OCI를 이용한 CORBA에서의 그룹 통신 지원 방법
(Method for Group Communication Support in CORBA using OCI)

남 덕 윤 † 이 동 만 ††
(Dukyun Nam) (Dongman Lee)

요 약 그룹 통신은 객체 복제를 지원하는 주요 기술 중 하나이다. 현재의 CORBA 표준은 객체 복제를 이용하여 지원할 수 있는 고장 감내와 고 가용성을 지원하지 못한다. 지금까지의 CORBA 기반 그룹 통신에 관한 연구들에서는 CORBA 애플리케이션 프로그래머가 그룹 통신 프로토콜을 직접 이용할 수 있는 방법이 없었다. 또한 CORBA 또는 OS의 수정이 요구되거나, 기존의 그룹 통신 프로토콜을 적용할 수 없었다. 본 연구에서는 표준 CORBA의 수정 없이 다양한 그룹 통신 프로토콜을 적용할 수 있는 일반적인 그룹 통신 프레임웍을 제안한다. 이를 위해 우리는 상호 운용성, 기존 그룹 통신 프로토콜의 재사용을 지원하고, ORB와 OS에 대한 독립성을 유지하며, 유연성 있는 하부 프로토콜 적용을 가능하게 하도록 하기 위해 OCI를 확장하였다. 또한 제안한 방법에는 기존의 다양한 그룹 통신 프로토콜이 적용될 수 있다.

키워드 : 오픈 커뮤니케이션 인터페이스, 그룹 통신, CORBA, 멀티캐스팅

Abstract Group communication is one of key components supporting object replication. CORBA provides little support for fault tolerance and high availability that can be supported by means of object replication. The existing approaches do not allow transparent plug-in of group communication protocols into CORBA with which CORBA application programmers are able to directly exploit group communication protocols. They either require modification of CORBA or OS, or provide no room for incorporating group communication transport protocols into CORBA. In this paper, we propose a generic group communication framework that allows transparent plug-in of various group communication protocols with no modification of standard CORBA. For this, we extend the Open Communications Interface(OCI) to support interoperability, reusability of existing group communication, and independency on ORB and OS. The proposed approach can also be applied to various group communication protocols.

Key words : Open Communications Interface, Group Communication, CORBA, Multicasting

1. 서 론

고장 감내와 고 가용성은 객체 복제 기법을 이용하여 지원될 수 있다[1]. 복제된 객체들은 일반적인 작업에 대해 객체 그룹을 형성하여 작업을 수행한다[2]. 한편 고장 감내를 지원하는 분산 애플리케이션을 만들기 위해서는 객체 그룹 멤버들의 상태가 동일하게 유지되어야 한다[3]. 그룹 통신 서비스(Group Communication Service; GCS)는 모든 멤버 객체들 사이의 일관성을 보장하는 데에 있어 유용한 기술로 객체 그룹 내에서 현재 연결되어 있는 멤버의 리스트를 뷰(view)로써 관리하며, 뷰 변경이 있을 때마다 최신의 뷰를 유지한다. 이에 뷰 내에 있는 멤버들 사이에서는 신뢰적 메시지 전송이 보장된다.

일반적으로 CORBA에서의 GCS 지원 방법들로는 통합 방식, 서비스 방식, 가로채기 방식이 있다[4]. 대표적 예로는 통합 방식의 Electra[5], 서비스 방식의 OGS (Object Group Service)[6, 8], 가로채기 방식의 Eternal[7]을 들 수 있다. 통합 방식에서 GCS 모듈이 ORB(Object Request Broker)에 통합되기 때문에, GCS를 지원하기 위해서는 ORB를 수정해야 한다. 서비스 방식은 CORBA의 점대점 통신인 GIOP(General Inter-ORB Protocol)를 활용하여 ORB 위에서 객체 서

· 본 연구는 과학기술부 국가지정연구실사업을 통해 수행된 것입니다.
† 학생회원 : 한국정보통신대학원대학교 공학부
paichu@icu.ac.kr
†† 정 회 원 : 한국정보통신대학원대학교 공학부 교수
dlee@icu.ac.kr
논문접수 : 2001년 8월 29일
심사완료 : 2002년 5월 9일

비스로써 그룹 통신을 지원하지만, 기존에 연구되어 있는 그룹 통신 프로토콜을 활용할 수 없다. 가로채기 방식은 인터셉터(interceptor)[9]를 이용하여 그룹 통신과 관련된 시스템 콜(system call)을 가로챔으로써 그룹 통신 서비스를 지원하는데, 인터셉터가 시스템 콜의 인터페이스와 그룹 통신 서비스의 인터페이스를 대응시킨다는 점에서 OS에 의존적이다. 또한 CORBA와 그룹 통신 서비스를 연결하기 위한 인터셉터가 CORBA의 일부가 아니기 때문에, CORBA 객체가 하부 그룹 통신 서비스를 바로 활용 할 수 없다. 결론적으로 기존의 방식들은 CORBA에 그룹 통신 프로토콜을 지원하기 위한 인터페이스를 제공하지 않으며, 애플리케이션 프로그래머들은 프로토콜을 바로 활용할 수 없었다.

CORBA는 이종의 분산 환경에서 분산된 객체들 간의 통신을 지원하기 위한 이식성(portability) 및 상호 운용성(interoperability)과 같은 특성들을 지원한다. 이런 측면에서 CORBA에 특정 그룹 통신 프로토콜만을 적용하는 것은 CORBA에서의 모든 그룹 애플리케이션들을 만족시킬 수 없다. 이를 해결하기 위해서는 표준 CORBA 인터페이스로 다양한 그룹 통신 프로토콜을 적용시킬 수 있는 일반적인 그룹 통신 프레임웍이 있어야 한다. 본 논문에서는 상호 운용성, 기존의 다양한 그룹 통신 프로토콜에 대한 재사용성, ORB와 OS에 대한 독립성을 유지하고, 유연하게 하부 프로토콜을 적용할 수 있는 OCI(Open Communications Interface)[10]를 활용하여, 기존의 CORBA를 수정하지 않고 그룹 통신 프로토콜을 적용할 수 있는 프레임웍을 제안한다. 우리는 기존 OCI에 객체 그룹을 관리하고, 그룹으로의 메쏘드 호출(method invocation)을 지원하기 위해 새로운 함수들을 추가하였다. GIOP는 IPX나 TCP/IP와 같은 네트워크 프로토콜에 독립적인 프로토콜이다. 이에 그룹 통신 프로토콜을 GIOP와 연결하기 위해, GIOP의 그룹 통신 인스턴스인 GCIOP(Group Communication Inter-ORB Protocol)을 정의하였다. 구체적으로 그룹 통신 Info 객체를 OCI에 추가하고, 그룹 함수 지원을 위해 OCI의 인터페이스를 확장하였다. 기존 OCI는 Connector, Connector Factory, Acceptor, Transport 모듈로 구성된다[10, 11]. 그룹 통신에서는 기본적으로 그룹 주소(group addressing), 메시지 전송 순서화(message delivery ordering), 상태 전송(state transfer)이 지원되어야 한다[12]. 고장 감내를 제외한 이러한 요구 사항들을 만족시키기 위해, 그룹 이름, 순서화 종류, 상태 정보를 Info 객체에 저장한다. 그리고 순서화 종류를 조정하기 위해 Connector의 Transport Info 객체에 순서화 종류를 나타내는 애트리뷰트(attribute)를 추가했으며, 그룹 함수 지원을 위해 Acceptor 객체에 그룹 함수 인터페이스를 추가하였다. 이에 본 논문에서 제안하는 프레임웍은 ORB와 OCI의 수정 없이 다양한 그룹 통신 프로토콜을 적용할 수 있다.

