11-2003

OCI-Based Group Communication Support in CORBA

Dongman Lee
Information and Communications University, dlee@icu.ac.kr

Dukyun Nam
Sung Kyun Kwan University, paichu@icu.ac.kr

Hee Yong Youn
Sung Kyun Kwan Uinversity, youn@ece.skku.ac.kr

Chansu Yu
Cleveland State University, c.yu91@csuohio.edu

Publisher's Statement

Original Citation

Dongman, L., Dukyun, N., Hee, Y. Y., & Chansu, Y. (2003). OCI-based group communication support in CORBA. IEEE Transactions on Parallel and Distributed Systems, 14, 11, 1126-1139.

Repository Citation

Lee, Dongman; Nam, Dukyun; Youn, Hee Yong; and Yu, Chansu, "OCI-Based Group Communication Support in CORBA" (2003). *Electrical Engineering & Computer Science Faculty Publications*. 94.
https://engagedscholarship.csuohio.edu/enece_facpub/94

OCI-Based Group Communication Support in CORBA

Dongman Lee, *Member*, *IEEE Computer Society*, Dukyun Nam,
Hee Yong Youn, *Senior Member*, *IEEE Computer Society*, and Chansu Yu, *Member*, *IEEE*

Abstract—Group communication is a useful mechanism guaranteeing consistency among replicated objects. The existing approaches do not allow transparent plug-in of group communication protocols into CORBA. They either require modification of CORBA or OS, or provide no room for incorporating group communication transport protocols into CORBA. We thus propose a generic group communication framework that allows transparent plug-in of various group communication protocols with no modification of existing CORBA. We extend the Open Communications Interface (OCI) to support interoperability, reusability of existing group communication, and independency on ORB and OS. We also define the Group Communication Inter-ORB Protocol (GCIOP) as a group communication instantiation of the General Inter-ORB Protocol (GIOP) that encapsulates underlying group communication protocols. The proposed scheme can be exploited for fault-tolerant CORBA (FT CORBA).

Index Terms—Group communication, CORBA, open communications interface, FT CORBA.

1 Introduction

Object replication is a technique enhancing fault tolerance and high availability [7]. An object group is a collection of object replicas that cooperatively work for a common task [16]. A consistent state for an object group can be used for constructing highly available and fault-tolerant distributed applications. Group communication service (GCS) is a useful mechanism guaranteeing consistency of the states of all the member objects. It maintains a view, a list of the currently active and connected members in an object group, and also informs the running objects of the updated view whenever it changes. The consistency can be guaranteed by reliable delivery of messages to the members in the current view.

There have been several approaches for supporting group communication service in CORBA. In general, they can be categorized into three approaches—*integration*, *service*, and *interception* approach. An example of each of the three approaches is Electra [13], Object Group Service (OGS) [4], and Eternal [19], [22], respectively. In the integration approach, the GCS module is an integral part of Object Request Broker (ORB) and, thus, introducing a new GCS requires modification of ORB. The service approach provides group communication as Object Service [5] on top of CORBA, leveraging the CORBA's point-to-point communications, General Inter-ORB Protocol (GIOP). That is, no room exists for group communication transport protocols to be incorporated into CORBA. The interception approach supports group communication service by intercepting the system calls related to group communications using an interceptor. It does not allow CORBA objects to directly exploit the underlying group communication services since the interceptor is not part of CORBA. In summary, the existing approaches do not support transparent plug-in of group communication protocols into CORBA and, thus, CORBA application programmers cannot directly exploit the protocols. In the above examples, there must be a generic group communication framework that allows transparent plug-in of various group communication protocols via a standard CORBA interface.

In this paper, we propose a mechanism that allows such framework with no modification of existing CORBA. We leverage the OCI (Open Communications Interface) [1] to support interoperability, reusability of existing group communication, and independency on ORB and OS. The OCI provides, as part of the CORBA, a set of interfaces by which various protocols instead of GIOP/IIOP over TCP/IP can be supported. We add new operations into the existing OCI by which an object group can be managed and invocations are made to the group. We define Group Communication Inter-ORB Protocol (GCIOP) as a group communication instantiation of GIOP that encapsulates underlying group communication protocols. We then add group communication Info Object to the OCI and extend the interfaces of the OCI to support group semantics. The existing OCI consists of a Connector, Connector Factory, Acceptor, and Transport module [1], [10]. The existing group communication basically requires group address expansion, delivery ordering, and state transfer [2]. To satisfy these requirements of group communication, a group name, ordering type, and state are saved in the Info object. We add an attribute to Connector's Transport Info for setting the ordering-type and add group maintenance operations to Acceptor. The proposed design can be used to wrap or embed various group communication protocols without any modification of ORB and the OCI. We devise a group Interoperable Object Reference (IOR) for GCIOP in order to support the non-FT CORBA compliant systems that

- D. Lee and D. Nam are with the School of Engineering, Information and Communications University, 58-4 Hwaam-dong, Yuseong-gu, Daejeon, 305-732 Korea. E-mail: {dlee, paichu}@icu.ac.kr.
- H.Y. Youn is with the School of Information and Communications Engineering, Sungkyunkwan University, 300 Chunchun-dong, Jangan-gu, Suwon 440-746 Korea. E-mail: youn@ece.skku.ac.kr.
- C. Yu is with the Department of Electrical and Computer Engineering, Cleveland State University, Stilwell Hall 340, Cleveland, OH 44115. E-mail: c.yu91@csuohio.edu.

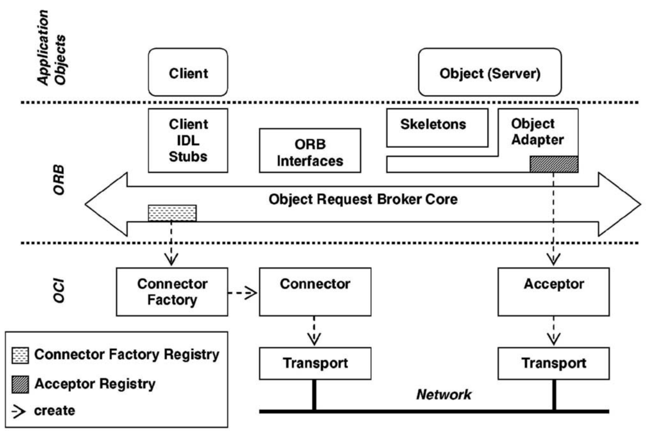


Fig. 1. The structure of OCI in CORBA.

do not support group object references. The proposed scheme can be exploited for fault-tolerant infrastructure based on the FT-CORBA standard [25]. The Interoperable Object Group Reference (IOGR) as a group reference is used instead of the GCIOP-based IOR when the proposed scheme is used by FT CORBA. For composing and invoking an object group except resolving the IOGR, the proposed scheme can be directly adapted to the FT CORBA standard. Resolving the IOGR should be included in the Connector Factory. We implement the proposed scheme on top of our group communication protocol [15] using ORBacus Java 4.01 [10] in which the OCI is provided as part of the CORBA interface. Experiment results show that the proposed scheme does not incur performance degradation as the number of members in a group increases.

The remainder of the paper is organized as follows: Section 2 describes the proposed group communication extension to the OCI for group communication and presents how the proposed extension is used with applications. Section 3 describes the implementation details of the proposed extension. The performance analysis of the proposed extension is presented in Section 4. Section 5 discusses the existing approaches for group communication in CORBA. A conclusion follows in Section 6.

2 The Proposed Mechanism for Group Communication

In this section, we describe the requirements of supporting group communication service and how a given transport protocol is integrated into it. We then present the proposed extension to OCI for group communication including the IDL specification and execution flow of the extension.

2.1 Communication Using OCI

The OCI [1] provides plug-in protocol interfaces for CORBA. The interfaces are Buffer, Acceptor, Transport, Connector, Connector Factory, Registries, and Info objects [10]. Buffer holds data in an array of octets and maintains a position counter. The array and counter are used in communications between a client and a server. Info objects provide information on Acceptor, Transport, Connector, and Connector Factory.

