A Real-Time IPC Service over ATM networks for the Chorus Distributed System

Christophe Lizzi †‡
†CS Technologies Informatiques
3, rue Maryse Bastié
69500 Bron, France
lizzi@{ csti.fr, cnam.fr }

Eric Gressier-Soudan ‡
‡CNAM, laboratoire CEDRIC
292, rue S^t-Martin
75141 Paris Cedex 03, France
Eric.Gressier@cnam.fr

Abstract

We describe a real-time interprocess communication service (RT-IPC) suitable for distributed systems built over ATM networks. This location-transparent communication service offers end-to-end quality of service guarantees, and end-to-end priority propagation. It relies on a specific real-time ATM Adaptation Layer (AAL-RT), that grants to preempt the transmission of a low priority message by a higher priority one, and on a transport service in charge of QoS negotiation, setup of the ATM virtual channel and priority inheritance. The RT-IPC service is implemented in the Chorus distributed operating system, as servers running in the micro-kernel address space. It is used for the real-time communication within ANTARA real-time clusters, and grants to communicate at a user rate up to 130 Mbps, over 155 Mbps ATM links. The end-to-end transfer delay between participants is less than 400 *μs.*

1. Introduction

Today's widely used networks, with the notable exception of FDDI, are designed to handle communication traffic in a best-effort fashion, meaning that no performance guarantee can be provided to user applications. By contrast, ATM networking technology can now offer such level of service. This leads ATM to be a privileged medium to support new multimedia and real-time distributed applications that have stringent performance requirements. In this paper, we therefore present an IPC (Inter-Process Communication) system that offers the abstractions and transparencies required by distributed systems, and provides end-to-end quality of service (QoS) guarantees.

This real-time IPC (RT-IPC) service uses a flexible traffic specification that grants admission control and resource reservation. It also uses an efficient end-to-end priority propagation mechanism that ensures a correct, prioritized, handling of messages in each communication level crossed, including the ATM network layer. Applications primarily targeted by this solution are the SCADA (Supervision Control And Data Acquisition), and remote process control systems, where both real-time supervision commands and video control streams are to be supported.

The first section surveys earlier work related to real-time IPC systems, while the second one reviews the features offered by the RT-IPC system. We then introduce the transport and ATM level features we rely on, and finally, we present the RT-IPC programming interface and describe our implementation.

2. Related work

Real-time communication systems with QoS management facilities have been widely studied and experimented during the last years. Two main approaches can be distinguished: the first one is to take advantage of the properties of a given networking technology, such as FDDI, to establish end-to-end communication channels. Other proposals, such as the Tenet [3] and IETF frameworks, address the setup of real-time circuits across hop-by-hop internetworks, and thus require resource check and reservation at each crossed node. Communications over ATM networks are hybrid in the sense that end-to-end virtual channels have to be opened prior to data transfer, but admission control is performed at connection setup on a per-switch basis, along followed virtual paths. Each of these communication systems introduces its own traffic and QoS models, although translation among them is possible [6]. They also provide their own API, that generally reflects the properties of the underlying network service. [1] surveys different existing network and transport APIs that support QoS specification. The authors advocate the definition of QoS semantics common to different architectures, and the extension of the QoS

guarantees from network services to IPC mechanisms.
In the context of distributed object architectures, a clause describing the required and offered QoS can be associated to object interfaces. Extensions to interface definition and solutions are proposed in [5]. Researches in this field have been carried out at CNET [11], where experiments were made over an FDDI ring [13]. These results account for previous work on IPC and priority inheritance at CMU on ARTS [19] and RT-Mach systems [16].
QoS-controlled communications over ATM networks were also studied at Lancaster University [10]. These works focus on resource management required in end-systems to support continuous media. Experiments were made on a Chorus-based testbed platform. They provide their own communication framework with goals similar than our work.
The use of ATM for hard real-time communications in industrial systems is also studied at MERL laboratory [15]. These researches notably address traffic management and connection admission control [18].

