CORBA와 DCOM의 통합

성결대학교 금영욱*
아주대학교 장연세

1. 서 론

최상의 응용 프로그램을 개발하려면 사용 가능한 최선의 기술과 도구의 접목이 필요하다. 다양한 기술이 요구되는 분산 응용 프로그램의 경우는 더욱 그렇다. 분산 객체를 통합하는 대표적인 기술로 Common Object Request Broker Architecture(CORBA)[1]와 Distributed Component Object Model(DCOM)[2]을 들 수 있다. CORBA는 OMG에서 분산 객체 지향 시스템을 구축하기 위해 제정한 산업체 표준이고 DCOM은 Microsoft에서 Common Object Model(COM) 객체가 네트워크 상에서 상호운용될 수 있도록 COM을 확장한 것으로 데스크 톱 컴퓨터에서의 실질적인 표준이다.

CORBA와 DCOM은 서로 경쟁적인 기술로 인식되었으나 사실은 상호 보완적인 측면이 강하다. CORBA는 중요한 컴퓨터 벤더들의 지원을 받고 있으며 중위의 시스템, 또는 데이터베이스나 트랜잭션 시스템과 통합된 후위에서 수행되는 Unix 서버나 메인 프레임 서버에 적합하다. COM/DCOM은 사용하기 쉬운 Visual Basic과 같은 개발 도구들을 제공하므로 응용 프로그램의 최종 사용자 인터페이스 개발에 최상의 해결책을 제공한다.

그러므로 CORBA와 DCOM의 양쪽 요소를 포함하여 분산 시스템을 만드는 것은 바람직하다. 이렇게 하면 기존의 응용 프로그램을 취하여 최소의 프로그래밍 노력으로 한 개의 분산 응용 프로그램으로 만들 수 있다.

CORBA와 DCOM은 각기 상위에서 하는 일은 비슷하지만 하위 수준으로 내려가면 상당히 다르다. 따라서 우리는 두 시스템의 차이점을 이해할 필요가 있으며 이런 차이점을 넘어 둘을 통합하는 효과적인 방법이 요구된다. Microsoft OLE의 객체 모델은 COM과 OLE 오토메이션(automation) 두 가지가 있는데 OMG에서는 CORBA와 OLE의 두 객체 시스템간의 상호운용을 보장하기 위해 COM-CORBA 상호작용 구조(interworking architecture)를 만들었고 서로 간에 변환하는 방법을 표준화하였다[1]. CORBA와 DCOM 두 시스템에서 만든 컴포넌트를 투명하게 통합하기 위해서는 두 시스템간에 서로 다른 자료형이나 객체 참조자 등을 변환하는 OMG의 상호작용 표준에 준하는 브리징 소프트웨어가 필요하다.

본 논문에서 DCOM에 대한 기술적인 고찰을 하고(CORBA는 이 회지의 다른 논문 참조) CORBA와 DCOM의 차이점을 비교한다. 실제 예로 OLE 오토메이션과 CORBA 프로그램의 통합을 보여준다. 또한 이런 통합을 지원하는 제품들을 소개한다.

2. DCOM의 기술적인 고찰

2장의 내용은 COM 객체 모델을 중심으로 다루었으며 필요한 경우 오토메이션 객체의 내용을 언급하였다.

2.1 DCOM의 역사

처음 등장한 것이 Dynamic Data Exchange

* 정회원

(DDE)와 Object Linking and Embedding (OLE)이다. 이 때의 object는 객체 프로그래밍과는 관계가 없으며 단순히 문서 객체를 의미하였다. 이름 그대로 문서 객체가 다른 객체에 링크되거나 또는 삽입되게 할 수 있다. OLE의 문서 재사용 기능이 일반적인 프로그래밍의 문제인 코드 재사용을 위한 기능으로 확대되어 OLE 2.0이 탄생하였다. OLE 2.0은 COM을 기초로 하여 그 위에 계층적으로 만든 객체 지향적 서비스들의 집합체이다. COM은 한 컴퓨터 내에서 컴포넌트를 생성하는 기본 모델과 런타임에 컴포넌트간에 대화하는 표준을 제시하는 아키텍처이다. OLE는 이제는 Object Linking and Embedding을 뜻하지 않으며 더 이상 버전 번호도 붙이지 않는데 이는 기본적인 구조를 바꾸지 않고 계속 확장하는 아키텍처이기 때문이다.

DCOM은 COM 객체가 근거리 통신망, 원거리 통신망 또는 인터넷에서 동작할 수 있도록 COM을 확장한 것이다. 네트워크 상에 있는 COM 객체를 활성화시키고 그 메소드(method)를 호출할 수 있는 기능을 제공한다. DCOM은 COM과 함께 또는 COM 위에 만들어진 명세이고 서비스이다.