본 논문의 구성은 다음과 같다. 2장은 CORBA에 그룹 통신이 적용된 기존 연구들을 분석하며, 3장은 그룹 통신을 위해 확장된 OCI의 디자인과 각 구성요소 별 내부 프로시저를 기술한다. 4장에서는 확장된 OCI의 구성요소별 구현을 설명하고, 5장에서는 확장된 OCI의 성능 평가를 위해, 확장 OCI를 활용한 그룹 객체 실행과 표준 CORBA의 일대일 객체 실행을 이용한 다중 객체 실행을 비교한다. 마지막으로 6장에서 결론을 맺는다.

2. 관련연구

이 장에서는 CORBA에서 그룹 통신을 지원하는 방법에 관한 연구들을 살펴보고자 한다. Electra[5]는 CORBA의 BOA(Basic Object Adapter)를 확장함으로써 그룹 통신 서비스를 지원하고자 하였다. 그룹 멤버쉽은 하부 그룹 통신 시스템에 의존하며, 객체 그룹 추상화(abstraction)을 위해 그룹 참조(group reference)를 새롭게 디자인하였다. 사용자들은 ORB의 Environment 클래스를 사용하여 동기 호출(synchronous call), 비동기 호출(asynchronous call), 그리고 지연된 동기 호출(deferred synchronous call)과 같은 호출 방법을 조정할 수 있다. 이 시스템은 ORB와 그룹 통신 시스템 사이에 중계 객체 없이 바로 연결이 되어 있으므로, 구현과 성능 측면에서 효율적이라 할 수 있다. 그러나 그룹 참조(group reference)와 그룹 관리(group management)를 지원하기 위해 표준 CORBA에서 정의한 ORB를 수정해야 한다는 단점이 있다.

OGS[6, 8]는 그룹 통신을 위한 새로운 CORBA 객체 서비스로 디자인되었다. OGS는 Groupable, Group Administrator, GroupAccessor로 구성된다. Groupable은 멤버 객체가 사용하는 메시지 수신, 그룹 구성 통보, 상태 전송을 위한 인터페이스를 가지고 있으며, Group Administrator는 join_group, leave_group과 같은 그룹 가입/탈퇴 관련 인터페이스를 가지고 있다. GroupAccessor는 멀티캐스트 전송을 위한 인터페이스를 가지고 있다. 그룹 뷰는 객체 서비스에서 지원하며, 메시지 멀티캐스트 또한 서비스에서 지원한다. 이 방식은 ORB에 대해 독립적이며, CORBA 객체 서비스로서 이식성을 보장하지만, 기존의 그룹통신 시스템을 이용할 수 없으며, 그룹 통신 프로토콜을 CORBA 서비스로 지원함으로써 성

능 측면에서 비효율적이라는 단점을 가지고 있다.

Eternal[7]과 Eternal 인터셉터[3]는 그룹 통신 시스템의 메시지와 관련된 IIOP 메시지를 가로채서 복제 메니저에게 전송한다. 특히 join/leave 그룹, send/receive 함수는 open/close와 write/read 시스템 콜에 매핑된다. 객체 그룹 식별자(object group identifier)는 각 인터페이스 이름으로 구성되며, 그룹 식별자와 인터페이스 이름을 포함한 매핑 테이블은 복제 메니저에 의해 관리된다[9]. 메시지 멀티캐스트는 하부 그룹 통신 시스템에 의해 지원된다. 이 방식은 인터셉터[3]를 이용하기 때문에 ORB 수정이 필요 없으나, 인터셉터가 시스템 콜 수준에서 구현되어 있기 때문에 OS에 종속된다는 단점이 있다.

CORBA에 TCP/IP 외에 멀티캐스트 프로토콜을 적용한 Halteren의 연구[13]에서는 OCI를 이용해서 IP 멀티캐스트를 CORBA에 적용하였다. OCI는 CORBA에서 TCP/IP를 대체할 수 있는 인터페이스를 제공하며, ORB 내에 클라이언트/서버 모델의 통신에서 연결 초기화 부분과 통신 부분을 나누어 놓은 Acceptor/Connector 모듈을 구현한 것이다[14]. CORBA는 전송 수준(transport layer)에 대해 연결 지향(connection oriented), 신뢰적 데이타 전송(reliable data transport), 스트림으로 전송되는 데이타(transported data as a stream), 연결 손실에 대한 통지(notification of Connection loss)를 만족해야 한다고 정하고 있다[15]. 이를 만족하기 위해 이 연구에서는 IP 멀티캐스트를 CORBA에 적용할 때, 긍정 응답(acknowledgement)과 소실된 패킷에 대한 재전송(retransmission)기능을 추가하였다. 그리고 IOR(Interoperable Object Reference)에 IP 멀티캐스트 주소(D class)와 시퀀스 번호를 추가하였다. 또한 이 연구는 Java IP 멀티캐스트 API를 그룹 함수로써 직접 이용하기 때문에 동적 그룹 멤버쉽(Dynamic group membership)을 지원하지 않으며, 기존에 연구되어 있는 다양한 그룹 통신 프로토콜을 CORBA에 적용할 수 없다는 단점이 있다.

마지막으로 CORBA에서 그룹통신을 지원할 수 있는 MGIOP(Multicast Group Internet Inter-ORB Protocol) 엔진이 MGIOP 명세서[16]에 제안되었다. 여기서 MGIOP는 그룹 ID와 도메인 ID를 포함하며, 특히 GIOP(General Inter-ORB Protocol) 메시지 자체를 포함하고 있다. MGIOP 엔진은 서로 다른 그룹 통신 프로토콜을 동시에 지원할 수 있다. 그러나 이 엔진은 ORB에 포함되거나 ORB와 연결이 되어야 하므로, 표준 ORB를 수정해야 한다는 단점이 있다.

3. CORBA에서의 그룹 통신을 위한 제안

이 장에서는 먼저 OCI의 개요와 OCI에 전송 프로토콜이 어떻게 적용되는지를 설명한다. 다음으로 IDL (Interface Definition Language)과 함께 그룹 통신을 적용하기 위해 OCI를 어떻게 확장했는지 기술한다.

3.1 Open Communications Interface

OCI는 CORBA에서 하부 프로토콜을 교체할 수 있는 인터페이스를 제공한다. 인터페이스들은 Buffer, Acceptor, Transport, Connector, Connector Factory, Registries, Info 객체 등이다[10]. Buffer는 데이타를 가지고 있으며, 클라이언트와 서버 사이의 통신에 사용하는 위치 카운터를 관리한다. Info 객체들은 Acceptor, Transport, Connector, Connector Factory에 대한 정보들을 가지고 있으며, 각 객체마다 자신의 Info 객체를 가지고 있다. 그림 1은 OCI가 ORB와 전송 프로토콜을 어떻게 연결하는지 보여준다. 클라이언트와 서버가 활성화 될 때, 객체 어댑터(Object Adapter; OA)내의 Acceptor Registry는 Acceptor를 생성하고, ORB core 안에 존재하는 Connector Factory Registry가 Connector Factory를 생성한다. 서버 측에서는 Acceptor가 IOR에 들어갈 프로파일 정보를 만들고, OA는 IOR을 생성한다. 한편 클라이언트 측에서는 Connector Factory가 IOR의 내용에 따라 Connector를 생성한다. 이 후, Connector와 Acceptor는 각각의 Transport 객체를 초기화하여 통신을 맺는다. 서버와 클라이언트 측의 Transport 객체 사이에 연결이 이루어지면, Transport는 Buffer 객체를 이용하여 메시지를 교환할 수 있다.

