Extending the UMIOP Specification for Reliable Multicast in CORBA*

Alysson Neves Bessani[1], Joni da Silva Fraga[1], and Lau Cheuk Lung[2]

[1] DAS - Departamento de Automação e Sistemas
UFSC - Universidade Federal de Santa Catarina
Florianópolis - Santa Catarina - Brazil
{neves, fraga}@das.ufsc.br
[2] Graduate Program in Applied Computer Science
Pontifical Catholic University of Paraná
Curitiba - Paraná - Brazil
lau@ppgia.pucpr.br

Abstract. OMG has published an unreliable multicast specification for distributed applications developed in CORBA (UMIOP). This mechanism can be implemented based on IP Multicast, a best-effort protocol, which provides no guarantees about the message delivery. However, many fault-tolerant or groupware applications demand more restrictive agreement and ordering guarantees (for instance, reliable multicast with FIFO, causal or total ordering) from the available support for group communication. OMG has not yet provided any specification for supporting those requirements. This paper presents an important contribution towards this direction. We proposed the ReMIOP, an extension to the UMIOP/OMG protocol, for the conception of a reliable multicast mechanism in CORBA middleware. Performance measures comparing ReMIOP, UMIOP and UDP sockets for IP multicast communication are presented in order to evidence the costs for adding reliable and unreliable multicast in middleware level.

1 Introduction

When CORBA architecture (*Common Object Request Broker*) [20] was introduced by OMG (*Object Management Group*), only point-to-point communications (using static or dynamic invocation) was available through the ORB (*Object Request Broker*). The messages that pass through this channel obey a proper transference syntax defined by the GIOP (*General Inter-ORB Protocol*). This syntax makes the messages involved in the communications independent from ORBs implementations and the consequences of an heterogeneous environment. The mapping of GIOP over TCP/IP transport layer is made by IIOP (*Internet Inter-ORB Protocol*) protocol. The IIOP and TCP/IP combination is a good solution for distributed objects communications in the client/server model, since it considers aspects like error control, FIFO ordering, etc.

* This work is supported by CNPq (Brazilian National Research Council) through processes 401802/2003-5 and 481523/2004-9.

Point-to-point communications have shown, in general, effective in distributed applications supported by CORBA. However, many of these applications would have a better performance, concerning time, memory and message complexity, if they could use multi-point communication mechanisms. Usually, these applications depend on abstractions like groups of objects or the need to disseminate data over several hosts of a network. Therefore, group oriented applications could have greater benefits from the network low-level services.

In an attempt to supply the need of multi-point communications in CORBA middleware-level, OMG published the UMIOP (*Unreliable multicast Inter-ORB Protocol*) specifications [19] in 2001. The UMIOP is a set of specification for an unreliable multicast service to be included as part of the ORB. The protocol defined in these specifications, the MIOP (*Multicast Inter-ORB Protocol*), is responsible by GIOP mapping over UDP/IP multicast stack. IP multicast is a set of IP protocol extensions that enables it to establish multi-point communications [7, 6]. This protocol is characterized by absence of guarantees and high performance, mainly in local networks. Many applications use IP multicast, especially those for distributed multimedia systems.

The unreliable multicast service defined by UMIOP, the less restrictive group communication model, can be used for some distributed applications, for example, video conference, in which the loss of some frames do not represent the degradation of transmitted information. However, fault-tolerant applications, groupware applications, among others, usually demand more restrictive guarantees concerning group communication supports reliability and ordering (for example FIFO, causal, total, etc). OMG has not yet provided any specification concerning these requirements. This problem is being treated by OMG in stages. The first step, therefore, was the publication of UMIOP specifications. We believe that initiative motivates OMG to publish another RFP (*Request for Proposal*), towards a Reliable and Ordered Multicast Inter-ORB Protocol. Initial submissions to this RFP has been already done [21].

The integration and implementation of UMIOP in an ORB were presented in [1, 3]. As a step forward, this paper presents our contributions in the conception of a "best-effort" reliable multicast support in the ORB, based on UMIOP specifications. The proposed model, called ReMIOP, is CORBA and UMIOP specifications compliant - the proposed extension does not change any interfaces of the current specification. Actually, we indicate how to integrate reliable multicast protocols into ORB without any change of the CORBA specifications. The inclusion of a reliable multicast protocol on top of the MIOP layer is implemented as a plugin mechanism. Some performance measures of the ReMIOP (the ReMIOP/MIOP/UDP/IP multicast stack), UMIOP (MIOP/UDP/IP multicast stack) and UDP sockets (UDP/IP multicast stack) are presented to show the costs of including reliable and unreliable multicast in middleware level.

This work is part of the GroupPac project [16, 2], which is a set of object services based on FT-CORBA specification (chapter 23 of [20]) and developed to make easier the implementation of fault-tolerant distributed applications.

This paper is organized as follow: section 2 presents the OMG initiatives for group communication introduction in CORBA. The MJaco is presented in section 3. In section 4, the ReMIOP protocol is presented as a MIOP extension for reliable multicast. Some implementation issues are described in section 5. In section 6, some experiments with our multicast ORB are presented. Finally, section 7 cites some related works, and in section 8 presents some final remarks of this research.

2 Group Communication in CORBA

Two significant initiatives were taken into account by OMG concerning the introduction of group communication mechanisms in CORBA. The first of them use the group abstraction to support fault-tolerant applications in object level (FT-CORBA [20, 9]), and the other considers to use the ORB as a high performance group communication mechanism without reliability (UMIOP [19]). So, these two specifications can be considered complementary and indicate a trend in OMG, the attempt to specify a standardized group communication mechanism with differentiated guarantee levels for different applications.

