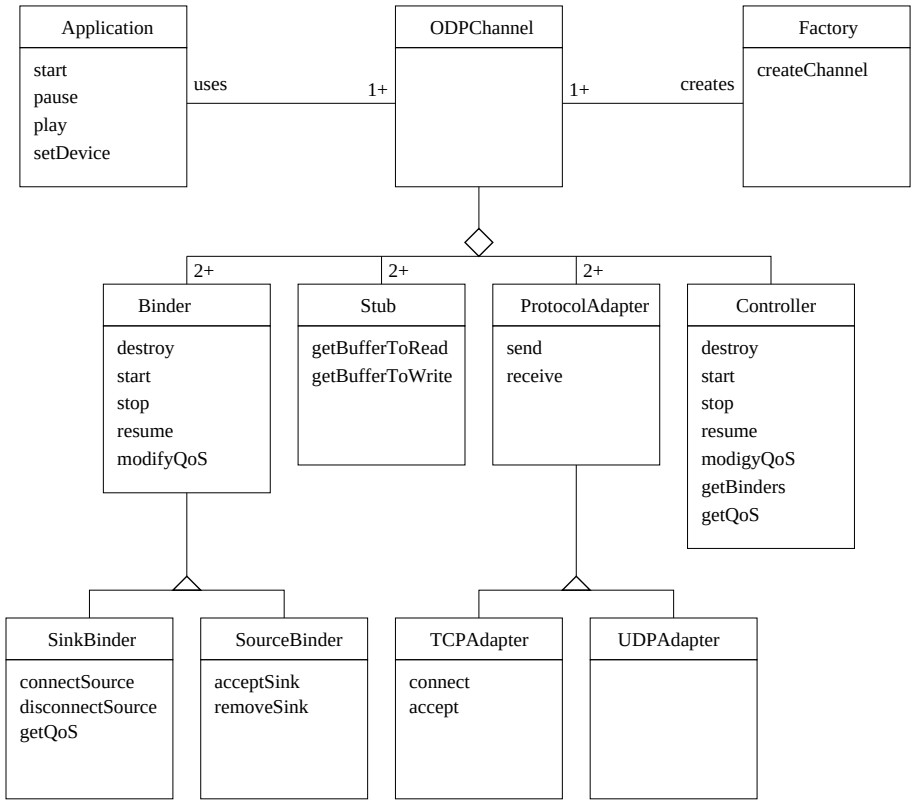

Fig. 5: Channel object classes in OMT notation. Boxes are classes (class name plus methods), diamonds state aggregation (part-of) relationships, triangles state generalization (is-a) relationships.

Notice that only 6% need to be written by the programmer. The code incorporated into the channel and its associated objects is application-independent.

The application objects are linked to a library containing the channel components: factory, stub, binder, protocol adapter and channel controller. Except for the stub, all of these objects are threads within a CORBA server. After the channel has being constructed, the source object (Microphone or Camera) captures a segment of media and simply writes it to the stub. At the sink side, sink object (Speaker or Display) read from the stub and present this segment of media at the corresponding device driver.

Figure 5 shows the channel object classes in a OMT class diagram [10].

4.1 Scenarios of Utilization

Figure 6 shows an application employing two channels for distributing audio and video from a source to a sink (a point-to-point channel). The client assembles the channel at the source side first. Then, at each sink node the client builds the sink side and connects to the source. This connection allows the channel controller to know the sinks connected to the source and where they are located. Activating a side of the channel is just a matter of starting the CORBA server housing the channel plus the application objects. This is achieved by issuing a binding[2] to the channel factory object at the corresponding server (Microphone, Camera, etc.).

In a teleconference application, for instance, point-to-multipoint channels must be employed.

[2] Orbix **_bind** call.