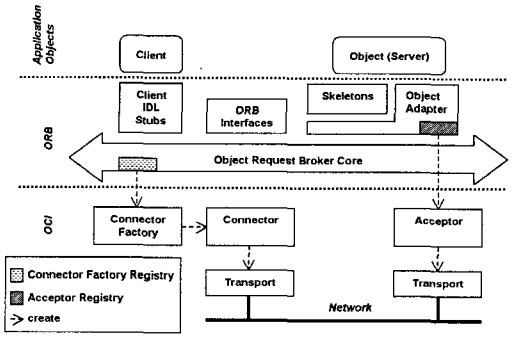


그림 1 표준 CORBA에서의 OCI 구조

클라이언트 측에는 서버 역할을 하는 로컬 프럭시 객체가 존재한다. 클라이언트가 로컬 프럭시 객체에게 함수 호출을 하면, 클라이언트 측의 ORB는 함수 호출을

마샬링(marshalling)한 후 서버에게 전송한다. 구체적으로 설명하면, ORB는 OCI의 Transport 객체가 가지고 있는 'send' 함수를 호출하고, 'send' 함수 내에서는 Buffer 객체에 그 내용을 담아 실제로 전송하게 된다. 서버 측에서는 Transport 객체가 받은 내용을 서버 쪽의 ORB에게 전달하면, ORB에서 그 내용을 언마샬링(unmarshalling)한 후, 실제 서버 객체를 호출하게 된다. OCI의 Info 객체까지 포함한 모든 객체들의 구현은 전송 프로토콜마다 다르다. OCI에서 Connector Factory와 Acceptor 객체만이 애플리케이션 객체에 드러난다.

3.2 그룹 통신을 위한 OCI 확장

CORBA 객체에 GCS를 적용하기 위해서는 그룹 주소(group address), 그룹 멤버쉽(group membership), 순서화(ordering)의 3가지 측면이 고려되어야 한다. 본 연구에서는 하부 그룹 통신 프로토콜이 메시지에 대해 신뢰성 있는 전송을 보장한다고 가정한다. 그룹 주소는 클라이언트가 그룹을 하나의 객체와 같이 인식하게 하도록 투명성을 보장한다. 여기서 제안하는 OCI 확장은 그룹 통신 프로토콜에서 제공하는 그룹 구별자를 사용하여 그룹 객체 레퍼런스를 만든다. 그룹 객체 레퍼런스는 CORBA 객체의 주소 값을 표현하는 IOR로 표현되며, OCI의 Transport에서 그룹 구별자를 확장한다. 클라이언트는 IOR에 있는 정보를 참조하여 그룹으로 통신을 위한 연결을 할 수 있다. GIOP가 그룹 주소와 같은 종단(end-point)의 정보를 표현할 수 없는 추상적인 프로토콜이기 때문에, 통신을 할 때에는 GIOP에 호환되어 실제 통신에 사용되는 전송 프로토콜에 맞게 표현할 수 있는 구현 프로토콜이 필요하다. 예를 들어 TCP/IP를 전송 프로토콜로 사용하는 경우, IIOP(Internet Inter-ORB Protocol)를 구현해서 사용한다. 이에 본 연구에서는 그룹 통신을 지원하기 위해 GIOP에 대한 구현 프로토콜인 GCIOP(Group Communication Inter-ORB Protocol)을 정의하였다.

그룹 멤버쉽에서는 그룹 내의 멤버들 사이의 일관성 유지를 위해 멤버 뷰를 관리한다. 이에 확장한 인터페이스에서는 그룹 초기화, 소멸, 가입, 탈퇴 등의 그룹 함수를 제공한다. 실제로 이 함수들은 OCI의 Acceptor에 추가된다. 상태 전송(state transfer) 인터페이스는 Acceptor를 통해 제공되며, 상태의 정보는 CORBA 객체 타입으로 Acceptor Info 객체에 저장된다.

순서화는 하나 이상의 클라이언트들이 그룹과 통신할 때 멤버들 사이의 일관성을 유지하기 위해 필요한 요소이다. 클라이언트는 Transport Info에 있는 순서화 타입의 값을 변경함으로써 순서화 타입을 조정할 수 있다. 여기서 기본 순서화 값은 Total 순서화로 한다. 그림 2는 CORBA에서의 하부 그룹 통신 프로토콜과 OCI 구조를 보여준다.

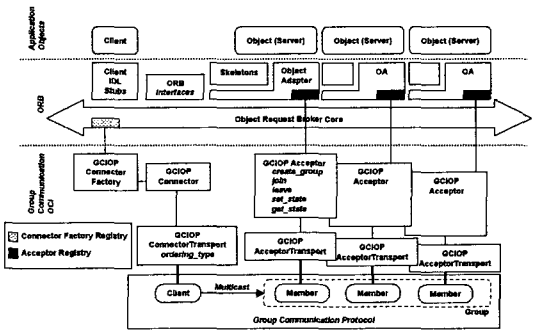


그림 2 그룹 통신 프로토콜과 확장 OCI 구조

3.2.1 그룹 IOR

기존의 CORBA에서 서버 객체가 ORB에서 제공하는 IOR에 의해 표현되는 것처럼, CORBA 그룹 통신에서 그룹을 식별하기 위해 그룹 통신 OCI는 그룹 IOR을 제공한다. 서버 측의 객체가 활성화 되었을 때, CORBA는 IP 주소와 포트 번호와 같은 전송 정보를 포함한 IOR을 만든다. 이와 같이 그룹 IOR은 그룹 이름과 각 멤버의 호스트 이름을 포함한다. IOR은 서버측에서 초기 실행 시 만들어지며, 클라이언트 애플리케이션 객체에 의해 분석된다. IOR은 Type ID, Profile Count, Tagged Profile로 구성된다. Type ID는 객체의 IDL 기반으로 저장소 ID 형식(repository ID format)에서 IOR의 인터페이스 형식을 제공한다. Profile Count는 Tagged Profile의 수를 의미한다. Tagged Profile은 IOR에 하나 이상 존재할 수 있으며, 통신에 이용되는 프로토콜의 정보를 포함하고 있다.

Tag	Group Name	Host Name	Object Key

그림 3 GCIOP에서의 Tagged Profile 형식

그림 3은 GCIOP에서 사용되는 Tagged Profile 형식이다. Tag는 고정 값으로서 객체 간의 통신을 위해 사용되는 프로토콜을 나타낸다. '01'은 이미 IIOP에서 쓰이고 있으므로, 그룹 통신을 위해서는 이 외의 다른 값을 사용한다. 그룹 이름(Group Name)은 ORB는 물론, 하부 그룹 통신 프로토콜에서도 그룹 구분자로 사용된다. 호스트 이름(Host Name)은 멤버 자체의 호스트 이

름을 나타내며, 객체 키(Object Key)는 특정 객체의 인스턴스(instance)를 나타낸다. 실제로 서버 측의 Acceptor는 IOR을 만들기 위한 프로파일 정보를 입력한다. 그룹 IOR의 ProfileBody는 IDL로부터 생성된 것이며, ProfileBody의 정보는 add_profile이라는 함수에 의해 GCIOP의 내용을 바탕으로 채워진다.

3.2.2 OCI 정보구조

그룹 통신 OCI 부분의 모든 구성 요소들인 Connector, Connector Factory, Acceptor, Transport는 각각의 정보 객체를 가지고 있다. 모든 Info 클래스들은 기존 OCI의 Info 클래스로부터 상속을 받아 확장된 것으로, OCI에서 다양한 프로토콜을 접합시킬 수 있도록 제공하는 특성에 의한 것이다. 그림 4는 그룹 통신을 위한 OCI Info 클래스의 IDL 명세서(specification)이다. ConnectorInfo는 그룹 이름을 저장하고 있으며, Connector는 메시지가 그룹에 보낼 때 ConnectorInfo를 사용한다. ConnectorInfo는 클라이언트 측의 OCI로 관리한다.