3. Real-Time IPC

3.1. Requirements

Basically, an IPC system grants its users to exchange messages between communication endpoints. The fundamental property expected from this service is the location transparency of the communicating entities. This means that the different participants can be named and accessed in a uniform way, whatever their distribution on the various nodes of the network. Real-time distributed applications introduce two additional requirements on the IPC system: it has to provide end-to-end quality of service guarantees, and has to handle priorities over the network. Quality of service requirements are expressed in terms of throughput, message size, delay and jitter. Admission control grants to ensure that the requested QoS requirements will actually be met, if adequate resource reservation is performed. In order to ensure that an IPC user will not overtake the traffic contract admitted by the service provider, some control on the quantity of data submitted for transmission must be provided.
Priority management relies on end-to-end priority inheritance mechanisms, that propagates message priority to receiver priority, combined to prioritized message queueing policies.
The last requirement is the only one to be relevant in case of local IPC interactions. There, messages may be copied from the address space of the sender to the one of the receiver, or remapped across address spaces. Zero-copy communications require the receivers to supply reception buffers in advance, in order to store incoming messages. Dynamic buffer allocation and on-demand remapping is time consuming and then forbidden. The control of kernel pre-allocated buffers can be shared between the service provider and the user, but induces some risks of faulty behaviour of the service provider, if the application doesn't release delivered buffers after message processing. We thus prefer a deterministic scheme where the user allocates a pool of buffers.
Also note that zero-copy transmission facilities can be implemented using adequate software features, but zero-copy reception generally requires a more sophisticated networking hardware. In the context of ATM networks, the network adapter is then expected to maintain per-connection buffer supply queues. A communication service provider that offers both zero-copy transmission and reception is therefore termed as *true zero-copy*.

3.2. Chorus IPC features

The IPC feature provided by the Chorus kernel grants *threads* hosted by *actors* to exchange *messages* through *ports* [8]. Message broadcast is also allowed via *port groups*. The IPC service is location transparent. It is based on a global uniform naming scheme, built using *unique identifiers*. These identifiers are guaranteed to be unique in time and space despite site reboots.
A message is an untyped octet sequence, termed as the message body. Its length is variable but limited in size. The message can be completed by a small fixed-size *annex*. The message body and the message annex are transferred with copy or move semantics across address spaces.
Messages are addressed to ports, named by their unique identifiers. The port is *attached* to a single actor at a time, but can *migrate*, from one actor to another, locally or across the network, along its lifetime.
Ports therefore provide an extra level of indirection between any two communicating threads, and thus grants transparent reconfiguration of application services.
Messages can be sent asynchronously, meaning that the sender is blocked only during the time of the local processing of the message, or can be sent using an *at most once* remote procedure call semantic. The sender is there blocked until a reply is returned by the receiver. The usual way for an actor to consume messages received by ports is to have threads explicitly waiting on these ports. Alternatively, privileged *supervisor* actors can attach a *message handler* to a port. This handler is a procedure called each time a message is received on the related port.
Ports can be gathered into groups. This offers multicast facilities to applications. Creating a group of ports just allocates a unique identifier for the group. Then, ports can later be inserted and removed from the group. Messages are sent to the group members according to the *addressing mode* specified by the caller:

- *broadcast*: the message is delivered to all ports in the group;
- *functional*: one port of the group is selected, arbitrarily, or on a given host, or on a host different from a given one.

The last release of the Chorus system is based on a componentized approach. The kernel is made of building blocks, selected according to application requirements. The selected components determine for instance the programming facilities that remain available, but also fundamental system features such as the virtual memory model or the scheduling policies. Consequently, the system can be targeted to a wide range of environments, from small embedded systems to large servers. Our real-time IPC framework could be inserted as a new building block in the Chorus componentized architecture.

