CORBA 구조 및 제품현황

동양시스템하우스 김석원*

1. 배 경

OMG(Object Management Group)는 1989년 4월 3COM, 아메리칸항공, 캐논, 데이타제너럴, 휴렛패커드, 필립스, 선, 유니시스 등을 포함하는 11개 회사에 의해 설립되었으며 현재는 800개이상의 회원을 가진 콘소시움으로 발전하였다. OMG의 목표에는 분산객체어플리케이션 개발을 위한 공통 프레임워크 가이드라인과 상세한 객체관리규약을 정립하는 것이 포함되어있다. OMG의 규약을 따르면 대부분의 하드웨어 플랫폼과 운영체제를 포함하는 이질적인 컴퓨팅환경에서 사용할 수 있는 공통 소프트웨어 환경을 개발할 수 있게 될 것이다[1].

CORBA(Common Object Request Broker Architecture)는 OMG에서 정의한 분산객체 컴퓨팅환경의 표준으로서 이질적인 분산환경에서 어플리케이션을 구현하는데 필요한 프레임워크, 객체서비스, 분야별 객체 클래스라이브러리가 정의되어있다. CORBA표준규약은 1991년에 발표된 CORBA 1.1에서 IDL(Interface Definition Language)과 ORB(Object Request Broker)를 이용한 클라이언트/서버 객체의 통신방식을 정의하였고, 1994년에 CORBA 2.0을 발표하여 서로 다른 벤더간의 ORB가 연동할 수 있도록 IIOP(Internet Inter-ORB Protocol)를 정의하였으며, 지금은 CORBA 2.2까지 발전시켰다.

* 종신회원

2. CORBA 아키텍처

2.1 OMG OMA 참조모델

OMA 참조모델은 OMG에서 CORBA 표준을 정의하면서 기준으로 삼고있는 모형이다. (그림 1).

2.1.1 Object Request Broker

ORB는 클라이언트와 서버간의 연산요청과 실행결과를 전달한다. 요청을 전달하기 위해 실제 서버의 위치를 알아내고, 연산에 필요한 매개변수를 서버에서 이해하는 형태로 변환한다. 실제 구현에서 ORB는 클라이언트와 서버 프로그램에 링크되는 라이브러리형태로 구현된다. ORB에 의해 클라이언트/서버가 완전히 분리될 수 있으며 이에따라 분산시스템의 확장성과 유연성이 크게 향상된다.

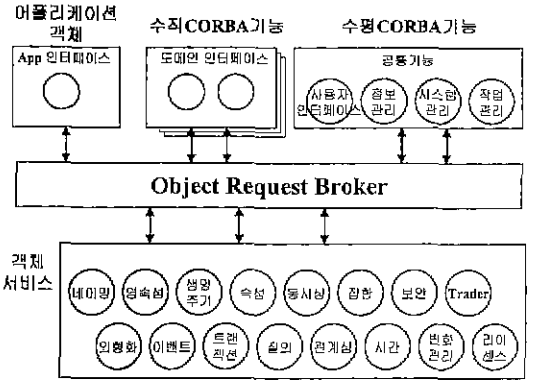

그림 1 OMA 참조모델

2.1.2 객체서비스(Object Services)

CORBA로 구축된 분산시스템의 기본 기능에 추가하여 시스템수준에서 제공되어야 하는 서비스를 정의한 표준이다. 네이밍, 이벤트, 영속성, 트랜잭션, 보안 등과 같은 서비스들이 정의되어 있으며 필요에 의해 계속 추가되거나 수정된다. 객체서비스를 ORB와 분리하여 ORB구현이 용이해지므로 많은 플랫폼에서 지원하게되고, 필요한 기능은 선택적으로 적용할 수 있게 된다.

2.1.3 공통기능(Common Facilities)

CORBA객체서비스 위에서 어플리케이션의 개발생산성을 높여주기 위해 정의된 클래스라이브러리이다. 이것은 어플리케이션의 적용분야와 무관하게 정의될 수 있는 공통기능인 수평CORBA기능과 산업분야별 공통기능인 수직CORBA기능으로 나뉜다. 수평CORBA기능에는 사용자인터페이스, 정보관리, 시스템관리, 작업관리 등이 있으며 수직CORBA기능은 금융, 제조, 통신, 의료, 전자상거래, 운송 등의 분야에 대하여 정의하고 있다.

2.1.4 어플리케이션객체(Application Objects)

최종적으로 사용하게되는 어플리케이션이다.

2.2 ORB 구조

ORB는 CORBA가 작동되는 공통의 플랫폼이며 일종의 소프트웨어버스라 할 수 있다. 모든 CORBA의 호출은 ORB를 통하여 전달된다. 그림 2는 ORB의 구조이다.