서버 측의 OCI는 그룹 이름과 호스트 이름을 가지고 있는 AcceptorInfo를 관리한다. 그룹 이름은 서버가 그룹을 만들거나 가입할 때 사용하는 것이며, 호스트 이름은 서버 자체의 호스트 이름을 나타낸다. 그룹 및 호스트 이름은 그림 3의 GCIOP 프로파일 정보를 만들 때 사용된다. 상태(state)는 그룹 멤버의 상태를 나타내며, 새로운 멤버가 그룹에 가입했을 때 상태 전송(state transfer)을 위해 사용된다. TransportInfo는 Connector와 Acceptor가 자신의 Transport 객체를 생성했을 때 클라이언트와 서버 양쪽에서 사용된다. Connector와 Acceptor는 TransportInfo에 같은 그룹 이름을 가지고 있다. 클라이언트 객체는 Connector의 TransportInfo에 있는 ordering_type의 값을 변경함으로써, 메시지 전송에 적용되는 순서화 타입을 변경할 수 있다.

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

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 Object state;
};

interface ConnectorTransportInfo : OCI::TransportInfo
{
    readonly attribute string group_name;
    readonly attribute string host_name;
    attribute OrderingType ordering_type;
};

interface AcceptorTransportInfo : OCI::TransportInfo
{
    readonly attribute string group_name;
    readonly attribute string host_name;
};

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

그림 4 정보 객체의 IDL 명세서

3.2.3 그룹 통신 서비스를 위한 인터페이스

그림 5는 그룹 통신을 위한 함수가 추가 된 OCI의 IDL 명세서이다. 그룹 통신 OCI의 Acceptor 인터페이스는 애플리케이션 객체들만이 이용할 수 있다. 기존 OCI의 모든 인터페이스는 변경 없이 그대로 사용되며, 그룹 멤버쉽과 상태 전송을 위한 create_group, join, leave, set_state, get_state 함수들만이 Acceptor에 추가되었다. 그룹 생성 후, 첫번 째 서버는 프라이머리 멤버로 그룹 주소 값을 제공한다. 이에 다른 멤버들은 기존의 그룹에 가입 또는 탈퇴할 수 있으며, 이들이 그룹에 가입했을 때 상태 전송이 실행된다.

```
module OCI
{
module GCIOP
{
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
```

그림 5 그룹 통신 OCI의 IDL 명세서

3.3 그룹 멤버쉽 모델

그룹 멤버쉽 모델은 동적인 그룹 멤버쉽을 제공하고, 그룹 가입/탈퇴 함수, 뷰 변경 메커니즘, 상태 전송 등을 이용하여, 그룹 뷰의 일관성을 보장한다. 그룹 멤버 객체들은 서버 측에서 동작하는데, 서버 측 OCI가 Acceptor 객체이므로, Acceptor에서 그룹 구성을 위한 함수들을 제공한다. 이 절에서는 그룹을 만들기 위해 멤버 객체들과 그룹 통신 시스템 사이에 어떤 메시지들이 교환되는지를 살펴본다. 다음으로 새로운 멤버가 가입을 요청했을 때, 상태 전송과 뷰 변경 메커니즘의 과정을 기술한다.

새로운 그룹을 만들고 처음 가입하는 멤버를 프라이머리 멤버라 가정한다. 각각의 멤버들은 뷰 변경 통지

메시지(view-change notification message)를 받았을 때, 자신의 뷰 정보를 프라이머리 멤버에게 보고한다. 이 후 모든 멤버들은 그룹 뷰의 일관성을 담당하는 프라이머리 멤버로부터 뷰 인스톨 메시지(view-install message)를 기다린다. 한편 프라이머리 멤버는 모든 멤버들로부터 동일한 뷰 정보를 받았을 때, 뷰 설치 메시지를 멀티캐스트 한다. 본 논문에서 제안한 메커니즘에서 프라이머리 멤버 객체는 그룹 통신 시스템의 함수와 일치되는 Acceptor의 함수들을 사용할 수 있다. 뷰 변경 메커니즘은 뷰 설치 메시지를 받음으로써 완료된다. 이 과정을 통해서 Acceptor는 하부 그룹 통신 시스템에 있는 그룹 멤버의 주소값을 가지고 있으며, 서버 객체들은 전송 프로토콜에 연결되어 있다.

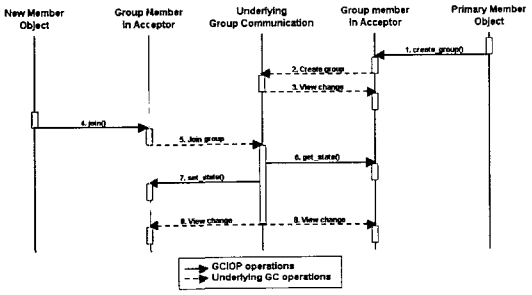


그림 6 그룹 멤버쉽 모델에서의 상호작용

그림 6은 그룹 구성을 위해 호출되는 함수들의 상호작용 도표이다. 현재 만들고자 하는 그룹은 존재하지 않는다고 가정하자. 프라이머리 멤버 객체는 Acceptor의 create_group()을 호출한다(1). create_group() 함수는 내부적으로 그룹 통신 시스템의 동일한 함수를 호출하고, 그룹에 가입한다(2). 하부 그룹 통신 시스템이 뷰 인스톨 메시지를 받았을 때, 그룹 멤버의 주소값은 Acceptor에게 전달되고(3), 그룹 생성 과정은 끝난다.

새로운 멤버가 그룹에 가입 할 때, 상태 전송 메커니즘이 요구된다. 새로운 멤버 객체가 Acceptor의 join()을 호출하면(4), 하부 그룹 통신 시스템에도 멤버로써 가입하게 된다(5). 상태 전송이 뷰 인스톨 메시지를 보내기 전에 실행되는 것은 멤버의 상태(state)가 애플리케이션에 의존적인 데이타이기 때문이다. 하부 그룹 통신 시스템은 프라이머리 멤버의 Acceptor 내 get_state() 함수를 이용하여 현재 멤버의 상태를 받아온 후(6), set_state() 함수를 이용하여 새로운 멤버에게 상태를 전달한다(7). 마지막으로 시스템은 모든 멤버에게 뷰 변경 완료 메시지를 멀티캐스트 한다(8).

3.4 그룹 멀티캐스트 모델

본 논문에서 제안하는 시스템의 통신 모델은 일대다(one-to-many) 통신이며, 이에 하나의 클라이언트가 그룹을 형성하고 있는 다수의 서버들에게 메시지를 전송한다. 클라이언트와 그룹 사이의 연결은 클라이언트의 Connector와 그룹의 Acceptor에 의해 이루어진다. 연결이 완료된 후, 하나의 Connector와 다수의 Acceptor는 각각 Transport 객체를 생성하며, 통신은 Transport 객체들 사이에서 멀티캐스트로 이루어진다. 본 절에서는 클라이언트와 그룹 사이의 연결을 위해 전송되는 메시지와 객체 호출(object invocation)을 위해 교환되는 메시지에 대해 기술한다.