The FT-CORBA standard, which introduced the concept of objects group in CORBA architecture, defines a set of object services that offer functionalities such as group management (membership), state transfer, fault detection and notification. One kind of support assumed by FT-CORBA, but not standardized by OMG, is the group communication service [25]. That specification defines that this service must support some communication properties in order to provide the underlying mechanisms for implementing active replication technique [27]; however, these specifications do not define the service semantics and protocols that must be implemented.

The UMIOP specifications, on the other hand, define an unreliable multicast service based on IP multicast. It can be considered as a basis for creating of a interoperable group communication mechanism standardized by OMG. Extensions for these specifications to define stronger properties for ordering and reliability would be appropriate for FT-CORBA standard.

2.1 UMIOP

In 1999, the OMG started a specification process for an unreliable multicast protocol based on IP multicast and objects group model to support this protocol in CORBA ORBs. This process culminated in UMIOP specifications release. This standard aims to support a multi-point communication mechanism in CORBA architecture, without any delivery guarantee. The protocol used by UMIOP is the MIOP. This protocol maps GIOP messages into UDP/IP multicast. The basic function of MIOP protocol is to segment and encapsulate GIOP messages, sent to the group, into packets. These packets contain a header (defined in the specifications) with a set of fields that allows the original message to be reassembled in the receiver side. Once the packets are properly arranged, the multicast

of message is made through UDP protocol, which provides an almost direct interface for IP services or in this case, for IP multicast. IP multicast defines a set of extensions to IP protocol enabling one-to-many communication (multicast). The main characteristics of this protocol are open groups (it is not necessary to be a member of the group to multicast a message to it), no membership (list of members), no reliability (such as IP) and accessibility through class D IP addresses (from `224.0.0.0` to `239.255.255.255`).

The use of MIOP, and IP multicast in a subjacent level, make it possible to transmit the GIOP messages between two different ORBs. However, the conventional CORBA object model, which specifies that one object reference must correspond to only one object implementation, is not appropriate for object group. In despite of that, the semantics of CORBA point-to-point invocation is reliable concerning messages delivery, and the order is defined by sender, which can be configured with or without reply, unlike the MIOP definition. Therefore, a new object model representing groups had to be defined in UMIOP. This model does not define an object identifier, but a group identifier that can be associated with multiple object ids used by the POA (*Portable Object Adapter*) to activate the corresponding servants [19]. The semantics of messages delivery and ordering in UMIOP has no guarantees, and the MIOP supports only messages with no replies.

An objects group in UMIOP is composed by group identifier, and information about how to reach it in the communication network (class D IP address and a port). These information are contained into UMIOP group references detailed bellow.

2.2 UMIOP Group Reference

A reference or IOR (*Interoperable Object Reference*) is used to identify a single object in CORBA. Each IOR contains one or more profiles which allow the ORB to locate a servant object through any network transport mechanism. For example, IIOP profiles contain in its fields the server ORB address (usually, an IP address and a port) and an object identifier in the server ORB (object key), used to access implementations through TCP/IP.

In order to support groups, the UMIOP specifications define a group IOR that addresses a set of zero or more objects. A group IOR uses a different type of profile to send messages through UDP/IP multicast. This UIPMC profile, defined in the specification, contains all necessary information to access a multicast group (a class D IP address and a port) in transport level. Another structure, with the logical group identifier, is used for identifying members at ORB level. The group IOR can also hold two IIOP profiles: one for requests that demand reply and another that specifies a gateway to multicast requests when the client is unable to do that.

The figure 1 presents the complete group IOR format defined by OMG as part of UMIOP specifications. The composing of a group IOR must be made through the specification of information about the group and the IORs for group IIOP object and gateway. This creation is made with these information specification

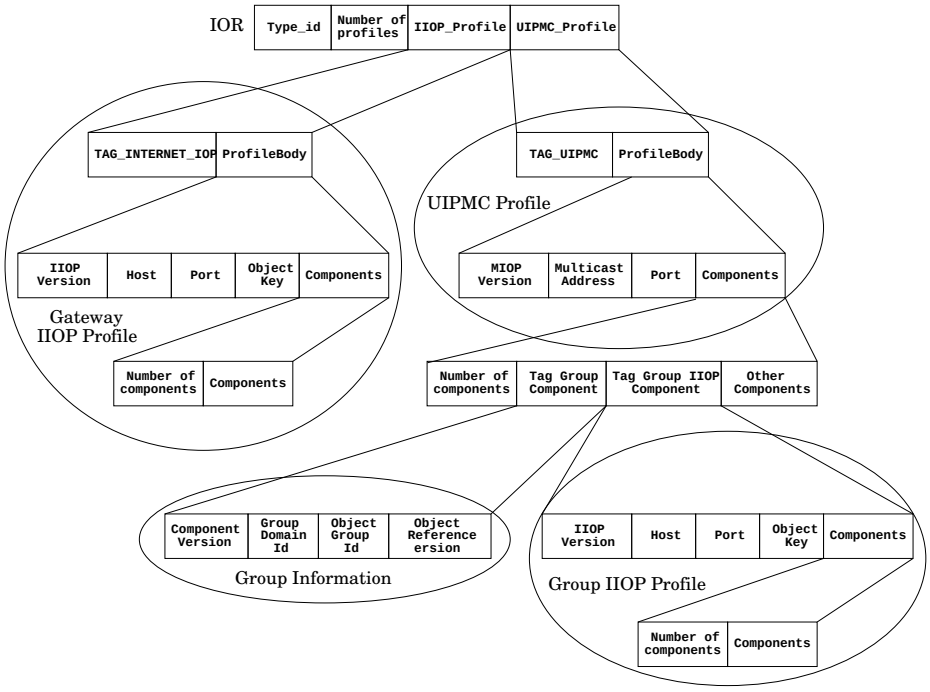

Fig. 1. Representation of the Group IOR.

in a "corbaloc" URL, or through MGM (*Multicast Group Manager*) methods, an optional object service that provides operations for groups management.