2.2.1 IDL 인터페이스

객체의 인터페이스는 OMG IDL을 이용하여 정의된다. 인터페이스 정의는 객체가 수행할 연산, 연산별 입출력 매개변수, 연산수행에서 발생할 수 있는 예외를 규정한다. 이 인터페이스를 통하여 클라이언트쪽 객체는 동일한 인터페이스를 지원하는 서버쪽 객체구현에 대하여 연산을 요청하고 결과를 받을 수 있다. 클라이언트와 서버가 오직 IDL정의만을 공유하여 연동하도록 하는 구조는 매우 유연한 설계를 할 수 있고, 클라이언트와 서버 각각에 대한 독립성을 유지시켜서 서로 다른 프로그램언어로 구현되더라도 요청을 처리할 수 있게 한다. OMG에서는 C, C++, Smalltalk, Java, 코볼, Ada 등의 주요 프로그램언어에 대하여 IDL정의를 매핑하는 방법을 규정해 놓고 있다.

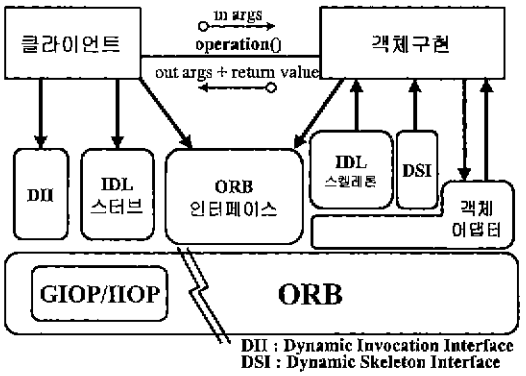

그림 2 CORBA ORB의 구조

2.2.2 호출방식

클라이언트에서 서버를 호출하는 방식은 정적호출방식과 동적호출방식의 두가지를 제공한다. 정적호출방식은 클라이언트의 요청이 IDL 인터페이스 정의로부터 생성된 스터브코드(Stub code)를 통하여 전달되는 방식이다. 정의된 IDL을 IDL컴파일러로 컴파일하여 클라이언트 프로그래밍언어의 스터브코드가 자동생성되고 이 코드가 클라이언트 프로그램과 링크되어서 사용된다.

동적호출방식은 클라이언트가 실행시간에 ORB 함수를 이용하여 요청을 만들어내고 이것을 상대편 ORB에 전달하여 호출하는 방식이다. 동적호출인터페이스(Dynamic Invocation Interface)는 유연한 어플리케이션을 만들 수 있다는 장점이 있으나 그 댓가로 코드의 양이 늘어난다. 클라이언트와 마찬가지로 서버쪽에서도 객체구현을 호출할 때 정적호출을 위한 인터페이스로 IDL 스켈레톤, 동적호출을 위한 DSI(Dynamic Skeleton Interface)가 정의된다.

2.2.3 IIOP

CORBA ORB는 일반적으로 IIOP를 사용하여 통신한다. IIOP는 ORB간의 통신을 위한 일반 규약인 GIOP(General Inter-ORB Protocol)을 인터넷 표준통신규약인 TCP/IP상에서 동작하도록 정의한 것이다. 즉 TCP/IP 네트워크에 연결된 ORB간의 통신을 위한 규약이다.

CORBA표준에서 어떤 ORB든 TCP/IP는 의무적으로 지원하도록 규정하고 있으므로 IIOP만을 지원하더라도 ORB간의 연동에 큰 무리가 없다.

2.2.4 객체어댑터

객체어댑터는 객체구현(Object Implementation)에서 ORB가 제공하는 서비스에 접근하는데 사용되는 요소이다. 객체어댑터에 대해서는 다음 장에서 설명하겠다.

3. CORBA의 새로운 기능들

3.1 Portable Object Adapter[2, 3, 4, 5]

CORBA는 초기의 규약에서 빠져있거나 명확히 정의되지않은 부분때문에 ORB를 구현하는 업체마다 서로 다른 방식으로 구현되었다. 이에 따라 어떤 ORB로 구현된 프로그램을 다른 ORB에서 실행하려면 많은 수정이 불가피했다. 그 중 대표적인 것이 객체어댑터규약이며 이 문제를 해결하기 위하여 OMG에서 제시한 새로운 규약이 POA(Portable Object Adapter)이다.

3.1.1 CORBA객체와 서번트

POA를 이해하기위해서는 새로 등장한 개념인 서번트(servant)와 CORBA객체와의 관계를 이해해야 한다. CORBA객체는 ORB에의해 검색되고 클라이언트의 요청이 전달될 수 있는 "가상의" 존재이다. 이것은 여전히 객체참조자를 통해 접근할 수 있다. 이 가상의 객체는 "서번트"라는 프로그래밍언어로 정의된 실체와 결합된 상태에서만 현실화될 수 있다. 서번트는 CORBA객체를 구현한 프로그램 실체이다. 즉 무형의 CORBA객체는 리소스를 갖는 서번트와 결합하여 비로소 클라이언트의 요청을 처리할 수 있게 된다. 서번트는 프로그래밍언어에 따라 형태를 달리하며 C++나 자바와 같은 객체지향언어의 경우에 서번트는 클래스의 객체인스턴스가 된다.