3.4.1 클라이언트에서 그룹으로의 연결 과정

그림 7은 클라이언트가 그룹에 연결하는 과정을 나타낸다. 서버 측의 ORB는 Acceptor의 accept() 함수를 호출한다(1). 이에 Acceptor는 자신이 사용할 수 있는 전송 프로토콜에서 요청 메시지 수신을 기다린다(2). 클라이언트는 IOR에 포함되어 있는 원격 객체 주소값(remote object reference)을 클라이언트 측에 있는 로컬 프락시(local proxy)에 연결하고(3), 클라이언트 측의 ORB는 Connector의 connect() 함수를 호출한다(4). Connector는 그룹 주소값을 이용하여 그룹에 연결을 시도한다(5). 이 후 그룹 통신 시스템은 Connector와 Acceptor에게 연결 완료에 해당하는 통보 메시지를 전달한다(6). 클라이언트와 서버 그룹 측에 있는 Connector와 Acceptor는 각각 Transport 객체를 생성하고, Connector와 Acceptor가 포함하고 있는 그룹 멤버 주소값들을 Transport에 전달한다(7). 클라이언트와 그룹 사이의 통신은 Transport 객체 사이에서 이루어진다.

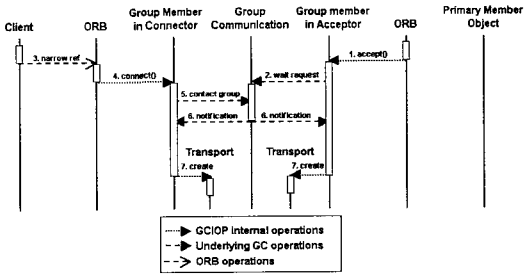


그림 7 클라이언트와 멤버들 사이의 연결을 위한 과정

3.4.2 객체 그룹 호출 과정

클라이언트가 ORB를 통해 원격 객체를 실행시킬 때, 기본적으로 하위 수준 통신은 메시지 전송(message

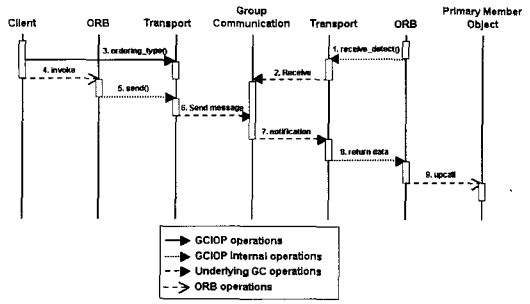


그림 8 객체 실행 과정

passing) 방식이다. 그룹 통신에서 데이타 전송의 순서화는 실행 때마다 변경할 수 있어야 한다. 그림 8은 객체 실행(object invocation)을 위한 과정을 나타낸 것이다. 제안된 시스템에서 클라이언트는 *TransportInfo* 클래스의 ordering_type 함수를 이용해서 순서화 타입을 결정할 수 있다(3). 이 후 클라이언트가 서버 측의 객체 실행을 요청한다면(4), 클라이언트 측의 ORB는 Transport의 send() 함수를 호출하고(5), 메시지는 그룹 통신 시스템의 SendMessage() 함수를 통해 그룹에게 전송된다(6).

서버 측의 ORB는 연속적으로 Transport 객체의 receive_detect() 함수를 이용해서 메시지의 도착을 조사한다(1). receive_detect() 함수는 불린(boolean) 값을 리턴한다. 즉, 리턴 값이 참이면, 서버는 receive_detect() 함수의 매개변수로 Buffer 객체를 사용함으로써 메시지를 받는다. 한편 거짓 값을 받으면, (1),(2),(7),(8) 과정을 반복한다. 결국 ORB는 서버 객체에게 요청을 전달하고, 객체 실행이 완료된다(9).

3.5 그룹 통신 OCI의 수행 과정

초기화 과정에서 클라이언트 객체와 서버객체는 그룹 통신 OCI 객체를 ORB에 끼워넣기 위한 초기화 클래스를 호출한다. 초기화 클래스는 객체 어뎁터(object adapter; OA)의 존재 유무에 따라 클라이언트 객체인지 서버 객체인지를 구분하며, 서버 측에서 OA 주소값은 매개변수로써 초기화 클래스에게 전달한다. 초기화 클래스는 기본 ORB 정책(policy)을 그룹 통신 정책으로 설정한다.

서버 측에서 그룹 통신을 위한 새로운 GCIOP Acceptor가 OCI에 추가되어야 한다. OA 관리자는 get_acc_registry 함수를 이용하여 Acceptor Registry에 접근할 수 있다. 새로운 Acceptor는 생성 후, Acceptor Registry의 add_acceptor 함수로 ORB에 추가된다. 클라이언트 측에서 Connector Factory는 생성 후, ORB에서 제공하는 Connector Factory Registry에 의해 추가된다. 서버는 OA와 Acceptor Registry를 통해 기본적인 Acceptor의 주소값을 얻을 수 있다. 이에 Acceptor 주소값은 IDL 컴파일러로 생성된 함수인 GCIOP.AcceptorHelper의 narrow 함수를 이용해 GCIOP Acceptor에 매핑된다. 이 후 GCIOP Acceptor의 create_group 함수가 호출되고, OA는 IOR를 만든다. 여기서 우리는 처음 그룹에 가입하는 멤버가 IOR을 만든다고 가정한다.

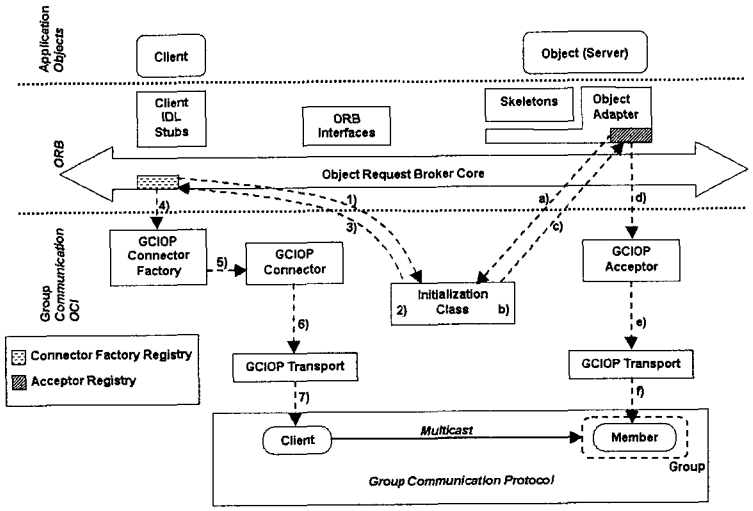
OA가 활성화 된 후, Acceptor는 Transport를 만든다. Transport는 하부 그룹 통신 서비스의 프로세스 그룹 멤버를 초기화한다. TransportInfo 객체의 정보에 따라, Transport는 프로세스 그룹 멤버를 그룹에 가입 또는 탈퇴한다. 여기서 주 개념은 서버 객체의 Transport를 프로세스 그룹 멤버에게 매핑하는 것이다. 새로운 그룹 멤버가 기존 그룹에 가입할 때에는 기존 멤버의 상태(state)를 새로운 멤버에게 전송한다. 서버는 GCIOP Acceptor의 get_state 함수와 set_state 함수를 이용하여, 상태를 얻거나 설정할 수 있다. 상태 정보는 object 타입으로 AcceptorInfo에 저장된다.

클라이언트가 초기화 된 후에는 ORB의 Connector Factory Registry가 Connector Factory를 만든다. GCIOP Connector Factory를 추가하기 위해, Connector Factory 주소값은 GCIOP Connector Factory의 주소값에 연결되어져야 한다. Connector Factory는 IOR의 정보를 통해 Connector를 생성하고, Connector는 Transport를 생성한다. 클라이언트는 GCIOP Connector TransportInfo의 OrderingType 변수를 사용하여 순서화 타입을 설정할 수 있다. 그림 9는 제안된 시스템에서 서버와 클라이언트가 초기화하는 과정을 설명한다.

