CORBA상에서의 그룹객체의 구현에 관한 연구

류 기 열† · 이 정 태†† · 변 광 준†

요 약

분산 환경에서 개발되는 응용 소프트웨어의 규모가 커짐에 따라, 생성되는 객체의 수가 기하 급수적으로 증가하게 되었고 객체간의 인터페이스 또한 매우 복잡하게 되었다. TINA에서 제안된 그룹객체 개념은 관련 있는 객체들을 하나의 그룹으로 묶어 캡슐화하여 그룹객체를 만들고, 그룹객체 내의 인터페이스를 체계적으로 제어하여 객체들 사이의 인터페이스 복잡도를 줄여 주는 일종의 고급 추상화 개념이다.

본 논문에서는 현재 분산 응용소프트웨어의 개발을 위한 표준 미들웨어로 정착되어 가는 CORBA상에서 이러한 그룹객체 개념을 지원하기 위한 구현 모델을 제안한다. 이를 위해 TINA에서 제안한 그룹객체 개념을 수정하여 간략화하고, 이러한 그룹객체를 지원하기 위해 기존의 CORBA ORB 구조를 그대로 유지하면서 자연스럽게 확장한다. 또한 그룹객체를 CORBA에서와 같이 언어-독립적으로 기술하기 위하여 CORBA의 IDL을 확장한다. 마지막으로 이러한 구현방법을 검증하기 위해 하나의 상용 CORBA 제품(Java 언어 지원)을 선택하여 제안한 구현모델을 적용해 본다.

An Implementation of Group Objects in CORBA

Ki-Yeol Ryu† · Jung-Tae Lee†† · Kwang-June Byeon†

ABSTRACT

As an application software in distributed computing environment becomes large, the number of objects to be created increases drastically and the interfaces among them become very complex. The concept of group object resolves this problem to some extent by grouping a set of related objects, encapsulating them, and controlling their interfaces systematically.

In this paper, we propose an implementation model of the group object concept in CORBA, which is a standard middleware for developing distributed application software on heterogeneous networks. To support group objects, we extends CORBA ORB without modifying its internal structure for the compatibility with existing CORBA applications. And we devise an interface definition language by extending CORBA IDL to describe group objects in a language -independent style, which is one of the most important characteristics in CORBA. Finally, we experiment the implementation model on a CORBA ORB compliant product which supports the Java language.

1. 서 론

현재의 분산 컴퓨팅 환경은 기존의 대형 서버 중심의 클라이언트/서버 컴퓨팅에서 벗어나 개방형 통신망의 영향으로 다양한 플랫폼에 적합한 분산 컴퓨팅 환경으로 바뀌어 가고 있다[1, 8]. 이러한 컴퓨팅 환경은 분산 응용 개발 환경을 변화시키고 있으며 이의 일환으로 분산객체(distributed objects) 기술에 대한 연구가 활발하다. 대표적인 분산객체 기술로 개발되고 있는 CORBA (Common Object Request Broker Archi-

※ 이 논문은 한국과학재단 '98핵심전문연구(과제번호 : 981-0923-393-1)의 지원에 의하여 수행된 결과임.
† 종신회원 : 아주대학교 정보 · 컴퓨터공학부 교수
†† 정 회 원 : 아주대학교 정보 · 컴퓨터공학부 교수
논문접수 : 1998년 8월 26일, 심사완료 : 1998년 11월 4일

tecture)는 개방형 통신망에 접속된 다양한 플랫폼에서 독립적으로 개발된 분산객체들을 접속하여 하나의 분산 응용 프로그램을 개발하기 위한 미들웨어 기술이다[6, 7]. CORBA에서는 플랫폼-독립적인 객체를 기술할 수 있도록 IDL (Interface Definition Language)이라는 언어를 제공하며 클라이언트/서버 사이의 서비스 연결을 위한 ORB (Object Request Broker)를 제공한다. 또한 다양한 분산 응용에 필요한 기본 서비스인 Common Object Services와 응용개발 프레임웍인 Common Facilities를 정의하고 있다.

다른 한편으로, 분산응용 소프트웨어의 규모가 확대되고 분산객체들 간의 인터페이스가 복잡해짐에 따라 객체들을 효과적으로 관리하고 객체들 간의 복잡한 인터페이스를 체계적으로 제어하기 위한 기술의 필요성이 증가하고 있다. TINA(Telecommunication Information Networking Architecture)[1]에서 제안한 그룹객체(group objects) 개념[3, 4]은 관련이 깊은 객체들을 하나의 그룹으로 캡슐화시켜 그룹 내의 정보를 은닉시키는 일종의 추상화(abstraction)개념이다. 그룹객체는 그룹 내의 객체(요소객체)들을 관리하고, 클라이언트 객체로부터 서비스 요청(메시지 전달)을 받으면 적절한 요소객체를 이용하여 서비스를 제공하는 역할을 한다. 그룹객체는 요소객체들의 모든 인터페이스를 외부로 노출시키지 않고 적절하게 제어함으로써 그룹객체의 정보를 은닉시킨다. 이렇게 함으로써 객체들 간의 인터페이스의 복잡도가 줄어들게 되고, 따라서 프로그램의 복잡도가 줄어 프로그램의 개발 및 유지보수의 비용을 감소시킨다. 또한 그룹객체는 그룹객체 내에 또 다른 그룹객체가 포함될 수 있는 계층적인 구조를 가짐으로써 분산 응용의 복잡한 객체 구조를 모델링하는 데 유용하다[9].

본 논문에서는 TINA에서 제안한 그룹객체의 개념을 CORBA 상에서 구현할 수 있는 방법을 제시하고자 한다. 이를 위해 우선 TINA의 그룹객체 개념을 도입하여 그룹객체 모델을 정립하고, 그룹객체를 언어-독립적으로 기술할 수 있도록, TINA ODL(Object Definition Language)[10]의 그룹객체 기술 방법을 이용하고 CORBA IDL을 확장하여 Group IDL(GIDL)을 설계한다. 그리고 클라이언트 객체로부터 서버 그룹객체에 대한 서비스 요청을 적절한 그룹객체의 서비스로 중개하기 위한 방안으로 CORBA ORB를 확장하여 Group ORB(GORB)를 설계한다. GORB는 CORBA ORB와의 호환성을 위해 CORBA의 ORB를 변경하지 않고 그대로 이용하여 확장한다.

본 논문에서 제안한 구현모델의 타당성을 검증하기 위해 CORBA 2.1 ORB[6, 7]를 구현한 상용 제품인 Visibroker 3.1(Java 언어를 지원하는 모델)[12]을 선택하여 구현모델을 적용하여 본다. 이를 위하여 우선 언어 독립적인 GIDL로 기술된 그룹객체의 각 요소가 특정 언어(본 논문에서는 Java언어)의 코드로 어떻게 매핑되는 지에 관한 규칙을 정의하고, 실제로 매핑을 수행하는 프리컴파일러를 정의한다. Visibroker 3.1[12]이 제공하는 요소와 프리컴파일러에 의해 생성되는 요소들에 의해 GORB의 각 요소들이 어떻게 구현될 수 있는 지를 설명한다. 마지막으로, GORB를 이용하여 그룹객체를 프로그래밍할 수 있는 프로그래밍 모델에 대하여 설명한다.

본 논문의 구성은 다음과 같다. 2절에서는 그룹객체 모델을 정의하고, 3절에서는 본 논문에서 제안한 CORBA 상에서의 그룹객체의 구현모델인 GORB와 GIDL의 설계에 대하여 설명한다. 4절에서는 상용 CORBA 제품에 본 논문에서 제안한 구현모델을 실험적으로 적용한 후 평가한다. 아울러 그룹객체를 프로그래밍 하기 위한 모델을 보여준다. 5절에서는 결론을 맺고 앞으로의 연구과제에 대하여 언급한다.