Fig. 1 shows how the OCI incorporates a given transport protocol with ORB. As a client and a server are activated, an Acceptor Registry in Object Adapter (OA) activates an Acceptor, and a Connector Factory Registry in ORB creates a Connector Factory. Then, at the server side, an Acceptor creates profile information for an Interoperable Object Reference (IOR), and OA creates an IOR using it. At the client side, a Connector Factory creates a Connector using the IOR. Then, the Connector and Acceptor instantiate their own Transport. Once a connection is made between the Transports at each side, the communications between a client and a group are done directly via the Transports.

At the client side, a stub exists which plays a role of a server. When a client invokes a method on a stub, the client's ORB marshals the method call and sends it to the server. In detail, ORB invokes the "send" operation of Transport that, in turn, sends a Buffer object including a request message via an associated transport protocol. At the server side, Transport passes the call to the server's ORB that unmarshals the call and invokes a corresponding operation in the server. All objects including each Info object of the OCI part must be implemented to support specific protocols. For example, a transport protocol for IIOP is TCP. Note that a Connector Factory and an Acceptor in OCI are exposed only to application objects.

Fig. 2 shows an IDL specification of OCI[1] that was proposed as an extension to CORBA for interworking with intelligent networks [1]. Buffer provides an interface for a buffer that holds an array of octets for data exchange between a client and a server. Its pos attribute gives information on how many octets have been sent or received. All interfaces except Buffer have a tag associated with the transport protocol. For example, the tag value is "org.omg.IOP.TAG_INTERNET_IOP.value" in the case of IIOP plug-in. The Acceptor interface is used for a server to accept a connection request from a client while the Connector interface is used for a client to make a connection to a server. The Transport interface provides methods for sending and receiving octet streams. A ConFactory serves as a factory for Connector objects. A ConFactoryRegistry and an AccRegistry serve as a registry for ConFactory objects and Acceptor object, respectively. These registries are provided by the plug-in implementors. A ConFactoryRegistry is created by the ORB while an AccRegistry is done by the OA. The Current interface provides methods that return Transport and Acceptor information objects related to the current request.

2.2 Group Communication Service Extension to OCI

To provide a group communication service to CORBA objects, three aspects should be considered—group address, group membership, and message ordering. We assume that the underlying group communication protocol is responsible for reliable delivery of group messages. A client application sending a message to the members of a group does not supply a member list. Instead, a membership service supplies a group address as a group identifier. It is mapped to a current membership list, hiding the group's internal structure from applications. The group membership service provides a set of operations to create and change the membership, and guarantees a mutually consistent view among group members. A membership list

1. In ORBacus [10], the OCI interface is extended to support methods such as callback and describe.

```
module OCI
{
interface Buffer {
      readonly attribute unsigned long length;
      attribute unsigned long pos;

      void advance(in unsigned long delta);
      unsigned long rest_length();
      boolean is_full();
};

interface AcceptorInfo {
      readonly attribute ProfileId tag;
};

interface ConnectorInfo {
      readonly attribute ProfileId tag;
};

interface TransportInfo {
      readonly attribute ProfileId tag;
};

interface Acceptor {
      readonly attribute ProfileId tag;
      readonly attribute Handle handle;

      Transport accept();
      void add_profile(in ObjectKey key, in IOR ior);
      boolean compare(in IOR ior1, in IOR ior2);
      AcceptorInfo get_info();
};

interface Connector {
      readonly attribute ProfileId tag;

      Transport connect();
      boolean compare(in IOR ior);
      boolean compare_with_policies(in IOR ior,
            in CORBA::PolicyList policies);
      ObjectKey extract_key(in IOR ior);
      ConnectorInfo get_info();
};
```

```
interface Transport {
      readonly attribute ProfileId tag;
      readonly attribute Handle handle;
      readonly attribute unsigned long fragmentation;

      void close();
      void receive(in Buffer buf, in boolean block);
      boolean receive_detect(in Buffer buf,
            in boolean block);
      void receive_timeout(in Buffer buf,
            in unsigned long timeout);
      void send(in Buffer buf,
            in boolean block);
      boolean send_detect(in Buffer buf,
            in boolean block);
      void send_timeout(in Buffer buf,
            in unsigned long timeout);
      TransportInfo get_info();
};

interface ConFactory {
      readonly attribute ProfileId tag;

      Connector create(in IOR ior);
      Connector create_with_policies(in IOR ior,
      in CORBA::PolicyList policies);
      boolean consider_reference(in IOR ior,
      in CORBA::PolicyList policies);
      boolean compare(in IOR ior1, in IOR ior2);
};

interface ConFactoryRegistry {
      void add_factory(in ConFactory Factory);
};

interface AccRegistry {
      void add_acceptor(in Acceptor Acceptor);
};

interface Current : CORBA::Current {
      TransportInfo get_oci_transport_info();
      AcceptorInfo get_oci_acceptor_info();
};
}; // end module OCI
```

Fig. 2. IDL specification of OCI.

can be managed by a membership server or by exchanging a view change message between the members. Delivery ordering determines whether messages should reach all of its members at the same time [2]. This guarantees a consistent state among the members. Ordering types are total and causal ordering that manage concurrent messages and sequences of related messages.

The existing CORBA object reference denotes exactly one CORBA object. In the proposed extension to the OCI, a **group address** is provided so that a client can make a call to a group of objects transparently as if it does to a singleton object. The proposed extension creates a group object reference using a group identifier provided by a group communication protocol. A group object reference is denoted as an IOR and expanded into a group identifier at OCI Transport. Client establishes a communication path to a group using the corresponding IOR. Here, concrete implementation of GIOP is needed for group communication because GIOP is an abstract protocol that does not have a group address available as endpoint information. We therefore extend GIOP to Group Communication Inter-ORB Protocol (GCIOP).

Group membership allows the currently active and connected member objects in a group to maintain a consistent group view by mapping a group identifier to a membership list. The proposed extension provides a set of interfaces to group membership operations such as group creation, destruction, join, and leave because this extension is the interfaces binding a group communication protocol with ORB. Group operations are provided as part of an Acceptor in OCI. The state transfer interface is provided via an Acceptor, and the state information is saved in Acceptor Info as a CORBA object type.

Message ordering is a key factor for enabling object group members to maintain a consistent state among them when more than one client interacts with the group. The object group members can exist in different machines and receive a single copy of a message in a different order. To maintain a consistent state, arrival messages at each member must be performed in the same order. Therefore, the client needs to have an ability of controlling the order of message delivery. In the proposed design, the client is able to set an ordering type by changing the value of an ordering type attribute in Connector's Transport Info. Default ordering type is "Total ordering," and it is stored in Transport Info.

Fig. 3 shows how the underlying group communication protocol is encapsulated through the OCI. The following

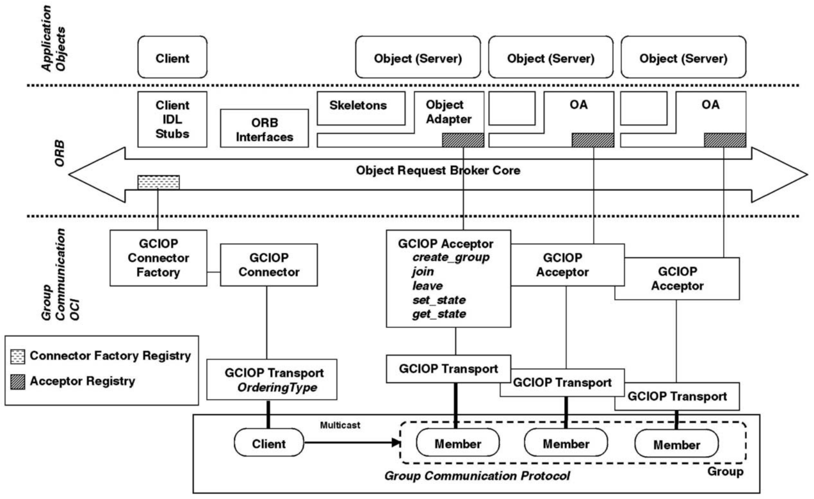


Fig. 3. Extended OCI and group communication protocol.

sections describe the details of the proposed extensions to the OCI for group communication service.