3.3. RT-IPC service

The standard Chorus IPC service processes messages in a best-effort fashion. Even if the underlying network supports performance guarantees, there is no provision for resource reservation nor QoS specification. Furthermore, messages are queued and delivered by ports in FIFO order, as there is no notion of message priority. This leads to priority inversions when critical messages are delayed by previous, less critical, ones.
In order to support the level of service expected by distributed hard real-time applications, we have designed a new IPC service for the Chorus system. This RT-IPC service retains most of the features offered by the classical Chorus IPC but also provides end-to-end quality of service guarantees and supports priority inheritance. Our system introduces new abstractions and communication semantic. The main differences with the classical Chorus IPC are described below.
The RT-IPC service allows exchange of *RT-messages* between *RT-ports*. A RT-message is similar to a classical Chorus message, except that it additionally conveys a priority. If the user does not explicitly precises a priority, then the message inherits the one of the sending thread. Depending on the option chosen at port creation time, RT-ports queue messages according to a FIFO order, or according to the message priority. The receiver thread also inherits the priority of the first message to be delivered, immediately after its reception by the port. The message reception makes the receiver thread eligible by the system. Priority propagation ensures that it will gain the processor according to the criticability of the message being delivered and processed.
By contrast to the classical Chorus IPC, there is no actor default port. RT-ports have to be explicitly created on both side of the communication. Transfer of messages between RT-ports is allowed as soon as the source and destination ports have been *connected*. The connection is established on request from the application, that precises the quality of service it expects. Explicit binding is our main change to the semantic of the Chorus IPC communication. Moreover, as one can not guarantee that a connection can be re-established with the same performance level to another host, RT-ports can not migrate.
Connection request is subject to admission control, in order to ensure that the QoS required by the application can actually be provided. In the case of a local communication, this step always succeeds.
Admission tests are based on a description of the connection, that details the characteristics of the data traffic being conveyed and the QoS expected from the RT-IPC service. Traffic specification is expressed in term of peak and substainable rates, and maximum message size, while the QoS specification include transfer delay and maximum admissible jitter. Traffic and QoS specifications compose the flow specification, or *flowspec*. For any connection establishment request, two flowspecs are submitted:

- a *target* flowspec, that denotes the nominal service level for the connection being requested;
- an *acceptable* flowspec, that defines the minimum service level acceptable for the connection. If the service provider is unable to establish the connection at the requested target level, it should try to establish it at the best possible level, greater or equal to the acceptable flowspec.

Connection request also includes a commitment level, that expresses the degree of certainty that the QoS requested will actually be provided. Three commitment levels are defined:

- *guaranteed*: QoS will be kept as long as the service provider functions properly [6];
- *predicted*: QoS will be kept under the assumption that the future load of the service provider corresponds to the load currently observed. Resources reserved can be preempted, if necessary, to establish new guaranteed-commitment connections;
- *best-effort*: QoS will be kept as good as possible. However, there is no guarantee that the required performance is ever provided. Hence, there is no admission control nor resource reservation.

The connection description is finally completed by a priority level parameter, used to control resource preemption. A predicted-commitment connection with a low-priority will have its resources preempted before one with a higher priority level. Also note that the current version of the UNI signalling protocol does not support QoS renegotiation. It

follows that degrading the QoS granted to a predicted-commitment connection means tearing down the used ATM channel, and re-establishing it at a lower performance level, inducing temporary service interrupt.
User requirements are translated into ATM traffic and QoS parameters by the underlying real-time transport service. This service is in charge of VC management and actual end-to-end message transfer with priority propagation, as described in [14].

4. Architecture

The RT-IPC system relies on underlying layers to achieve real-time communication. It mainly implements protocols for location-transparent naming of RT-ports and groups, and provides mechanisms for local interactions.
Two underlying software layers are used to support real-time remote communication. At the ATM adapter level, we introduce a new, priority inversion free, adaptation layer to support real-time flows. Then, directly running on top of this real-time AAL, we describe the transport protocol in charge of end-to-end management of ATM VCs. This communication architecture is depicted by figure 1.

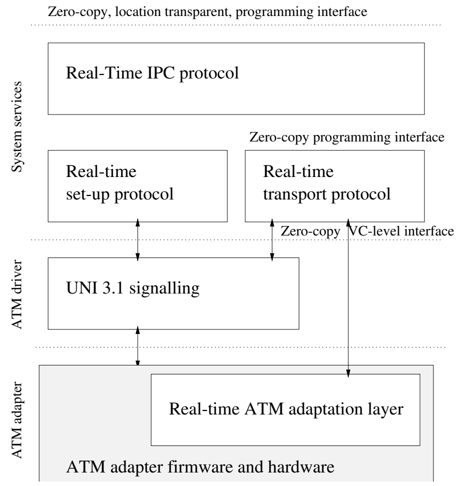

Figure 1. Architecture of the communication system.

