분산 객체 컴퓨팅에서 객체 그룹화를 위한 모델 설계

송기범* · 홍성표* · 이 준**
*조선대학교 컴퓨터공학과
**조선대학교 컴퓨터공학부

Design of Model for Object's Grouping in Distributed Object Computing

Gi-beom Song* · Seong-pyo Hong* · Joon Lee**
*Dept. of Computer Engineering, Graduate School, Chosun University
**School of Computer Engineering, Chosun University

요 약

분산환경에서 효율적인 분산 서비스를 제공하기 위해 TINA 콘소시엄과 OMG CORBA에서는 객체지향기술을 적용한 분산 객체 플랫폼의 제안과 다양한 서비스의 요구사항들 정립하고 있다. 그러나, 어플리케이션의 규모가 점점 커지고 분산화 됨에 따라 객체들간의 서비스 및 관리 인터페이스가 매우 복잡해지고 있다. 이러한 문제들을 해결하기 위해서 새로운 객체그룹 모델의 제안과 객체그룹하에서 도입될 수 있는 객체 관리 및 서비스를 위한 요구사항들이 필요하다.

본 논문에서는 TINA에서 제안한 그룹객체 정의를 도입하여 현재 분산 시스템의 표준으로 사용하는 CORBA 기반에서 분산된 객체들을 효율적으로 관리할 수 있는 시스템을 제안한다.

ABSTRACT

For efficiently providing distributed services, distributed computing environments are specified the requirements of various services and distributed object platforms applied an object-oriented technology by TINA Consortium and OMG CORBA. Because applications are becoming large and distributing, their servicing and managing interfaces among objects are being complicated.

In order to solve these defects, it is necessary to suggest a new object grouping model and specify object service/management requirements can be introduced under the object groups.

키워드
Object Orinted, Distributed Computing, Object Group, CORBA

I. 서론

이 기종 분산환경에서 날로 복잡해지는 분산 어플리케이션 서비스 플랫폼으로써, 새로운 개방형 통신망의 구조 정립을 위해서 객체지향 모델링 기법을 적용한 개방형 정보 통신망 구조의 연

구가 활발하게 진행 중에 있다. 분산 객체 시스템은 전형적으로 분산 컴퓨팅 노드들 상에 펼쳐진 객체 시스템은 전형적으로 분산 컴퓨팅 노드들 상에 펼쳐진 객체들의 인스턴스들과 바인딩되는 객체 인스턴스들의 분산된 집합으로 구성되므로 그 관리 및 서비스가 매우 복잡하다. 따라서 개방형 통신망인 TINA-C(Telecommunication Information Network Architecture-Consortium)[1][5]의 TINA 권고안에서는 개별적인 객체뿐만 아니라, 객체들의 집합을 관리하는 구조화 메카니즘으로써 그룹객체를 정의하고있다. 객체그룹화의 목적은 다양하고 복잡한 분산 멀티미디어 서비스를 단순하게 관리하고 서비스를 지원하는데 있다.

본 논문에서는 TINA에서 제안한 그룹객체 모델 구조를 이용하여 현재 분산 시스템의 표준으로 사용하는 CORBA 기반에서 그룹객체에 대한 구성요소의 기능을 정립하고 정립된 기능을 수행할 수 있는 그룹객체 관리구조를 표현한다.

II. 관련 연구

분산객체 시스템은 모든 개체(entity)가 객체로 모델링된 분산된 시스템이다. 분산객체 시스템은 잘 알려진 객체지향 분산 어플리케이션의 패러다임이다. 이 어플리케이션은 상호작용하는 객체들의 집합으로 이루어져 있기 때문에 매우 자연스럽게 분산 시스템의 서비스를 배치할 수 있다.