2. 그룹객체

2.1 그룹객체 모델

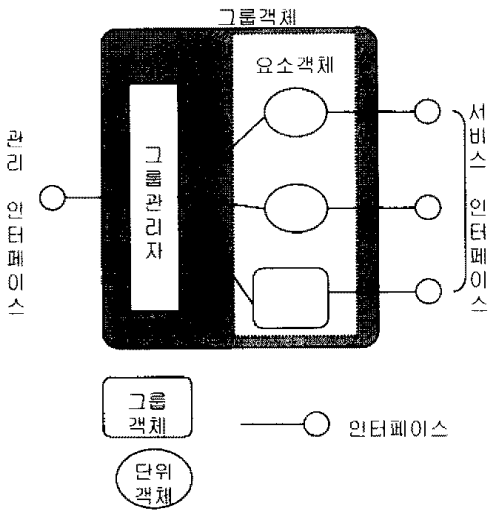
본 논문에서 구현하고자 하는 그룹객체(group object)모델은 TINA에서 정의한 객체그룹(object group)의 모델을 CORBA 상에서 구현하기 위하여 간략화하고 수정한 모델이다[1]. 그룹객체란 하나 이상의 객체들을 캡슐화(encapsulation)하여 기술하고 관리하는 소프트웨어 단위이다. 그룹객체는 그 자신이 또 하나의 객체이면서 그 그룹객체가 속한 요소객체(element object)들을 관리하고 외부로 유출되는 요소객체들의 인터페이스를 제어한다. 요소객체란 그룹에 포함된 단위객체(unit object)[2]나 그룹객체를 말한다. 그룹객체의 주요 구성 요소로는 요소객체들과 그룹관리자(group manager), 그리고 인터페이스가 있다. 그림 1은 그룹객체의

1) 본 논문에서는 객체의 그룹도 하나의 일반객체와 동일하다는 의미에서 이름을 '그룹객체'라고 부르기로 한다.

2) 단위객체(unit object)란 그룹이 아닌 객체를 말한다.

전형적인 구조를 보여주고 있다.

그룹객체의 인터페이스는 그룹객체가 외부의 객체들에 제공하는 서비스를 나타내는 것으로, 관리 인터페이스(management interface)와 서비스 인터페이스(service interface) 두 가지로 나뉘어 진다. 관리 인터페이스는 그룹객체 내의 그룹관리자가 제공하는 관리기능을 위한 인터페이스이며, 서비스 인터페이스는 그룹객체가 클라이언트 객체들에 응용 고유의 서비스를 제공하기 위한 인터페이스이다. 각각의 서비스 인터페이스는 그룹객체가 직접 제공하는 것이 아니라 그룹객체 내의 하나의 요소객체에 의해 제공되는 것이다. 그룹객체의 서비스 인터페이스를 제외한 요소객체의 다른 인터페이스는 외부로 노출되지 않는다. 그룹객체의 하나의 인터페이스는 그룹객체의 한 가지 역할(role)에 해당한다. 즉, 외부에서 볼 때 그룹객체는 여러 가지 역할을 수행할 수 있다. 가령, 관리 인터페이스는 그룹객체의 관리자 역할을 의미한다. 이 역할의 개념은 그룹객체를 하나의 단위로 사용하지만 사용자에 따라 다른 객체처럼 사용할 수 있다는 것을 의미한다.



(그림 1) 그룹객체의 구조
(Fig. 1) Structure of Group Object

그룹관리자는 그룹 내의 모든 요소객체들을 관리하고 인터페이스의 제어기능을 맡고 있다. 이 관리기능은 그룹객체를 사용하는 응용 프로그램(controlling application)에 하나의 인터페이스(관리 인터페이스)를 제공한다. 대표적인 관리기능으로는 요소객체의 생성(creation) 및 요소객체의 삭제(deletion), 요소객체의 활성화(activation) 및 비활성화(deactivation), 그리고 객체들의 인터페이스 제어 등의 기능이 있다. 요소객체 생성은 한 객체를 생성하여 그룹객체의 요소객체로 등록하는 기능이고, 요소객체 삭제는 그룹으로부터 요소객체를 삭제하는 기능이다. 요소객체의 활성화 기능은 비활성화 되어 있는 요소객체의 서비스를 처리할 수 있는 활성화 상태로 만들어주며, 비활성화는 요소객체가 서비스를 처리할 수 없는 상태로 만든다. 그룹객체에 등록된 모든 객체를 동시에 활성화/비활성화시키는 기능으로서 그룹객체의 활성화/비활성화 기능을 제공할 수 있다. 객체의 인터페이스 제어기능은 그룹객체의 서비스 인터페이스를 제외한 요소객체의 인터페이스는 외부에 감추고 서비스 인터페이스를 외부로 노출시켜 그룹객체에 대한 서비스 요청을 처리할 수 있도록 하는 기능이다.

TINA의 그룹관리자는 위에서 언급한 관리기능 외에도 응용 프로그램마다 고유한 그룹 관리기능을 동시에 가지고 있다. 이를 위해서 TINA에서는 그룹 내 특정 요소객체를 그룹관리자로 지정한다. 그러나 본 논문에서는 그룹객체의 개념을 지원하기 위해 하부구조가 제공해야 하는 관리기능과 응용 프로그램의 고유한 관리기능은 분명하게 구분해야 한다고 판단하여, 그룹관리자는 단지 앞에서 언급한 관리기능만을 제공하고, 응용 프로그램의 고유한 관리기능은 그룹객체가 제공하는 하나의 서비스 인터페이스로 간주한다.

그룹객체를 다루는 연산으로 그룹객체 자체의 생성과 삭제기능이 있다. 본 논문에서는 그룹객체 또한 하나의 CORBA객체로 간주하기 때문에 다른 객체를 생성하는 방식과 거의 같다. 그러나 그룹객체를 생성할 때 그룹 내의 초기 요소객체들의 배치(configuration)를 할 수 있어야 한다는 점에서는 다르다. 그룹객체를 삭제하는 기능 역시 CORBA의 객체를 삭제하는 방법과 같으나 하나의 그룹객체를 삭제할 경우 그룹객체 내의 모든 요소객체를 동시에 삭제할 수 있어야 한다.

2.2 기존의 개념과의 비교

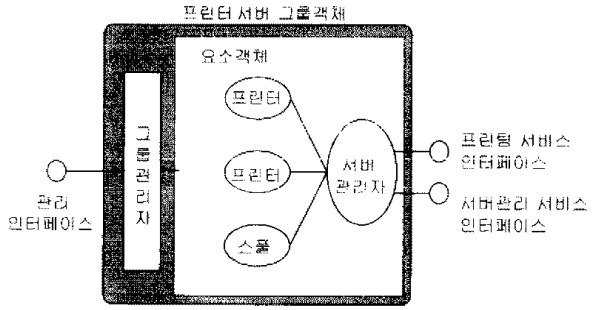
그룹객체는 객체들을 조직화하고 인터페이스를 체계적으로 제어하여 소프트웨어의 복잡도를 줄인다는 점에서 기존의 객체지향 언어의 복합객체(composite object)나 Java 언어의 패키지(package)[2], 컴포넌트(component)[5, 10] 등의 개념과 매우 유사하다. 복합객체와의 차이점은 복합객체는 요소객체의 관리와 메

기서의 요소객체로의 전달 및 결과의 반환 등 모든 일은 자기 프로그래머에 의해 기술되어야 한다. 그러나 그룹객체는 그룹객체를 지원하는 시스템에서 이러한 일들을 자동으로 처리해 준다. 그리고 패키지 개념은 클래스 또는 수준에서 정적으로 객체들을 조직화하는 개념으로 동적인 객체 개념이 아니다. 그러나 가시성을 제어할 수 있으며 캡슐화 기능이라는 점에서 유사하다. 컴포넌트의 경우에는 아직 통일된 정의는 없으나, 대부분 관련 있는 여러 객체나 클래스의 집합으로 정의되고, 외부로 노출되는 인터페이스를 명확하게 정의한다는 점에서 그룹객체와 매우 유사하다. 그룹객체 모델도 일종의 컴포넌트 모델로 간주할 수 있다. 대표적인 컴포넌트 모델의 한 가지인 COM/DCOM 모델[10]의 컴포넌트와 비교해 보면 COM 객체는 객체들을 그룹핑하는 관점에서는 복합객체와 동일하며, 하나의 COM객체가 여러 가지의 인터페이스를 제공하는 점은 그룹객체의 그것과 동일하다. 위에서 언급한 이러한 개념들은 현재 언어나 시스템에 종속적인 경우가 대부분이다. 그러나 CORBA상에서의 그룹객체의 개념은 IDL과 같은 플랫폼-독립적인 인터페이스 언어로 기술하여 실제 구현 언어와는 분리함으로써 다양한 플랫폼을 이용하여 분산 응용시스템을 개발할 수 있어야 한다.