3 MJaco

From the UMIOP specification study we developed an ORB to fulfill these specification. This ORB was called MJaco [1, 3], which is an extension of JacORB, a high-performance and open source CORBA ORB that implements CORBA 2.3 specifications (*http://www.jacorb.org*). The MJaco architecture was defined to allow the compatibility of two protocol stacks (IIOP/TCP/IP and MIOP/UDP/IP multicast) in the same ORB, contributing for better interoperability and portability.

Figure 2 illustrates the UMIOP and MJaco ORB integration architecture. In this figure the ORB is presented with the two protocol stacks: one for point-to-point communication based on IIOP using TCP/IP services, and other for multi-point communication based on MIOP, using UDP/IP multicast as transport mechanism. Our integration model presents some elements defined in the specification that compose the support for theses two models of communication. Other components and extensions, not defined in that specification, has also

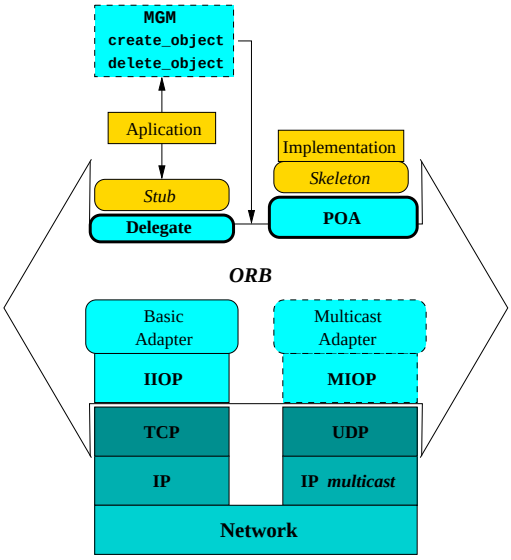

Fig. 2. MJaco Architecture.

been added. Their purpose is to facilitate the integration of the different stacks and to improve its efficiency.

The Multicast Adapter is a fundamental part of our integration model. It is responsible for managing the multicast sockets used in the reception of the MIOP packets and for the delivery of group messages to the POAs in the ORB.

The `POA` and the `Delegate` are the main components of the ORB to be modified to add the UMIOP. The modifications on `Delegate` are made in some points to support multicast GIOP message to groups, since it is the first ORB internal component to be activated when a stub method call is executed. Four new methods for handling objects group, described in the OMG specifications, are added to the POA. In addition, the POA has to be modified in order to process requests addressed to groups. For every group upcoming request, a search in the active groups table is made in order to obtain the group member implementations to which the request is addressed in each POA.

4 ReMIOP - Reliable MIOP

As mentioned before, the MIOP is unreliable, thus inadequate for many kinds of applications, like fault-tolerant or groupware systems, which do not allow message losses. We propose a set of extensions in the MIOP, called ReMIOP (*Reliable Multicast Inter-ORB Protocol*), which provides high probability in the message delivery guarantee. Basically, the ReMIOP protocol is a receiver-initiated "best-effort" reliable multicast protocol (it uses NACKs to ask for lost message [14, 24]) such as SRM [10], LRMP [15] and TRM [26]. The ReMIOP may be seen as a

minimal reliable multicast protocol in the sense that it uses only two very weak mechanisms - lost message recovery and flow control - in order to implement "best-effort" reliability.

The premise of ReMIOP reliability properties assume a communication support with asynchronous systems characteristics, therefore, with no time limits guarantees for message transmission and remote operations execution. The fault model considers only crash faults for hosts, and omission faults in the communication system. Another fundamental assumption concerning the consistency of the ReMIOP is the statement: after $Od+1$ multicast of the same message, there is no correct host which did not receive this message (see section 4.1).

The ReMIOP operates as the following: messages (MIOP packets) are multicast by the sender to all receivers. The sender has no knowledge about the group members identity. The receivers detect lost packets by gaps in the sequence of received messages[3]. When a member detects a missing packet, it multicast a control message (NACK) to the group, asking for the lost packets. Any member that receives this message, either the sender or any other receiver that owns the required packet, can multicast it to the group. This protocol also includes session messages that are multicast by the receivers to report to the senders its buffers state, allowing a dynamic adjustment of transmission rate through the flow control algorithm. The algorithm presented in the figure 3 describes the executed procedures to multicast and to receive messages through ReMIOP.

Before describing the algorithm of the figure 3 we shall explain some primitives used in it:

- $calculate\_delay()$: This primitive is used to calculate the schedule time to the next multicast according to the flow control algorithm;
- $schedule\_multicast(time, message)$: It is used to schedule the multicast of a message at a specified local clock time;
- $cancel\_scheduled(message\_id)$: Cancels the multicast of a message with specified id;
- $cancel\_scheduled\_nack\_for(message\_ids)$: Cancels the multicast of a NACK for specified messages. The $message\_ids$ is a set of message ids;
- $missing\_messages(buffer)$: This function searches in the specified buffer for gaps in messages sequence and return a set with all the ids for these missing messages;
- $random(limit)$: This primitive chooses a random integer value between 0 and a specified limit;
- $nack(message\_ids)$: Builds and returns a NACK requesting the messages with the specified ids;
- $update\_send\_rate(states)$: Applies the flow control rule to define the new send rate.