분산 컴퓨팅은 다른 사용자나 컴퓨터가 정보를 공유할 수 있도록 한다. 즉, 하나의 컴퓨터에 있는 어플리케이션으로 하여금 다른 컴퓨터의 연산능력, 메모리, 저장장치 등을 함께 사용할 수 있게 한다. 이것은 분산 컴퓨팅이 독립형(stand-alone) 어플리케이션의 성능을 향상시킬수 있게 만든다. 워드프로세싱같은 어플리케이션은 분산 컴퓨팅의 이점을 가지지 못하지만, 만약 한 어플리케이션의 수행중 많은 작업량을 요구하는 계산을 수행할 필요가 있을 때, 이 계산을 분산 컴퓨팅 환경으로 분산시키면 성능의 향상을 꾀할 수 있고 분산컴퓨팅이 필요한 좋은 예가 된다.

분산객체는 다른 메모리 도메인에 존재하므로, 접근하기 위해서는 네트워크를 통해야 하는 객체를 말한다. 이것은 일반 객체와 마찬가지로 데이터와 메소드로 이루어져 있다. 분산 객체 기술은 다양한 환경 즉, 이종의 플랫폼과 프로그램언어에 독립적으로 분산 객체를 사용할수 있게 해주는 기술이다. 네트워크에의해 컴퓨터가 클라이언트/서버 환경으로 분산됨에 따라 정보 시스템은 다양한 이 기종 컴퓨터 환경에서 분산되어 운용된다. 즉, 데이터와 응용프로그램을 분산시켜 운용하면서, 프로그램의 상호 운영성과 분산의 투명성을 제공해야할 필요성이 대두되었다. 한편, 소프트웨어 위기의 극복 방법으로 객체지향 기술이 재조명되고 있고, 객체지향 기술이 제공하는 추상화, 캡슐화, 상속성, 다형성 등의 성질은 컴포너트 프로그램을 가능하게 하고, 소프트웨어 생산성을 높일 수 있다. 따라서, 객체지향 방법론은 소프트웨어 개발 방법론으로 계속 성장하고 있고, 객체들은 데이터와 그 데이터를 조작할 수 있는 메소드로 구성되어 있으며, 프로그램은 메시지에 의한 객체간 상호작용을 기술한 것으로 객체간 메시지 교환으로 문제를 해결한다.

이러한 분산객체 컴퓨팅은 분산 컴퓨팅과 객체지향 기술의 장점을 효과적으로 통합하여주는 기술이며, 개발자에게 어플리케이션 개발의 생산성을 향상을 제공하고, 사용자에게 분산환경에 투명하게 통합된 정보를 제공해준다.[2][3][4]

현재 컴퓨터 사용자들은 개인적으로 생산성있는 응용 프로그램을 원할뿐만 아니라 파일이나 데이터베이스처럼 다양한 데이터 저장공간으로부터 정보를 얻길 원한다. 뿐만 아니라 다른 사요아들과 자료를 공유하고 서로 통신하길 원하고 있다. 이렇듯 다양한 요구사항을 만족시키는 프로그램을 작성하기 위해서는 서로다른 운영체제와 네트환경 등 이종의 환경하에서 작동가능한 클라이언트/서버 소프트웨어를 개발해야 한다. 뿐만 아니라, 개발된 소프트웨어는 분산환경에서 서로다른 시스템과도 쉽게 통합이 가능해야 한다. 그러나, 이러한 요구사항을 만족시키는 작성하는데 있어서 많은 어려움이 존재한다. 먼저, 기존의 C와 같이 로직의 흐름에따라 프로그램을 작성하는 방식으로는 이 같은 요구사항을 만족시키기 어렵고, 이종의 분산 환경에서 여러 종류의 프로그램을 통합하기위해서는 일정한 결합방식이 필요하기 때문이다.

이러한 문제를 해결하는 방식으로, 기존의 RPC[4][7] 방식이나 OLE[8][9] 같은 방식을 사용했다. 그러나, RPC의 경우 데이터 상호교환에 대한 자신의 프로토콜을 정의하기위해 네트워크에

대한 상호작용을 설계하는데 많은 시간을 소모하였고, 복잡한 프로그램을 작성하는데 있어 많은 한계를 가지고 있다. OLE의 경우에도 현재 윈도우즈 객체들만의 호환을 지원하기 때문에 진정한 분산 시스템 표준이라고 말하기 어렵다. 이러한 불편함으로 공통 구조에대한 개념의 필요성이 대두되었고, 1990년 경 OMG[4][7]가 분산 시스템을 위해 OMA를 도입하여, CORBA가 OMA 추상객체 모델위에 세워지면서 표준으로 정의되었다.