2.3 예제

일반적으로 복잡한 시스템은 대부분 계층적인 구조를 가지고 있다. 즉, 하나의 시스템은 부 시스템(sub-system)들로 이루어져 있고 그 부 시스템은 다시 더 작은 부 시스템으로 이루어져 있다. 하나의 시스템에 의해 제공되는 서비스는 내부 부 시스템들이 제공하는 일부 서비스들에 의하여 구현되며 대부분의 내부 시스템들의 서비스는 외부에 감추어진다. 이러한 계층 구조는 그룹객체의 개념에 의해 쉽게 표현될 수 있다.

프린터 서버 시스템을 예로 들어보자. 프린터 서버 시스템은 하나의 서버 관리자(컴퓨터)와 스풀(spool), 그리고 여러 개의 프린터로 구성되어 있다. 서버 관리자는 클라이언트가 출력 작업에 대한 서비스나 출력제어(출력 작업의 삭제나 상태 정보를 보는 등) 서비스를 제공하며 또한 시스템 관리자를 위한 관리(프린터의 추가나 스풀의 공간 확장 등) 서비스를 제공한다. 그러나 프린터나 스풀의 서비스는 서버 시스템 외부로 직접적으로 노출되지 않는다. 이러한 시스템의 구조를 그림 2와 같은 하나의 그룹객체로 쉽게 표현할 수 있다.



(그림 2) 프린터 서버시스템을 위한 그룹객체 구조
(Fig. 2) A Group Object for the Printer Server System

프린터 서버를 위한 그룹객체에서 서버 관리자가 제공하는 두 가지 서비스는 외부 서비스로 제공되며 프린터나 스풀에 제공하는 서비스는 내부 서비스로 처리된다. 프린터 서버 시스템의 내부 구조는 외부에 보이지 않고 클라이언트는 단지 서버 관리자의 외부 서비스를 이용함으로써 프린터 서비스를 제공받는다. 또한 프린터 서버 시스템은 클라이언트에게는 노출되지 않고 시스템의 배치 (configuration)를 동적으로 변경할 수 있다. 가령, 동적으로 프린터를 시스템에 추가하거나 삭제할 수 있다. 또한 이러한 서버 시스템은 각각의 요소들의 단위로 취급(생성이나 삭제, 또는 트레이더에의 등록 등)되기보다는 하나의 단위로 취급되는 것이 보다 바람직하다. 그룹객체로서의 서버 시스템은 다른 객체와 마찬가지로 하나의 처리 단위(first-class object)가 된다. 그룹객체의 응용성은 본 논문의 주제가 아니므로, 보다 복잡한 예제를 보기 위해서는 [3]을 참조하기 바란다.

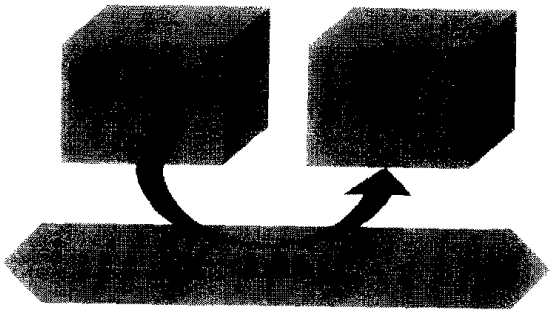
3. 그룹객체의 구현모델: GORB

본 절에서는 CORBA 2.1 ORB 상에서 그룹객체를 구현할 수 있는 한 가지 구현모델을 제안한다. 구현모델에서 채택한 방법은 그룹객체를 지원할 수 있도록 ORB를 확장한 GORB(Group Object Request Broker)이다. GORB를 설계하면서 고려한 사항은 기존의 CORBA 2.1 호환 프로그램을 그대로 사용할 수 있도록 ORB Core를 변경하지 않고 그대로 유지하고 그 위에서 확장하였다는 점이다. 또 하나의 고려사항은 그룹객체를 기술할 때 특정 언어와 무관하게 기술할 수 있도록 하여 특정 사용 언어의 확장을 피하도록 한 것이다. 본 논문에서는 그룹객체를 언어-독립적으로

기술하기 위하여 CORBA IDL을 확장한 언어인 GIDL을 설계하였다.

3.1 GORB의 설계

GORB는 클라이언트의 객체와 서버의 단위객체 및 그룹객체들 사이에서 중개역할을 한다. 즉, CORBA ORB와 유사하게 클라이언트 객체가 요청한 서비스를 서버의 해당 객체에게 전달하고, 그 결과를 클라이언트 객체에게 전달하는 역할을 한다. ORB와의 근본적인 차이점은 GORB는 서비스 요청이 그룹객체에게 발생했을 때 그룹객체 내의 적절한 요소객체에게 중개할 수 있다는 점이다. 그림 3은 GORB를 이용한 클라이언트/서버 모델을 보여주고 있다.

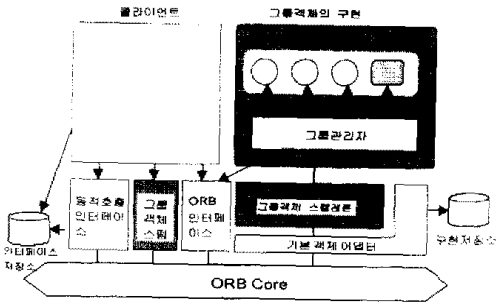


(그림 3) GORB를 이용한 클라이언트/서버모델
(Fig. 3) Client/Server Model using GORB

(1) GORB의 구조

GORB는 CORBA ORB Core를 변경 없이 사용하면서 CORBA ORB를 확장한다. CORBA ORB가 서버의 객체에 대한 환경을 제공하는 것처럼 GORB는 단위객체 뿐만 아니라 그룹객체들에게 ORB 서비스를 제공한다. 그림 4는 본 논문에서 제안한 GORB의 전체적인 구조를 보여준다. 기본적인 구조는 CORBA 2.1 ORB[7]를 따르고 있다. 그림에서는 서버 측에서 제공하는 객체가 단위객체인 경우는 CORBA ORB와 동일하므로 생략하고, 그룹객체의 경우만을 보여주고 있다.

구성 요소를 보면 클라이언트 쪽에는 동적 호출 인터페이스(DII, Dynamic Invocation Interface), 클라이언트 스텁(Client Stub), 인터페이스 저장소(Interface Repository) 등이 있다. CORBA ORB와 다른 점은 스텁 코드에는 서버의 단위객체로의 인터페이스를 위한 스텁 코드 외에 그룹객체로의 중개를 위한 그룹객체 스텁이 있다는 점이다. 서버 측의 기본 객체 어댑터(BOA, Basic Object Adaptor)나 구현 저장소 (Implementation Repository)는 CORBA ORB와 동일하며, 구현 스켈레톤 부분에 단위객체를 위한 스켈레톤 코드 외에 각 그룹객체를 위한 그룹객체 스켈레톤(Group Object Skeleton)과 그룹객체의 관리기능을 구현한 그룹관리자가 추가되었다. 다음 절들에서 각각에 대하여 좀더 자세하게 알아보기로 한다.



