Integrating the Unreliable Multicast Inter-ORB Protocol in MJaco

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

[1] Laboratório de Controle e Microinformática - DAS
Universidade Federal de Santa Catarina - Florianópolis - SC Brazil
{neves,fraga}@das.ufsc.br,
[2] Graduate Program in Applied Computer Science - CCET
Pontifical Catholic University of Paraná - Curitiba - Paraná - Brazil
{lau,alcides}@ppgia.pucpr.br

Abstract. This paper presents our experience in implementing OMG Unreliable Multicast Inter-ORB Protocol specifications into an ORB. An integration model is proposed to allow the coexistence of two different protocol stacks (IIOP/TCP/IP and MIOP/UDP/IP multicast) making possible a large spectrum of middleware support for distributed objects communication. That integration model is discussed in this paper, giving evidence of the compatibility of our approach with the CORBA specifications. In order to evaluate that integration, different tests were made considering interoperability, performance and scalability aspects.

1 Introduction

Nowadays, the CORBA architecture (Common Object Request Broker Architecture) [9] constitutes the most important middleware specification for supporting distributed objects. The main component of this architecture is the ORB (Object Request Broker) which, among other things, implements the communication semantics defined in the architecture. Messages and formats follow the General Inter-ORB Protocol (GIOP) which is a generic protocol used to support remote invocation. GIOP is a protocol that allows communication despite of different ORB implementations and transport technologies. The Internet Inter-ORB Protocol (IIOP) specifies how GIOP messages are exchanged using TCP/IP connections. Although the IIOP and TCP/IP combination provides a reliable solution for point-to-point communications - handling flow control errors, FIFO order, etc. - many other communication paradigms, when implemented over these protocols, may not use some important characteristics of lower levels of the network. This difficulty always reflects in the performance costs of these paradigms.

Some distributed applications depend on abstractions such as group communication to allow a sender to multicast messages to many receivers. These applications (e.g. groupware systems) require a better employment of network services. In 1999, the OMG published a request for proposal defining a set of requirements for an unreliable multicast service based in IP multicast. IP multicast

is an extension of the Internet Protocol (IP) that enables multipoint communications [2]. This protocol is characterized by the absence of guarantees and by it's high performance, particularly in local network. Therefore, in 2001, the OMG published the UMIOP specifications (Unreliable Multicast Inter-ORB Protocol) [10], which specifies a concretization of GIOP over the stack UDP/IP multicast. UMIOP is the key for providing an unreliable multicast into the ORB.

The group paradigm in open systems has been the theme of many research projects [6, 4, 7] and standardization proposals [3, 8, 10]. To provide the concept of group to distributed applications, it is necessary a combination of protocols that deal with group management and group communication. Within the OMG, these facilities are being standardized separately. Group management (e.g. fault detection and membership) is defined in the FT-CORBA specifications [8][3]. However, OMG has not yet published a specification for group communication in the CORBA architecture that meets different levels of guarantees and semantics available in the usual group processing models [5]. OMG started to deal with group communication with the publication of the UMIOP specifications, for unreliable multicast - the least restrictive model of group communication.

This paper presents our experiences with the integration of UMIOP into an ORB that complies with OMG specifications. Our integration model is discussed in this text, which shows evidence of the compliance of our approach to the CORBA specifications. For evaluating this implementation, tests were performed concerning interoperability, performance and scalability aspects. We present comparative measures collected with our unreliable multicast mechanism (based on the MIOP[4]/UDP/IP multicast stack), and the use of UDP sockets for IP multicast communication.

In section 2 the paper presents a short description of the IP multicast. Aspects of the specification describing the MIOP and the necessary structures to the ORB for providing unreliable multicast are presented in section 3. In section 4, we propose an integration model aiming to preserve two protocol stacks: one for point-to-point, and the other for multipoint communications. Details of implementation are described in section 5. In section 6, some tests and measures are presented and discussed and, finally, in section 7, the final considerations of this work are presented.