Besides these primitives, the algorithm for messages reception executes sequentially, and uses a scheduler that obeys the specified time schedule with

[3] For the first message of each sender, the receiver creates a buffer for controlling sender messages.

```
procedure R-multicast(m):
  T_d ← calculate_delay() // Sender flow control
  schedule_multicast(T_d, m) // Message multicast scheduled

To R-deliver(m) do:
  U-receive(m)
  if m.type = DATA then // m.type: type of the message m
      cancel_scheduled(m.id) // Cancels m multicast, if scheduled
      if m ∉ buffer_{m.sender} then // m.sender: sender of m
          buffer_{m.sender} ← buffer ∪ {m}
          R-deliver(m)
          missing ← missing_messages(buffer)
          if missing ≠ ∅ then
              schedule_multicast(random(T_nack), nack(missing))
          end if
      end if
  else if m.type = NACK then
      cancel_scheduled_nack_for(m.nacked)
      for all buffer_s do // m.nacked: required messages list
          for all m_r ∈ buffer_s : m_r.id ∈ m.nacked
              if nacks_{m_r} ≤ Od then
                  schedule_multicast(random(T_repair), m_r) // Repair
                  nacks_{m_r} ← nacks_{m_r} + 1
              end if
          end for
      end for
  else if m.type = STATE then
      update_send_rate(m.states)
  end if
```

Fig. 3. Simplified ReMIOP algorithm.

minor deviations from this. The *R-multicast*(m) procedure, defined in figure 3, implements message transmission in two steps: the computation of the wait time to multicast the message (this calculation follows the flow control algorithm); and the unreliable multicast scheduling. The reception and delivery of messages is also illustrated in the figure 3. For reliable message delivery, we first receive it in an unreliable way using the *U-receive*(m) primitive. Only after that, the algorithm treats each of the three types of messages defined by ReMIOP protocol in a differentiated manner:

– If the incoming message is a data message (a GIOP message fragment) it is verified if this message was already received before ($m \notin buffer_{m.sender}$); if it is true, no action is taken. If the message was received for the first time, it is added in the reception buffer of this sender and then delivered to the

application. After that, receivers verify if there are missing messages and multicast NACKs on the group for error recovering;
- If the received message is a NACK, then the NACK suppression mechanism is activated (cancelling NACKs that was already received). For each requested message id in the NACK a repair message is prepared (when the receiver is capable to repair it). The retransmission of this message is scheduled for a posterior time randomly defined, which avoids an explosion of repair messages. Note that a repair message multicast is conditioned to the *Od* limitations (see next subsection);
- Finally, if the message is a state notification from a group receiver (buffers state), the system takes this state into account to update the transmission rate of the protocol.

The mechanisms used to add reliability to ReMIOP are detailed in the next subsections.

A Note on Garbage Collection: As already mentioned, the ReMIOP implements probabilistic reliability mainly because, in real systems, it can not maintains the received messages into the buffer indefinitely. The buffers used in the algorithm of figure 3 (represented as a set) are not infinite. They have a predefined fixed size and when this limit is reached, the older messages are thrown away. The subsection 4.2 discuss some more issues related to the buffers of the ReMIOP.

4.1 Lost Message Recoveries

As the possibility of message loss exists and may be substantial, specially in large scale systems, the ReMIOP includes a kind of control message to allow requests for retransmitting lost messages: NACK messages. This message contains the identifiers of lost MIOP packets, so that host, which receives the message, is able to multicast the asked packets. Despite this mechanism, characterized for being initiated by receiver, some improvements had been added to the protocol in order to prevent, as possible, flooding of NACKs and repair messages. Among these modifications it can be mentioned the use of a RINA (*Receiver Initiated Nack Avoidance*) mechanism [24] and repair delay. These two improvements cause a delay on the diffusion of NACKs and repairs for random periods of time in the expectation that another group member does the retransmission.

Moreover, the omission degree parameter (*Od*) was introduced as an *optional improvement*. In this case, the sender and the receivers involved in the group interactions can retransmit the same message again until the limit defined by $Od+1$ is reached. In communication supports with omission faults it is acceptable to consider that no more than *Od* retransmissions of one single packet is lost in a reference period of time. Tests can be executed in real networks to determine *Od* in any degree of probability [28]. If a receiver does not receive a packet after $Od+1$ transmissions from a sender, then it is possible to assume that the receiver is faulty (*crash*). Note that *Od* is only one parameter that can be used in the protocol, even when the system assumed to be asynchronous. If *Od* is a too high value, then the protocol will execute like those ones which assume *reliable*

channels [12]. For this environment (since it is asynchronous, bursts of messages may be over-delayed, instead of lost) this artificial hypothesis (omission degree) can make a too slow process (or slowly connected) be treated as a crashed one. This hypothesis can be considered acceptable because it allows progress of the protocol, however this method is subject to inconsistencies if failures are not correctly detected. Therefore, this parameter is useful only for practical ends. As mentioned earlier, the omission degree is an optional improvement, so, if it is not used, then, in our algorithm, the *Od* variable is set to ∞.

4.2 Flow Control

Flow control is a fundamental mechanism for any reliable multicast service. The absence of membership information in ReMIOP environment makes impossible the use of more refined flow control algorithms like those defined in [5]. Therefore, we use a simple mechanism, inspired in LRMP flow control [15], that provides packets loss prevention in hosts, and the consequent NACKs explosion.