또한, 1992년에, 몇몇의 통신회사가 네트워크 전역에 광범위의 서비스를 제공하기위한 텔레커뮤니케이션 네트워크에 기반을 둔 분산 객체지향 구조의 설계와 기술을 위한 목적으로 TINA-C[1][5]를 설립하였다.

1. CORBA

CORBA는 분산객체 시스템의 설계와 개발에 대한 새로운 표준을 제시한다. CORBA에 있는 기본적인 개념은 한 서비스 요구자가 임의의 객체나 심지어 전형적인 비 객체 지향적 프로그램을 요구할 때, 그 클라이언트로부터 하나의 요구를 받아들이고 응답하는 임의의 객체를 위한 균일한 방법을 제공하자는 것이다. 일단 그러한 하나의 요구가 만들어지면, ORB는 그 요구가 적절한 서비스 제공 객체(어디에 있든지 어떻게 구현되어졌든지 상관없이)에 배달되어지는 것을 보장해야 한다.

이를 위해 클라이언트는 인터페이스 정의어(IDL)로 기술된 미리 정의된 인터페이스를 통하여 객체 구현으로부터 서비스를 요구한다. 서버로 전달된 요구는 객체 레퍼런스와 파라미터로 구성되며, 이 요구의 형태는 광역의 이질적인 네트워크에서 서로 주고받을 수 있도록 표준화되어진다. 따라서, 클라이언트는 서비스를 제공하는 객체의 자료 표현이나 실행 코드등과 같은 객체구현에 독립적이다. IDL의 목적은 메소드나 함수와 접근 가능한 속성의 완전한 시그니쳐(signiture; 이름, 파라미터와 타입 등)를 포함하는 인터페이스들의 언어 독립적인 표현을 허용하는 것이다.

모든 이들 능력을 제공하기위해 CORBA 기술서는 다른 벤더(vender)들에 의해 다른 방법으로 구현되어질수 있는 인터페이스들의 구조를 정의한다. CORBA의 구조는 인터페이스와 구현을 분리하여 기술되도록 설계되어졌다. CORBA 구조의 주요 요소는 (그림 1.)처럼 클라이언트 측, 구현 객체 측, 그리고 ORB core 등 세가지 범주로 나누어진다. CORBA는 서버라는 용어를 사용하지 않는다. 왜냐하면, CORBA는 객체지향 개념을 바탕으로하여 원격지의 클라이언트가 원격지에 있는 서버를 호출하는 것이 아니라 객체의 메소드를 호출함으로서 서비스가 이루어지기 때문이다. 클라이언트와 구현객체 측은 IDL을 통하여 ORB와 인터페이스를 하게된다.

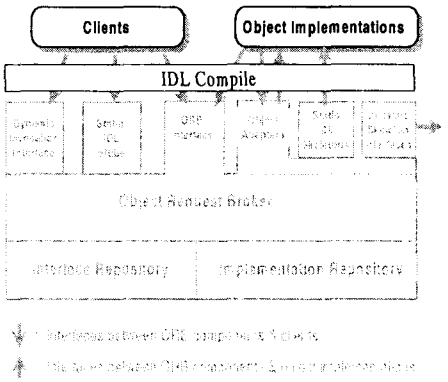

그림 1. CORBA 구조

◆ ORB(Object Request Broker)

ORB는 클라이언트와 원격에 있는 객체구현간에 통신할 수 있도록 하고, 클라이언트의 요청을 서비스하기 위해 구현객체를 찾고, 요청을 받아들일 수 있도록 구현객체를 준비시킨 후, 서버로 요청을 보내고 서버가 수행한 요청의 결과를 클라이언트로 보내는 등의 모든 서비스를 제공할 수 있어야 한다. 그러므로, 클라이언트는 서버의 인터페이스를 구현객체의 위치에 독립적으로 요구할 수 있다.