2.2.1 Group IOR

As a server object is represented by an IOR that ORB offers in the existing CORBA communication, the group communication OCI provides a group IOR to identify a group in CORBA group communication. When an object exports its service, ORB generates a corresponding IOR that includes transport information such as an IP address and a port number. ORB uses an IOR as a universal means of identifying an object. As such, a group IOR consists of a group name and a host name. An IOR is generated at runtime by ORB at the server, and interpreted by a client application object. It consists of Type ID, Profile Count, and Tagged Profiles. Type ID is based on the IDL of an object and provides the interface type of the IOR in the repository ID format. Profile Count is the number of Tagged Profiles. One or more Tagged Profiles exist in an IOR and the Tagged Profile contains the information on the protocol supported by a target object.

Fig. 4 shows the format of a Tagged Profile used in GCIOP. **Tag** indicates the protocol used for object communications that is represented by a constant value. Since "01" has already been assigned for IIOP, we choose a different number for the proposed scheme. **Group Name** is used as a group identifier not only in ORB, but also in the underlying group communication protocol. **Host Name** and **Port** represent a group member. **Object Key** identifies a particular object instance.

FT CORBA [25] defines the Interoperable Object Group Reference (IOGR) as a group object reference. We devise the GCIOP-based Interoperable Object Reference (IOR) for non-FT CORBA compliant systems that do not support group object references. When the proposed scheme is applied to the FT CORBA infrastructure, we can use the IOGR instead of the GCIOP-based IOR that encompasses all the attributes required by the proposed scheme. Fig. 5 shows the IIOP profile for the IOGR in FT CORBA.

2.2.2 OCI Information Structure

Connector, Connector Factory, Acceptor, and Transport have a corresponding Info object, respectively. All Info classes are derived from Info class of the existing OCI by adding some attributes, while the operations remain the same. This allows generic specification and implementation of various plug-in protocols in OCI. Fig. 6 shows the IDL specification of OCI Info classes for group communication. ConnectorInfo stores a group name, and Connector uses ConnectorInfo when a message is sent to a group. ConnectorInfo is maintained at the client side's OCI.

A server OCI maintains AcceptorInfo that consists of a group name and a host name. The group name identifies the group that the server either creates or joins. The host name indicates a host where the server resides and, thus, it is actually a server name. Both the names are used when constructing GCIOP profile information as shown in Fig. 4. State of AcceptorInfo represents the state of a group member in a group. It is used for state transfer when a new member joins a group. The replication policy of the underlying group communication protocol can be set by replication_style attribute of AcceptorInfo. TransportInfo is used at both the client side and server side when a Connector and an Acceptor create their corresponding Transport, respectively. Connector and Acceptor have the

Tag	Group Name	Host Name	Port	Object Key

Fig. 4. Tagged profile in GCIOP.

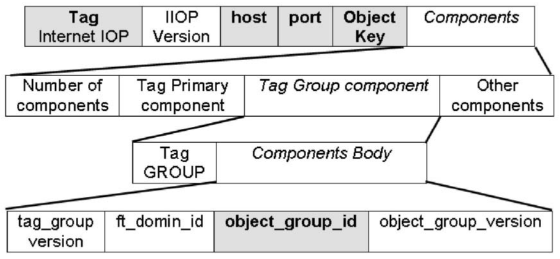


Fig. 5. IIOP profile for IOGR.

```
module OCI
{
module GCIOP
{
enum OrderingType {Total, Causal};
enum ReplicationStyle {Passive, Active};

interface ConnectorInfo : OCI::ConnectorInfo
{
   readonly attribute string group_name;
};

interface AcceptorInfo : OCI::AcceptorInfo{
   readonly attribute string group_name;
   readonly attribute string host_name;
   readonly attribute unsigned short port;
   readonly attribute Object state;
   attribute ReplicationStyle replication_style;
};

interface TransportInfo : OCI::TransportInfo
{
   readonly attribute string group_name;
   readonly attribute string host_name;
   readonly attribute unsigned short port;
   attribute OrderingType ordering_type;
};
}; // End module OCI::GCIOP
}; // End module OCI
```

Fig. 6. IDL specification of information object.

same group name in Transport Info. A client object can change the ordering type at runtime by modifying the ordering_type attribute of Connector's TransportInfo.

2.2.3 *Group Communication Service Interface*

Figs. 6 and 7 show the IDL specification of Information Object and the proposed extension to OCI, respectively. The Acceptor interface is exposed only to application objects. All the existing interfaces of OCI are used without any change. We extend only the Acceptor interface to provide group membership and state transfer such as create_group, join, leave, set_state, and get_state. After creating a group, the first member provides a group reference as a group creator. Then, other members can join or leave the group, and execute state transfer when they join the group.

2.3 Group Membership

The group membership component provides dynamic group membership and guarantees consistency of a group view by using group operations, a view change mechanism, and state transfer. Acceptor's operations of the extended OCI support group composition. This section describes what kinds of messages are exchanged between member objects and a group communication system to create a group and, then, explains the procedure of state transfer and a view change mechanism required when a new member joins.

The replication style of an object group is set to one of the replication styles such as passive replication, active replication, etc., depending on its fault tolerance requirements. In FT CORBA [25], the replication style is set by invoking the method of the property manager within the replication manager. The value for the replication style can be applied to the underlying group communication protocol through the replication_style attribute of AcceptorInfo class.

Fig. 8 shows an interaction diagram for group composition for the passive replication case. Initially, a group is empty. A primary member object calls the create_group() method of Acceptor (1). Then, the create_group() method internally calls a corresponding function and joins a group using the underlying group communication protocol (2). When a view-install message has arrived at the underlying group communication protocol, the reference of the group member is passed to Acceptor (3) and the group creation procedure is finished. When the second member joins the group as a newcomer, the state transfer mechanism is required in addition to the composition of a group. The newcomer object calls the join() method of Acceptor (4) and joins a group in the underlying group communication protocol (5). State transfer is performed before a view install message is sent since the state of a member can contain application-dependent data. The underlying group communication protocol gets the current member's state with the get_state() method of the primary member's Acceptor (6) and duplicates this state to a newcomer's state using the set_state() method (7). Last, a view change complete message is multicasted to all members (8).

```
module OCI
{
module GCIOP
{
typedef IOP::IOR IOR;
enum OrderingType {Total, Causal};

interface Acceptor : OCI::Acceptor
{
   void create_group(in string group_name);
   void join(in string group_name);
   void leave(in string group_name);
   void set_state(in Object state);
   Object get_state();
};

}; // End module GCIOP
}; // End module OCI
```

Fig. 7. IDL specification of group communication extension to OCI.

In FT CORBA [25], the replication manager is responsible for managing a replica and provides interfaces for creating object groups. To create an object group, an external object calls the create_object() method of the replication manager (RM) and then the RM invokes the corresponding method of a local factory in server ORBs. Each factory creates a server replica. In the proposed scheme, the server replica joins a group via the Acceptor's method after creating an object group, i.e., the object group creation procedures of FT CORBA. Fig. 9 depicts a group composition diagram of the proposed scheme based on the FT CORBA standard.