The mechanism applied to implement the flow control in the ReMIOP protocol uses information provided by NACKs (that contain lost messages indication) and state messages (that contain the reception buffers state of the members) received[4]. Through these information the sender can estimate the speed of its receivers and can apply a rate update function according to the receivers capacity. This mechanism uses two types of buffers: one for senders and another for receivers.

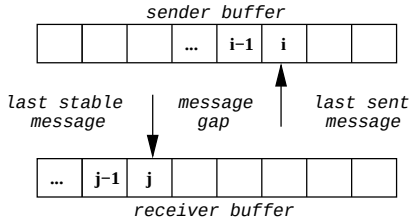

Fig. 4. ReMIOP flow control.

In figure 4, the sender buffer is used to store sent messages as well as messages to be sent. The size of this buffer determines how many old messages can be resent by this host in case of a NACK reception. In the receiver, the buffer stores the received messages. The difference between the sequence number of the last message sent by the sender (i in sender buffer) and the last received stable message (j in receiver buffer) is the parameter used to adjust the sending rate.

Let the difference $\delta = i - j$ be such that: the bigger is δ, the lower must be the sending rate so that slow receivers (whose j is much lesser that i) are able to consume the sender messages. The main objective of this algorithm is to provide

[4] Each receiver has a reception buffer for each sender.

transmissions adjusting the sending rate in a manner that all group members, including the ones that are in congested areas of the network, can receive the messages.

The transmission rate of the senders always varies between the interval $[R_{min}, R_{max}]$. Where R_{min} and R_{max} are defined by the application. The initial rate is defined as $R_0 = (R_{min} + R_{max})/2$. We consider the adjustment band as $R_{max} - R_{min} = L$ in which, with a discretization we assume ten levels of transmission rate: $R_{max} - R_{min} = 10(0.1L)$. For each $size/10$ packets sent, the sender adjusts the transmission rate R according to the following rule (where $size$ is the fixed size of the sender buffer, i.e. its maximum capacity):

$$R_i = \begin{cases} R_{min} & \text{if } \delta > \frac{size}{2} \\ R_{i-1} + (-1)^{\lceil \frac{\delta - \frac{size}{2}}{size} \rceil} 0.1L & \text{otherwise} \end{cases} \quad (1)$$

In the equation 1, R_i defines a new sending rate based: (i) on the current rate (R_{i-1}), (ii) on the greater δ collected in the period and (iii) on the size of the sender buffer ($size$). This adjustment rule states that the sending rate will always be determined by slowest receiver.

Through this flow control algorithm and the (informal) assumption that the system operates most of the time in normal conditions (without congestions and omission faults), it is possible to guarantee that all the messages sent will be delivered to the group members.

5 ReMIOP Implementation

In order to implement ReMIOP as a communication service of MJaco, we have to consider three important issues: control messages definition, ReMIOP/MIOP interoperability and where in the protocol stack we will implement the ReMIOP algorithm (presented in figure 3). This section considers these and others issues.

5.1 Control Messages Definition

As mentioned before, the ReMIOP reliability is supported by two important mechanisms: message recovery and flow control. Each of these mechanisms requires some type control messages, that are defined in the figure 5.

In this IDL two types of messages are defined: NACK and STATE. These two kinds of messages are defined by the `MessageType` enumeration. The `ReMIOPControl` structure defines the fields of the control messages used by protocol. The first field of this structure defines the type of message, following this, the *senderId* field contains the IP address of the control message sender. The last field of the structure, the `messages` field, have different purposes depending the type of the message: if it is NACK, then this field contains an array of ids of messages (represented by the `MessageId` structure) identifying the required messages that this host has lost. Otherwise, if it is a session message, then this field contains the maximum stable message for each sender of the group.

```
module ReMIOP {
    enum MessageType {
        NACK, STATE;
    };
    struct MessageId {
        string senderId;
        unsigned long long sequenceNumber;
    };
    typedef sequence<MessageIds> MessageIds;
    struct ReMIOPControl {
        MessageType type;
        string senderId;
        MessageIds messages;
    };
};
```

Fig. 5. Extension on UMIOP: ReMIOP Messages.

Independently of what type of the control message defined in the `ReMIOPControl` structure, it will be serialized such as any other IDL definition and encapsulated in MIOP packets as stated in the next subsection.

5.2 ReMIOP/MIOP Integration

To ensure the interoperability requirement, the data packets sent by ReMIOP are exactly the same as the ones sent by MIOP. So the data sent by ReMIOP can be received by MIOP receivers as well.

Each ReMIOP control message is encapsulated in one MIOP packet and transmitted to the IP multicast group just like a data packet. However, MIOP receivers do not process these control packets, they are ignored. That means that basic issues must be considered when fulfilling the MIOP packets header fields to ensure they are discarded.

The filling of MIOP packets header containing ReMIOP control messages obeys some basic rules:

1. the packet id must have the same value, and this value cannot be used by data messages;
2. the packet number field of the header is always set to 0 value;
3. the field defining the number of packets that are part of this message is always set to 2 value;
4. a bit is marked in the field `flags` indicating that the packet is a ReMIOP control message.

The rest of the fields of the MIOP packet are filled in conventional way according to the MIOP protocol specification.

If the MIOP header fields are set as indicated above, the receivers of ReMIOP packets capable to process it will detect flag pointed in the `flags` field and will treat them adequately (as a ReMIOP control packet). The receivers that do not implement ReMIOP will process the packet as the first element of a size 2 collection of packets. As the second packet of this collection do never arrives, it is never released to the ORB upper layers, and will be discarded after a timeout, just as defined by UMIOP specifications [19].