CORBA객체가 클라이언트의 요청을 처리하기 위해서는 활성화되어야 하는데(activate) 이것은 적절한 서번트와 결합되었다는 의미이다. 반대로 이 CORBA객체의 일이 끝나면 비활성화되며(deactivate), 이것은 CORBA객체와 서번트간의 결합이 끊어졌다는 뜻이다. 서번트입장에서 보면 단지 프로그램의 클래스 인스턴스이던 서번트가 CORBA객체와 결합되면서 CORBA세계에서 실현(incarnate)되고, CORBA객체와의 결합이 끊어지면서 CORBA세계에서 승화(etherealize)된다. incarnate, etherealize는 CORBA객체를 영혼으로보고, 서번트를 육체로 보는 맥락에서 사용된 것으로 보이며 정확한 용어를 찾기가 어려워 본 글에서는 실현과 승화라는 말로 사용하였다.

3.1.2 객체어댑터의 기능

- 클라이언트의 CORBA객체에 대한 요청을 적절한 서번트로 분배한다.
- 목표서번트에게 요청된 연산을 전달한다. 이때 스켈레톤을 사용하여 매개변수를 목표서번트에 맞도록 변환하여 전달한다.
- 객체어댑터는 CORBA객체를 활성화시킬 수 있다. 이 과정에서 서번트가 이 객체에 대한 요청을 처리할 수 있도록 실현된다(incarnate). 마찬가지로 객체어댑터는 객체를 비활성화시키고 여기에 대응하는 서번트가 더 이상 필요없으면 승화시킨다(etherealize).
- 객체어댑터는 등록된 CORBA객체에 대한 객체참조자를 생성한다. 객체참조자는 CORBA객체를 지시하며 분산시스템에서 어떻게 그 객체에 접근할 수 있는가를 나타내는 위치정보를 포함하고 있다. IIOP를 쓴다면 이 위치정보는 인터넷주소와 포트번호가 된다.

3.1.3 BOA의 문제점

POA이전에 사용되던 CORBA규약인 BOA (Basic Object Adapter)에는 다음과 같은 문제가 있었다.

· 스켈레톤과 서번트의 결합방법

스켈레톤과 서번트를 결합하는 방법의 이식성이 없었다. BOA규약은 스켈레톤의 형태나 서번트와 스켈레톤이 결합하는 방식에 대하여 명시하지 않았으므로 업체별로 서로 다른 방식을 사용하게 되어 코드의 이식이 매우 어렵게 되었다.

· 서번트의 등록방법

BOA를 구현하려면 서번트가 암시적으로든 명시적으로든 ORB에 등록이 되어야 한다. BOA규약은 이것을 위한 API를 정의하고 있지 않으며 따라서 ORB마다 서로 다르게 구현해왔다.

· 멀티쓰레딩

서버 프로세스의 멀티쓰레딩문제에 대한 언급이 없으며 ORB 개발자의 결정에 맡긴다. 결과적으로 각 ORB마다 멀티쓰레딩을 작위적으로 구현하여 제공하고 있다.

· 애매한 함수

BOA에서는 서버가 요청을 처리하기 시작하도록 만드는 함수인 impl_is_ready와 obj_ is_ready의 의미가 불명확하여 벤더마다 다른 의미로 사용하였다.

3.1.4 POA규약

1997년 3월에 POA규약이 BOA를 대치하도록 OMG에 제출되었고 CORBA2.2에 정식으로 포함되었다. 주의할 점은 POA에 의해 기존의 BOA는 사라지게 되었다는 점이다. 그러나 상용ORB제품에서는 기존고객을 위해서 BOA를 계속 지원하고 있으며 POA가 구현된 상용ORB제품의 수도 많지 않다.

POA는 ORB와 서번트 사이에 위치하여 둘 사이의 요청을 중계한다(그림 3). 클라이언트

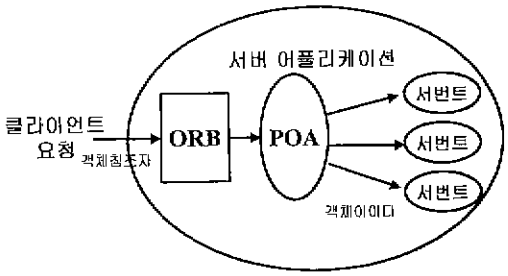

그림 3 ORB, POA, 서번트의 관계

는 객체참조자를 사용하여 대상객체에 대한 연산을 요청한다. ORB는 객체참조자를 사용하여 서버 어플리케이션에서 대상객체를 담당하는 POA를 찾을 수 있다. POA에서는 객체참조자에 포함되어있는 객체아이디를 사용해서 대상객체가 어느 서번트에 결합되어있는지 판단할 수 있다. POA는 서번트에게 요청을 넘겨주고, 서번트는 그것을 수행하여 결과를 POA에게 넘겨주며 POA는 ORB를 통하여 최종적으로 클라이언트에게 전달되게 된다.