클라이언트 측 구조는 세 가지 요소로 구성되어 진다;

▶ 동적호출 인터페이스(DII: Dynamic Invocation Interface)

실행 시간동안 호출될 메소드를 발견할 수 있도록 해주는 메커니즘이다. 클라이언트 컴파일시에 정의와 인터페이스를 모르던 구현객체에 메소드 호출을 가능하도록 한다. 이 경우, 호출은 객체 레퍼런스, 오퍼레이션 이름, 파라미터의 데이터와 데이터 타입들의 리스트로 구성된다.

▶ IDL 스터브 인터페이스(IDL Stub Interface)

IDL 스터브는 객체 서비스에 대한 정적 인터페이스를 제공한다. 미리 컴파일된 스터브는 클라이언트가 서버상의 해당 서비스를 호출하는 방법을 정의한다. 클라이언트의 관점에서 스터브는 마치 로컬 호출처럼 행동한다. 즉, 스터브는 원격 서버 객체에 대한 로컬 프록시(Proxy)이다. 서비스들은 IDL을 사용하여 정의되며 클라이언트 및 서버 스터브는 모두 IDL 컴파일러에 의해서 생성된다. 클라이언트는 자신이 서버 상에서 사용하는 각 인터페이스에 대한 IDL스커브를 가져야만 한다. 스터브는 마샬링을 수행하기위한 코드를 포함한다. 이것은 스터브가 오퍼레이션과 자신의 파라미터를 서버에 전송할 수 있는 메시지 형식으로 코드화 및 해독할 수 있다는 것을 의미한다. 스터브는 또한 기반 프로토콜 또는 데이터 마샬링과 같은 이슈들을 고려하지 않고 C, C++, Java, Smalltalk 등과 같은 고급언어로부터 서버상의 메소드를 호출할 수 있게하는 헤더 파일을 포함할 수 있다. 이 때 원격서비스를 얻기 위해서는 프로그램 내에서 단순히 메소드를 호출하면 된다.[2][4][7]

▶ ORB 서비스인터페이스(ORB service Interface)

클라이언트와 서버에 의해 요구되는 대부분의 공통 서비스들을 포함한다. ORB 인터페이스는 ORB core로 직접가고, 모든 ORB에 동일하며 객체의 인터페이스나 객체 어댑터에 독립적이다. 클라이언트 코드에 의해 직접 앱스되어질 수 있는 많은 기능들(즉, 객체에대한 레퍼런스를 검색 등)을 가지며, 구현 객체에 의해 고유되어진다.

구현객체 측에 대한 다른 두가지 요소는 다음과 같다;

▶ IDL 스켈레톤 인터페이스(IDL skeleton Interface)

클라이언트 IDL 스터브 인터페이스에 상응되며, 클라이언트에 의한 하나의 요청에 대한 구현 객체의 메소드를 ORB가 호출하는 인터페이스이다.

▶ 객체 아답터(Object Adapter)

객체 아답터는 ORB의 핵심 통신 서비스 상에 존재하며 서버의 객체를 대신하여 서비스에대한 요청을 수용한다. 즉, ORB에 의해 제공된 서비스들을 구현 객체가 엑세스할 수 있도록 하는 방법으로, 객체 레퍼런스의 생성과 해석, 메소드 호출, 객체 활성화 비활성화, 등록 등의 기능을 갖는다. 객체 아답터는 구현객체를 인스턴스화하고, 요청을 구현객체에 전달하며 이들에게 객체 레퍼런스를 할당하기 위한 런타임 환경을 제공한다. 객체 아답터는 또한 자신이 지원하는 클래스와 런타임 인스턴스를 구현 저장소에 등록한다. CORBA에서는 각각의 ORB가 기본 객체 아답터(BOA)라고 불리는 표준 아답터를 지원해야 한다고 명시한다.