2.4 Group Object Invocation

The proposed scheme assumes one-to-many multicast and, thus, one client transmits messages to multiple servers, that is, a group. A connection between a client and a group is accomplished by the client's Connector and the group's Acceptor. After connection establishment, one Connector and several Acceptors instantiate a Transport object per Connector and Acceptor, and these Transport objects use multicast communication among them. For reply coordination, we assume that one of the group members, e.g., the primary member, collects replies from other members and then returns the result to the client. In this section, we describe how a connection is made between a client and group members using Connector and Acceptors, respectively, and

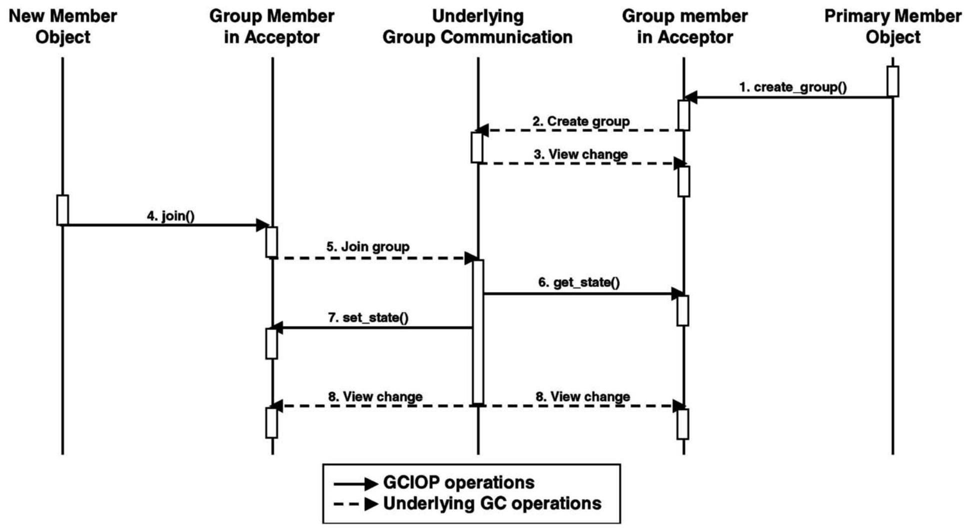


Fig. 8. Interaction diagram for group membership model.

how the methods of an object group are invoked via Transports.

Fig. 10 illustrates the procedure for group object invocation. The server side's ORB first calls the accept() method of an Acceptor (1). The Acceptor waits for a client connect request delivered by the underlying group communication protocol (2). At the client side, the client maps a local proxy onto a remote object reference contained in the IOR (3) and invokes a method of the server (4). The client side's ORB initiates a Connector Factory (5). It creates a Connector based on the IOR and, then, the Connector initiates the client module of the underlying group communication protocol (6-9). After the client side's ORB gets the Connector from the Connector Factory (10), it calls the connect() method of the Connector (11) that initiates a Transport and returns it to the ORB (12-14). For transmitting the client's invocation to the server group, the client side's ORB calls the send() method of the Transport (15) that, in turn, calls the SendMessage() method of the underlying group communication protocol (16). Then, the client side's ORB waits for a reply by calling the receive() methods of the Transport (17). At the server side, as the Acceptor is notified of message arrival (18), it initiates a Transport (19-20) and returns the Transport to the ORB (21). Then, the server side's ORB gets a received message from the underlying group communication protocol by calling the receive_detect() method of the Transport (22-25). Finally, the ORB makes a call to the server object (26). It also calls the send_detect() method of the Transport (27) to notify the client of the success of an invocation. The server side's Transport sends a message to the client (28). The primary member of the underlying group communication protocol gathers the server objects' replies and transmits only one reply to the client. As the client side's Transport recognizes message arrival (29), it gets a message from the underlying group communication protocol (30-31). The client side's ORB obtains the data by using a Buffer object as a parameter of the receive() method (32) of the Transport and, then, the object invocation is complete. In Fig. 10, even though specific interfaces for group invocation are not described, they can be substituted with the corresponding interfaces of underlying group communication protocols.

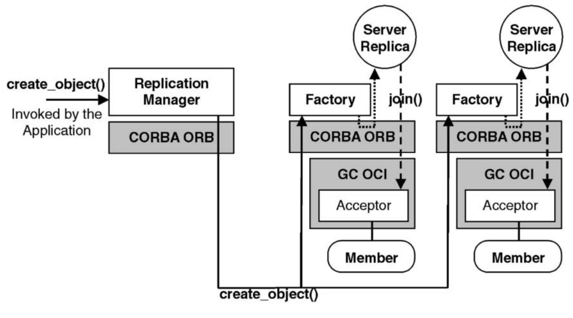


Fig. 9. Group composition based on a fault-tolerant CORBA.

FT CORBA [25] permits three implementation strategies for the IOGR such as "access via IIOP directly to a member of a server object group," "access via IIOP and a gateway," and "access via a proprietary multicast group communication protocol." The last two cases use a group communication protocol as a transport protocol. Fig. 11 shows these cases with the proposed scheme. When a client invokes an object group, a Connector Factory in the client side ORB resolves the information required by the proposed scheme from the IOGR and then fills ConnectorInfo and TransportInfo with it. If a gateway is not used, the proposed scheme can be directly applied to the overall system like Fig. 11a. Otherwise, the gateway should be located between a client object and server replicas. As shown in Fig. 11b, it makes two connections with a client object through IIOP and also with server replicas using the proposed scheme. After the gateway receives a request from a client, it forwards the request to the server replicas.

2.5 Execution Flow of Group Communication OCI

To enable group communication, a client object and a server object perform the initialization procedure by which group communication OCI objects are plugged into ORB. They are Connector Factory, Connector, Acceptor, and Transport. During the initialization, an ORB and POA manager register a GCIOP Connector and a GCIOP Acceptor, respectively.

At the server side, a new GCIOP Acceptor and Acceptor for group communication need to be added to OCI. For this,

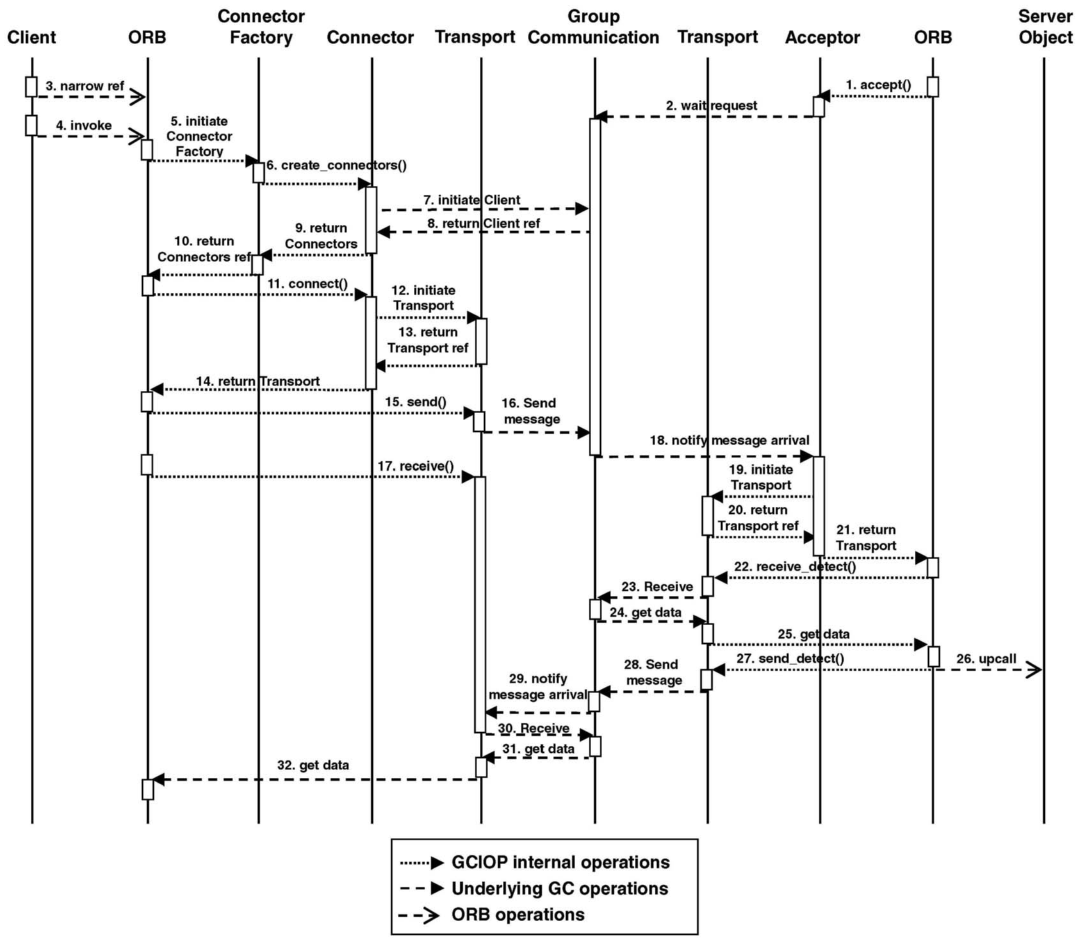


Fig. 10. Procedure for group invocation.