2 IP Multicast

Through IP multicast service, it is possible to send an IP packet to a group of hosts identified by an IP address. When a datagram is sent to a group, an attempt is made to deliver it to each host member of this group. However, as in IP unicast, there is no guarantee of delivery. That is, this service does not provide any guarantees concerning: the delivery of packets to all members of the group, the integrity of the packets and the packet delivery order.

[3] Currently the FT-CORBA specification is part of the CORBA specification [9].

[4] UMIOP is the name of the specification, MIOP is the name of the protocol defined in this specification.

Operation	Description
CreateGroup	Create a temporary group with the invoking host as its only member
JoinGroup	Add the invoking host to the specified group (permanent or temporary)
LeaveGroup	Remove the invoking host from the specified group

Table 1. IP Multicast Management Operations.

The IP multicast specifications [3] define two types of groups: permanent groups and temporary groups. Furthermore, there is no membership service. Each host knows what group it belongs to, but does not know about the other members. The management of members of a group is dynamic, so the association of IP addresses with hosts is quite flexible. Thus, at a given instant, a host may belong to one or more groups, or to no group. This dynamic association is carried out through three management operations, provide by the IP module. These operations are shown in table 1.Note that there is no operation for the destruction of groups, which is justified by the fact that permanent groups cannot be destroyed and temporary groups only exist as long as their number of members is more than zero.

Management operations are implemented through the IGMP protocol (Internet Group Management Protocol) [2], which is used in communication among local network hosts and multicast agents. Every network that supports IP multicast must contain at least one active multicast agent, usually implemented in the gateway of the network. The agent is the entity encharged for groups creation and maintenance, and for messages exchange in the Internet. Each multicast agent must know which groups have members in its network.

3 UMIOP Object Model

The current CORBA objects model (stack IIOP/TCP/IP) specifies an Interoperable Object Reference (IOR) which is associated to a single implementation. Furthermore, the CORBA remote method invocation implements an at-most-once semantic based on TCP connections that provides error handling that adds reliability to the point-to-point communication regarding the message delivery and, also, the FIFO ordering that can be defined to wait or do not for a reply.

The delivery of GIOP messages via unreliable multicast service establishes a different communication context from that cited above. The UMIOP specification defines an unreliable multicast mechanism for communications among distributed objects. At distributed object level, these mechanisms are reduced to the mapping of GIOP messages in UDP/IP packets - the MIOP protocol - and some facilities for group management. Information about the group is added in its corresponding reference (group IOR) for the group management. The group management is simplified since reliability is not required in MIOP and a membership service is unnecessary as there is no need to know how many or who are the group members.

Actually, a group of objects in the UMIOP specification consists of information about how to access it through the network. This information is contained in the UIPMC profile, available through the group reference (figure 1). The UIPMC profile differs from the traditional IIOP profile presented in a single object IOR by the following points:

- There is no object key[5];
- The host IP address of the IIOP profile is replaced on the UIPMC by a field that must contain an IP multicast address, or an alias of this address.

Object keys and host addresses absences in the UIPMC profile is pertinent: the unreliable multicast service is defined under the assumption of non-existence of membership. Even so, it is possible to reach group members (an object) from the UIPMC profile, using POAs (Portable Object Adapter[6]) in the hosts that received the multicast message.

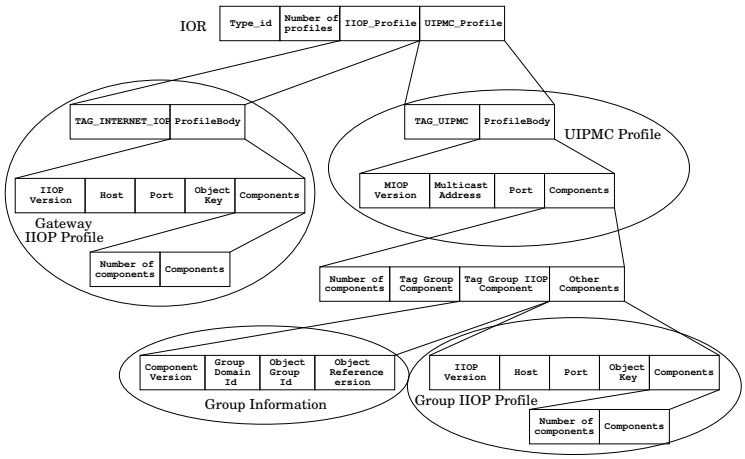

Fig. 1. Group Interoperable Object Reference.