한편 Visual Basic의 Visual Basic Extensions(VBX)는 16 비트에 기반을 두었고 개방형 인터페이스를 제공하지 않았으므로 32 비트와 COM의 객체 지향적 인터페이스를 지원하는 OLE Control Extension(OCX)로 교체되었다.

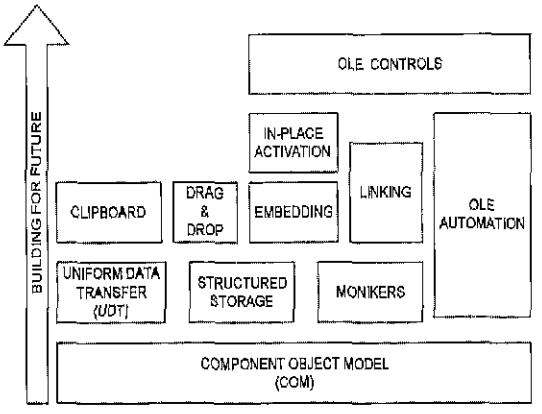


그림 1 OLE의 구조

Visual Basic이 제공하는 프로그래밍 모델은 OLE 오토메이션에 기반을 두고 있다. 오토메이션은 COM 위에 만든 객체 모델이다. 오토메이션 객체 모델은 COM 객체 모델과 유사한 점도 많지만 인터페이스 구조나 데이터 타입과 같은 중요한 점에서 다르다.

ActiveX는 Microsoft가 Java에 대응하기 위해 OCX를 확장한 것으로 다양한 인터넷 애플리케이션을 지원하기 위한 인터넷과 멀티미디어 서비스를 제공한다. ActiveX는 COM 객체와 오토메이션 객체 시스템의 재사용 가능한 콤포넌트를 구현하는 표준을 제공한다.

2.2 DCOM의 구조

그림 2에 있는 것처럼 DCOM은 COM 애플리케이션이 원격 컴포넌트를 프록시(proxy)와 스텁을 이용하여 마치 같은 기계 내에 있는 것처럼 사용하게 한다. DCOM을 사용하면 분산된 컴포넌트와 교류하기 위해 필요한 네트워크 코드를 프로그래머가 코딩할 필요가 없다.

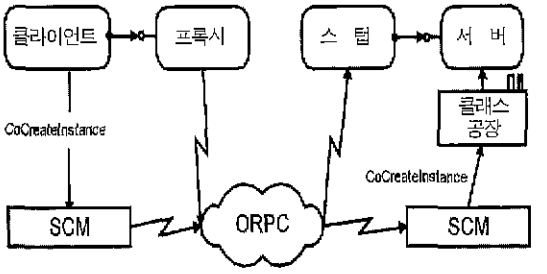


그림 2 DCOM의 구조

2.2.1 인터페이스

인터페이스는 객체가 제공하는 서비스로 메소드와 속성으로 구성된다. 각 인터페이스는 다른 인터페이스와 구별하기 위해 Interface Identifier(*IID*)라고 불리는 Universally Unique ID(UUID)를 할당받는다. DCE RPC의 IDL을 확장한 Microsoft IDL(MIDL)을 사용하여 COM의 인터페이스를 정의한다.

모든 COM 객체는 자신의 인터페이스 구현하며 또한 IUnknown 인터페이스를 반드시 상속받아야 한다. IUnknown 인터페이스는 COM 객체의 모든 다른 인터페이스를 접근하

게 하는 QueryInterface() 메소드와 AddRef()와 Release() 메소드를 가지고 있다.

오토메이션 객체는 IUnknown 인터페이스 외에 IDispatch 인터페이스를 포함하는 COM 객체이다. 오토메이션의 인터페이스 구조는 클라이언트가 인터페이스를 사용하기 위해 인터페이스의 구조를 컴파일할 때 전혀 모르도록 설계된 것이다. 많은 점에서 CORBA의 Dynamic Invocation Interface(DII)와 유사하다.

2.2.2 이진 상호 운용(Binary Interoperability)

COM은 컴포넌트간의 상호 운용성을 보장하기 위해 메소드의 호출을 위한 이진 호출 표준을 정의하였다. 이진 표준 호출은 객체에서 구현한 메소드를 가리키는 포인터의 배열인 vtable(virtual table)을 사용한다. vtable은 C++에서 유래한 것으로 C++ 객체가 자신의 가상 함수를 사용하기 위해 정의한 메모리 배열과 동일하다.

인터페이스는 메소드의 집합체이므로 인터페이스 포인터는 vtable을 가리키는 포인터를 가리키고 있다. 클라이언트가 어떤 객체의 메소드를 사용하려면 CoCreateInstance() 메소드를 호출하여 먼저 인터페이스 IUknown의 인터페이스 포인터를 얻는다. 이 포인터를 사용하여 QueryInterface() 메소드를 호출하면 원하는 객체의 인터페이스 포인터가 반환된다. 이 포인터로 그 객체의 모든 메소드를 호출할 수 있다. 이런 과정이 그림 3에 나와 있다.