(그림 4) GORB의 구조
(Fig. 4) Architecture of GORB

(2) 그룹 관리자

그룹 관리자는 하나의 그룹객체 마다 반드시 하나씩 존재하며, 그룹 내의 요소객체들과 그룹객체 외부에 관리 인터페이스(management interface)를 제공한다. 그룹객체 관리자는 그룹객체의 요소객체를 생성하여 등록하고 삭제하는 기능을 제공하고 또한 객체의 활성화 및 비활성화 기능을 제공한다. 또한 이 그룹객체 관리자는 그룹객체 스켈레톤과 함께 그룹객체로 전달되는 클라이언트로부터의 서비스 요청을 적절한 요소객체로 전달하는 역할도 수행한다. 이 때 전달된 요청이 그룹객체의 서비스 인터페이스에 속하는 지를 판단하여 그렇지 않다면 적절한 예외 상황을 발생시킨다. 이러한 그룹관리자 기능은 ORB의 BOA가 서버객체를 등록하여 관리하고 클라이언트로부터 서비스 요청이 오면 활성화되어 있지 않다면 해당 객체를 활성화시키고 서비스 요청을 그 객체에 전달하는 기능과 유사하다.

(3) 스켈레톤 및 스텁 코드

클라이언트 스텁 코드는 서버 객체에 대한 클라이언트의 정적 인터페이스를 제공한다. 즉, 클라이언트 객체가 서버 측의 단위객체나 그룹객체에게 서비스를 요청할 경우, 클라이언트는 이 스텁 코드를 통해 서비스를 요청한다. 이 요청은 ORB를 통하여 서버 측에 전달되고 다시 서버의 구현스켈레톤을 통해서 서버 객

세에 전달된다. 클라이언트 측의 관점으로 보면, 서버 객체에 대한 요청은 이 클라이언트 스텁에 대한 지역적인 메쏘드 호출 (local method call)과 같다. 서버 객체의 서비스는 GIDL이라는 언어로 기술되는데 스텁이나 스켈레톤 코드는 이 GIDL로터 GIDL 컴파일러에 의해 자동 생성된다.

이 스텁이나 스켈레톤 코드에는 앞에서 설명한 바와 같이, 단위객체를 위한 스텁이나 스켈레톤 외에도 각 그룹객체 마다 하나씩의 그룹객체 스텁과 그룹객체 스켈레톤이 포함되어 있다. 그러나 그룹객체도 하나의 객체에 불과하고, 또한 모두 CORBA ORB Core 인터페이스를 이용하여 같은 방식으로 구현된다는 점에서 단위객체를 위한 스텁 코드와 별로 차이가 없다. 단위객체와의 차이점은 스텁코드 속에 GIDL로 기술된, 그룹객체의 정의부분에 명시된 그룹객체의 서비스 인터페이스들에 대하여 적절한 정적 인터페이스를 만든다는 점과, 그룹객체 스켈레톤에 그룹관리자를 이용하여 해당 요소 객체로 접속할 수 있는 인터페이스 코드를 생성한다는 점이다.

클라이언트로부터 그룹객체로의 서비스 요청과 그 응답에 대한 절차를 보면 다음과 같다. 먼저 클라이언트는 그룹객체를 위한 스텁 코드에 서비스를 요청한다. 스텁 코드는 서비스 요청(method call)을 표준 문자열로 변환(marshaling)하여 ORB Core를 통하여 서버 측의 BOA에 보낸다. 다시 BOA는 이 요청을 그룹객체 스켈레톤 코드를 통하여 해당 그룹객체에 보낸다. 요청을 받은 스켈레톤 코드는 그룹관리자에게 이 서비스요청이 어떤 요소객체에 전달된 것인지를 판별하도록 요청한다. 그룹관리자는 메시지를 위한 요소객체를 판별하고 그 객체가 활성화되었는 지를 조사하여 그 결과를 스켈레톤 코드에 전달한다. 요소객체가 결정되면 스켈레톤 코드는 표준 문자열로 변환된 메쏘드 호출을 다시 메시지로 복원(unmarshaling)하여 해당 요소객체에게 전달한다. 요소객체의 수행결과는 메시지가 전달된 반대의 경로를 따라 클라이언트에 전달된다.

3.2 그룹객체를 위한 인터페이스 정의 언어(GIDL)

(1) GIDL의 정의

앞에서도 언급했듯이 그룹객체의 개념을 플랫폼-독립적으로 지원하기 위해서는 OMG IDL과 같은 언어 독립적인 언어로 기술되어야 한다. 그러나 OMG IDL은 그룹객체 개념을 지원하고 있지 않고 있다. 따라서 본 논문에서는 그룹객체를 정의할 수 있도록 OMG IDL을 확장하여 GIDL(Group Interface Definition Language)이라는 언어를 정의하였다. OMG IDL과의 호환을 위해 OMG IDL을 변경하지 않고 사용하면서 TINA ODL[8]의 그룹객체 정의 부분을 도입하여 확장하였다. TINA ODL은 OMG IDL을 포함하는 언어로서, IDL에서 기술하는 객체를 위한 연산 인터페이스(operational interface)의 정의나 자료형, 예외(exception)에 대한 정의 외에도 다중 인터페이스 객체(multi-interface object)나 그룹객체에 대한 템플리트를 정의할 수 있으며, 이외에도 스트림 인터페이스(stream interface), 인터페이스나 객체에 대한 행위(behavior)나 QoS 등도 기술할 수 있다[2, 7]. 본 논문에서 TINA ODL을 그룹객체를 위한 인터페이스 기술 언어로 사용하지 않은 이유는, TINA ODL이 본 연구에서 추구한 분산 프로그램의 고수준 추상화기법으로서의 기능보다 훨씬 더 복잡하고 많은 기능을 포함하고 있기 때문이다. GIDL이 추후에 확장되어야 할 필요가 있다면 TINA ODL과 같이 확장하는 것은 그리 어렵지 않다고 본다.

본 논문에서는 정의한 GIDL은 크게 네 가지 구조적인 요소를 포함한다. 그 중 두 가지는 모듈(module)과 인터페이스 정의 부분으로 OMG IDL에서 이미 지원되는 부분이고 나머지 둘은 객체정의 및 그룹객체 정의 부분으로 TINA ODL로부터 도입한 것이다. GIDL에 객체의 구조가 정의되는 이유는 그룹객체를 도입함에 따라 그룹객체에 포함되는 요소객체를 정의하기 위해서이다. 그러나 객체의 정의 부분에서 실제 객체의 구현(code)이 포함되는 것이 아니고 그 객체가 제공하는 인터페이스만을 기술한다. GIDL의 객체의 정의 부분이 OMG IDL의 인터페이스 정의와 다른 점은 OMG IDL에서의 인터페이스는 반드시 하나의 클래스로 매핑되지 않을 수도 있다. 가령, 구현 객체에서 여러 개의 인터페이스를 지원하는 객체를 정의할 수 있다. 그러나 GIDL에서 정의되는 객체는 반드시 하나의 구현객체의 정의(클래스)로 매핑되어야 한다. 구현 언어와의 매핑 문제는 다음 절에서 상세하게 설명하기로 한다.

GIDL의 그룹객체 정의가 TINA ODL의 그룹객체 정의와 다른 점은 TINA ODL에서는 그룹객체 정의에서 그 그룹을 위한 그룹관리자를 지정하는데 반해, GIDL에서는 그룹관리자를 지정하지 않는다는 점이다. 그 이유는 본 논문에서는 GORB가 각 그룹객체 마다

하나의 그룹관리자를 제공하기 때문이다. 그룹객체 마다 그룹관리자를 제공하는 방법은 구현의 문제이므로 4절에서 상세하게 설명하기로 한다.

GIDL에 대한 개략적인 구문구조는 다음과 같다.

1) 모듈의 정의

```
module <identifier>{
        group <group_identifier>{ ... };
        object <object_identifier>{ ... };
        interface <interface_identifier>{ ... };
};
```

모듈의 정의는 세 가지 요소 즉, 그룹객체, 객체, 및 인터페이스의 정의부분으로 이루어진다. 각각에 대하여는 아래에서 설명한다.