This scenario is much more complex than that of IIOP profiles, which identify only one interface implementation. Besides protocol version, IP multicast address, and port information, the UIPMC profile also provides a set of other components such as: an IIOP profile for requests demanding replies (multicast is used only for oneway requests) and a group information structure which contains group id, domain and version.

[5] The object key is the object id into the ORB. It is used, with the IP address and ORB port, to locate an object implementation.

[6] ORB component responsible to activate implementations and also to forward received messages to these.

Besides the UIPMC profile (mandatory), a group IOR can optionally have one or more IIOP gateway profiles to be used by clients that do not support IP multicast. A group IOR is then used, basically, in three types of invocations:

- When the client is unable to execute multicast through its ORB, an IIOP gateway supplies the information needed in this case;
- When client and servers have UMIOP support, implement the same interface and belong to the same group, UIPMC profile is used for multicasting messages among the group members;
- When a group member is on an IIOP server and communicates using two-way operations, IIOP group profile is needed to support it.

Figure 1 presents the complete structure of a group IOR, with the UIPMC profile, group information, IIOP gateway and IIOP group profile of an object group. As mentioned before, the UIPMC profile does not define object keys to access members of a group. Thus, a POA must use the associated group information (id, domain and version) described at the UIPMC profile to identify local members of the group.

Association among groups and local members implementations requires a new internal table in the POA. The Active Groups Map table, using UIPMC group information, makes accessible through its adaptor the corresponding implementation of local group members. In this way, the inclusion of members in a group is always performed locally through calls to the activated POAs. For that, UMIOP specifications provide four new operations on the interface `PortableObject:POA`. These operations make it possible to associate/dissociate registered implementations to the group references; as well as to make lists of the implementations that belong to a given group, offering, a sort of local membership service.

In figure 2, the Java code shows the association of an object implementation to a group reference. Lines 1 - 4, show how ORB and POA references are obtained. Next, in lines 5 - 7, a group IOR is created through a corbaloc[7] URL. This URL defines a UIPMC profile to the group and an IIOP profile for operations with reply.

After getting the group IOR, lines 8 and 9 present the instantiation of the object `member` and its activation by the POA. The line 10 shows the operation `associate_reference_with_id`, an extension of the POA interface, used for associating the activated implementation to the group reference, and finally, at line 11 the ORB is activated, making it ready to receive requests.

When compared to other group supports (e.g., FT-CORBA), group creation and member association showed in figure 2 can be considered very complex, involving low-level structures manipulations. To avoid this problem, the UMIOP specification defines an optional service object called MGM (Multicast Group Manager), which is introduced to offer a high level group management interface. The MGM service supplies less complex operations for creating and destroying

[7] Used to represent object references as URLs.

```
1  ORB orb = ORB.init(args,null);
2  Object poaObj = orb.resolve_initial_references("RootPOA");
3  POA poa = POAHelper.narrow(poaObj);
4  poa.the_POAManager().activate();
5  Object group = orb.string_to_object("corbaloc:" +
6    "miop:1.0@1.0-mydomain-1-2/225.1.2.5:7676;" +
7    "iiop:1.1@localhost:1236/MyObjectKey");
8  MyGroupMemberImpl member = new MyGroupMemberImpl();
9  byte[] memberId = poa.activate_object(member);
10 poa.associate_reference_with_id(group,memberId);
```

Fig. 2. Association in POA of a Group IOR with an Implementation Object.

groups. Other operations for managing groups properties (group IIOP components, gateway, multicast address and port) are available too.