4. 구 현

4.1 시스템 구조

본 논문에서 제안하는 시스템은 그룹 통신 프로토콜, ORB, 확장된 OCI로 구성된다. 하부 그룹 통신 프로토콜인 FTGCS(Fault Tolerant Group Communication Service)[17]는 그룹 통신 컴포넌트와 그룹 멤버쉽 관리 컴포넌트를 가지고 있다. 그룹 통신 컴포넌트는 메시지 멀티캐스팅과 신뢰적 전송을 지원하며, 그룹 멤버쉽 관리 컴포넌트는 동적으로 멤버쉽 리스트를 관리하며, 그룹 뷰의 일관성을 보장한다. OCI를 지원하는 CORBA는 ORBaus Java 4.01을 사용하였다[11]. 구현된 내용들은 Java Archive(JAR) 파일로 제공되며, 확



클라이언트 측
1)Connector Factory Registry를 얻는다.
2)GCIOP Connector Factory를 생성한다.
3)Connector Factory를 Connector Factory Registry에 추가한다.
4)Connector Factory를 생성한다.
5)GCIOP Connector를 생성한다.
6)GCIOP Transport와 GCIOP ConnectorTransportInfo를 생성한다.
7)그룹 통신 프로토콜에서 클라이언트를 초기화 한다.

서버 측
a)Acceptor Registry를 얻는다.
b)GCIOP Acceptor를 생성한다.
c)Acceptor를 Acceptor Registry에 추가한다.
d)Acceptor를 생성한다.
e)GCIOP Transport와 GCIOP Acceptor TransportInfo를 생성한다.
f)그룹 통신 프로토콜에서 그룹 멤버를 초기화 한다.

그림 9 그룹 통신 OCI의 초기화 과정

장된 OCI는 GCIOP IDL에 의해 생성된 파일을 포함하여 213K bytes, FTGCS는 73K bytes를 차지한다. 모든 시스템은 JDK1.3으로 구현되었다.

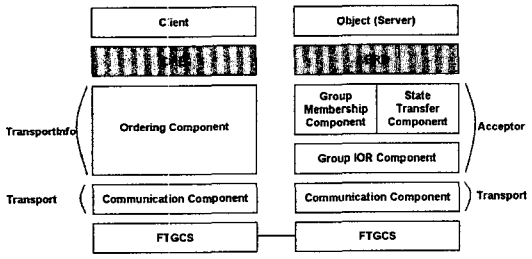


그림 10 시스템 구조

그림 10은 제안된 시스템 구조이다. 클라이언트 부분은 메시지 전송에 대한 순서화 타입을 설정할 수 있는 순서화 컴포넌트와 그룹에게 멀티캐스트 통신을 지원하는 통신 컴포넌트로 구성된다. 확장된 OCI에서 순서화 컴포넌트와 통신 컴포넌트는 각각 TransportInfo와 Transport 객체에 포함된다. 서버 부분에서 그룹 멤버쉽은 서버 객체가 Acceptor의 함수들을 이용해서 그룹을 구성할 수 있게 한다. 상태 전송 컴포넌트는 상태 전송 메커니즘을 지원하며, 그룹 IOR 컴포넌트는 그룹 주소값으로써 그룹 IOR을 만든다. Transport 객체의 한 부분으로 통신 컴포넌트는 FTGCS를 사용한다. 그리고 OCI에서의 통신은 Transport 객체들 사이에서 이루어진다.

그룹 멤버는 Acceptor의 create_group(), join() 함수에서 초기화된다. 클라이언트 측의 Transport 객체는 Connector의 connect() 함수에서, 서버 측의 Transport 객체는 Acceptor의 accept() 함수에서 생성된다. 이 후 그룹 멤버는 Transport 객체에 매핑된다. 즉 그룹 통신이 CORBA의 애플리케이션 객체들에게 제공되는 것이다. 다음 절부터는 그림 11의 GCIOP 함수들을 설명한다.

4.2 그룹 멤버쉽 컴포넌트

동적 그룹 멤버쉽을 지원하기 위한 그룹 관리 함수들은 Acceptor 객체에서 제공된다. 한편 그룹 뷰와 그룹 멤버들 사이의 일관성 관리는 FTGCS에 의존한다. 객체 애플리케이션이 Acceptor의 함수들에 의해 FTGCS

```
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);

  // State Transfer Component
  //
  // 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 an 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 which 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);
  // Receive a buffer's contents through FTGCS.
  public boolean receive_detect(OCI.Buffer buf, boolean
block);
}
```

그림 11 GCIOP 함수들

의 그룹 멤버에게 연결되었을 때, 함수들은 객체 애플리케이션과 연결할 그룹의 정보가 필요하다. 이에 Acceptor의 로컬 변수로써 그룹 이름, 호스트 이름, 하부 그룹 통신 프로토콜의 그룹 멤버의 주소값을 가지고 있으며, Acceptor의 함수들은 이 변수들을 함수의 매개변수로 이용한다. 애플리케이션 객체들이 인식하는 그룹 이름은 하부 그룹 통신 시스템에서의 그룹 이름과 같으며, 스트링 타입의 group_name_ 변수에 저장된다. 호스트 이름은 객체 애플리케이션이 존재하는 호스트의 이름으로 프로파일 정보를 만드는 데 사용되며, 스트링 타입으로 host_ 변수에 저장된다. 그룹 멤버 gm_은 FTGCS의 그룹 멤버를 가리킨다. 이 값은 함수의 매개변수로써 Transport 객체에 전달되며, 오직 하나의 Transport 객체에 매핑된다.

Acceptor의 함수들은 Acceptor의 IDL 명세서에 언급되었듯이 그룹 멤버쉽과 상태 전송, 멤버들 사이의 통신을 지원하며, 이 함수들에서는 FTGCS의 해당 함수들이 호출된다. 예를 들어, 그룹 함수에 해당하는 Acceptor 함수들은 'gm_.Create_group(group_name_)', 'gm_.Join (group_name_)', 'gm_.Leave()'를 호출한다. 상태 전송 함수 또한 FTGCS의 해당 함수인 'set_state(org.omg. CORBA.Object state)', 'get_state()'를 호출한다.

4.3 순서화 컴포넌트

ConnectorTransportInfo와 AcceptorTransportInfo의 차이점은 GCIOP TransportInfo 객체에서 순서화 설정 여부이다. *ConnectorTransportInfo* 객체에서만 애플리케이션 프로그래머는 순서화 타입을 설정할 수 있는 함수를 사용할 수 있다. ConnectorTransportInfo는 인과관계 순서화(causal ordering)와 전 순서화(total ordering)를 표시할 수 있는 OrderingType 변수를 가지고 있다. ordering_type() 함수는 OrderingType을 리턴하며, 이 함수를 이용하여 순서화 타입을 확인 할 수 있다. ordering_type(OrderingType value) 함수는 value 매개변수 값을 value_ 라는 로컬 변수에 저장한다. 실제로 이 값은 Transport 객체에서 메시지 전송 시 적용된다. Transport의 send 함수는 실제 하부 그룹 통신 프로토콜의 함수인 SendMessage(buf.data(), info_.ordering _type(), atomic)을 호출 함으로써 메시지 전송을 한다.

4.4 그룹 IOR 컴포넌트

Acceptor는 서버 측에서 IOR을 만들기 위해 프로파일 정보를 채워 넣는다. IDL 컴파일러에 의해 생성된 그룹 IOR의 ProfileBody 클래스의 생성자에서는 3.2.1. 절에서의 변수를 초기화 하고, Acceptor의 add_profile 함수에서 ProfileBody의 정보를 완성한다.