객체아이디는 한 POA내에서 CORBA객체를 나타내는데 사용되는 바이트스트링이다. 객체아이디는 그것이 정의된 POA내에서만 의미를 가지므로 클라이언트가 CORBA객체를 나타내는데에는 여전히 객체참조자를 사용하여야 한다.

결국 POA는 클라이언트로부터 대상객체로의 요청을 객체참조와 객체아이디를 사용하여 대상객체를 찾고, 대상객체와 결합된 서번트로 전달하여 요청을 수행하도록 하는 대상객체와 서번트의 연결고리라고 할 수 있다.

3.1.5 서버 프로그램의 구조

그림 4는 POA를 사용하여 구현한 간단한 서버 프로그램이다[4]. 본 예제는 워싱턴대학에서 개발된 공개ORB인 TAO를 이용하여 만든 C++예제이다.

```
int main (int argc, char *argv[])
{
 // 1. ORB초기화
 ORB_var orb = ORB_init (argc, argv);
 // 2. RootPOA 검색. 필요시 새POA 정의
 Object_var obj =
```

```
orb->resolve_initial_references("RootPOA");
POA_var poa = POA::_narrow(obj);
// 3. 서번트 생성
My_Servant_Impl servant;
// 4. 필요시 CORBA객체 생성. 서번트 등록
obj = servant._this ();
// 5. 필요시 객체참조자를 서비스에 등록
// 6. POA 활성화
poa->the_POAManager ()->activate ();
// 7. ORB 이벤트루프 시작
orb->run ();
// 8. 종료는 다른 곳에서 일어남.
}
```

그림 4 POA 서버 프로그램

POA를 구현한 서버 프로그램은 일반적으로 다음과 같은 구조를 갖는다.

1) ORB_init를 사용하여 ORB를 초기화한다.
2) POA에 대한 객체참조자를 획득한다. 모든 서버 프로그램은 RootPOA라는 적어도 하나의 POA를 가진다. 예제의 POA::_narrow는 obj가 Object 유형이므로 POA 유형으로 바꾸어주기 위하여 사용한다. 사용자는 RootPOA를 그대로 사용할 수도 있고 RootPOA의 자식노드로서 새로운 POA를 생성하여 사용할 수도 있다. 같은 POA에 속하는 서번트는 같은 정책을 사용하게 되므로 RootPOA와 다른 정책을 사용하려면 새로운 POA를 생성하여 사용한다.
3) POA에서 사용할 서번트를 생성한다. 객체지향프로그래밍언어에서는 클래스의 인스턴스를 만드는 것이 서번트를 생성하는 것이 된다. 예제에서는 My_Servant_Impl이라는 클래스의 서번트를 사용하고 있다.
4) 서번트와 결합될 CORBA객체를 생성하고 서번트를 POA에 등록한다. 이 부분은 POA와 CORBA객체의 성격에 따라 다양하게 변화될 수 있으나 본 예제에서는 간단히 _this 메소드를 이용하여 CORBA객체를 만들고 서번트를 RootPOA에 등록해주었다. 이것은 객체가 일시객체(transient object)이고 RootPOA를 사용할 경우에 편리하게 이용할 수 있다. 객체가 영속객체(persistent object)이거나 사용자가 정의한 POA를 사용한다면 이 부분을 수정해주어야 한다.
5) 본 예제에서는 명시되지 않았지만 객체참조자가 새로 정의되었으면 이 객체참조자를 네이밍서비스나 트레이딩서비스에 등록하여 클라이언트가 검색할 수 있게 한다.
6) POA를 활성화한다. POA가 활성화되면 클라이언트의 요청을 처리할 수 있게 된다.
7) ORB의 이벤트루프를 시작한다.
8) ORB의 종료는 클라이언트의 요청을 처리하는 도중 ORB::shutdown을 만나게 되면 일어나게 되며 시스템관리자가 프로세스를 죽이는 경우에도 ORB가 종료된다.

이상 예제를 보면 POA를 적용하면 서버 프로그램의 구조가 명확하게 정의된다는 것을 알 수 있다. 즉 표준규약만을 보고도 서버 프로그램을 초기화하는 부분을 구현할 수 있다. 그동안 BOA를 사용하여 애매하게 정의되었던 객체의 활성화방식, 관리방식, 유형 등이 명확히 정의됨으로써 제품간의 구현에서 나타나는 애매성이 제거되고 서버 프로그램의 이식성을 보장하게 되었다.