CORBA 표준의 주요부분은 객체 서비스이다. 일부 객체 서비스는 객체의 범주를 넓히기 위해 응용가능한 오퍼레이션의 특정 부분만을 제공하도록 만들어졌다. 예를 들면, 어떤 객체 서비스는 객체를 저장하고 검색하거나, 객체들간의 관계를 관리하거나 다른 객체들에 의해 허가받지않은 접근으로부터 객체를 보호하는 등의 일을 하는 오퍼레이션들만 제공한다. 객체 서비스는 ORB의 간단한 요청관리 기능 위에 객체의 상호작용성과 기능성을 확장시키는 역할을 하는 것이다.[2][3][4][7][9]

2. 객체그룹(Object Group)

본 절에서는 분산 객체들의 관리를 용이하게하기위한 개체들의 논리적인 집합체로서 객체그룹에대한 필요성과 그에 따른 객체그룹의 기능과 구조에 대해 설명한다.

표 1. 로컬객체와 분산객체의 비교

특성 \ 객체	연관된 객체	분산된 객체
통신(Communication)	빠름	느림
실패(Failure)	객체 모두 실패	객체는 독립적으로 실패
동시 접근(Concurrent access)	다중 스레드일 경우만 가능	가능
보안(Secure)	가능	불가

1) 객체 그룹의 필요성

분산 어플리케에션 개발자는 모든 로직이 동일 운영시스템 프로세스에서 실행되는 로컬 프로그램에서 당연하게 생각되어지는 몇 가지 문제들을 처리해야만 한다.

(표 1.)은 동일 프로세스에서 서로 연관된 객체와 프로세스 또는 시스템 영역간에 상호작용하는 객체사이의 기본적인 차이점에 대한 몇가지를 요약해 보여준다.

동일 프로세스에서 객체간의 통신은 다른 머신상의 객체간의 통신보다 최대 10배까지 빠르다. 이 사실은 둘이나 그 이상의 분산객체가 매우 강한 상호작용을 하는 분산 어플리케이션을 설계하는 것을 피해야만 한다는 것을 의미한다. 만약 객체들이 강한 상호작용을 하고있다면 객체들은 상호관계를 가져야 한다. 두 개의 객체가 서로 상호관계를 가지고 있을 때 객체들은 함께 실패한다. 만약 객체들이 실행되고 있는 프로세스가 실패하면 두 객체 모두 실패한다. 객체의 설계자는 만약 객체중의 하나가 사용가능하고 다른 하나가 그렇지 않다면 어플리케이션의 행위에 관여할 필요는 없다. 그러나, 두 객체가 프로세스 경계를 가로질러 분산되어 있다면 객체는 독립적으로 실패한다. 이런 경우 객체의 설계자는 다른 하나의 객체가 실패할 경우 각각의 객체의 행위에 관여해야만 한다. 유사하게 분산된 시스템에서 네트워크는 분할할 수 있고 두 객체는 다른 객체들은 실패했다고 간주하고 실행할 수 있다.

대부분의 로컬 프로그램의 기본 모드는 단일 스레드 제어로 작동된다. 단일 스레드 프로그래밍에서 객체들은 프로그램의 알고리즘에 따라 잘 정의된 순서로 접근되므로 쉽다. 그러나, 분산 어플리케이션에서 다중 스레드 제어는 필요하다. 각각의 분산된 객체는 서로다른 제어 스레드에서 실행된다. 분산된 객체는 여러명의 동시접속 클라리언트를 가질수 있다. 객체의 개발자와 클라이언트의 개발자는 객체에의 동시접근에 대해서 고려해야만 하고 동기화 메커니즘을 사용해야만 한다. 두 객체가 동일 프로세스상에서 서로 연관되어 있을 때, 보안에 대해 고려할 필요는 없다. 그러나, 객체가 다른 머신상에 존재할 때 다른 객체의 식별(Identity)을 인증하기위해 보안 메커니즘을 사용할 필요가 있다.

2) 객체그룹의 개념

앞서 살펴보았던 분산객체컴퓨팅 환경인 TINA에서는 서비스나 어플리케이션 또는 시스템이 다수의 컴퓨팅 노드에 분산되어있는 객체들의 집합으로 구성된 객체그룹이라는 객체를 정의하였다. 커다란 TINA 시스템을 설계할 때, 복잡한 설계와 관리의 어려움을 해결하기위해, 개별의 객체들 대신에 하나의 단위로써 객체들을 집합을 관리하는 객체그룹을 도입한 것이다. 그러나, TINA에서 제시된 객체그룹들은 구조적인 도식과 객체그룹을 이용하여 설계와 관리를 용이하도록 하기위한 필요한 구성요소들의 명세만이 정의되어있다.[1][5][6]