Acceptor Registry is accessed using the OA manager (i.e., get_acc_registry method). Then, a new Acceptor is created and added to ORB by calling the add_acceptor method of Acceptor Registry. At the client side, Connector Factory is created and added by Connector Factory Registry that is provided by ORB. A server gets a reference to a generic Acceptor through OA and Acceptor Registry. Therefore, the Acceptor reference needs to be narrowed to a reference to a GCIOP Acceptor using the narrow method of GCIOP.AcceptorHelper (generated code by an IDL compiler). Then, the "create group" method of GCIOP Acceptor is called and, then, OA creates an IOR. We assume that the first group member creates and publishes the IOR.

When OA is activated, Acceptor creates Transport. Transport instantiates a process group member of the underlying group communication service. According to the information in Transport, i.e., TransportInfo, Transport lets a process group member join or leave a group. Here, a key concept is to map Transport of a server object into a process group member. A member can dynamically join or leave a group using the methods of GCIOP Acceptor. When a new group member joins an existing group, the state of the existing members should be transferred to it. A server gets or sets a state by calling the "get_state" and "set_state" methods of GCIOP Acceptor. The state information is saved in Acceptor Info as an object type. When the state in

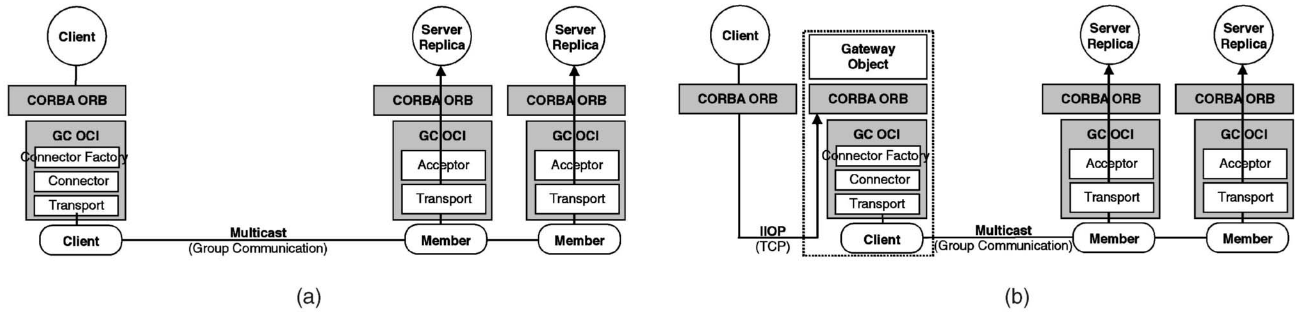
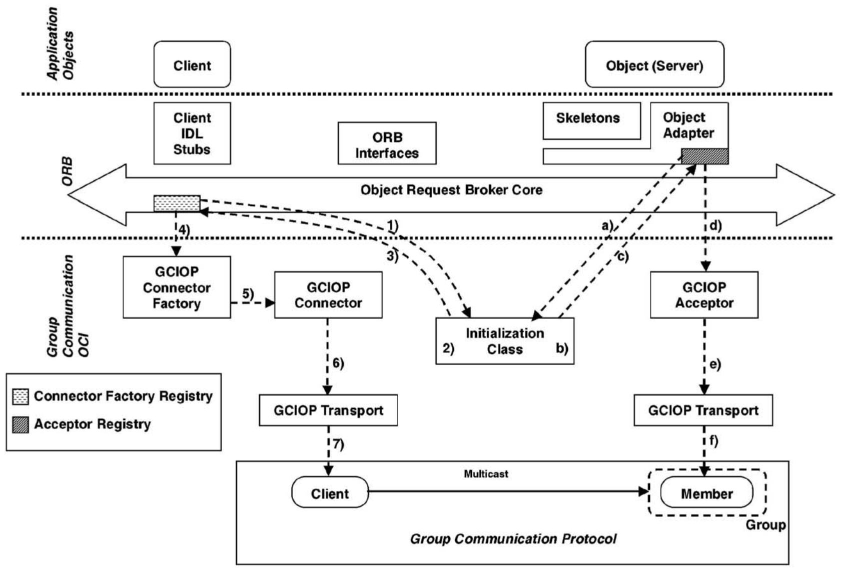


Fig. 11. Group object invocation based on a fault-tolerant CORBA. (a) A case without gateway and (b) a case with gateway.



Client Side
1) get a Connector Factory Registry.
2) create a GCIOP Connector Factory.
3) add the Connector Factory into a Connector Factory Registry.
4) create the Connector Factory.
5) create a GCIOP Connector.
6) create a GCIOP Transport and a GCIOP ConnectorTransportInfo.
7) instantiate a client in Group Communication Protocol.

Server Side
a) get a Acceptor Registry
b) create a GCIOP Acceptor
c) add the Acceptor into a Acceptor Registry
d) create the Acceptor
e) create a GCIOP Transport and a GCIOP AcceptorTransportInfo
f) instantiate a group member in Group Communication Protocol

Fig. 12. Usage of group communication OCI.

Acceptor Info is changed, the state in the underlying group communication service is changed as well.

After the initialization of a client, Connector Factory Registry in ORB creates Connector Factory. To get GCIOP Connector Factory, the Connector Factory reference needs to be narrowed to a reference to GCIOP Connector Factory. Connector Factory creates Connector based on the IOR that OA has published. Then, Connector creates Transport. A client can set an ordering type using the OrderingType attribute of GCIOP Connector's TransportInfo. Fig. 12 depicts the overall flow of operations occurring at the server and the client with the proposed scheme [21].

3 Implementation

3.1 Implementation Environments

We implemented the proposed scheme on Microsoft Windows 2000 [14]. Each Windows machine is connected by 10Mbps Ethernet. As the underlying group communication protocol, we exploit the fault-tolerant group communication service (FTGCS) implemented in Java [15] that we proposed for CORBA. FTGCS guarantees reliable communications between group members. We use ORBacus Java 4.01 [10] supporting OCI. The proposed scheme is implemented as a library, i.e., the Java Archive (JAR) file format. The library consumes 213K bytes, including the files generated by GCIOP IDL. The entire system is implemented in JDK 1.3.

3.2 System Architecture

Fig. 13 shows the system architecture. The client part consists of the ordering component setting an ordering type for message delivery and the communication component supporting multicast communication in a group. In the proposed scheme, the ordering component and the communication component are implemented asTransportInfo and Transport, respectively. At the server side, the group membership component allows a server object to compose a group via Acceptor operations. The state transfer component supports the state transfer mechanism and the Group IOR component constructs a group IOR as a group reference. The communication component as part of Transport uses FTGCS, but other group communication protocols can be used. Communication in OCI is performed between Transport objects. GCIOP operations are shown in Fig. 14 and described in detail in the following sections.