3.1.6 POA의 정책

하나의 서버 어플리케이션은 여러개의 POA를 가질 수 있다. 각 POA는 여러 종류의 CORBA객체를 지원하거나 여러 종류의 서번트를 지원하도록 정의할 수 있다. 서버에서 정의하는 POA는 생성당시 다른 정책을 가지도록 정의할 수 있으며 사용할 수 있는 정책은 다음과 같다.

· LifeSpan Policy

영속객체는 객체의 존속기간(Lifespan)이 그것을 활성화시킨 서버 프로세스의 존속기간과 무관한 객체를 말한다. 즉 영속객체는 일단

생성되면 그것을 만든 서버 프로세스가 종료되더라도 계속 존재하며 제거하려면 어플리케이션에서 명시적으로 제거해주어야 한다. 일시객체는 서버 프로세스가 종료되면 사라진다.

영속객체는 객체의 상태가 영속적인 반면 POA는 영속적인 상태를 가질 수 없으며 객체의 영속적 상태를 저장하는 기능을 제공하지도 않는다. 그러므로 영속객체의 상태를 보존하는 일은 객체의 구현부분에서 제공해 주어야 한다.

영속객체를 이용하려면 그 객체가 속하는 POA의 LifeSpan정책이 PERSISTENT이어야 한다. RootPOA의 정책은 TRANSIENT이므로 LifeSpan정책이 PERSISTENT인 POA를 정의하여 RootPOA의 자식노드로 등록하여 사용한다.

클라이언트가 현재 비활성인 영속객체에 대하여 요청을 하면 ORB는 클라이언트가 알지 못하는 사이에 이 요청을 처리할 객체를 포함하는 프로세스를 생성하여 요청이 해당 객체에서 처리되도록 해야 한다. 이렇게 요청에 대해 서버 프로세스를 준비하는 것은 ORB의 책임이고, 객체가 올바른 상태를 가지도록 하는 것은 객체를 구현하는 사람의 몫이다.

· 멀티쓰레딩

POA규약에서는 각 POA별로 멀티쓰레딩 정책을 지정할 수 있게 되었다. 멀티쓰레딩정책은 ORB제어정책과 단일쓰레드정책으로 나뉜다. ORB제어정책은 ORB가 적절한 쓰레딩 정책을 결정하며 단일쓰레드정책은 항상 단일 쓰레드만을 지원하게 된다. ORB제어정책을 사용하는 경우에는 POA에서 쓰레드풀이나, 요청별 쓰레드 할당, 객체별 쓰레드 할당 등의 다양한 세부기능을 제공할 수 있으므로 필요에 의해 원하는 기능을 추가할 수 있다. 이런 유연성은 향후 CORBA규약의 확장을 위해서는 유리하겠지만 벤더에서 나름대로의 기능을 추가할 가능성이 있으므로 이식성면에서는 새로운 문제를 야기시킬 수도 있다.

· 기타 정책

- Servant Retention Policy : CORBA객체와 서번트와의 결합관계를 POA가 보유하고 있는지 여부를 정의한다. 이 정책이 NON-RETAIN인 경우에는 결합관계를 어플리케이션에서 지정해 주어야 한다.
- Request Processing Policy : 요청에 대해 적절한 서번트를 찾아주는 방법을 지정한다.
- Implicit Activation Policy : 서번트의 활성화를 암시적으로 할 수 있는지의 여부를 정의한다. 일시객체의 경우에는 서번트가 암시적으로 활성화되는 것이 편리하다.
- ObjectID Uniqueness Policy : 서번트가 오직 하나의 객체만을 처리할 수 있는지, 여러 객체를 처리할 수 있는지를 정의한다. 시스템에 CORBA객체의 수가 많아지면 모든 객체에 서번트를 할당하는 것이 어려우므로 하나의 서번트가 여러개의 객체를 처리할 수 있도록 정의하여 확장성을 높일 수 있다.
- Object Assignment Policy : 객체아이디가 POA에 의해서 지정되는지, 어플리케이션에서 제공하는지를 정의한다.

3.2 메시징[3, 6, 7, 8]

3.2.1 개요

비동기메시징(Asynchronous Messaging)은 오랜동안 CORBA규약의 미개척분야로 남아있었다. 비동기 메시징이 지원되면 시스템요소가 항상 연결된 상태로 있을 필요가 없어지기 때문에 랩탑, 팜탑, 핸드헬드장치 등과 같은 모빌시스템을 지원할 수 있게되며 네트워크 장애나 시스템장애에 대해서도 신뢰성있는 서비스를 제공할 수 있게 된다. 이런 비동기 통신방식의 부재때문에 CORBA는 확장성에서 제약이 있는 것으로 인정되어 왔다.

이런 문제를 해결하기 위해 OMG에서 내놓은 표준이 CORBA메시징규약이다[6]. 메시징규약은 현재 기술안이 상정된 상태이며 기술채택투표, 수정안심의 등을 거쳐 2000년에 확정될 예정이다. CORBA메시징규약에는 비동기 메시징(AMI:Asynchrounous Method Invocation), TII(Time-Independent Invocation), 메시징의 QoS(Quality of Service)를 명시하

기 위한 방식 등이 정의되었다.