At message level, a GIOP message is always encapsulated in a set of MIOP packets. A MIOP packet is composed by a header and a GIOP block of data. The maximum size of a GIOP data block that can be contained in a MIOP packet usually depends on the frame size supported by the network utilized.

4 UMIOP Integration Model

Figure 3 presents our UMIOP integration model, implemented in the MJaco project. MJaco is an extension of JacORB [1], a CORBA compliant ORB. The architecture MJaco is proposed to allow the co-existence IIOP/TCP/IP and MIOP/UDP/IP multicast protocol stacks in the same ORB, contributing in this way, for a better interoperability and portability. In figure 3, we presents a ORB integrating two protocol stacks: one for point-to-point communication based on IIOP mapped on TCP/IP services; and the other one for multipoint communication composed by MIOP, using UDP/IP multicast as a transfer mechanism for its packages. Our integration model presents several elements defined in the specification in its support with the two communication models. Also, extensions and other components not defined in the specifications were added to the support whose purpose is to facilitate the different stacks integration and improve the efficiency of the set.

Our approach proposes a local object service MGM+ which extends the MGM of the UMIOP specifications, by adding functionalities for group management at object level. By creating group references, implemented through ORB calls, MGM+ registers the created group IOR on a CORBA name server, making the group reference automatically available to any other application. Other MGM+ facilities concerning the membership changes (locally) are available through `add_member` and `delete_member` operations. These operations centralize in the MGM+ the interactions for associating groups and implementations of local members through active groups map of the POA (section 3).

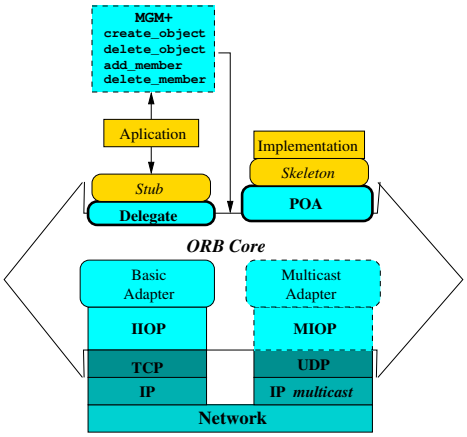

Fig. 3. MJaco Architecture.

Another component, part of our integration model, is the Multicast Adaptor which is responsible for managing the multicast sockets used in the reception of MIOP packets and for delivering messages addressed to group members by the active POAs. MIOP module executes tasks described in the UMIOP specification which refers to the translation of the GIOP messages into collections of MIOP packets and vice-versa. POA and Delegate are the main ORB components that were extended. Delegate is altered in some points in order to support the sending of GIOP messages to groups; it is the first internal component of the ORB to be activated when a method call is executed on the stub. In our approach, the delegate decides, based on the corresponding method call, which protocol stacks will be used to send a GIOP message. The POA, besides the addition of the four primitives described in the UMIOP specification, must be altered to search the active groups map in order to obtain implementations of local group members. The local group members receive through the POAs the messages addressed to the group to process the corresponding method requests. POA is also responsible for the activation of the multicast adaptor which executes operations on IP multicast interface for group management. For example, when an `associate_reference_with_id` call is made to register the first member of a group in the ORB, POA activates the multicast adaptor to create a socket and to execute the IP multicast operation *JoinGroup* at the address defined on the UIPMC profile present in the group reference. From then on, the ORB receives messages addressed to this group. The other POA operations related to groups result in the execution of IP multicast management operations.

5 MJaco: Implementing UMIOP on JacORB

For implementing the UMIOP specifications and the model presented in the section 4, the JacORB platform (*http://www.jacorb.org*) was selected. JacORB

is a open source ORB that was chosen because of its recognized quality and performance and our experience with it in other two projects: GroupPac and JaCoWeb (*http://www.das.ufsc.br/grouppac*).