4.1. Real-Time AAL

The *ATM Adaptation Layer* (AAL) adapts user traffic to cell-based networking [4]. Its sublayers are responsible for segmentation and reassembly of messages, and for the enforcement of possible timing relationships between source and destination systems. Several AALs have been defined, to accommodate to a wide variety of traffic, from connectionless, asynchronous data to connection-oriented isochronous audio or video streams. The only AALs that support timing requirements between source and destination systems are AAL1 and AAL2. AAL1 is devoted to synchronous constant bit rate, while AAL2 handles synchronous variable bit rate traffic.
In our opinion, these AALs do not provide the level of service required for hard real-time applications, because they do not take into account the relative *priority* of messages submitted for transmission, leading to priority inversions when messages are queued. In order to support real-time traffic with end-to-end priority propagation, a new adaptation layer, called *Real-Time AAL* (AAL-RT), has been defined at CSTI. This AAL provides features similar to the standard AAL3/4 [12], but is devoted to real-time CBR traffic instead of non-time sensitive VBR traffic.
The AAL-RT uses the same PDUs as the AAL3/4, but makes a different usage of the the *Message Identification* (MID) field, defined in the PDU of the *segmentation and reassembly* (SAR) sublayer. In standard AAL3/4 scheme, the MID field grants to multiplex different user message fragments into one VC. In the AAL-RT scheme we propose, this 10-bit field is used to convey an end-to-end *priority* associated to the message. Using the *streaming* transfer mode, this priority is used to *preempt* a low priority message by a higher priority one. Transfer of the preempted message is interrupted, and restarts after the completion of the sending of the higher priority message. This ensures a correct, prioritized, message processing, within the scope of a virtual channel, when crossing the AAL layer.

4.2. Real-Time transport

The underlying real-time transport (RT-transport) service basically grants to establish end-to-end *unidirectional* VCs, whereas its programming interface also provides abstractions to use bidirectional communications. This service is based on two protocols. The first one is in charge of connection setup while the second one handles data transmission.
The connection setup service is responsible for connection admission control and resource reservation in the network and in end-systems. Admission tests are derived from those presented in [7]. Remaining bandwidth of the ATM network and worst-case transfer delays are determinated using the admission algorithms proposed in [20], by the way of a centralized connection manager. On completion of these setup steps, the ATM virtual channel is finally established.
The real-time transport protocol conveys user data after the connection was established. It also ensures that the accepted traffic contract is not exceeded by the sender. Data requested for transmission are therefore submitted to a *traf-*

fic shaper, that limits the rate of the data injected into the network, by rejecting the message or blocking the sender until the amount of data submitted can truly be accepted. A message is rejected or delayed unless its priority is strictly greater than the one of the message currently transmitted. Combined to the preemption mechanism run by the AAL-RT, this shaping policy grants to avoid priority inversion. However, in order to ensure that every message will be transmitted in a bounded delay, the number of times a message can be preempted must be bounded. Otherwise, the transmission of a low priority message can potentially never be completed, if it is continually preempted by the transmission of higher priority messages.
Determining the number of times a message of a given priority can be preempted requires an in-deep knowledge of the user application, notably the minimum inter-arrival delay between two messages of the same priority to be sent into a given VC. This bound is thus provided by the user at connection setup time.

5. RT-IPC interface

Most of the abstractions and primitives here are inherited from the off-the-shelf Chorus IPC service. Inherited calls requires additional parameters to handle the QoS requirements, and new calls have been introduced to support our zero-copy architecture. The whole set of primitive offered by the RT-IPC system is distinct from those of the standard Chorus IPC, to ensure a compatible approach to the componentized architecture of the micro-kernel. Our framework can therefore be added to the Chorus system as a third-party building block for distributed application that requires QoS support.
The most meaningful entry points of the RT-IPC service are reviewed below.