3.2.2 기존의 호출방식

원래 CORBA에서는 다음과 같은 요청호출 방식을 지원해왔다.

- 동기식 : 클라이언트가 연산을 호출하고, 결과가 반환될 때까지 수행이 중단된다. 그러므로 단일쓰레드 클라이언트는 응답이 올 때까지 완전히 수행이 중단되며 성능을 중시하는 어플리케이션에서는 만족스럽운 결과를 얻지 못하게 된다.
- 지연동기화(Deferred Synchrounous) : 클라이언트가 연산을 호출하고서 수행결과를 기다리지 않고 곧장 수행을 계속한다. 클라이언트는 다른 부분에 대한 수행을 진행하면서 정기적으로 polling을 하여 연산결과가 반환되었는가를 검사할 수 있다. 또는 다른 일부터 수행한 후에 연산결과를 기다리도록 구성할 수도 있다. 이 방식의 문제점은 DII를 사용하는 클라이언트만 사용가능하기 때문에 프로그램이 복잡하고 길어진다는 것이다.
- 원웨이(Oneway) : 클라이언트가 원웨이 연산을 호출하면 ORB는 이 요청을 목표에 전달하기 위해 "최선의 노력"을 다하도록 규정한다. 이 연산은 실행결과가 없다. 즉 반환값을 기다릴 필요가 없다는 뜻이다. 대신 이 요청이 목표에 도달한다는 보장은 하지 못한다. CORBA규약에서는 이것을 *best effort guarantee*로 정의하였다. best effort semantic이란 대상서버가 현재 네트워크에 연결이 되어있지 않거나 중간에 오류가 발생하여 ORB에서 이 요청을 전달할 수 없는 경우에도 ORB가 오류를 보고할 필요가 없다는 것이다. 대신 CORBA에서는 ORB가 전달과정에서 오류나 예외를 겪지 않은 경우에 "정확히 한 번"만 요청이 전달되고, 오류가 발생하거나 예외를 겪은 경우에는 "최대한 한 번" 전달된다는 것을 보장한다.

3.2.3 비동기메시징

메시징규약에서 비동기호출방식을 위해 추가된 새로운 기능에서 가장 중요한 점은 정적호출에서도 지연동기화호출을 할 수 있게 한 것이다. 정적호출을 사용하면 동적호출에서 파생되는 많은 불편함, 즉 코드가 길고 복잡해지며, 매개변수의 유형검사를 컴파일시에 할 수 없다는 점 등을 해소할 수 있다. 다음은 CORBA 메시징규약에서 추가된 모델이다.

- 콜백 : 클라이언트는 요청메마다 실행결과를 받아서 처리할 ReplyHandler라는 콜백용 객체참조자를 매개변수로 전달하고 서버의 실행결과가 도착하면 클라이언트쪽 ORB에서 그 객체참조자의 적절한 콜백메소드를 호출해 줌으로써 실행결과를 전달한다.
- 폴링 : 클라이언트가 연산을 호출하면 결과를 폴링하거나 대기하는데 사용되는 Poller 밸류타입을 즉시 반환한다. 밸류타입은 OBV(Objects By Value)에서 새롭게 정의된 유형이며 일종의 클래스라고 볼 수 있다. 클라이언트 프로그램에서는 나중에 이 Poller에 정의된 메소드를 이용하여 실행결과가 도착했는지 검사할 수 있다.

3.2.4 Time-Independent Invocation

클라이언트가 호출할 당시에 활성화되어있지 않거나 네트워크에 연결되어있지 않은 객체에 대한 호출을 허용하기 위하여 표준 GIOP와 IIOP ORB 상호운용규약에 중간라우팅에이전트에게 요청을 전달하는 기능을 추가하였다. 이 에이전트는 store-and-forward 기능을 제공하여 목표가 현재 활성화되지않거나 연결이 되지않은 경우에도 메시지를 보관했다가 나중에 전달해 줄 수 있다. 응답도 마찬가지로 라우팅에이전트를 통하여 전달되며 요청한 클라이언트에 연결할 수 없는 경우에도 에이전트가 보관하였다가 나중에 전달해준다. 보관되는 조건이나 시한은 서비스수준(QoS)을 적절히 지정하여 제어한다.

3.2.5 서비스수준(Quality of Service)

메시지전달, 메시지큐잉, 메시지우선순위 등에 대한 서비스의 수준을 클라이언트가 지정할 수 있도록 한다. 이것은 ORB수준으로 정의할

수도 있고, 쓰레드 수준으로 정의할 수도 있으며, 객체참조자 수준으로 정의하여 이 객체참조자를 이용하는 요청에 대해서만 적용되도록 지정할 수도 있다.