Figure 4 presents a class diagram in UML describing classes that process requests within the client JacORB. The set of classes shown are activated when a CORBA client makes a method call on the stub. The stub is responsible for the serialization of the method invocation; the message is sent using the `Delegate` class. This class maintains an open TCP connection to the server ORB, where the implementation corresponding to the stub is registered and it implements the operations related to the `CORBA::Object`, the base interface of all defined for CORBA objects.

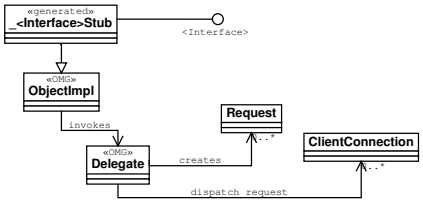

Fig. 4. Request Processing on CORBA Clients.

On the server side, the requests processing is much more complex. Figure 5 presents a class diagram with the main components in the request processing activated by a GIOP message arrival in the socket.

An object of the `Listener` class waits for requests on one TCP connection. When a request is received in this port, the object creates a new instance of the `RequestReceptor` class, which is responsible for interpreting the request and forwarding it to a destination POA[8]. Once the POA is located, the request is placed in a list to be processed by the target implementation (subclass of `Servant`). Note that some classes showed in figure 5 have the stereotype thread; these classes represent the points of JacORB structure that implement their multithreading architecture.

To process MIOP packets, a new structure of classes must be added to the JacORB diagrams (showed in figures 4 and 5) to represent classes that send and receive MIOP packet collections. Figure 6 presents a new diagram with these new classes. In this diagram we identify `Delegate` class, within the ORB, as the point for communication stack selection. In this class, when the IOR of the destination object corresponds to a group reference and the requested operation is oneway, the processing is deviated to the class `MulticastSender`, which encapsulates the GIOP request in a collection of MIOP packets to be sent via IP multicast.

On the server, an object of `MulticastListener` class is created for each group in which the ORB has members registered (one listener for each port and

[8] The POA in which the target implementation is registered.

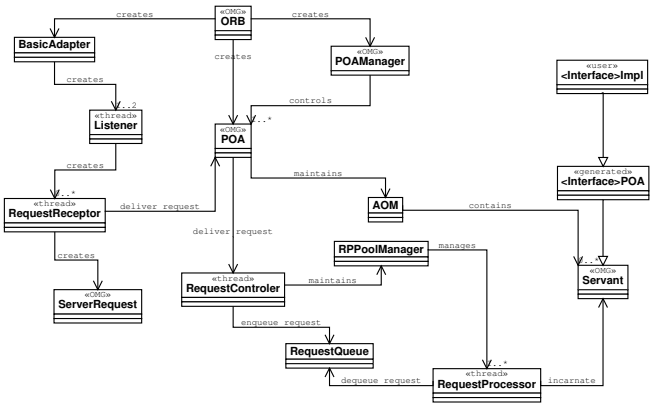

Fig. 5. Request Processing on CORBA Servers.

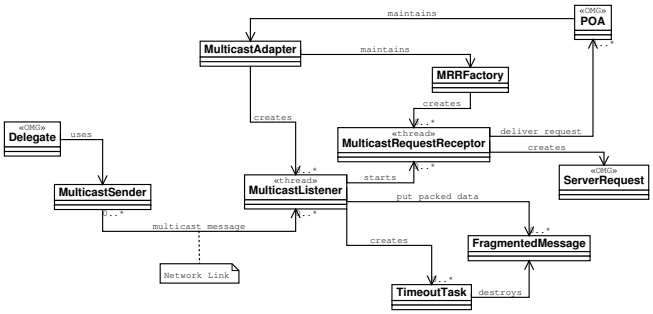

Fig. 6. UMIOP Extensions.

one port for each group) by the multicast adaptor (class `MulticastAdapter`). These listeners receive and store MIOP packets until to complete the corresponding GIOP message. When that message is completed, a thread (instance of class `MulticastRequestReceptor`) is activated from the thread pool to receive the original GIOP message and pass it to all POAs of the ORB. In each POA, using the active groups map (`AGM`) and group information contained in the header of the GIOP message, a set of local object identifiers belonging to the target group are located in this POA.