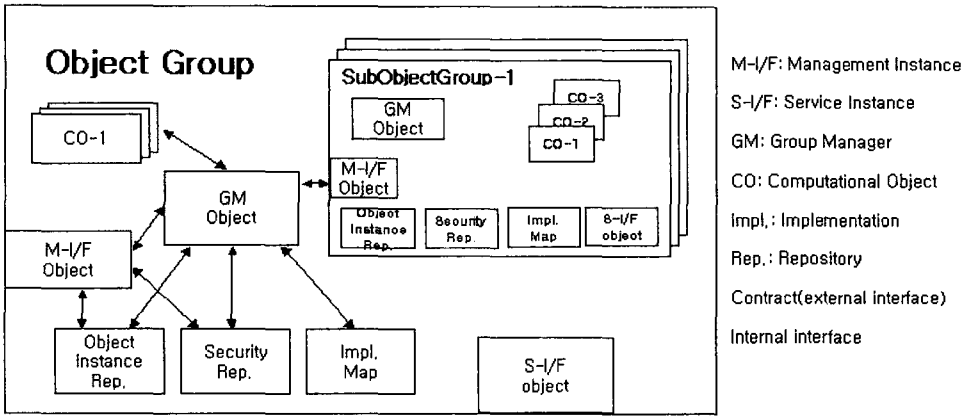

그림 2. 객체그룹 구조

III. 그룹객체 모델의 설계 및 구현

1. 객체그룹의 구조

본 논문에서 제안한 객체그룹은 외부와의 인터페이스를 위해 각각의 관리적 측면의 접근이 가능한 관리 인터페이스객체(M-I/F Object)와 서비스 인터페이스 객체(S-I/F Object)로 나누어 설계하고, 객체그룹을 관리하기위한 그룹관리자 객체(GM Object), 객체들의 정보와 보안을 위해 각각 객체 인스턴스 레포지트리(Object Instance Repository)와 보안 레포지트리(Security Repository), 객체그룹 내에 생성가능한 객체들에 대한 정보를 담고있는 구현 맵(Implemnetation Map)등이 있으며, 실질적인 서비스를 제공하는 객체들이 존재한다. 객체그룹은 또한 서브객체그룹을 내포할 수 있으며, 그 기능과 구성요소들은 상위 객체그룹과 동일하다. (그림 2.)는 본 논문에서 제안한 객체그룹의 구조이다.

2. 그룹객체의 구현

본 장에서는 그룹 관리, 객체의 상태관리, 그룹정보의 감시기능을 포함할 수 있는 그룹과 관련된 역할을 담당할 수 있도록 그룹객체를 구성하는 그룹관리자, 보안객체, 객체 인스턴스 레포지토리에 대한 기능들을 각각 OMG IDL(Interface Definition Language)을 이용하여 정의 설계한다.

그룹관리자는 객체이름과 그룹이름을 그룹생성(group information)정보에 추가하여 그룹객체의 관리적 측면으로 그룹객체 생성자가 특정한 기준에 맞는 그룹개체 내에 객체들을 소속시킬 수 있도록 하는 오퍼레이션과 소속되어있는 객체들을 탈퇴시킬 수 있는 오퍼레이션을 제공한다.

그룹관리자 클래스의 IDL 인터페이스의 주요부분은 아래와 같다.

```
interface GroupManager {
boolean create_subgroup(in Group_Name grp_name);
boolean register(in Object obj);
boolean delete(in Object obj);
boolean deleteAll();
boolean activate(in Object obj);
boolean deactivate(in Object obj);
boolean activateAll(in Object obj);
boolean deactivateAll(in Object obj);
Object getElementObj(in unsigned interfaceId);
};
```

보안 객체는 그룹객체의 외부에서 서비스를 요청한 클라이언트가 서비스 수행객체에 대한 접근권한 여부를 검사하는 기능을 수행한다. 따라서, 그룹객체에 소속되는 서비스객체들의 접근권한 정보를 저장하기 위한 오퍼레이션과 탈퇴하는 서비스객체들의 접근권한 정보들을 삭제하기 위한 오퍼레이션을 제공한다.