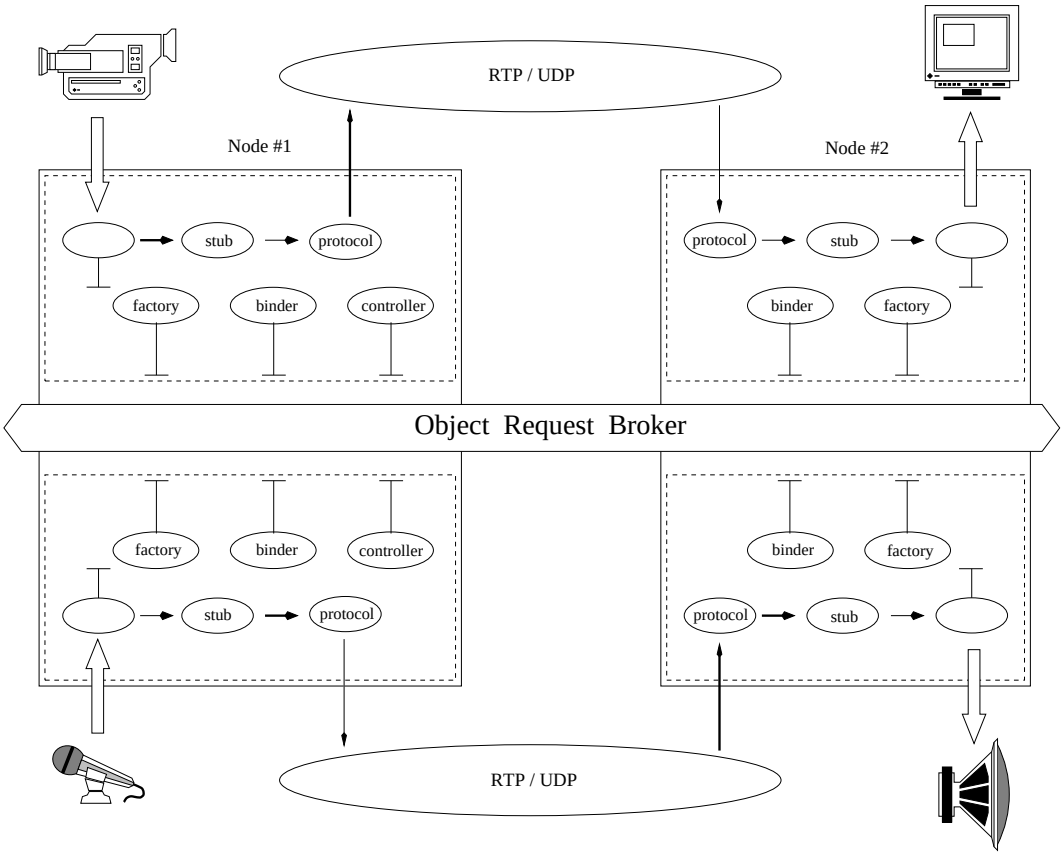

Fig. 6: Distribution of audio and video through point-to-point channels.

A videoconference application builds two source endpoints (one for audio and one for video) and waits the invitation to join from the conference moderator. Suppose the invitation carries the location (node) of all participants. For each participant the invited application builds a video sink (Display) and connects to the participant's video source (Camera). Since video sinks are X Window screens an application can open as many video sinks as the number of sources (see the bottom part of Fig. 7).

For receiving audio from many sources the procedure is slightly different. Since there is a single audio sink device per node (the speaker), all the audio channels having sinks at this node must share the application object connected to this device. An application object at the sink side connected to multiple channels selects what channel it wishes to receive by selecting the proper stub. The upper part of Fig. 7 illustrates this situation. Building another channel endpoint at a given CORBA server is trivial, being just a matter to ask the factory at that server to instantiate a new channel (stub, binder and protocol adapter objects).

Once the participant finishes the joining procedure it notifies the moderator that informs all members the location of the newcomer. At this moment any participant: (i) creates a new video sink and connects it to the newcomer's video source; (ii) creates a channel endpoint at the CORBA server housing the audio sink and connects it to the newcomer's audio source. The ingress procedure is now complete.