3.3 Objects By Value(OBV)[3, 9]

그동안 CORBA연산은 객체참조자를 원격어플리케이션에 넘겨주고 실행결과가 반환되는 방식이었며, 객체자체가 로컬어플리케이션으로 넘어와서 연산이 일어나는 것은 아니었다. 이런 단점을 보완하기 위하여 객체를 pass-by-value로 전달하는 기능이 OMG에 제안되어 현재 기술채택투표단계에 있다.

이를 위하여 밸류타입이라는 새로운 유형이 OMG IDL에 추가되었다. 밸류타입은 C++나 자바의 클래스정의와 마찬가지로 데이터멤버와 연산을 동시에 지원하므로 스트럭쳐와 인터페이스를 합친 것이라고 볼 수 있다. 인터페이스와 마찬가지로 밸류타입은 단일상속에 한하여 다른 밸류타입으로부터 상속받을 수 있다. 이 밸류타입을 이용하여 원격프로세스에 데이터와 메소드를 함께 넘겨주는 객체의 Pass-By-Value를 부분적으로 할 수 있게 되었다.

밸류타입이 원격연산의 인자로 사용되면 수신주소공간에 복사본이 생성된다. 밸류타입의 복사본은 원본과는 완전히 구분되며 한쪽에 어떤 연산을 하더라도 다른 쪽에 영향을 미치지 않는다. 복사본이므로 밸류타입에 대해 적용되는 연산은 항상 밸류타입이 존재하는 프로세스에서 지역적이다. 즉 CORBA객체에 대한 연산과는 달리 밸류타입의 호출은 네트워크를 넘어서 요청을 전송하거나 결과를 전달받지 않는다.

원격어플리케이션에 연산인자로 밸류타입의 상태, 즉 데이터멤버를 전달하는 것은 다른 데이터타입을 전달하는 것과 동일하다. 그러나 대부분의 밸류타입은 메소드도 포함하고 있다. 메소드는 네트워크를 통해 전달하기가 쉽지 않다. CORBA가 사용되는 환경은 대개 서로 이질적인 환경이 섞여있다는 점을 감안할 때 밸류타입 연산의 메소드를 넘겨주는 것은 불가능하다.

자바와 스몰톡 어플리케이션은 각 밸류타입과 함께 충분한 정보가 정리되어 수신어플리케이션이 그 밸류타입연산의 구현에대한 바이트코드를 다운로드받아서 사용할 수 있도록 구성할 수 있다. 그러나 이런 경우를 제외한다면 OBV에서 메소드를 넘겨주는 방식은 아주 특수한 환경, 즉 라이브러리가 동적으로 전달될 수 있는 환경같은 경우에만 적용될 수 있을 것이다. OMG에서도 OBV는 자바위주로 표준이 만들어지고 있다.

이상과 같은 밸류타입의 제약 때문에 OBV는 제한적으로 연결리스트, 그래프, 날자, 시퀀스생성기 등과 같이 가볍고 캡슐화된 데이터타입을 정의하는데 사용되도록 권유되고 있다.

4. CORBA를 구현한 제품들

상용 CORBA제품 중에서 널리 사용되는 제품은 BEA의 ObjectBroker, IONA 사의 Orbix, Inprise의 VisiBroker이다. 공개소프트웨어로는 OmniORB나 TAO 등이 상용 제품과 버금가는 인기를 가지고 있다. 여기에서는 세가지 상용ORB제품에 대해서 공급업체의 동향과 주요 기능에 대해서 설명하겠다.

4.1 BEA 사의 ObjectBroker[10]

BEA 사의 ObjectBroker는 원래 디지털 사에서 개발한 것으로 미들웨어 시장의 통합 솔루션을 제공하기 위해 BEA 사가 인수한 제품이다. BEA 사는 Tuxedo 제품으로써 트랜잭션 모니터 시장에서 확고한 입지를 구축하였으며 Tuxedo 상위에 CORBA API를 제공하는 제품인 M3를 98년에 출시하여 CORBA 시장에도 진출을 시도했다. 그러나 BEA사는 최근 M3 제품을 자사의 웹로직 어플리케이션 서버에 포함시키는 전략을 발표하여 CORBA 단독 제품이 아니라 어플리케이션서버의 일부로서 CORBA를 지원하고 있다.