2) 그룹객체의 정의

```
group <group_identifier>{
  components objectType1, objectType2, ... ;
  contracts   interfaceType1, interfaceType2, ... ;
};
```

그룹객체의 정의부분은 크게 두 가지, 즉 components 부분과 contracts 부분으로 나누어진다. components 부분은 그룹객체 내에 있는 요소객체의 유형을 기술하고, contracts 부분은 서비스 인터페이스들을 기술한다. contracts 부분에 기술된 서비스 인터페이스는 components 부분에 기술된 요소객체가 제공하는 (supports) 인터페이스 중에서 기술되어야 한다.

3) 객체의 정의

```
object <object_identifier>{
      supports interfaceType1, interfaceType2, ... ;
};
```

supports 부분에는 객체가 제공하는 서비스들을 나타내는데, 이들 각 서비스는 하나의 인터페이스에 해당한다.

4) 인터페이스의 정의

```
interface <interface_identifier>{
      <operation_type> <operation_identifier>( ... );
      <dataType> dataType1, dataType2, ... ;
};
```

인터페이스의 정의부분은 OMG IDL과 동일하다. 인터페이스에는 자료형이나 연산(operation 또는 method)이 기술된다.

(2) GIDL 예제 프로그램

이 절에서는 2.2절에서 설명한 프린터 서버를 그룹객체로 구현하기 위한 GIDL의 예를 보여주고 있다.

```
module PrinterServerModule{
  // 그룹객체
  group PrinterServer {
     components PrinterManager, Printer, Spool;
     contracts MangementService, PrintingService;
  };
  //요소객체
     object PrinterManager {
     supports ManagementService,
     PrintingService;
  };
  object Printer { supports PrinterInterface; }
  object Spool  { supports SpoolInterface; }
  // 인터페이스
  interface ManagementService {
        void addOnePrinter(String printer);
        void removeOnePrinter(String printer);
  };
  interface PrintingService {
        void printFile(String file);
        void showStatus(String printer);
        void removeOneEntry(String file);
        void removeAllEntries();
  };
    // 이하 생략
};
```

위 프린터 서버를 위한 GIDL 정의에서 PrinterServer 라는 그룹객체의 한 그룹객체는 세 가지의 클래스형 즉, PrinterManager, Printer, Spool 클래스의 객체를 요소객체로 가질 수 있다. 일반적으로 PrinterServer 그룹객체는 하나의 PrinterManager 객체와 여러 개의 Printer 객체 그리고 하나의 Spool 객체를 가질 것이다. PrinterServer 그룹객체는 두 개의 인터페이스 즉, ManagementService와 PrintingService를 외부로 공개하고 Printer 객체나 Spool 객체가 제공하는 인터페이스들은 외부에 숨긴다. 즉, 외부에서는 개개의 Printer나 Spool 객체에는 직접 접근할 수 없다.

4. GORB의 구현

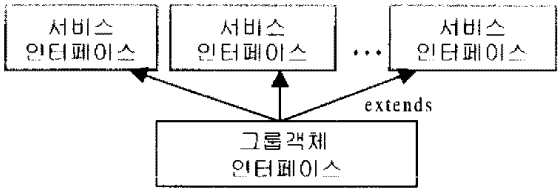
이 절에서는 앞 절에서 제안한 GORB를 CORBA 2.1 제품인 Visibroker 3.1(Java 언어 지원)에 적용하여 실험적으로 구현한다. GORB는 ORB의 메시지 요청 중개방식을 거의 그대로 따르고 있기 때문에 CORBA 구조를 따르고 있는 다른 상용 제품에도 쉽게 적용될 수 있다. 우선, GORB를 구현하기 위해서는 GIDL을 특정 언어로 매핑하여야 한다. 본 논문에서는 매핑하고자 하는 언어로 현재 대표적인 객체지향 언어의 하나인 Java를 선택하였다. 그리고 그룹객체를 위한 클라이언트 스텁 코드나 서버 스켈레톤 코드의 구조를 설계하여야 한다. 이러한 GIDL의 Java 언어 매핑규칙과 스텁/스켈레톤 코드의 구조가 결정되면 GIDL 프리컴파일러(precompiler)를 구현하여야 한다. 또한 그룹관리자를 어떻게 구현할 지를 결정하여야 한다. 마지막으로 응용프로그램에서 그룹객체를 프로그래밍하는 방법과 서버와 클라이언트의 프로그래밍 하는 방법을 결정하여야 한다.

4.1 GIDL로부터 Java 언어로의 매핑

GORB가 Java언어를 지원하기 위해서는 3절에서 설명한 GIDL에서 정의되는 다양한 내용이 Java언어의 구문구조로 매핑되어야 한다. GIDL에서 OMG IDL에 해당하는 부분에 대하여는 CORBA 2.1 명세서[2]에서 정의되어 있는 매핑을 그대로 따른다. 가령, 하나의 GIDL 인터페이스는 하나의 Java 인터페이스에 매핑된다.

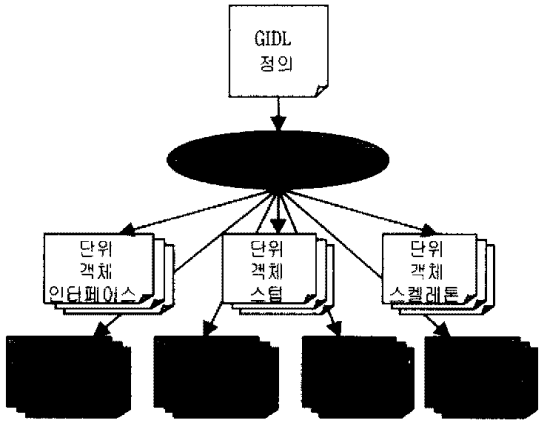
OMG IDL에 비해 GIDL에 추가된 부분으로 그룹객체와 요소객체가 있다. 이들은 모두 Java언어의 인터페이스 구문구조로 매핑된다. 즉, 하나의 그룹객체나 요소객체는 하나의 Java 인터페이스로 매핑된다. 그룹객체 정의로부터 생성되는 Java 인터페이스는 그림 5와 같이 그 그룹객체에 정의되어 있는 contracts 부분의 모든 서비스 인터페이스를 상속(extends)하여 정의된다. 이 Java 인터페이스는 클라이언트와 서버에서 그룹객체를 전체 역할로서 선언할 때 사용된다. 각각의 서비스 인터페이스는 그룹객체를 하나의 역할로 사용하고자 할 때 선언하여 사용한다.

요소객체에 대해서도 그룹객체와 유사한 방법으로 Java 인터페이스로 매핑된다. 하나의 요소객체가 여러 개의 GIDL 인터페이스를 지원할 수 있으므로 요소객체를 위한 Java 인터페이스는 요소객체가 지원하는 이



(그림 5) 그룹객체를 위한 Java 인터페이스
(Fig. 5) Java Interface for a Group Object

들 GIDL 인터페이스에 대응하는 Java 인터페이스들을 모두 상속하여 생성한다. 그리고 서버 측에서 요소객체를 구현할 때, 이 Java 인터페이스를 구현(implements)해야 한다.



(그림 6) GIDL 프리컴파일러의 입출력
(Fig. 6) GIDL Precompiler

GIDL 프리컴파일러는 이러한 매핑규칙 외에도 3절에서 설명한 다양한 스텁 및 스켈레톤 코드들을 생성해야 한다. GIDL 프리컴파일러가 생성하는 코드는 위에서 언급한 Java 매핑에서 정의한 다양한 Java 인터페이스 코드 즉, 요소객체를 위한 Java 인터페이스, 그룹객체를 위한 Java 인터페이스, GIDL 인터페이스를 위한 Java 인터페이스 외에도 클라이언트 GIDL 스텁 코드(GIDL 인터페이스를 위한 스텁과 그룹객체 인터페이스를 위한 스텁), 서버를 위한 스켈레톤 코드(일반 단위객체를 위한 스켈레톤, 그룹객체를 위한 스켈레톤) 등이 있다. 요소객체를 위한 클라이언트 스텁과 서버 스켈레톤 코드가 없는 이유는 클라이언트에게는 요소객체가 보이지 않고 그룹객체만 보이기 때문이다. 그림 6은 GIDL로부터 생성되는 Java 인터페이스 및 스텁 코드들을 보여준다. 스켈레톤 코드나 스텁 코드들은 CORBA ORB Core가 제공하는 기능들을 이용한다.