5.3 Plugable Strategies for MJaco

In order to make available a reliable multicast service provided by ReMIOP in MJaco, a plugin mechanism was implemented. This mechanism allows the integration of reliability strategies capable to extend the ORB multicast stack. So, many other protocols, with distinct features, could be implemented over this unreliable multicast service. The figure 6 illustrates the architecture of this mechanism.

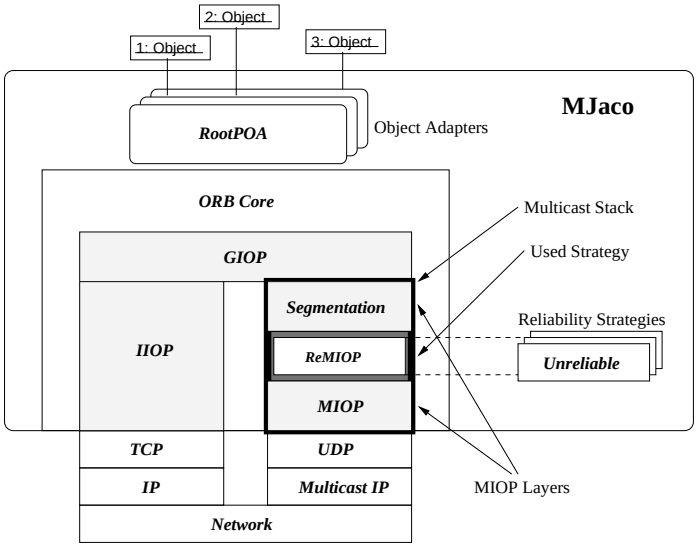

Fig. 6. MJaco architecture with plugable strategies.

In figure 6, we have the reliability strategy loaded as a multicast stack layer. This layer, which can be plugged, is in between two other layers (segmentation and MIOP) which characterizes the MIOP implementation. In a lower level layer we have the MIOP encapsulating data blocks in MIOP packets and transmitting them using UDP/IP multicast. The layer above the plugin is a segmentation layer, responsible for disassembling (marshalling) long messages in collections of MIOP packets and reassembling (unmarshalling) them on the receivers.

6 Results

In order to verify the performance of our reliable multicast service in a CORBA ORB, we executed a set of simple comparative tests concerning the use of MJaco with the ReMIOP strategy, pure MIOP, and using multicast sockets. These tests had been accomplished in four equally configured machines[5] in a LAN with minimal external network load. The test objective was to validate our implementation measuring the MJaco+ReMIOP performance, so more complex network architectures were not considered. The test program measured the time needed by a group member to multicast a variable size message and receive the reply message from all group members (including itself). It is called round trip time[6].

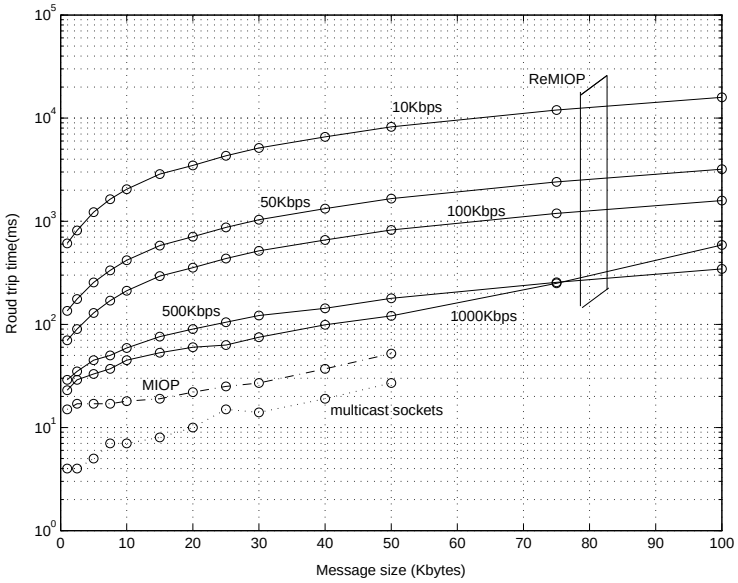

Fig. 7. MJaco performance.

Several experiments were executed with different values for R_{max} (see section 4.2). R_{max} defines the maximum transmission rate (in bits per second) of the ORB to multicast messages to a group. The larger R_{max} is, the faster is the transmission, and greater is the possibility of message losses. Experiments with MJaco using pure MIOP and with multicast sockets were also run to find out the costs of reliable multicast in middleware level. The graph of figure 7 presents these experiments results in a logarithmic scale.

[5] Pentium IV 1.6GHz, 256 Mbytes of RAM memory, Mandrake Linux 9 Operating System (kernel 2,4) and 100Mbps Ethernet network card.

[6] All communication (message and replies) are done via the tested multicast protocol: ReMIOP, MIOP and UDP/IP multicast.

Figure 7 shows that the higher the value of R_{max} the closer is the behavior of ReMIOP to the original MIOP. The behavior described by the curves with lower R_{max} values ($10Kbps$, $50Kbps$ and $100Kbps$) is extremely reliable, so, in our experiments, no packet was lost, and therefore, no NACK was multicast to the group. However, the round-trip times obtained are much larger than when using MIOP since each sender waits much longer to transmit each packet. In fact, the perceived round-trip overhead is a direct consequence of the value of R_{max} and the number of data packets[7] (message size). The ReMIOP layer overhead (compared with MIOP) is about 5ms per round-trip for small messages (one data packet) and large values of R_{max}, as can be seen in figure 7.