4.2 IONA 사의 Orbix[11]

아이오나사의 오빅스는 가장 먼저 발표된 제품으로 많은 구현사례가 있어서 상대적으로 안정된 제품이라고 할 수 있다. 가장 많은 제품

표 1 ORB제품의 기능 및 특성 비교

	Orbix 3.0	ObjectBroker 2.7	VisiBroker 3.4
CORBA 버젼	2.1	2.2	2.1
프로그램언어	C, C++, ADA95, Java, OLE	C, C++, VB	C, C++, Java, OLE
서비스	네이밍, 이벤트, 트랜잭션, 트레이더, 노티피케이션, 영속성, 질의	네이밍	네이밍, 이벤트, 트랜잭션
POA 지원	베타	Y	베타
보안	ACL, 방화벽	DCE, DECNET	방화벽
멀티쓰레딩	Y	N	Y
비동기메시징	자체솔루션지원(OrbixTalk)	N	N
ODBMS어댑터	Y	N	N
OLE/COM 통합	Y	Y	Y
COM/CORBA 게이트웨이	Y	N	N
플랫폼	솔라리스, HP-UX, IRIX, AIX, 디지털유닉스, MVS, 윈도우NT (인텔), 윈도우95, Vx웍스, QNX, IBMOS/390	솔라리스, HP-UX, AIX, 디지털유닉스(Alpha), OpenVMS(VAX, Alpha), 윈도우NT(Intel, Alpha), 윈도우95, Mac, Win3.1	솔라리스, HP-UX, 디지털유닉스, 윈도우NT, 윈도우95, IBM OS/390
코드 자동 생성	OrbixGenerator	N	C++ Builder, Jbuilder
IDE 환경	OrbixStudio(MSDEV 환경통합), Visual Café	N	C++ Builder, Jbuilder
객체 디자인 도구	OrbixComposer	N	N

군을 가지고 있어서 다양한 요구사항이 존재하는 프로젝트에서 상대적으로 유리하고 메인프레임을 포함하여 가장 많은 플랫폼을 지원한다. 아이오나사는 이번 2월에 자사의 아키텍쳐 프레임워크로서 C3(Component, Container, Connector)를 발표하여 CORBA를 통한 상호운영솔루션의 폭을 넓히고 있다.

최근에 발표된 Orbix3는 CORBA 2.1을 준수하는 제품이다. Orbix3는 이전의 Orbix2.3에 비해 성능을 대폭 향상시키고 MS Visual-Studio와의 통합개발환경을 제공하여 개발 생산성을 향상시키고 프로그래밍 기술의 빠른 습득을 가능하게 한다. 또한 네이밍서비스와 방화벽을 함께 제공하여 시스템개발에 필요한 최소한의 기능을 모두 포함하고 있다.

4.3 Inprise 사의 VisiBroker[12]

VisiBroker는 출시가 타제품에 비해 늦은 편이었지만 처리속도 면에서 우수했기 때문에 빠른 속도로 인정받은 제품으로 오라클이나 넷스케이프 등에서 내부 CORBA 엔진으로 채택이 되는 등 시장에서 큰 비중을 차지하고 있다. 또한 작년 말부터 IBM 메인프레임을 지원하면서 강력한 통합 솔루션을 제공할 수 있게 되었다. VisiBroker는 타제품과는 달리 공급회사가 IDE 제품을 함께 공급하기 때문에 IDE와 통합되어 개발자에게 편의성을 제공한다는 특징이 있다.

5. 맺음말

이상으로 간략하게나마 OMG와 CORBA의 개요, 최근에 정의가 되고 있는 새로운 기능, 주요 상용제품 등에 대하여 설명하였다.

참고문헌

[1] Object Management Group. http://www.omg.org.

[2] Object Management Group. The

Common Object Request Broker : Architecture and Specification. Revision 2.2. Object Management Group, Framingham, MA 1998.

[3] S. Vinoski, "New Features for CORBA 3.0," CACM, vol. 41, no. 10, pp 44-52, October 1998.

[4] D. Schmidt and S. Vinoski, "Object Adapters : Concepts and Terminology," C++ Report, vol. 11, Oct. 1997.

[5] D. Schmidt and S. Vinoski, " Using the Portable Object Adapter for Transient and Persistent CORBA Objects," C++ Report, vol. 12, April 1998.

[6] Object Management Group. CORBA Messaging Joint Revised Submission. Object Management Group, Framingham, MA 1998, document orbos/98-05-05.

[7] D. Schmidt and S. Vinoski, "An Introduction to CORBA Messaging," C++ Report, vol. 11, Nov/Dec 1998.

[8] D. Schmidt and S. Vinoski, "Programming Asynchrounous Method Invocations with CORBA Messaging," C++ Report, vol. 12, Feb 1999.

[9] Object Management Group. Objects By Value Joint Revised Submission. Object Management Group, Framingham, MA 1998, document orbos/98-01-28.

[10] BEA Systems, http://www.beasys.com/weblogicserver4/.

[11] IONA Technologies, http://www.iona.com.

[12] Inprise Corporation, http://www.inprise.com/visibroker/.

김 석 원

1987 서울대학교 공과대학 컴퓨터공학과(학사)
1989 한국과학기술원 전산학과(석사)
1991 미국 IBM T.J. Waston 연구소 객원연구원
1994 한국과학기술원 전산학과(박사)
1994~현재 동양시스템하우스(주) 기술연구소 책임연구원
관심분야 : 분산시스템, 데이터베이스, 소프트웨어공학, 인공지능
E-mail : swkim@tysystemhouse.com