3.2.1 Group Membership Component

Group management operations are provided in Acceptor that supports dynamic group membership. Group view and consensus of group members rely on the underlying group communication protocol, that is, FTGCS in our implementation. When Acceptor binds an application object with group members in FTGCS, it needs the information on the application object and the target group. Local variables such as group_name_, host_, and gm_ are stored in Acceptor and used as parameters to Acceptor. An application object assigns a group name. Then, the group name is stored in group_name_ of Acceptor and passed to FTGCS when a group is created or a member joins the group. host_ stores the local machine name and is used for constructing profile information. gm_ indicates a group member in FTGCS, and this value is transmitted to Transport object as a parameter of a Transport constructor. Especially, one group member is

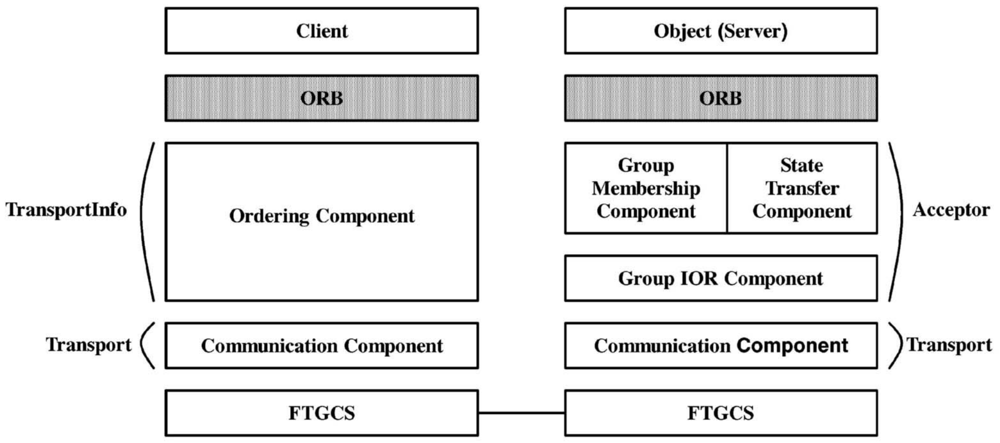


Fig. 13. System architecture.

mapped onto only one Transport and all group members share a group name as a static type.

The methods of Acceptor are implemented using the corresponding methods of FTGCS, i.e., "gm_.Create_group (group_name_),""gm_.Join(group_name_),""gm_.Leave()," set_state(), and get_state().

3.2.2 Ordering Component

ConnectorTransportInfo provides the ordering methods, ordering_type(), and ordering_type(OrderingType value). ConnectorTransportInfo has an ordering variable as an OrderingType that is enumerated as causal and total ordering. An ordering value is obtained by ordering_type(). The ordering_type(OrderingType value) method stores the value parameter to value_. This value determines the message delivery order in a Transport.

3.2.3 Group IOR Component

Acceptor fills up the profile information to construct an IOR at the server side. Fig. 15 shows ProfileBody of a group IOR generated by an IDL compiler. Actually, the add_profile(O CI.ProfileInfo profileInfo, org.omg.IOP.IORHolder ior) method provides the information on ProfileBody.

3.2.4 Communication Component

The send, receive, send_detect, receive_detect, send_time-out, and receive_timeout methods of Transport are implemented using the two methods in FTGCS, i.e., void SendMessage(byte[] data, OrderingType ordering, int atom ic) and McastMsg Receive(). McastMsg is a proprietary data type defined in FTGCS and contains a sequence number, a vector timestamp, etc. A byte array is used in FTGCS when messages are exchanged between a client and members, while a Buffer object is used as an assistant object in the OCI. For this, conversion from a byte array into McastMsg and vice versa is included in Transport. The send and receive methods fill a Buffer object with the received data. The send_detect and receive_detect methods perform the same job as the send and receive methods, respectively, but they return FALSE when an error occurs. The send_timeout and receive_timeout methods allow ORB to set a specific timeout value. The methods of Transport are only called by ORB as internal operations.

3.3 Group Object Key

An IOR includes an object key that allows OA to designate a target object instance. An object key in an IOR representing a group is supposed to designate all the member objects of a group. However, existing BOA and POA models do not allow all object implementations of a group to be represented by a single object key [3]. That is, there is no guarantee that OA for each member object will assign the same object key. Thus, there must be a way to map an object key in a group IOR into an actual object key designating each member object instance [6]. We extend OA such that it substitutes each member's object key with an object key included in a client's request message. We assume that all members know a representative object key for a group, which indicates the object key of a member that creates a group, by means of some extra mechanisms such as a naming service. Every member's ORB creates an object key for its own object implementation. Each member's GCIOP Transport keeps its object key as a local variable. When a member's GCIOP Transport gets a message, it checks whether the message is sent to the group that the member belongs to. If the object key in the message is equal to the representative object key of the group, it replaces the key with a local object key. Then, the operation requested by a client will be invoked.

4 Performance Evaluation

This section describes the experimental results, measuring the time taken for group object invocations using the proposed scheme. We run the experiments using 10 hosts (Pentium III processor with 384MB or 256MB RAM) running Windows 2000 connected by 10Mbit Ethernet. Each machine runs a single server object. A client runs on one of the hosts and makes an invocation to a server object group. We run the experiments 50 times.

Table 1 shows the latency of group invocation using the proposed scheme that includes request/reply time, process member instantiation time, and Connector Factory, Connector, and Transport object creation time at the client side. Invocation time using GCIOP includes the cost consumed by GCS when the proposed scheme makes multiple object invocations. FTGCS is implemented atop UDP/IP multicast and maintains a group view and a delivery queue to support group communication properties such as group

```
final public class Acceptor_impl extends CORBA.LocalObject implements OCI.GCIOP.Acceptor
{
  // Group Membership Component
  //
  // Create an object group.
  public void create_group(String group_name);
  // Add a member object to the group.
  public void join(String group_name);
  // Remove a member object from the group.
   public void leave(String group_name);
  // Transfer the state to the application object.
  public void set_state(org.omg.CORBA.Object state);
  // Return the state of the application object.
  public org.omg.CORBA.Object get_state();

  // Group IOR Component
  //
  // Add a new pofile that matches an Acceptor to IOR.
  public void add_profile(OCI.ProfileInfo profileInfo, org.omg.IOP.IORHolder ior);
}

final public class TransportInfo_impl extends CORBA.LocalObject implements OCI.GCIOP.TransportInfo
{
  // Ordering Component
  //
  // Return the ordering value.
  public OrderingType ordering_type();
  // Set the ordering value that a client requires.
  public void ordering_type(OrderingType value);
}

final public class Transport_impl extends CORBA.LocalObject implements OCI.GCIOP.Transport
{
  // Communication Component
  //
  // Send a buffer's contents through FTGCS.
    public void send(OCI.Buffer buf, boolean block);
    public boolean send_detect(OCI.Buffer buf, boolean block)
    public void send_timeout(OCI.Buffer buf, int t)
    // Receive a buffer's contents through FTGCS.
    public void receive(OCI.Buffer buf, boolean block)
    public boolean receive_detect(OCI.Buffer buf, boolean block);
    public void receive_timeout(OCI.Buffer buf, int t)
}
```

Fig. 14. GCIOP operations.

membership and reliable group communication. Invocation latency in GCIOP consists of latency in the ORB and that in FTGCS. It depends mainly on that of FTGCS. The latency in the ORB is almost constant, about 125.77ms on the average, regardless of the number of server objects.

The invocation latency in the proposed scheme can be divided into three parts—client side ORB, underlying group communication, and server side ORB—and is measured after composing the group. Table 2 shows the overhead consumed by each part. The value for the server side is measured in a member object that creates a group. The client side's processing time consists of client ORB processing time, Connector creation time, and ConnectorTransport creation time. Here, the client ORB processing part is one of most time consuming portions. When a client invokes a method of a server group, the client ORB creates a Connector by calling the ConnectorFactory's create_connectors method and gets information about all the profiles for which the Connector can be used from the given IOR and a list of protocol policies. The client ORB internally executes the Connector's get_usable_profile method to extract the profile information from the IOR and determine if its profile is usable with the protocol policies or not. In the proposed scheme, the protocol policy indicates that GCIOP is the protocol used in the ORB. The time taken in the creation of a Connector and a ConnectorTransport is relatively small and the proposed scheme incurs no extra overhead into them.