ODP channels allow several videoconference policies be implemented directly over the communication infrastructure. For example, the conference moderator may allow one participant at a time to speak. In our implementation this policy is easily realized by the application: except for

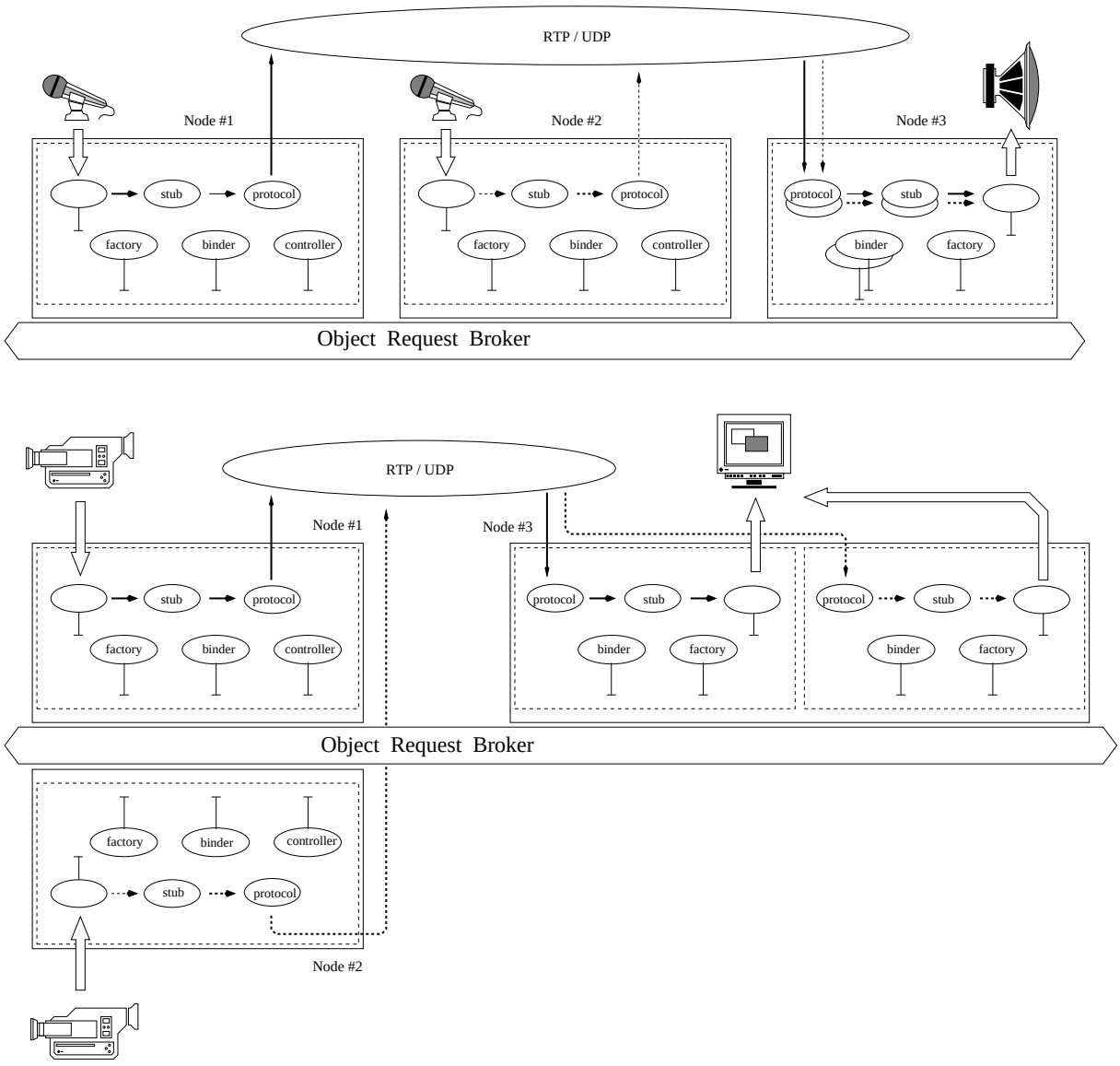

Fig. 7: Multiple audio and video channels ending at a single node.

the speaker, all the audio channels have their flow stopped at the source side (operation stop at source binders).