5.1. Port management

RT-ports are managed by the following calls:

- `rtPortCreate()` creates a RT-port and allocates a unique identifier. The user also precises the message queueing policy (i.e. FIFO or prioritized), the priority inheritance options, and the pools of shared buffers, for transmission and reception or RT-messages, associated to the RT-port;
- `rtPortDeclare()` is similar to the preceding call, except that the unique identifier of the RT-port is provided by the user, which is therefore responsible for its unicity in time and space;
- `rtPortDelete()` destroys a RT-port previously created. When the port is deleted, queued messages are lost;

Ports can be gathered into groups by the following calls:

- `rtGrpAllocate()` allocates a unique identifier for the RT-port group. Initially, the group is empty;
- `rtGrpPortInsert()` inserts a RT-port in a group. The insertion may fail if the RT-IPC service provider is unable to bind the port being inserted to remote RT-ports that are already connected to the group, at the QoS level they requests;
- `rtGrpPortRemove()` grants to remove a RT-port from a group.

RT-ports have to be bound before message transfer. The binding is done through a connection setup step. The binding request conveys the target and acceptable flowspecs, that describe the performance level expected by the application. Therefore remote bindings may fail if sufficient resources cannot be reserved in crossed switches or in the end-systems. Connection management is performed by the following calls:

- `rtPortConnect()` tries to establish a connection between a RT-port and another port or group. If all the ports involved in the binding are located on the same host, this call just remaps the transmission buffer pool of the caller to the address space of the actors hosting the receiver ports. Otherwise, the request is submitted to the admission controller;
- `rtPortDisconnect()` releases a connection established by the preceding call.

5.2. Buffer management

As the RT-IPC system is based on a true zero-copy architecture, buffers have to be provided by the user prior to the reception of incoming messages. Buffer management is based on two abstractions: shared buffers and buffer pools. Buffers are the unit of allocation, and pools are the unit of mapping between address spaces. Related calls are described below:

- `rtBufferAllocate()` allocates a buffer. Privileged actors can allocate buffers using physically contiguous memory pages. This grants the underlying real-time transport service to produce only one DMA descriptor to submit the whole buffer to the ATM adapter.
- `rtBufferFree()` frees a buffer previously allocated;
- `rtPoolAllocate()` grants to allocate a buffer pool;

- `rtPoolInsert()` inserts a buffer in a pool;
- `rtPoolMap()` remaps the buffers inserted in a pool to the address space of the RT-IPC service provider;
- `rtPoolGet()` retrieves a free buffer from a pool. The buffer is subsequently declared as busy. The buffer being returned is selected according to a FIFO policy;
- `rtPoolRelease()` grants to release a busy buffer to its pool, when it is no longer used;
- `rtPoolFree()` frees a pool allocated by the previous call and, if required, frees inserted buffers.

Two functions are used along the data transfer phase. The `rtPoolGet()` function is called by the user in order to retrieve a free buffer from the pool. The buffer can then be used as the body of a RT-message being transmitted. `rtPoolRelease()` is later called by the underlying RT-transport server on completion of the RT-message transmission. Then, during reception process, `rtPoolGet()` is called by the RT-transport server each time a buffer is required to store an incoming message. More precisely, buffers are provided in advance to the per-VC supply queues of the ATM adapter, depending on a low-water/high-water marks policy. The `rtPoolRelease()` function is also called by the receiving application after the processing of a delivered RT-message.

5.3. Data transfer

After RT-ports have been connected, data transfer operations are similar to those granted by the classical Chorus IPC framework. The main related calls are:

- `rtIpcSend()` sends a RT-message to a RT-port or to a port group. The message is transferred asynchronously, and the caller is blocked only during the time required to enqueue the RT-message. The related shared buffer will later be implicitly released to its pool by the underlying transport server. If the user has not set a priority for the message being sent, then it inherits the priority of the caller's thread;
- `rtIpcReceive()` blocks until a message is received. According to inheritance option of the target RT-port, the priority of the incoming message may be propagated to the receiver's thread;
- `rtIpcCall()` sends a message to a RT-port and blocks the caller until it gets a reply from the destination port. As before, the receiver may inherit the priority of the RT-message. This call may be invoked only if two distinct connections have been established between pair RT-ports, one in each direction, because such binding is unidirectional;
- `rtIpcReply()` grants a receiver to answer a RT-message;
- `rtIpcTarget()` specifies the addressing mode (i.e. broadcast or one of the functional modes reviewed in section 3.2) of a group. It defines which group members will be reached by a RT-message targeted to a group.