4.5 통신 컴포넌트

통신 컴포넌트의 함수들은 CORBA의 내부 함수들로써 ORB에 의해 요청된다. Transport 객체들이 클라이언트와 서버 사이의 연결을 완료한 후, Buffer 객체에 교환할 메시지의 내용을 저장하여 send, receive 함수로 Buffer 객체를 주고받음으로써 메시지를 교환한다. 전송되는 데이타는 바이트 스트림(byte stream)이다. Transport 객체들 사이의 기본 통신은 IIOP이고, 그룹 통신을 위한 인터페이스 또한 같은 이름의 인터페이스를 사용하기 때문에, 하부 통신 프로토콜을 그룹 통신 프로토콜로 사용하더라도 send_detect, receive_detect, send_timeout, receive_timeout 함수와 같이 IIOP 기반의 함수들과 같은 이름의 인터페이스들을 구현해야 한다. 이 함수들의 내부에서는 하부 그룹 통신 프로토콜인 FTGCS의 SendMessage(byte[] data, OrderingType

표 1 GCIOP를 이용한 다중 객체 실행(msec)

서버 객체수	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
전체 시간	174.28	172.62	174.98	171.36	171.13	174.70	171.60	174.75	173.60	173.85
그룹 인터페이스를 사용하는 ORB에서 소모되는 시간	49.26	48.28	50.66	46.90	48.08	50.45	47.25	49.95	50.05	49.55
FTGCS에서 소모되는 시간	125.02	124.34	124.32	124.46	123.04	124.25	124.35	124.80	123.55	124.30

ordering, int atomic), Receive() 함수를 호출한다. ORB의 초기 환경 설정에 따라 send, receive에 해당하는 다음 함수들이 호출된다.

- void send(OCI.Buffer buf, boolean block)

Buffer 객체는 클라이언트와 그룹 사이의 메시지 교환을 위해 사용되는 매개 객체이다. Block 옵션은 제안된 방법에서는 고려하지 않는다. 이 함수 내부에서 SendMessage(buf.data(), info_.ordering_type(), atomic)가 호출되며, buf.data()는 바이트 배열로써 메시지 자체를 의미한다. info_.ordering_type()는 순서화 값이며, atomic 옵션은 FTGCS의 기본 값을 따른다.

- boolean receive_detect(OCI.Buffer buf, boolean block)

receive_detect 함수는 클라이언트로부터 보내진 메시지가 FTGCS의 receive 큐에 도착했는지를 확인하기 위해 반복적으로 실행된다. 도착한 메시지가 없다면, 이 함수는 FALSE 값을 리턴한다. 한편 도착한 메시지가 있다면, 이 함수에서는 내부적으로 FTGCS의 gm_.Receive()을 호출하고, TRUE 값을 리턴한다. 그리고 받은 메시지는 Buffer 객체로 전달된다.

4.6 그룹 객체 키

CORBA에서의 객체 호출에 있어서 수신자가 메시지를 받았을 때, 이 메시지로 인해 호출되는 객체의 객체 키(object key)가 요청된다. 이와 마찬가지로 그룹 통신이 CORBA에 적용한 경우에도, 그룹을 이루고 있는 멤버 측에 그룹 IOR을 포함하는 객체 키가 요청된다. 본 연구에서는 기존의 일대일 통신에서 사용하는 객체 키를 재사용하였다. 그러나 기존의 BOA 또는 POA 모델에서는 그룹의 모든 객체 구현들을 하나의 객체 키로 표현할 수 없기 때문에 [18], 각 멤버들의 객체들을 구분하지 못한다. CORBA에서 그룹 객체 모델을 지원하기 위해서는 그룹 객체 키를 생성하는 새로운 OA가 디자인 되어야 한다. 본 논문에서 제안하는 시스템은 표준 ORB를 수정하지 않고 지원하기 위해, 실제 구현에서 클라이언트의 요청 메시지에 포함된 객체 키를 각 멤버의 객체 키로 교체되도록 하였다. 여기서 모든 멤버들은 네이밍 서비스와 같은 추가적인 메커니즘을 이용함으로써 자신이 속한 그룹을 대표하는 객체 키를 알고 있다고 가정한다. 모든 멤버들의 ORB는 자신의 객체 구현에 해당하는 객체 키를 생성하며, 각 멤버들의 GCIOP Transport는 로컬 변수로써 객체 키를 저장한다. 클라이언트로부터 요청된 메시지가 GCIOP Transport에 도착했을 때, 멤버들은 도착한 메시지가 현재 가입되어 있는 그룹으로 보내진 것인지를 확인한다. 받은 메시지에 포함되어 있는 객체 키가 대표 객체 키와 같다면, 이는 로컬 객체 키로 교체된다. 이 후 클라이언트에서 요청한 함수가 호출된다.

5. 성능 평가

본 장에서는 IIOP를 이용한 일대일 원격 실행(one-to-one remote invocation)과 확장된 OCI를 활용한 원격 실행을 비교한다. 실험은 10Mbit Ethernet으로 연결된 Windows 2000 기반의 호스트 10대(펜티엄 III 프로세서, 384MB 또는 256MB 램)로 수행하였다. 각 머신은 하나의 서버 객체를 실행하며, 클라이언트는 호스트들 중 임의의 하나에서 실행한다. 이 실험은 50번 반복 수행하여 평균값을 취하였다.

클라이언트가 서버 객체들에게 첫 실행을 요청할 때, 클라이언트와 서버 객체들 사이의 통신을 위해 클라이언트 및 서버 객체들에 대한 Transport 객체들이 생성된다. Acceptor에서 프로세스 그룹 멤버를 생성한 후, 서버 객체들은 클라이언트와의 연결 시에 프로세스 멤버를 Transport 객체에 전달한다. 한편 클라이언트는 Transport 객체 내에서 프로세스 멤버를 생성한다. 표 1은 GCIOP를 이용하여 통신하는 ORB에서 동작하는 클라이언트가 원격 실행을 요청한 후, 그 결과를 받을 때까지 소비된 시간으로 요청/응답 시간, Transport 객체 생성 시간, 프로세스 멤버 생성 시간을 포함한 것이다. GCIOP를 이용하는 경우에는 한번의 요청(request)으로 다중 객체 실행(multiple object invocation)이 가능하기 때문에 서버 객체의 수가 증가하더라도 전체 소비 시간은 거의 일정하게 나타난다. 전체 시간은 GCS

표 2 IIOP를 이용한 다중 객체 실행(msec)

서버 객체수	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
전체 시간	91.56	141.62	208.66	263.16	350.86	434.38	506.12	602.10	672.82	777.90
IIOP 인터페이스를 사용하는 ORB에서 소모되는 시간	48.54	96.78	148.03	199.93	252.30	298.40	354.20	403.40	458.90	505.85
Socket에서 소모되는 시간	43.02	44.84	60.64	63.23	98.56	135.98	151.92	198.70	213.92	272.05

에서 소비되는 시간도 포함되어 있다. FTGCS는 UDP/IP 멀티캐스트를 기반으로 구현되어 있으며, 그룹 멤버쉽과 신뢰성 있는 전송과 같은 그룹 통신 특성들을 지원하기 위해 그룹 뷰와 전송 큐(delivery queue)를 관리한다. 결국 GCIOP를 이용한 실행 시간은 그룹 인터페이스를 이용하는 ORB에서 소비되는 시간과 FTGCS에서 소비되는 시간으로 나눌 수 있다. 표 1에서 알 수 있듯이 전체 시간은 FTGCS에서 소비되는 시간에 의존적이다. 결국 확장된 OCI에서는 서버 객체 수와 상관없이 거의 일정한 시간(약 49.04ms)이 소모된다.