Note that the request is forwarded to all the active POAs, even those that do not register objects implementations of the destination group. This algorithm for delivery was developed with the objective of making the class `AGM` as simple as possible and of avoiding the processing overhead imposed by the destination POA search. Another factor that justifies this option is the fact that in the majority of the applications, not many POAs are activated at the same time.

6 Obtained Results

This section presents some tests performed with the aim of evaluating the implementation performance of MJaco. The tests were executed on a local network using computers with the same configuration[9], connected to the same hub. Two versions of a distributed program were implemented for measuring round-trip times[10]: the first one using multicast sockets and the other using MJaco.

The first experiment was setup as follows: two instances of the test program were initiated on four machines, thus eight members in the group. One of these members was the sender that sent message of variable size to the group. To configure the round-trip, the sender was kept waiting for all the confirmations from members who received that message. This procedure was repeated 10000 times. The experiment was executed with each of the program versions built, and the results obtained are shown in figure 7.

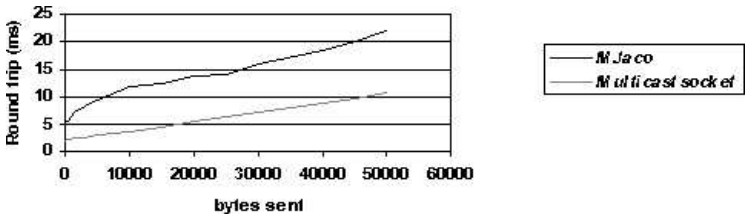

Fig. 7. MJaco Performance.

In figure 7, it can be verified that the use of multicast sockets program results in a performance approximately 60% better (on the average) than the one using MJaco. This was expected, since MJaco places a whole software layer for object and request management, as well as for message serialization (see figure 3); but distributed programs using multicast sockets in some part of the application code might have to perform the same functionalities that our little multicast sockets program does not implement. It is also worth pointing out that the MJaco implementation was realized with the least possible dependence on JacORB, which caused some performance losses that must be minimized since the implementations are evolving.

An important point to be emphasized is that, in this current version of the prototype, starting with 50 Kbyte messages the system begins to lose some messages. That is due to the weak reliability of the UDP/IP communications which does not avoid packet losses.

In the second experiment we evaluated the scalability of MJaco. The tests were executed in following way: on each host an instance of the test program

[9] Pentium III 900 Mhz with 198 Mb RAM memory over Ethernet 10Mbps.

[10] round-trip time represents the time period between the beginning of the message multicasting to all the group members and finishing with the sender reception of the last confirmation message sent by the receiving members.

was initiated, specifying how many objects of the group had to be instantiated and registered on the MJaco support. One of the object members created on the system hosts was the sender, the one that multicast the message with 4 Kbytes of data and was responsible for measuring the round-trip times. The number of messages multicast during a round-trip was $4n + 1$ where n is the number of group objects registered in each host (one request message and an acknowledgement for each member object of the group). The round-trip process was repeated for a different number of objects per host and the results of these experiments are shown in figure 8.

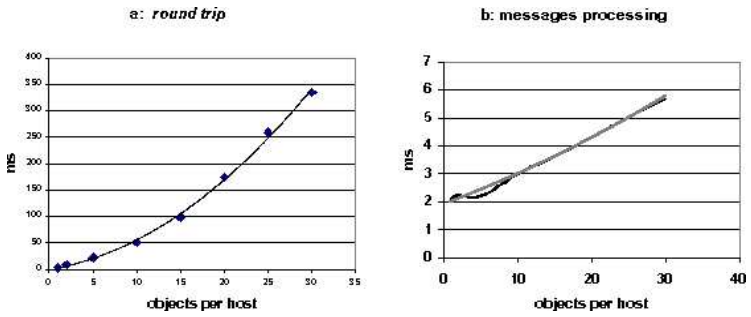

Fig. 8. MJaco Scalability.