In curves with bigger values of R_{max}, the round trip time is lower, and the amount of lost packets, NACKs and transmissions is higher. In the curve with $R_{max} = 1000Kbps$, the round trip time is equal or higher than that of the curve $R_{max} = 500Kbps$ for long messages ($\geq 75K$). This result is caused by the great amount of lost packets (NACKs also) and repairs.

The graph of figure 7 also describes curves with the behavior of the MIOP and multicast sockets to round trip time. These curves describes messages of at most 50 Kbytes, because for longer messages the packet losses disable the use of our test program since the round-trip cannot be completed.

7 Related Work

There are some works aiming to incorporate group communication in CORBA architecture [17, 8, 18]. These works, in general, focuses in integrating existing group communication systems and protocols in CORBA middleware without concerning interoperability issues[8]. However, only recently, with the publication of UMIOP specifications, the possibility of implementing interoperable group communication mechanisms based on IP multicast had been opened. A similar study of ours is the RMIOP protocol [23] that proposes an extension of MIOP for reliability purposes. This protocol uses a NACK-based approach and ACKs messages to signal the acknowledgement of each received GIOP message (each receiver sends an ACK to the sender after receiving all the packets from a collection). This type of policy needs membership information, so the sender has to know who are the members of the group in order to collect the ACKs. The use of membership protocols increases the complexity of the RMIOP and degrades the performance in highly dynamic groups (where processes join and leave groups all the time). The approach presented in this work does not need membership, and in our conception, this type of service must be implemented at object level, using the FT-CORBA facilities (like `ObjectGroupManager` interface) [20]. Moreover, in [23] mechanisms of flow control are not used (or not reported). The RMIOP implementation was made in C++ on ORBacus 3.1, a

[7] According to the flow control algorithm, some packets must wait some time before being sent.

[8] An excellent review of some of these efforts for fault-tolerance is the paper by Felber and Narasimhan [9].

proprietary ORB that provides sufficiently generic plugin mechanism for integrating new transport protocols. We used an open-source ORB because of the possibility of unrestricted extensions implementation.

In [11] it is described another approach integrating reliable multicast in a CORBA ORB that was developed by the same group that idealized the RMIOP. In that work the reliable multicast is provided through a LRMP library integrated to the ORBacus as a plugin. Therefore, it is not a standardized and interoperable solution. It is important to remember that although it represents a simple solution, this work precedes the publication of UMIOP specifications and it brought some ideas that were included later in that specification.

The literature for "best effort" reliable multicast protocols is broad [14]. These protocols are characterized by the absence of acknowledgements message and scalability. As some of the protocols of this type we can mention the SRM [10] and TRM [26] and LRMP [15], this last one has great influence on the ReMIOP conception. Despite the mentioned successful protocols, the IETF (Internet Engineering Task Force) did not adopt any of them as the standard multipoint reliable transport protocol for the Internet. Instead, they defined a series of mechanisms that compose a reliable protocol so that different applications, with distinct requirements, can build a variety of protocols from standardized mechanisms [13, 29].

Two OMG specifications are related with our work in different aspects. The already mentioned ROMIOP (Reliable and Ordered Multicast Inter-ORB Protocol) upcoming specifications [21] may have an important role as it evolves, since it defines formats for various types of control messages used in multicast protocols (e.g. ACKs, NACKs and order enforcement). Another OMG specification of great interest is the ETF (Extensible Transport Framework) [22]. These specifications define a framework to integrate new transport protocols in CORBA ORBs. Unfortunately this framework assumes that the protocol to be integrated is reliable, point-to-point and connection-oriented. These assumptions make it difficult to implement group communication protocols as ETF plugins. The work presented in [4] is an initial attempt to integrate these two specifications devising an interoperable total order multicast protocol. This kind of protocol satisfies more restrictive properties and consequently is much heavier than ReMIOP.

8 Conclusions

The main objective of this paper was to propose extensions to standardized MIOP specification to obtain a reliable multicast support. The resulting protocol is better suited to implement more restrictive ordering and agreement guarantees and is developed as part of a CORBA ORB. The integration model, which uses the plugins mechanisms, is very flexible and do not compromise the ORB interoperability and portability aspects. The ORB is capable to make invocations using ReMIOP, MIOP or IIOP.

The ReMIOP implementation makes possible the development of other CORBA architecture group communication solutions (such as [2, 4]). These solu-

tions are being used in GroupPac project [16, 2], which implements the Fault-Tolerant CORBA specification.

Despite of that, it was also presented some ReMIOP implementation performance measures, comparing it to MIOP and IP sockets multicast to verify the costs related to this service quality available on middleware level. These developments, which were based on the proposed integration model, were built on JacORB. These implementations can be obtained on the web in the following address: *http://grouppac.sourceforge.net/*.