6. Implementation

The services above described are used for real-time communication in the ANTARA distributed real-time servers. An ANTARA server is a cluster of processing nodes interconnected by ATM links. In order to ensure the real-time capabilities of the system, ANTARA nodes uses a specific ATM adapter, designed by CSTI. It supports all the service classes defined by ATM specifications, and provides enhanced real-time functionalities, such as a true zero-copy interface and the AAL-RT adaptation layer.

6.1. Operating system support

Each processing node runs the Chorus operating system. The RT-IPC and RT-transport services are implemented as system actors running in the supervisor address space, also shared by the kernel. The ATM driver is hosted by the Chorus *IOM* (Input/Output Manager) actor, that also embeds the TCP/IP protocols. The IOM presents to device drivers an *IFNET* interface, identical to the one offered by BSD kernels. This interface is used by the TCP/IP stack and, subsequently, by upper level Chorus protocols, including the standard Chorus IPC service. The IFNET interface eases the port of devices driver initially written for BSD systems, but induces an important CPU load, due to systematic copy of user data to/from *mbuf* buffer chains. Moreover, context switches and non-real-time synchronization primitives degrade the predictability of the communication subsystem.
To bypass these drawbacks, the ATM driver also implements an event-driven asynchronous interface, that provides a true zero-copy access to the ATM adapter. This VC-level interface, called *LLE* (Low-Level Entry), grants its callers to access the UNI signalling protocol and the AAL-RT layer embedded in the ATM adapter, by running minimum ATM driver code.
Like the RT-IPC server, the RT-transport and IOM actors share the same address space, together with the microkernel. The LLE interface is thus invoked, after a dynamic attachment step, by direct function calls, avoiding any context switch. Calls are non-blocking and reports are therefore delivered to LLE callers by callback handlers. The RT-IPC user interface is implemented using Chorus local access ports (LAPs), a lightweight facility used to implement system calls. Figure 2 depicts this whole architecture.

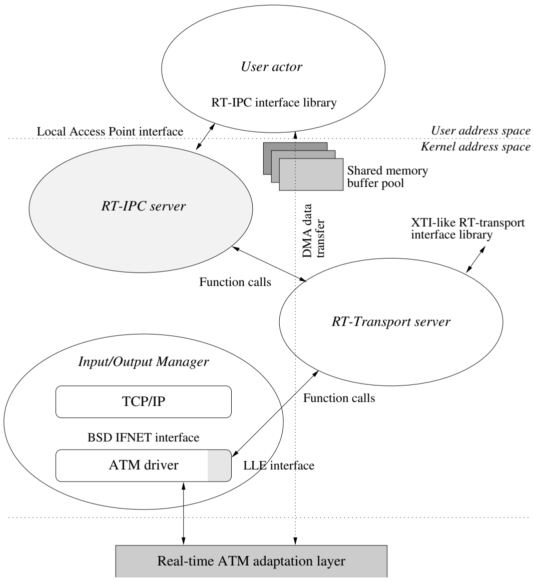

Figure 2. Implementation of the real-time IPC system.

6.2. Performances

The RT-IPC system is used for inter-process communication between real-time applications run in the ANTARA servers. The processing nodes of a cluster are connected by 155 Mbps ATM links around a 16 port ATM switch. The real-time IPC service grants to reach the user rate of 130 Mbps, over a 155 Mbps ATM link, between 133 MHz *PowerPC* hosts. This is close to the maximum theoretical rate, accounting for the overhead induced by the AAL-RT layer, the ATM cell headers, and the SDH physical frames. The end-to-end transfer delay, between the time the message is submitted to the RT-IPC service, until its reception by the destination RT-port is less than 400 μs, crossing one unloaded ATM switch within an ANTARA cluster.

7. Conclusion

We have presented a complete real-time interprocess communication service suitable for distributed systems built over an ATM network. We are able to provide end-to-end QoS guarantees to distributed real-time applications, through a location-transparent, true zero-copy, communication service. The IPC service provider relies on the underlying communication architecture to achieve end-to-end guarantees: the AAL-RT adaptation layer grants a priority inversion-free processing of messages submitted for transmission, while the upper RT-transport protocol handles end-to-end priority propagation. Our RT-IPC framework is going to be used in the CSTI video server for interprocess communication within ANTARA clusters.