4.2 Performance Considerations

In order to assess the channel performance two channels (audio and video) with one source and three sinks were established. In the audio channel the application object samples audio at 8 KHz and 8 bits/sample, while in the video channel the application object samples video frames of 640x480 pixels (16 bits/pixel) at rate of 30 frames/s.

Typical values for QoS obtained for the audio channel are shown in Table 1. Notice that the application is not able to transfer the demanded bandwidth (8 Kbytes/s for 8 Khz/8 bit audio and 88.2 Kbytes/s for 44.1 Khz/16 bit audio). This is explained by the lack of a real-time scheduler at the node and a synchronous or isochronous network.

Type	Bandwidth (Kbytes/s)	Delay (ms)	Jitter (ms)	PER (%)
A	7.89	6.5	0.03	0
A	7.38	7.8	0.04	0
B	39.27	8.7	0.06	24
B	41.29	9.1	0.05	18

Table 1: Typical QoS measurements for an audio channel. Audio type A has sampling rate of 8 Khz and precision of 8 bits; audio type B has sampling rate of 44.1 KHz and precision of 16 bits.

The overhead introduced by the channel can not be perceived when compared with the same application build above the plain socket interface. In fact, compared with MBONE tools such as NV (videoconference tool) and VAT (audioconference tool), audio and video transported through ODP channels perform equally as well. We claim that tools build over our multithreaded implementation of ODP channels will perform much better in multiprocessed workstations where true parallelism is achieved when manipulating multiple media flows.

5 Quality of Service Issues

We developed a very elaborated architecture for quality of service negotiation and management based on mobile agent technology [11]. References [12, 13] report such architecture, while references [14, 15] provide a comprehensive survey on QoS for distributed multimedia applications. In this paper we present a simplified scheme based on fixed objects.

During the establishment of a channel, some parameters related to quality of service are passed to the factory. At the moment we have no way to guarantee that these parameters will be honored by the network. At most we can monitor these parameters and take some corrective action when a severe degradation of QoS is detected.

Our current implementation of ODP channels handles the following QoS parameters:

- network delay: the elapsed time between the submission of a RTP packet to the network and its receiving at the sink side;
- delay jitter: the variation of network delay;
- bandwidth: byte flow (bytes/s) from source to sink(s);
- package error rate (PER): the percentage of packets carrying media segments discarded by the network.

These parameters are computed at the sink side only. Jitter and PER are computed using the information carried in the RTP header. Bandwidth is computed simply by counting the amount of bytes received during a time interval. Delay is computed by sending a echo request to the UDP port number 7 (echo port) at the source's node and measuring the round trip time. A weighted-average scheme is employed in order smooth delay peaks observed during a short network congestion.

A global data structure storing these parameters are updated when a certain amount of RTP packets are received. This same structure is accessed by the binder when a QoS report is requested. We implemented a monitoring scheme for keeping track of the QoS in the channel. The monitoring facility can be employed for presentation purposes or for adapting the channel to a particular combination of network speed, application demands and multimedia devices.

The adaptation scheme performs a task similar to that performed by the Real Time Control Protocol (RTCP) [7], that is, to report back to the source how the media segments are being received at the sink side. However, different of the RTCP approach, the monitoring scheme is decoupled from the end-user application, being a part of the communication infrastructure.

5.1 A Simple QoS Management Scheme

Two objects are employed for the purpose of QoS management: a QoS monitor and a QoS adaptor. These objects are not part of the channel, but use the control interfaces exported by the binder and the application objects. For each channel there is a QoS adaptor and as many QoS monitors as the number of sinks. QoS adaptor and monitors can run anywhere, but it is recommended to place them at the same host of the source (QoS adaptor) and sinks (QoS monitors).