4.2 그룹관리자의 구현

그룹관리자는 그룹객체의 요소객체들을 관리하고 그룹객체에 들어오는 요청을 연결해주는 역할을 담당한다. 본 논문에서 그룹관리자를 구현하는 방식은 그룹관리자의 기능이 각 그룹객체의 일부분으로 지원되도록 구현한다. 이를 위해 그룹관리자를 하나의 클래스로 구현하고 각 그룹객체가 이를 상속할 수 있도록 한다. 그룹객체를 정의하는 방법에 대한 설명은 다음 절에서 설명하기로 한다.

그룹관리자 클래스의 IDL 인터페이스의 주요부분은 아래와 같다.

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

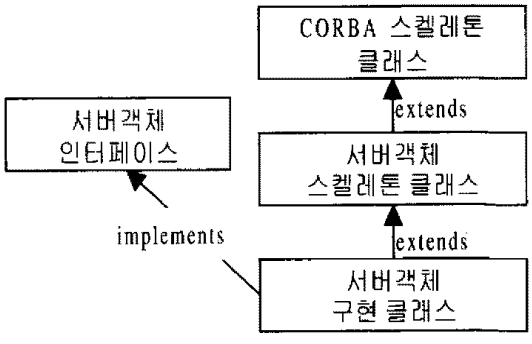
register() 연산은 요소객체를 등록하는 기능이고, deleate() 연산은 요소객체를 그룹객체로부터 삭제하는 기능이다. activate()와 deactivate() 연산은 각각 요소객체를 활성화하고 비활성화하는 기능이다. delateAll(), activateAll(), 및 deactivateAll()은 그룹객체에 속하는 요소객체 전체에 대한 연산이다. getElementObj()는 클라이언트로가 요청한 메시지가 어떤 객체로 전달될 것인지를 판별하기 위한 연산이다. 이들 기능을 구현하기 위해 그룹객체 관리자는 요소객체를 관리할 수 있어야 한다. 그룹객체 관리자가 요소객체의 관리를 위해 사용하는 자료구조의 IDL 정의는 다음과 같다.

```
enum ObjectState {ACTIVE, INACTIVE,
                  NO_AVAIL};
struct ObjectMgmt {
    Long   interfaceId;
    Object objectReference;
    enum ObjectState objectState;
};
```

ObjectMgmt 구조는 하나의 요소객체의 정보를 유지하기 위한 자료구조이다. interfaceId와 objectRerference는 클라이언트로부터 전달되어 오는 메시지가 어떤 요소객체로 전달되어야 할 지 판별할 때 사용하기 위한 정보이다. interfaceId는 GIDL에 정의된 GIDL의 인터페이스를 식별하는 식별자이고, objectReference는 요소객체의 등록시 저장되는 요소객체의 식별자이다. 클라이언트 스텁 코드로부터 서버 스켈레톤 코드로 전달되는 메시지는 그 메시지가 정의되어 있는 인터페이스에 대한 정보를 포함하고 있다. objectState변수는 객체가 등록되었는 지, 활성화(ACTIVE)또는 비활성화(INACTIVE) 되었는 지에 관한 정보를 포함하고 있다. 비활성화되어 있다면 활성화 시킨 후 메시지를 전달한다. 활성화의 방법은 BOA가 제공하는 기능을 이용한다. 메시지가 전송될 객체가 등록되어 있지 않다면 (NO_AVAIL) 적절한 예외 상황을 발생시킨다.

4.3 스텁 및 스켈레톤의 구현

스텁 및 스켈레톤의 구현방식은 어떤 CORBA 제품을 사용하느냐에 따라 다소 차이가 있을 수 있으나 그 기본적인 방법은 비슷하다. Visibroker 3.1에서는 서버객체를 구현할 때 그 객체가 구현하는 인터페이스를 위한 스켈레톤 클래스를 상속받아 구현한다(그림 7 참조).



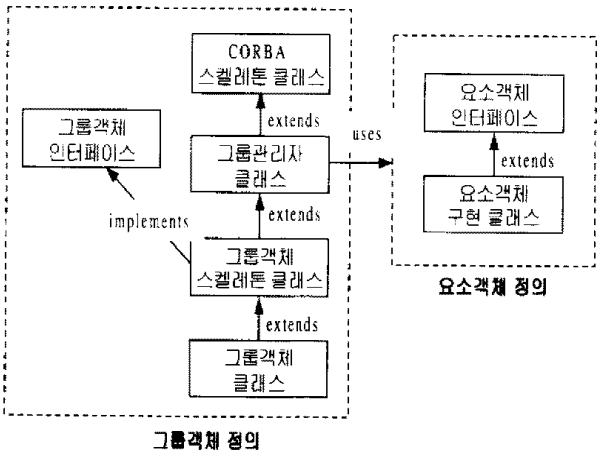
(그림 7) Visibroker 3.1 서버객체의 구현
(Fig. 7) Implementation of Server Object in Visibroker 3.1

본 논문에서도 일반 단위객체에 대해서는 이 방식을 그대로 따른다. 그런데 그룹객체를 지원하기 위해 추가된 그룹객체를 위한 스텁 코드나 스켈레톤 코드가 생성되어야 하는데 이 역시 그림 7의 방식과 유사한 방식으로 해결한다.

우선, 스텁 코드에 대해서는 그룹객체의 스텁 코드가 추가적으로 생성되어야 하는데 클라이언트 측에서

의 그룹객체를 하나의 객체로 간주하게 되므로 각 그룹객체에 대한 하나의 스텁코드를 생성하면 된다. 이 그룹객체에 대한 스텁 코드는 OMG IDL의 다중 상속 인터페이스의 스텁 코드를 생성하는 방식과 동일하므로 여기서는 설명을 생략하기로 한다.

스켈레톤 코드의 구현에 있어서는 기존의 방식과 큰 차이가 있다. 기존의 CORBA의 구현에서는 모든 서버객체마다 각각 하나씩의 스켈레톤 코드가 필요하나 본 논문의 그룹객체의 경우에는 그 그룹에 속하는 모든 객체는 그룹객체를 통해 메시지가 전달되므로 그룹객체를 위한 하나의 스켈레톤 코드만이 존재한다. 즉, 그룹객체의 요소객체로 전달되는 모든 메시지는 이 그룹객체의 스켈레톤 코드를 통하여 각 요소객체에 전달된다. 이들의 구현 방식은 서버측에서 그룹객체나 요소객체의 구현 클래스를 어떻게 정의할 것인가에 많은 영향을 주고 있다. 그림 8은 그룹객체와 요소객체의 구현 클래스와 스켈레톤 클래스들과의 관계를 보여주고 있다.

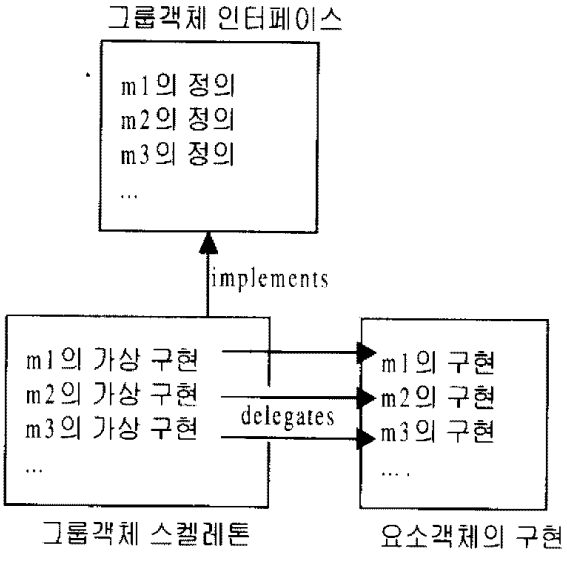


(그림 8) 그룹객체와 요소객체의 구현
(Fig. 8) Implementation of Group Object and Element Object

그룹객체의 구현 클래스는 그룹객체 관리자 클래스와 그룹객체 스켈레톤 클래스들을 다중 상속받아 구현되는데 Java 언어가 다중 상속을 지원하지 않으므로 이들 수퍼클래스들을 선형화(linearlization)하여 구현하였다. 실제적인 메시지의 수행은 요소객체가 담당하므로 그룹객체 클래스 내에서는 메시지를 수행하기 위한 아무런 코드가 없이 단지 수퍼클래스들을 상속받기만 한다. 그룹객체 클래스는 단지 요소객체의 초기화를 위한 생성자(constructor)만 정의한다(이 부분은 다음 절에서 예를 들어 설명한다).