This communication architecture provides an efficient basis to support real-time QoS requirements at an upper abstraction level. It could enhance software bus environments to address real-time distributed objects platforms. We intend to integrate the results presented here in an object-oriented distributed environment. First candidates to this new step are COOL and JAZZ, respectively the object request broker and the Java virtual machine proposed by Chorus Systems. The ACE system [17], designed at Washington University, could be another targeted environment.

References

[1] W. Almesberger, S. Sasyan, S. Wright, *Quality of Service (QoS) in Communication APIs*, IFIP International Workshop on Quality of Service, May 1997.

[2] ATM Forum, *Traffic Management Specification Version 4.0*, ATM Forum specification af–tm–0056-000, April 1996.

[3] A. Banerjea, D. Ferrari, B. Mah, M. Moran, D. Verma, H. Zhang, *The Tenet Real-Time Protocol Suite: Design, Implementation and Experiences*, Technical Report UCB/CSD-94-059, Computer Science Division, University of California, Berkeley, November 1994.

[4] U. Black, *ATM: Foundations for Broadband Networks*, Prentice Hall, 1995.

[5] G. Blair, J.-B. Stefani, *Open Distributed Processing and Multimedia*, Addison-Wesley, 1997.

[6] S. Böcking, *Communication Performance Models*, Technical Report TR-95-013, ICSI, University of California, Berkeley, March 1995.

[7] A.T. Campbell, *A Quality of Service Architecture*, Ph.D Thesis, Computing Dept., Lancaster University, January 1996.

[8] Chorus Systems, *CHORUS/ClassiX r3 Technical Overview*, Technical Report CS/TR-96-119.9, 1996.

[9] D. Clark, S. Shenker, L. Zhang, *Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanisms*, ACM SIGCOMM'92, 106-114, August 1992.

[10] G. Coulson, A. Campbell, P. Robin, G. Blair, M. Papathomas, D. Hutchison, *The Design of a QoS Controlled ATM Based Communications System in Chorus*, IEEE Journal on Selected Areas in Communications, 13(4):686-699, May 1995.

[11] F. Dang Tran, V. Perebaskine, *TORBboyau: Architecture and Implementation* (in french), Technical Report NT/PAA/TSA/TLR/4587, CNET, 1995.

[12] International Telecommunication Union, *B-ISDN ATM Adaptation Layer (AAL) Specification*, ITU-T Recommendation I.363, 1993.

[13] L. Leboucher, J.-B. Stefani, *Admission Control for End-to-End Distributed Bindings*, Lectures Notes in Computer Science, 1995.

[14] C. Lizzi, E. Gressier-Soudan, J. Montiel, *A Real-Time Communication Service for ATM-based Distributed Systems*, to appear, IEEE International Conference on ATM, Colmar, June 1998.

[15] I. Mizunuma, C. Shen, M. Takegaki, *Middleware for Distributed Industrial Real-Time Systems on ATM Networks*, 17^{th} IEEE Real-Time Symposium, September 1996.

[16] T. Nakajima, T. Kitayama, H. Arakawa, H. Tokuda, *Integrated Management of Priority Inversion in Real-Time Mach*, IEEE Real-Time Systems Symposium, 1993.

[17] D.C. Schmidt, A. Gokhale, T.H. Harrison, G. Parulkar, *A High-Performance Endsystem Architecture for Real-Time CORBA*, IEEE Communications Magazine, 14(2), February 1997.

[18] C. Shen, *On ATM Support for Distributed Real-Time Applications*, IEEE Real-Time Technology and Applications Symposium, June 1996.

[19] H. Tokuda, Y. Tobe, S. Chou, J. Moura, *Continuous Media Communication with Dynamic QoS Control using ARTS with an FDDI Network*, School of Computer Science, Carnegie Mellon University, 1992.

[20] Q. Zheng, T. Yokotani, T. Ichihashi, Y. Nemoto, *Connection Admission Control for Hard Real-Time Communication in ATM Networks*, Technical Report 96-21, Mistubishi Electric Research Laboratory, September 1996.