A QoS monitor collects statistics from a sink endpoint by interacting with the binder at this endpoint. The period of interaction is a parameter passed to the QoS monitor, typically half second. The QoS monitor checks the QoS parameters against those specified for the channel. If a severe degradation of these parameters is detected the QoS monitor reports to the QoS adaptor. In our application "severe degradation" is defined when the network is loosing more than 25% percent of packages in three subsequent measurements. Surely, more clever schemes that monitor also delay, jitter and bandwidth must be devised.

If the QoS adaptor receives notifications from many monitors reporting QoS degradation, the adaptor concludes that a network congestion is in course. In this case the adaptor degrades the source's QoS parameters[3] while the congestion persists. After an amount of time without receiving notification from the monitors, the adaptor sets the QoS parameters back to their original values.

The rules the QoS adaptor relies are very simple, although sophisticated adaptation schemes based on artificial intelligence techniques can be employed[4]. The rules for QoS degradation act in the following parameters:

- for video: sampling rate (frames per second), frame size (number of pixels) and precision (color or grayscale);
- for audio: sampling rate and precision (bits per sampling).

This adaptation scheme provides a back-off mechanism that acts in presence of congestion (TCP has such mechanism but UDP has not). Contrary to the RTCP approach, the source side receives the reports only when the QoS parameters went out of range. Figure 8 illustrates the QoS management scheme.

[3] By interacting with the application object.

[4] Fuzzy rules may provide the best adaptation behavior.

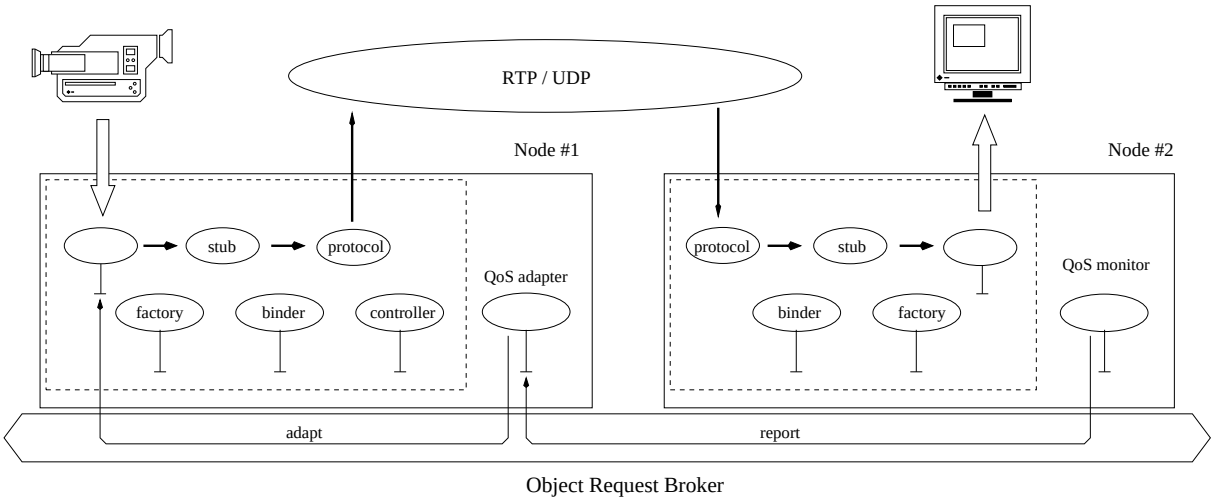

Fig. 8: Quality of service management employing monitor and adaptor objects.

6 Concluding Remarks

ODP channel is a generic infrastructure for multimedia communication in distributed environments. ODP channel is also an important component of the TINA's Distributed Processing Environment (DPE). The implementation reported in this paper is a foundation for other developments in the line of TINA at the State University of Campinas:

- integration of the channel with other ODP components (cluster, capsule, nucleous, etc.);
- implementation of the Service Architecture components;
- methodologies for the development of TINA-compliant telemedia services over high-speed networks.