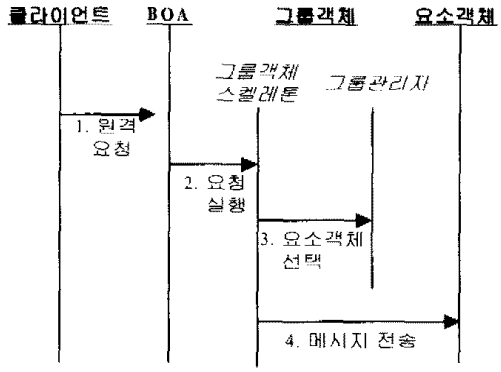
그룹객체 스켈레톤 클래스는 ORB로부터 원격 메시지 호출을 전달받아 수퍼클래스인 그룹객체 관리자로부터 해당 요소객체를 찾아 그 메시지를 전달하고 그 결과를 다시 ORB를 통해 클라이언트로 전달하는 역할을 한다.

그림에서 보듯이 이 클래스는 그룹객체를 위한 인터페이스를 구현(implements)한다. 이 인터페이스는 그룹객체의 자료형을 정확하게 부여하여 클라이언트와 서버 프로그램에서 그룹객체의 자료형으로 사용할 수 있게 한다. 그런데 그룹 객체 인터페이스에서 정의된 모든 메쏘드를 그룹객체 클래스에서 구현해야 하는 문제점이 발생한다. 단위객체의 경우에는 스켈레톤 클래스를 추상클래스(abstract class)로 정의하고 서버 객체 인터페이스에 포함된 모든 메쏘드는 스켈레톤 클래스의 서브클래스(서버객체의 구현 클래스)에서 구현하는데 반해, 그룹객체의 경우에는 메쏘드가 그룹객체 클래스에서 정의되지 않고 요소객체에서 정의되기 때문이다. 본 논문의 구현 방법에서는 스켈레톤 클래스에서 그룹객체 인터페이스에 정의된 각 메쏘드를 가상으로 구현(virtual implementation)한다. 그러나 실제적인 메쏘드의 구현(real implementation)은 요소객체에 있으므로 이 메쏘드의 정의는 단지 메시지를 요소객체로 전달(delegation)하는 일만 한다. 그림 9는 이러한 관계를 보여주고 있다.



(그림 9) 그룹객체의 메쏘드의 구현
(Fig. 9) Implementation of Methods in Group Object

그림 10은 클라이언트로부터 그룹객체로 보내진 메시지의 수행 구조를 보여주고 있다.



(그림 10) 클라이언트 요청의 수행구조
(Fig. 10) Execution Path of Client Request

4.4 프로그래밍 모델

(1) 그룹객체 클래스의 구현 및 그룹객체의 생성

4절의 스켈레톤의 구현에서 설명했듯이 그룹객체 구현클래스는 GIDL 컴파일러로부터 생성된 그룹객체 스켈레톤 클래스를 상속받아 정의된다. 그룹객체 스켈레톤 클래스는 다시 그룹관리자 클래스를 상속받고 있으므로, 그룹객체 클래스로부터 생성되는 그룹객체는 그룹관리자의 기능을 가지고 있다. 아래 프로그램은 3절에서 사용한 예제의 그룹객체를 위한 클래스 정의를 Java 언어로 작성하여 보여주고 있다. PrinterServer는 그룹객체 클래스이고 _PrinterServerImplBase는 프리컴파일러가 생성한 PrinterServer를 위한 그룹객체 스켈레톤 클래스이다.

```
package PrinterServerModule;
class PrinterServer extends _PrinterSserverImplBase {
        public PrinterServer() {
            Printer pr = new Printer("lp0");
            Spool sp = new Spool();
            register(pr);
            register(sp);
            PrinterManager pm =
            new PrinterManager(pr,sp);
            register(pm);
            activateAll();
        }
}
```

PrinterServer 클래스에서 정의한 생성자는 그룹객체를 생성할 때 그룹객체를 초기화하는 부분이다. 여기서 그룹객체의 초기 요소객체들을 등록한다.

(2) 클라이언트/서버 응용프로그램의 구조

그룹객체를 사용하는 클라이언트/서버 응용프로그램의 구조는 일반적인 CORBA 객체를 사용하는 프로그램 구조와 거의 동일하다. 클라이언트 측에서는 다음과 같은 순서로 작업을 진행한다. 우선 ORB를 초기화한 후, 클라이언트 측에서 사용할 그룹객체 인터페이스(전체 역할로서 사용할 경우)나 그룹객체의 한 서비스 인터페이스(하나의 역할로 사용할 경우) 자료형을 가진 변수를 정의하여 서버의 그룹객체에 바인딩한다. 바인딩된 그룹객체에게 일반 객체에 메시지를 전달하는 방식과 동일한 방식으로 서비스를 요청한다.

서버 측 프로그램은 단일객체를 이용하는 경우와 다소 차이가 있다. 서버 프로그램이 그룹객체를 이용하여 클라이언트 객체와 상호작용을 하는 과정은 다음과 같다. ORB를 초기화하고 ORB를 사용하여 BOA를 초기화한다. 그 다음 그룹객체 클래스를 이용하여 그룹객체를 생성하고 그 그룹객체에 요소객체를 생성하여 등록하고 요소객체를 활성화시킨다. 그룹객체를 생성시킨 후 BOA에게 그룹객체를 등록하고 활성화시킴으로써 서버 측의 그룹객체는 클라이언트의 서비스 요청을 대기하게 된다.

4.5 구현 모델의 평가 및 기존의 연구와의 비교

본 논문에서 제안한 구현 모델의 특징으로는 우선 그룹객체도 CORBA의 일반 객체와 마찬가지로 IDL의 확장언어인 GIDL이라는 플랫폼(언어, 운영체제, 하드웨어)-독립적인 인터페이스 언어로 기술한다는 점이다. 이렇게 함으로써 이기종의 다양한 플랫폼에서 분산 응용 시스템을 개발할 수 있게 된다. 또한, 구현모델인 GORB는 CORBA ORB가 원격 객체에의 요청을 중개하는 방식을 따라 그룹객체에의 요청을 지원할 수 있도록 자연스럽게 확장하였다. 이러한 확장이 CORBA 규격을 따르는 기존의 상용 제품에 쉽게 적용될 수 있다는 것을 앞 절에서 실험적인 구현을 통하여 보여주었다. 현재 Visibroker 3.1(Java 언어 지원)에서 구현하였지만 다른 제품에도 같은 방식으로 쉽게 확장할 수 있을 것이다.