보안객체 클래스의 IDL 인터페이스의 주요부분은 아래와 같다.

```
interface Security {
boolean create_ACL(in Object obj,in Object_Permission obj_permission);
boolean del_ACL(in Object obj, in Object_Permission obj_permission);
};
```

객체 인스턴스 레포지토리는 그룹객체의 객체들에 대한 정보를 유지한다. 새로 멤버가 되는 객체정보(object information)를 저장하는 오퍼레이션과 그룹객체에서 탈퇴하는 객체에 대한 정보를 삭제하기 위한 오퍼레이션을 가진다. 이들 오퍼레이션은 그룹관리자의 호출에 의하여 수행된다.

객체인스턴스 레포지토리 클래스의 IDL 인터페이스의 주요부분은 아래와 같다.

```
interface ObjectInformation {
create_obj(in Object obj);
del_obj(in Object obj);
};
```

IV. 결론

아직까지 CORBA 상에서 그룹객체의 구현에 관한 연구는 많이 이루어지고 있지 않은 실정이다. 현재 그룹객체의 구현에 대한 연구로는 CORBA 트레이더를 이용하여 객체와 그룹객체 상호간의 접속방법을 제시한 연구가 있다. 이 연구에서는 그룹객체가 서비스를 트레이더에 등록하고 클라이언트가 이를 이용할 수 있는 방법으로써 그룹객체의 위치 투명성과 동적 접속을 제

공하는등의 장범을 가지고 있다. 반면, 이 방법에서는 클라이언트와 그룹객체의 상호접속이 항상 동적으로 일어나며 그룹객체가 제공하는 서비스를 명시적으로 트레이더에 등록해야 한다.

또한, 클라이언트로부터 그룹객체의 접속 절차가 매우 복잡하며 클라이언트는 트레이더를 통하여 요소객체에 직접 접속함으로써 요소객체가 그룹객체내에 캡슐화되지않고 클라이언트에게 노출되는 문제점이 있다. 또한 동적 접속은 융통성을 증가시키는 장점은 있지만 항상 동적으로 접속하는 것은 시간적으로 비효율적인 방법으로 알려져 있다.

본 논문에서는 CORBA에서 그룹객체를 지원하기위한 그룹객체 모델을 정의하고 CORBA 상에서 그룹객체를 구현하기위한 방법을 제안하였다. 본 논문에서 그룹객체를 지원하는 방식은 IDL을 이용하여 그룹객체를 기술함으로써 클라이언트 객체와 서버 그룹객체들 사이를 정적으로 연결할 있다.

따라서, 대표적인 분산 응용 프레임웍인 CORBA에서 그룹객체 개념을 지원할 수 있게 함으로써 대형 분산 소프트웨어의 개발과 유지.보수시에 그 복잡도를 크게 줄여줄 수 있다.

참고문헌

[1] Tom Handegard, Horoaki Hara, Ajeet Parhar, "Group Managers and ODL", EN_AP.001_1.0_95, TINA-C, version 1.0, 15 Nov. 1995.

[2] "The CORBA Object Group Service", http://lsewww.eplf.ch/OGS/thesis

[3] Silvano Maffeis, "The Object Group Design Pattern", Dept. of Computer Science, Cornell Univ. 1996

[4] OMG, "OMG RFP5 Submission: Trading Object Service", 1996

[5] Nguyen Duy Hoa, "Distrbuted Object Computing with TINA and CORBA", Technical Report Nr. 97/7

[6] Pier Giorgio Bosco and Corrado Mosio, "A Distributed processing Model for Telecommunications Service Management", The Proceedings of DSOM '95, 1995

[7] OMG, "CORBA Services: Common Object Service Specification", 1997

[8] OBJECT MANAGEMENT GROUP. The Common Object Request Broker: Architecture and Specification, 2.0 ed., July 1995.

[9] Distributed Systems Group, Technical University of Vienna, "CORBA Software", http://www.infosys.tuwien.ac.at/Research/Corba/software.html, 1997