Other efforts in the direction of distributed multimedia applications can benefit from our channel implementation as well. For instance, channels allow real time audio and video be incorporated into an existing, text-based collaborative environment.

Realizing ODP channels over CORBA gives a very straighforward access to the channel functionalities. This design decision is in line with current trends in distributed computing: the channel is a *component* to be (easily) integrated into an application. In fact, with a generic communication infrastructure at hand, the efforts for developing a distributed multimedia application decreases dramatically.

Finally, some open issues need to be addressed. Quality of service must be taken into account in a more comprehensive manner, considering network infrastructures, application requirements, processing platforms, among other requirements. Another interesting issue is mobility: how mobile agent technology fits into ODP/TINA environments? Issues such as security, management and reliability, can not be neglected in mission-critical implementations.

Acknowledgments

This research is partially support by FAPESP (grant 92/3507-0), CNPq (grant 300723/93-8) and CAPES (which provides PhD scholarships to the first, third and fourth authors).

References

[1] ISO/IEC, *Reference Model of Open Distributed Processing*, Standards 10746-1 (Overview) and 10746-3 (Architecture), 1995.

[2] Telecommunication Information Network Architecture Consortium, *Overall Concepts and Principles of TINA*, Version 1.0, 1995, `http://www.tinac.com/`

[3] Object Management Group, *Common Object Request Broker Architecture 2.0 Specification*, OMG Technical Document PTC/96-03-04.

[4] M.K. Durmosch, K.D. Engel, *The Tangram DPE - A Distributed Processing Environment in a Heterogeneous Corba 2 World*, 30^{th} Hawaii Intl. Conference on System Science (HICSS'30), Maui, Hawaii, Jan 1997.

[5] *ReTINA's WWW Page*, `http://www.chorus.com/Research/retina.html`

[6] P.G.Bosco, G. Martini, D. LoGiudice, C. Moiso, *ACE: An Environment for Specifying and Generating TINA Services*, 5^{th} IFIP/IEEE Intl. Symposium on Integrated Network Management (IM'97), San Diego, USA, May 1997.

[7] H.Schulzrinne, R. Frederick and V. Jacobson, *RTP: A Transport Protocol for Real-Time Systems*, Computer Communications, Jan. 96.

[8] Iona Technologies Ltd., *Iona's Orbix WWW Page*, `http://www.iona.com/Orbix/index.html`

[9] Sun Microsystems Inc., *Solaris XIL 1.2 Programmer's Guide and Reference Manual*, 1994.

[10] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, *Object-oriented Modeling and Design*, Prentice-Hall International Editions, 1991.

[11] H. S. Nwana, *Software Agents: An Overview*, Knowledge Engineering Review, Vol. 11, No 3, pp. 1-40, September, 1996.

[12] L.A. Guedes, P.C. Oliveira, E. Cardozo, *An Agent-based Approach for Quality of Service Negotiation and Management in Distributed Multimedia Systems*, Lecture Notes in Computer Science, Mobile Agents, K. Rothermel and R. Popescu-Zeletin (Eds.), Springer Verlag, 1997.

[13] L. A. Guedes, P.C. Oliveira, L.F. Faina, E. Cardozo, *QoS Agency: An Agent-based Architecture for Supporting Quality of Service in Distributed Multimedia Systems*, IEEE Conference on Protocols for Multimedia Systems - Multimedia Networking (PROMSMmNet'97), Santiago, Chile, Nov 1997.

[14] C. Aurrecoechea, A. Campbell, L. Hauw, *A Review of Quality of Services Architectures*, ACM Multimedia Systems Journal, November, 1995.

[15] A. Vogel et al., *Distributed Multimedia and QoS: A Survey*, IEEE Multimedia, Summer 1995, pp. 10-18.