References

1. Alysson Neves Bessani, Joni da Silva Fraga, and Lau Cheuk Lung. Implementing the multicast inter-ORB protocol. In *Proceedings of the 6th IEEE International Symposium on Object-oriented Real-time distributed Computing - ISORC'03*, Hakodate - Hokkaido - Japan, 2003.
2. Alysson Neves Bessani, Joni da Silva Fraga, Lau Cheuk Lung, and Eduardo Adílio Alchieri. Active replication in CORBA: Standards, protocols and implementation framework. In *Proceedings of International Symposium on Distributed Objects and Applications (DOA'04)*, volume 3291 of *Lecture Notes in Computer Science*, Larnaca, Cyprus, October 2004. Springer-Verlag.
3. Alysson Neves Bessani, Lau Cheuk Lung, and Joni da Silva Fraga. Integrating the unreliable multicast inter-ORB protocol in MJaco. In *Proceedings of the 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems - IFIP DAIS'03*, volume 2893 of *Lecture Notes in Computer Science*, pages 200–211, Paris - France, 2003. Springer-Verlag.
4. Daniel Borusch, Lau Cheuk Lung, Alysson Neves Bessani, and Joni da Silva Fraga. Integrating the ROMIOP and ETF specifications for atomic multicast in CORBA. In *Proceedings of International Symposium on Distributed Objects and Applications (DOA'05)*, Lecture Notes in Computer Science, Larnaca, Cyprus, October 2005. Springer-Verlag.
5. Raimundo José de Araujo Macêdo, Paul D. Ezhilchelvan, and Santosh K. Shrivastava. Flow control schemes for a fault-tolerant multicast protocol. In *Proceedings of Pacific Rim International Symposium on Fault-Tolerant Systems (PRFTS'95)*, Newport Beach, California, USA, December 1995. IEEE Computer Society.
6. S. E. Deering. Host extensions for IP multicasting (rfc 988). IETF Request For Comments, July 1986.
7. S. E. Deering and D. R. Cheriton. Host groups: A multicast extension to the internet protocol (rfc 966). IETF Request For Comments, December 1985.
8. Pascal Felber, Benoît Garbinato, and Rachid Guerraoui. The design of a CORBA group communication service. In *Proceedings of the 15th Symposium on Reliable Distributed Systems (SRDS'96)*, pages 150–159, Niagara-on-the-Lake, Canada, 1996.
9. Pascal Felber and Priya Narasimhan. Experiences, strategies, and challenges in building fault-tolerant CORBA systems. *IEEE Transactions on Computers*, 53(5):497–511, 2004.
10. S. Floyd, V. Jacobson, C.-Gung Liu, S. McCane, and L. Zhang. A reliable multicast framework for light-weight session and application level framing. *IEEE/ACM Transactions on Networking*, December 1997.

11. C. Gransart and J.-M. Geib. Using an ORB with multicast IP. In *Proceedings of PCS99: Parallel Computing Systems Conference*, Ensenada - Mexico, 1999.
12. V. Hadzilacos and S. Toueg. A modular approach to the specification and implementation of fault-tolerant broadcasts. Technical report, Department of Computer Science, Cornell University, New York - USA, May 1994.
13. M. Handley, S. Floyd, B. Whetten, R. Kermode, L. Vicisano, and M. Luby. The reliable multicast design for bulk data transfer (rfc 2887). IETF Request For Comments, August 2000.
14. B. N. Levine and J. J. G.-L.-Aceves. A comparison of reliable multicast protocols. *Multimedia Systems*, 6(5):334–348, 1998.
15. T. Liao. Light-weight reliable multicast protocol. Avaliable at http://webcanal.inria.fr/lrmp/, 1998.
16. Lau Cheuk Lung, Joni da Silva Fraga, Jean-Marie Farines, Michael Ogg, and Aletta Ricciardi. CosNamingFT - a fault-tolerant CORBA naming service. In *Proceeding of the 18th International Symposium on Reliable Distributed Systems - SRDS'99*, Lausanne - Suice, 1999.
17. Silvano Maffeis. Constructing reliable distributed communication systems with CORBA. *IEEE Communications Magazine*, 14(2), 1997.
18. L.E. Moser, P.M. Melliar-Smith, P. Narasimhan, R.R. Koch, and K. Berke. Multicast group communication for CORBA. In *Proceedings of International Symposium on Distributed Objects and Applications*, pages 98–107, Edinburgh, United Kingdom, September 1999.
19. Object Management Group. Unreliable multicast inter-ORB protocol specification v1.0. OMG Standart ptc/03-01-11, October 2001.
20. Object Management Group. The common object request broker architecture: Core specification v3.0. OMG Standart formal/02-12-06, December 2002.
21. Object Management Group. Reliable, ordered, multicast inter-ORB protocol (revised submission). OMG TC Document realtime/2003-10-04, October 2003.
22. Object Management Group. Extensible transport framework v1.0. OMG TC Document ptc/2004-01-04, January 2004.
23. S. L. Dit Picard, S. Degrande, and C. Gransart. A CORBA based platform as communication support for synchronous collaborative virtual environments. In *9th ACM Multimedia Conference*, Ottawa - Canada, 2001.
24. S. Pingali, D. Towsley, and J. F. Kurose. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. In *Proceedings of the Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 221–230, New York, NY, USA, 1994. ACM Press.
25. David Powel. Group communication. *Communications of the ACM*, 39(4):50–53, April 1996.
26. B. Sabata, M. Brown, B. Denny, and C. H. Heo. Transport protocol for reliable multicast: TRM. In *Proceedings of the International Conference on Networks*, Orlando - Flórida - USA, 1996.
27. F. B. Schneider. Implementing fault-tolerant service using the state machine aproach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.
28. P. Veríssimo, L. Rodrigues, and A. Casimiro. Cesiumspray: a precise and accurate global clock service for large-scale systems. *Journal of Real-Time Systems*, 12(3):243–294, 1997.
29. B. Whetten, L. Vicisano, R. Kermode, M. Handley, S. Floyd, and M. Luby. Reliable multicast transport building blocks for one-to-many bulk-data transfer (rfc 3048). IETF Request For Comments, January 2001.