The time taken in the underlying group communication is composed of FTGCS client initiation and data transmission. The time varies depending on the underlying group communication protocol. The time taken in the server ORB processing and AcceptorTransport creation is not a major portion. In summary, the most time consuming operations in the proposed scheme are the client side's ORB processing and the client initiation and data exchange in FTGCS as shown in Table 2.

5 Related Work

In this section, we describe the existing approaches supporting group communication service for CORBA.

```
final public class ProfileBody
    implements org.omg.CORBA.portable.IDLEntity
{
  public
  ProfileBody(Version _ob_a0,
          String _ob_a1,
          String _ob_a2,
          short _ob_a3,
          byte[] _ob_a4)
  {
    gciop_version = _ob_a0;
    group_name = _ob_a1;
    host_name = _ob_a2;
    port = _ob_a3;
    object_key = _ob_a4;
  }

  public Version gciop_version;
  public String group_name;
  public String host_name;
  public short port;
  public byte[] object_key;
}
```

Fig. 15. ProfileBody class.

Electra [13] integrates group communication service with CORBA by extending Basic Object Adapter (BOA). The group membership relies on underlying group communication systems. Here, in order to support object group abstraction, group reference is specifically designed. Users can set call semantics such as synchronous, asynchronous, and deferred synchronous calls using the ORB Environment class. The implementation and performance of the system are efficient since there is no intermediate object between ORB and the group communication system. However, this approach requires modification of an ORB for creating group reference and supporting group communication.

Object Group Service (OGS) [4], [5] was proposed as a CORBA object service for group communication. OGS consists of three interfaces: Groupable, GroupAdministrator, and GroupAccessor. Groupable has interfaces for message reception, group composition notification, and state transfer operation, which are used by the member objects. GroupAdministrator has interfaces for group operations, which are join_group and leave_group. GroupAccessor has interfaces related to a multicast operation [4]. In OGS, a group view is managed by the service itself. Message multicast is supported by using a proprietary messaging service. This approach is independent of ORB and guarantees portability with the CORBA Object Service. However, it does not utilize existing group communication protocols and, thus, has a potential drawback in terms of performance.

Mishra et al. [18] also designed CORBA group communication service as a service approach and evaluated the effect of CORBA on the performance of group communication service. For this, they implemented atomic broadcast protocol and group membership protocol in three ways. The first one uses the UDP socket interface as the communication interface between group members that are implemented by a single process. The second one, called the pure CORBA implementation, uses the CORBA instead of UDP socket. The last one also uses the CORBA, but the group member is composed of two separate processes—a client process and a server process. Interprocess communication between client and server processes uses UDP socket and, therefore, this case is called the hybrid CORBA implementation. In terms of performance, the UDP socket-based implementation of a group communication service is most efficient and the pure CORBA implementation is the worst. However, they conclude that the CORBA is suitable for implementing heterogeneous group communication services and the performance degradation can be reduced by controlling the protocol parameters such as buffer sizes and timer values.

Eternal [19] and Eternal interceptor [22] capture and transmit an Internet Inter-ORB Protocol (IIOP) message to the replication mechanism that maps messages onto the group communication system. Especially, join/leave group and send/receive operations are mapped to open/close and write/read system calls, respectively. Here, a unique object group identifier is associated with each interface name. The replication manager maintains a mapping table that contains group identifiers and interface names and is globally accessible [19]. Message multicast is performed using underlying group communication system. This approach need not modify ORB as in the case of using the interceptor. However, it is dependent on OS since the interceptor is implemented in the system call level.

The Multicast Group Internet Inter-ORB Protocol (MGIOP) engine was designed with MGIOP specification in [20]. MGIOP consists of group ID and domain ID. Especially, it does not instantiate GIOP, but encapsulates a GIOP message itself. The MGIOP engine concurrently supports different group communication protocols. It, however, requires modification of ORB because its engine must be included in ORB or connected to it.

Fault Tolerant CORBA (FT CORBA) [25] is composed of replication management, fault management, and logging and recovery management to support entity redundancy, fault detection, and recovery. For group addresses, an Interoperable Object Reference (IOR) is extended such that clients transparently access an object group with a single object reference called Interoperable Object Group Reference (IOGR). Group operations, e.g., create_member, add_member, remove_member, set_primary_member, etc., are provided by ObjectGroupManager of the replication manager. For the IOGR implementation, the FT CORBA specification recommends usage of a proprietary group

TABLE 1
Group Object Invocation Using GCIOP (MSEC)

# of Server objects	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
Invocation latency	258.10	259.70	261.94	259.34	259.98	257.84	259.04	258.18	256.56	258.70
ORB using *group interface*	129.06	126.64	124.08	125.86	128.06	125.68	121.74	126.26	127.18	123.16
FTGCS	129.04	133.06	137.86	133.48	131.92	132.16	137.30	131.92	129.38	135.54

TABLE 2
Analysis of Invocation Latency of the Proposed Scheme

Activity		Percentage
Client Side	*ORB Processing*	22.47%
	Connector Creation	7.86%
	ConnectorTransport Creation	7.94%
Underlying Group Communication	*FTGCS Client Initiation*	24.03%
	Data Exchange	24.54%
Server Side	*ORB Processing*	7.07%
	AcceptorTransport Creation	6.09%

communication protocol as the strategies, i.e., "access via IIOP and gateway" and "access via a proprietary multicast group communication protocol," for connecting a client to a server object group. However, it does not describe how to integrate a group communication protocol into the ORB.

DOORS (Distributed Object-Oriented Reliable Service) [23], Eternal [19], [22], and IRL (Interoperable Replication Logic) [17] implement all or part of the FT CORBA standard. To provide fault tolerance, DOORS proposes a framework that is implemented as a CORBA service and IRL defines Replication Logic that is a set of protocols, mechanisms, and services allowing a CORBA system to handle object replication. FTS [9] consists of portable interceptors and Group Object Adapter (GOA) [8]. It does not follow the FT CORBA standard. The GOA is implemented atop standard POAs and uses the underlying group communication protocol through the Group Communication Interface (GCI).

To apply various protocols into CORBA rather than GIOP/IIOP over TCP/IP, a plug-in framework was proposed, e.g., Open Communications Interface (OCI) [1] and TAO's Pluggable Protocol Framework (PPF) [12]. The OCI provides open interfaces that can substitute TCP/IP in CORBA. It implements the Acceptor/Connector module [26] in ORB. The module distinguishes connection establishment from service initialization occurring in the communication of a client/server model. The OCI consists of an acceptor (at the server side) and a connector (at the client side) for connection establishment, and a transport as a communication component. TAO's PPF is based on the OCI and, therefore, the basic mechanism of that is the same as the OCI. But, it enhances the OCI for high-performance and real-time systems.

Halteren et al. [27] applied IP Multicast to CORBA by extending the OCI. In order to meet the requirements on the transport layer such as connection-oriented, reliable data transport, transported data as a stream, and notification of connection loss in the CORBA specification [24], this approach adds the schemes for acknowledgement and retransmission of packets to the intermediate protocol. It uses the modified Interoperable Object Reference (IOR) that includes an IP multicast address (D class) and a sequence number. However, the approach does not provide a generic framework that can accommodate various group communication protocols in CORBA. Moreover, dynamic group membership is not supported because Java IP Multicast API is directly used as group operations.

Pluggable FT CORBA Infrastructure [28] is a FT CORBA compliant implementation using a TAO's PPF. To apply the group communication protocol into the ORB, the FT adaptor of server-side FT protocol plug-in is proposed. The FT adaptor manages the connection and exchanging messages to the underlying group communication protocol. In viewpoint of extending the plug-in framework, this approach is similar to the proposed scheme in this paper. However, the pluggable FT CORBA infrastructure focuses on the FT CORBA compliant framework, whereas the proposed scheme provides extended interfaces for group communication service to both FT CORBA and non-FT CORBA.