아직까지 CORBA상에서 그룹객체의 구현에 관한

전히 많은 연구가 이루어지고 있지 않은 실정이다. 현재 그룹객체의 구현에 대한 연구로는 CORBA 트레이더를 이용하여 객체와 그룹객체 상호간의 접속방법을 제시한 연구가 있다[14]. 이 연구에서는 그룹객체가 서비스를 트레이더에 등록하고 클라이언트가 이를 이용할 수 있는 방법으로써 그룹객체의 위치 투명성(location transparency)과 동적(dynamic) 접속을 제공하는 등의 장점을 가지고 있다. 반면, 이 방법에서는 클라이언트와 그룹객체의 상호 접속이 항상 동적으로 일어나며 그룹객체가 제공하는 서비스를 명시적으로 트레이더에 등록하여야 한다. 또한, 클라이언트로부터 그룹객체의 접속 절차가 매우 복잡하며 클라이언트는 트레이더를 통하여 요소객체에 직접 접속함으로써 요소객체가 그룹객체 내에 캡슐화되지 않고 클라이언트에게 노출되는 문제점이 있다. 또한 동적 접속은 융통성을 증가시키는 장점은 있지만 항상 동적으로 접속하는 것은 시간적으로 비효율적인 방법으로 알려져 있다.

본 연구에서는 그룹객체를 IDL과 유사한 언어인 GIDL로 기술하고 클라이언트나 서버에서 프리컴파일러를 이용하여 미리 컴파일함으로써 정적(static)으로 클라이언트 객체와 그룹객체를 접속할 수 있도록 한다. 또 그룹객체는 일반객체와 같이 클라이언트에게 하나의 객체로 보이게 함으로써 클라이언트와 그룹객체의 상호 접속이 일반적인 객체지향 프로그램에서의 객체간 접속과 동일한 방식을 제공하도록 한다. 또한 그룹객체 내의 요소객체들은 노출되지 않고 숨김으로써 보다 캡슐화(encapsulation) 기능이 강화된다. 그러면서도 본 연구에서 제공하는 방법에서는 그룹객체를 하나의 객체로 볼 수 있기 때문에 기존의 트레이더를 이용하여 동적으로 객체간 상호 접속하는 방법은 그대로 유효하다. 앞에서의 방식에서 처럼 그룹객체가 자신의 각각의 서비스와 그 서비스를 제공하는 요소객체를 등록하는 것이 아니라 그룹객체 자신을 하나의 객체로서 등록하는 방식이다.

5. 결론 및 향후 연구 계획

본 논문에서는 CORBA에서 그룹객체를 지원하기 위한 그룹객체 모델을 정의하고 CORBA상에서 그룹객체를 구현하기 위하여 방법을 제안하였다. 본 논문에서 그룹객체를 지원하는 방식은 IDL과 유사한 GIDL언어로 그룹객체를 기술함으로써 클라이언트 객체와 서버 그룹객체는 자이를 정적으로 컴파일하는 방식이다. 물론, CORBA의 트레이더와 DII를 이용하여 동적으로 서버 그룹객체의 서비스를 이용할 수도 있다.

본 연구의 의의는 대표적인 분산 응용 프레임웍인 CORBA에서 그룹객체 개념을 지원할 수 있게 함으로써 대형 분산 응용 소프트웨어의 개발과 유지·보수시에 그 복잡도를 크게 줄여 준다는 점이다. 또 다른 장점은 구현모델이 CORBA ORB의 구조를 그대로 확장함으로써 CORBA ORB를 따르는 CORBA 제품에 쉽게 적용될 수 있다는 점이다. 현재, 제안된 그룹객체의 구현 모델은 CORBA 2.1 규격을 따르는 상용 제품인 Visibroker 3.1 (Java 언어 지원)에 기반을 두고 구현하였다.

본 논문에서 제안한 그룹객체의 구현 방안은 몇 가지 문제점을 가지고 있다. 그룹객체가 중첩될 경우 클라이언트로부터 중첩된 안쪽의 요소객체로의 연결은 많은 중간코드(가상 메쏘드 코드)를 거쳐 가야하는 문제점이 있다. 따라서 앞으로 GORB의 구현에 대한 성능을 분석한 후 중첩된 그룹객체를 효율적으로 지원할 수 있는 방안을 연구해야 한다. 그리고 클라이언트의 요청 시 메시지를 처리할 수 있는 요소객체가 없을 때나 또는 삭제하고자 하는 요소객체가 없을 때와 같은 예외에 대한 처리를 아직 지원하지 않고 있다. 마지막으로, 그룹객체를 동적인 환경에 적응하기 위해서는 동적으로 새로운 요소객체의 클래스형을 추가할 수 있어야 한다. 기존의 GIDL 인터페이스에 기술되지 않은 이러한 새로운 요소객체 클래스를 추가하기 위해서, 앞으로 CORBA의 DSI(Dynamic Skeleton Interface)와 DII(Dynamic Invocation Interface)기능을 이용하여 이를 지원하는 방안을 모색하고자 한다.

참 고 문 헌

[1] Martin Chapman and Stefano Montesi, "Overall Concepts and Principles of TINA," TB_MDC.018_1.0_94, TINA-C, Feb. 1995.

[2] James Gosling, Bill Joy, and Guy Steele, *The Java language Specification*, Addison-Wesley, 1996.

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

[4] Tom Handegard, "Object Grouping and Configuration Management," EN_TH.003_1.1_95, TINA-C, Aug. 1995.

[5] Scott M. Lewandowski, "Frameworks for Component-Based Client/Server Computing," *ACM Computing Surveys*, Vol.30, No.1, 1998.

[6] OMG, *The Common Object Request Broker: Architecture and Specification*, The OMG Publication, version 2.0, 1995.

[7] OMG, *The Common Object Request Broker : Architecture and Specification*, The OMG Publication, revision 2.1, Aug. 1997.

[8] Rober Orfali, Dan Harkey, and Jeri Edwards, *The Essential Distributed Objects Survival Guide*, John Wiley & Sons, Inc., 1996.

[9] Ajeet Parhar, "Grouping Example," EN_AP.002_0.1_95, TINA-C, Dec. 1995

[10] Ajeet Parhar (ed.), "TINA Object Definition Language Manual," TR_NM.002_2.2_96, TINA-C, version 2.3, July 22 1996.

[11] Frank E. Redmond, *DCOM : Microsoft Distributed Component Object Model*, IDG Books Worldwide, Inc., 1997.

[12] Visigenic, *Visibroker for Java, Version 3.1*, Visigenic Software Inc., 1998.

[13] Andress Vogel, Keith Duddy, *Java Programming with CORBA*, Wiley Computer Publishing Inc., 1997.

[14] 최은주, 정창원, 주수종, 오현주, "트레이더를 통한 분산 객체그룹들의 상호 접속 방안", *'97 가을학술 논문집*, 한국정보과학회, 이화여자대학교, 1997년 10월.

류 기 열

e-mail : kryu@madang.ajou.ac.kr

1985년 서울대학교 전자계산기공학과 졸업(학사)

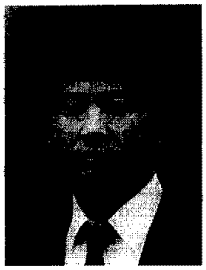
1987년 한국과학기술원 전산학과(공학석사)

1992년 한국과학기술원 전산학과(공학박사)

1993년~1994년 일본 토교대학 객원연구원

1994년~현재 아주대학교 정보및컴퓨터공학부 부교수

관심분야 : 객체지향 프로그래밍언어, 분산객체, 컴포넌트 모델, 객체지향 설계 및 분석



이 정 태

e-mail : jungtae@madang.ajou.ac.kr

1979년 서울대학교 농과대학 졸업(학사)

1981년 서울대학교 계산통계학과 졸업(이학석사)

1988년 서울대학교 계산통계학과 졸업(이학박사)

1981년~1983년 울산대학교 전자계산학과 전임강사

1983년~1988년 아주대학교 전자계산학과 전임강사

1988년~현재 아주대학교 정보·컴퓨터공학부 부교수

관심분야 : 객체 지향 시스템, 문서 처리 시스템, 병렬 처리 언어

변 광 준

e-mail : byeou@madang.ajou.ac.kr

1985년 서울대학교 전자계산기공학과 졸업(학사)

1987년 Pennsylvania State Univ., Computer Science(공학석사)

1993년 Univ. of Southern California, Compter Science (공학박사)

1994년~현재 아주대학교 정보·컴퓨터공학부 부교수

관심분야 : 데이터베이스, 분산객체