Figure 8.a shows a situation in which, when we increase the group members, the round-trip time also increases considerably. This is due to the substantial increasing of the number of messages generated on the network. For instance, with two objects per host (eight members in the group), nine messages are exchanged for one round-trip (one data message and eight acknowledgements). For ten objects per host, 41 messages are exchanged in the network. So, if the round-trip duration is divided by its corresponding message number, an estimative is obtained from average transfer time for one message in the system. Figure 8.b shows these values with respect to the number of group members. In this graph, it is possible to verify that this average time is linear, and increases approximately 0.1 ms for each new object included in each system host. According to the results of this last test, we conclude that the solutions adopted in MJaco prototype presents a good scalability.

7 Conclusions

The Unreliable Multicast Inter-ORB Protocol is the first step towards a complete group communication solution in the CORBA architecture. We are developing studies on multicast protocols and evaluating different ways to adapt them to the CORBA, especially the UMIOP specifications, since the objective of this project is to implement a reliable multicast mechanism over MIOP on MJaco.

For this reason, questions involving loss of packets, performance analysis, fault-tolerance, etc, were not discussed here. Our intention, in this paper, is to have an unreliable multicast support available into an ORB.

In this paper our solutions were presented for the integration of IP multicast into a CORBA ORB. The integration model proposed does not jeopardize aspects of interoperability and portability of ORB as a whole. The ORB is capable of making invocations using both IIOP and MIOP. This model can very well be adopted to integrate other communication protocols, since they have an available API. Furthermore, our experiences with MJaco implementation was also presented in this paper. This implementation, which had as a basis the integration model proposed, were built over JacORB, a Java free ORB. These implementations can be obtained on the Web at *http://grouppac.sourceforge.net/*.

We are now focusing our efforts to integrate MIOP to the FT-CORBA infrastructure. The solutions presented here are part of GroupPac project, which uses the Fault-Tolerant CORBA specifications, towards the conception of active replication models [11], inexistent in current OMG specifications.

References

1. Gerald Brose. Jacorb: Implementation and design of a javaorb. In *Proceedings of IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems*, 1997.
2. S. E. Deering. Host extensions for ip multicasting (rfc 988). IETF Request For Comments, July 1986.
3. S. E. Deering and D. R. Cheriton. Host groups: A multicast extension to the internet protocol (rfc 966). IETF Request For Comments, December 1985.
4. Pascal Felber, Benot Garbinato, and Rachid Guerraoui. The design of a CORBA group communication service. In *Proceedings of the 15th Symposium on Reliable Distributed Systems*, pages 150–159, Niagara-on-the-Lake, Canada, 1996.
5. Vassos Hadzilacos and Sam 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.
6. Silvano Maffeis. Adding group communication and fault-tolerance to CORBA. In *Proceedings of the USENIX Conference on Object Oriented Technologies*, pages 135–146, Monterey, Canada, June 1995.
7. L. Moser, P. Melliar-Smith, P. Narasimhan, R. Koch, and K. Berket. A multicast group communication protocol, engine, and bridge for corba. *Concurrency and Computation Pratice and Experience*, 13(7):579–603, June 2001.
8. Object Management Group. Fault-tolerant corba specification v1.0. OMG Standart, 2000.
9. Object Management Group. The common object request broker architecture: Core specification v3.0. OMG Standart formal/02-12-06, December 2002.
10. OMG. Unreliable multicast inter-orb protocol specification v1.0. OMG Standart ptc/03-01-11, October 2001.
11. Fred B. Schneider. Implementing fault-tolerant service using the state machine aproach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.