6 Conclusion

Group communication is one of the key components supporting object replication. Here, we have extended OCI, the proposed CORBA standard for intelligent network systems [1], in order to provide a generic group communication framework for CORBA. We first define GCIOP that has end-point information such as group name. Then, we design the group communication Info Object and the OCI to use group semantics. The proposed scheme consists of a group membership component, a group IOR component, and a group multicast component. The group membership component provides operations for dynamic group membership and guarantees the consistency of a group view through Acceptor's operations. To identify a group in CORBA, the group IOR component constructs a group IOR that is filled up based on the GCIOP information in an Acceptor. The group multicast component provides multicast communication within a group via each Transport object under a client and server objects. All group semantics are exposed to CORBA application objects by mapping group operations in the proposed scheme into the corresponding operations in the underlying group communication protocol. The proposed scheme can be used to support FT-CORBA compliant applications. In that case, the IOGR in FT CORBA is used instead of the GCIOP-based group IOR. To create an object group, the replication manager of FT CORBA invokes the create_object() method of a local factory in server ORBs. Server replicas created by the factory join the group through the group membership procedure of the proposed scheme. When a client invokes the object group atop FT CORBA regardless of a gateway, the group object invocation procedure is the same as that in the proposed scheme. The experiment results show that the proposed scheme does not incur performance degradation, even though the number of server objects increases.

We assume that the group communication model is one-to-many at present. Here, a sender is located atop a Connector with multiple receivers atop an Acceptor. If application objects are constructed as a reactive client/server playing a role of both a client and a server, the proposed design can also be applied to a peer model. Furthermore, it supports not only group communication in CORBA, but also existing group communication without any modification of ORB and dependency on the OS. We plan to apply other group communication protocols to the ORB using the proposed approach and evaluate the performance comparing with existing group communication systems in CORBA. We are currently exploring a mechanism for transparent manipulation of a group object key [11], since the existing BOA and POA models do not allow member object implementations for a group to be represented by a single object key. We also plan to design a gateway for FT CORBA so that FT CORBA compliant objects can be supported.

Acknowledgments

This work was supported in part by the National Research Laboratory Program funded by the Ministry of Science and Technology, the Republic of Korea.

References

[1] AT&T, Internetworking between CORBA and Intelligent Network Systems, OMG Document telecom/98-06-03, 1998.

[2] K.P. Birman, "The Process Group Approach to Reliable Distributed Computing," *Comm. ACM,* vol. 36, no. 12, pp. 37-53, Dec. 1993.

[3] Eternal Systems, Inc., Unreliable Multicast Inter-ORB Protocol: Initial Submission, OMG Document orbos/2000-02-03, 2000.

[4] P. Felber, B. Garbinato, and R. Guerraoui, "The Design of a CORBA Group Communication Service," *Proc. 15th Symp. Reliable Distributed Systems,* pp. 150-159, Oct. 1996.

[5] P. Felber and R. Guerraoui, "Programming with Object Groups in CORBA," *IEEE Concurrency,* vol. 8, no. 1, pp. 48-58, Jan./Mar. 2000.

[6] R. Guerraoui, P. Eugster, P. Felber, B. Garbinato, and K. Mazouni, "Experiences with Object Group Systems," *Software-Practice and Experience,* vol. 30, no. 12, pp. 1375-1404, Oct. 2000.

[7] R. Guerraoui and A. Schiper, "Software-Based Replication for Fault Tolerance," *Computer,* vol. 30, no. 4, pp. 68-74, Sept. 1999.

[8] R. Friedman and E. Hadad, "A Group Object Adaptor-Based Approach to CORBA Fault-Tolerance," *IEEE Distributed Systems Online,* vol. 2, no. 7, http://dsonline.computer.org/0107/features/fri0107.htm, Nov. 2001.

[9] R. Friedman and E. Hadad, "FTS: A High-Performance CORBA Fault-Tolerance Service," *Proc. Seventh Int'l Workshop Object-Oriented Real-Time Dependable Systems,* pp. 61-68, Jan. 2002.

[10] IONA's ORBacus page, http://www.iona.com/products/orbacus_home.htm, 2003.

[11] Y. Joe, D. Lee, and D. Nam, "A Transparent Object Group Reference Management Scheme for Group Communication in CORBA," *Proc. Confederated Int'l Confs. CoopIS, DOA, and ODBASE,* pp. 25-28, Oct. 2002.

[12] F. Kuhns, C. O'Ryan, D.C. Schmidt, O. Othman, and J. Parsons, "The Design and Performance of a Pluggable Protocols Framework for Object Request Broker Middleware," *Proc. IFIP Sixth Int'l Workshop Protocols for High-Speed Networks,* pp. 81-98, Aug. 1999.

[13] S. Landis and S. Maffeis, "Building Reliable Distributed Systems with CORBA," *Theory and Practice of Object Systems,* vol. 3, no. 1, pp. 31-43, 1997.

[14] D. Lee, D. Nam, H.Y. Youn, and C. Yu, "The Implementation and Analysis of OCI-Based Group Communication Support in CORBA," *Proc. Pacific Rim Int'l Symp. Dependable Computing,* pp. 281-288, Dec. 2001.

[15] D. Lee et al., "Development of Reliable Group Communication Module for Object Group," ICU CDS&N Laboratory, Daejeon, Korea, Technical Report CDSN-1999-TR005, http://cds.icu.ac.kr/, June 1999.

[16] S. Maffeis, "The Object Group Design Pattern," *Proc. USENIX Conf. Object-Oriented Technologies,* June 1996.

[17] C. Marchetti, M. Mecella, A. Virgillito, and R. Baldoni, "An Interoperable Replication Logic for CORBA Systems," *Proc. Int'l Symp. Distributed Objects and Applications,* pp. 7-16, Sept. 2000.

[18] S. Mishra, L. Fei, X. Lin, and G. Xing, "On Group Communication Support in CORBA," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 2, pp. 193-208, Feb. 2001.

[19] L.E. Moser, P.M. Melliar-Smith, and P. Narasimhan, "Consistent Object Replication in the Eternal System," *Theory and Practice of Object Systems,* vol. 4, no. 2, pp. 81-92, 1998.

[20] L.E. Moser, P.M. Melliar-Smith, P. Narasimhan, R.R. Koch, and K. Berket, "Multicast Group Communication for CORBA," *Proc. Int'l Symp. Distributed Objects and Applications,* pp. 98-107, Sept. 1999.

[21] D. Nam, D. Lee, H.Y. Youn, and C. Yu, "Group Communication Support for CORBA Using OCI," *Proc. 12th IASTED Int'l Conf. Parallel and Distributed Computing and Systems,* pp. 106-111, Nov. 2000.

[22] P. Narasimhan, L.E. Moser, and P.M. Melliar-Smith, "Using Interceptors to Enhance CORBA," *Computer,* vol. 32, no. 7, pp. 62-68, July 1999.

[23] B. Natarajan, A. Gokhale, S. Yajnik, and D.C. Schmidt, "DOORS: Towards High-Performance Fault-Tolerant CORBA," *Proc. Int'l Symp. Distributed Objects and Applications,* pp. 39-48, Sept. 2000.

[24] Object Management Group, The Common Object Request Broker: Architecture and Specification, Rev. 2.3, OMG Document formal/98-12-01, June 1999.

[25] Object Management Group, Fault Tolerant CORBA, OMG Document formal/2001-09-29, Sept. 2001.

[26] D.C. Schmidt, "Acceptor-Connector: An Object Creational Pattern for Connecting and Initializing Communication Services," *Pattern Languages of Program Design 3,* R. Martin, F. Buschman, and D. Riehle, eds., Addison-Wesley, 1997.

[27] A.T. van Halteren, A. Noutash, L.J.M. Nieuwenhuis, and M. Wegdam, "Extending CORBA with Specialised Protocols for QoS Provisioning," *Proc. Int'l Symp. Distributed Objects and Applications,* pp. 318-327, Sept. 1999.

[28] W. Zhao, L.E. Moser, and P.M. Melliar-Smith, "Design and Implementation of a Pluggable Fault-Tolerant CORBA Infrastructure," *Proc. Int'l Parallel and Distributed Processing Symp.,* pp. 343-352, Apr. 2002.