표준 CORBA는 멀티캐스트 통신을 지원하지 않기 때문에, 표준 CORBA에서의 그룹 객체 실행은 다중 일대일 원격 실행(multiple one-to-one remote invocation)으로 수행할 수 있다. 표 2는 서버 객체 수에 따른 요청/응답까지 소모되는 시간을 클라이언트에서 측정한 값이다. 이 경우도 GCIOP를 활용한 경우와 마찬가지로 OCI를 지원하는 ORBacus를 이용했기 때문에 Transport 객체 생성 시간이 포함되어 있으며, 전체 시간 또한 ORB에서 소모되는 시간과 통신 부분에서 소모되는 시간으로 나뉘어 있다. 서버 객체 수가 증가함에 따라 전체 시간은 일정하게 증가하는데, 이것은 서버가 추가될 때마다 클라이언트와 서버 사이의 추가적인 연결이 이루지기 때문이다. 실험 결과에서 서버 객체수가 3개 미만일 때는 그룹 실행을 하는 경우가 IIOP를 이용하는 경우보다 나쁘게 나타난다. 이것은 전자의 경우 전송된 메시지들이 그룹 통신 프로토콜 내에 있는 전송 큐에 저장되지만, 후자의 경우에는 이러한 오버헤드가 포함되어 있지 않기 때문이다.

6. 결 론

본 논문에서는 CORBA에 기존의 다양한 그룹 통신 프로토콜을 지원하기 위해 OCI[10]를 확장하였다. 이를 위해 그룹 이름과 같은 종단의 정보를 갖는 GCIOP를 정의하고, 그룹 함수들을 사용하기 위해 그룹 통신 Info 객체와 OCI를 확장하였다. 서버 측의 Acceptor Info는 그룹 이름, 호스트 이름, 상태 정보를 가지고 있으며, 클라이언트 측의 Connector Info는 오직 그룹 이름 만을 가지고 있다. Transport Info는 하부 그룹 통신 프로토콜에 전달되는 그룹 이름, 호스트 이름, 순서화 정보를 가지고 있다. 서버 객체들은 Acceptor의 함수들을 이용하여 그룹 멤버쉽 관리를 하며, 서버에서 제공하는 함수들은 클라이언트가 Transport 객체를 통해 멀티캐스트한 메시지에 의해 호출된다. 순서화는 실행 시에 Transport Info의 순서화 변수 설정 값에 따라 결정된다. 또한 본 디자인에는 ORB와 OCI의 수정 없이 기존에 나와있는 다양한 그룹 통신 프로토콜이 적용될 수 있다.

제안된 디자인에서는 그룹 통신 모델을 일대다 통신으로 가정하였다. 송신자는 Connector 위에서 동작하며, 수신자들 또한 Acceptor 위에서 동작한다. 애플리케이션 객체들이 클라이언트와 서버의 역할을 모두 할 수 있도록 구현된다면, 제안된 디자인 또한 다대다 통신에 적용될 수 있을 것이다. 결론적으로 본 연구에서 제안한 OCI 기반 그룹 통신 프레임웍은 표준 ORB를 수정하지 않고, OS에 대한 의존성 없이 기존의 그룹 통신 프로토콜을 CORBA에 적용할 수 있다. 향후 CORBA에서의 그룹 통신 시스템과 본 논문에서 제안한 방법으로 기존의 그룹 통신 프로토콜을 CORBA에 적용하여 성능을 비교할 계획이다. 이와 함께 그룹 객체 키를 효율적으로 처리할 수 있는 방법에 대해 연구해 갈 것이다.

참 고 문 헌

[1] Guerraoui, R. and Schiper, A., "Software-Based Replication for Fault Tolerance," *IEEE Computer*, Vol. 30, No. 4, pp. 68-74, April 1997.

[2] Maffeis, S., "The Object Group Design Pattern," *The 1996 USENIX Conference on Object-Oriented Technologies*, Toronto, Canada, June 1996.

[3] Narasimhan, P., Moser, L.E., and Melliar-Smith, P.M., "Using Interceptors to Enhance CORBA," *IEEE Computer*, pp. 62-68, July 1999.

[4] Narasimhan, P., Moser, L.E., and Melliar-Smith, P.M., "The Interception Approach to Reliable Distriubted CORBA Objects," *Third USENIX*

Conference on Object-Oriented Technologies and Systems, pp. 245-248, June 1997.
[5] Maffeis, S., "Adding Group Communication and Fault-Tolerance to CORBA," *The 1995 USENIX Conference on Object-Oriented Technologies*, Monterey, CA, June 1995.
[6] Felber, P., Garbinato, B., and Guerraoui, R., "The Design of a CORBA Group Communication Service," *15th IEEE Symposium on Reliable Distributed Systems*, pp.150-159, October 1996.
[7] Moser, L.E., Mellar-Smith, P.M., Narasimhan, P., Tewksbury, L.A. and Kalogeraki, V., "The Eternal System : An Architecture for Enterprise Application," *International Enterprise Distributed Object Computing*, pp. 214-222, September 1999.
[8] Felber, P. and Guerraoui, R., "Programming with Object Groups in CORBA," *IEEE Concurrency*, Vol. 8, No. 1, pp. 48-58, Jan.-March 2000.
[9] Narasimhan, P., Moser, L.E., and Melliar-Smith, P.M., "Exploiting the Internet Inter-ORB Protocol Interface to Provide CORBA with Fault Tolerance," *Third USENIX Conference on Object-Oriented Technologies and Systems*, pp. 81-90, June 1997.
[10] AT&T et al. *Internetworking Between CORBA and Intelligent network Systems*, OMG Document telecom/98-06-03, 1998.
[11] *ORBacus Web Site*: http://www.ooc.com/ob/, Object-Oriented Concepts, Inc.
[12] Birman, K.P., "The Process Group Approach to Reliable Distributed Computing," *Communications of the ACM, 36(12)*, pp. 37-53, December 1993.
[13] Halteren, A.T. van, Noutash, A., Nieuwenhuis, L.J.M., and Wegdam, M., "Extending CORBA with Specialised Protocols for QoS Provisioning," *International Symposium on Distributed Objects and Applications*, pp. 318-327, September 1999.
[14] Schmidt, D.C., "Acceptor-Connector: An Object Creational Pattern for Connecting and Initializing Communication Services," In: Martin, R., Buschman, F., and Riehle, D., *Pattern Languages of Program Design 3*, Addison-Wesley, 1997.
[15] *The Common Object Request Broker: Architecture and Specification, Rev. 2.3*, Object Management Group, OMG Document formal/98-12-01, June 1999.
[16] Moser, L.E., Melliar-Smith, P.M., Narasimhan, P., Koch, R.R. and Berket, K., "Multicast Group Communication for CORBA," *International Symposium on Distributed Objects and Applications*, pp. 98-107, September 1999.
[17] Lee, D. et al., "Development of reliable group communication module for object group," *Project Report, CDS&N Lab., Information and Communications University(ICU)*, December 1999.
[18] *Unreliable Multicast Inter-ORB Protocol: Initial Submission*, Eternal Systems, Inc. et al., OMG Document orbos/2000-02-03, February 2000.

남 덕 윤
1999년 2월 포항공과대학교 컴퓨터공학 학사. 2001년 2월 ICU 공학 석사. 2001년 3월 ~ 현재 ICU 박사과정 재학중. 관심분야는 그룹 통신, CORBA

이 동 만
1982년 2월 서울대학교 컴퓨터공학 학사. 1984년 2월 KAIST 전산학 석사. 1987년 2월 KAIST 전산학 박사. 1987년 3월 ~ 1988년 3월 KAIST post doc. 1988년 4월 ~ 1997. 9월 Hewlett-Packard 책임연구원. 1997년 10월 ~ 현재 ICU 부교수. 관심분야는 네트워크 및 분산 시스템, scalable network architecture for distributed virtual enviroment, reliable multicast protocol, fault-tolerant group communication, layered multimedia multicast, web caching, collaborative computing framework