오토메이션 객체는 vtable을 사용하지 않고

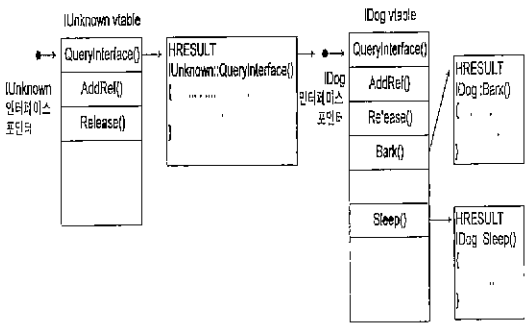


그림 3 COM의 메소드 호출 과정

IDispatch 인터페이스의 Invoke() 메소드를 사용하여 다른 오토메이션 메소드를 호출한다.

2.2.3 클래스와 서버

COM의 클래스는 한 개 이상의 COM 인터페이스를 구현한 코드의 집합체이다. 클래스는 자신이 지원하는 인터페이스를 구현한 실제 함수를 제공한다. 인터페이스가 유일한 IID에 의해 서로 구별되는 것처럼 COM 클래스도 유일한 식별자인 CLSID를 가지고 있다. 다른 컴포넌트와 교류하기 위해서 클라이언트는 적어도 한 클래스의 CLSID와 그 클래스가 지원하는 IID를 알아야 한다. 이 정보를 이용하여 클라이언트는 COM에게 객체 생성과 인터페이스 포인터를 요청할 수 있다. 한 개 또는 그 이상의 클래스가 결합하여 서버를 구성한다. 서버에 크게 두 가지 종류가 있다.

- in-process 서버는 COM 클래스를 DLL로 만들어서 서버의 클래스가 클라이언트에 의해 처음 호출될 때 클라이언트 프로세스에 로드되어 수행된다. 인터페이스 포인터를 사용하여 in-process COM 객체의 메소드를 직접 호출할 수 있다.
- out-of-process는 독립적으로 수행하는 프로그램(exe)으로 별도의 기억 공간을 갖으며 Service Control Manager(SCM)가 out-of-process 서버의 시작과 종료를 담당한다. 클라이언트는 프록시의 인터페이스 포인터를 가지며 이 포인터로 클라이언트가 프록시를 호출하고 프록시는 원격 호출을 사용하여 원격 객체의 스텁을 호출하면 스텁이 서버 객체의 메소드를 호출한다. 이 과정은 그림 4에 표시되었다.

2.2.4 객체의 생명 주기

COM 객체는 클래스 공장(class factory)에서 만들어진다. 클라이언트가 CLSID와 관련된 COM 서버의 정보를 제공하여 CoCreateInstance() 메소드로 새로운 객체의 생성을 요청하면 SCM은 서버를 찾아서 수행시키고 클래스 공장에 객체 생성을 요청한다. 객체 생성 후 COM은 클라이언트에게 프록시의 인터페이스 포인터를 알려주고 객체는 프록시를 호출함

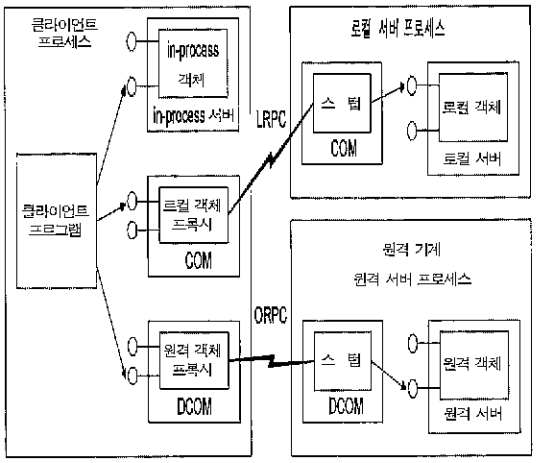


그림 4 DCOM의 클라이언트와 서버

으로 COM 객체를 직접 호출할 수 있다.

COM 객체는 참조 회수를 갖는데 AddRef()와 Release() 메소드를 사용하여 참조 회수를 증가 또는 감소시킨다. 인터페이스 포인터를 반환하는 모든 메소드는 AddRef() 메소드를 호출해야 하며 클라이언트는 인터페이스 포인터 참조가 끝나면 Release() 메소드를 호출해야 한다. 참조 회수가 0이 되면 객체를 참조하는 클라이언트가 더 이상 없는 것으로 간주하기 때문에 객체는 제거될 수 있다.

2.2.5 DCOM의 확장

DCOM은 COM이 제공하는 프로그래밍 모델을 네트워크 상에서도 동작하도록 확장한 것이다. DCOM에서 확장된 중요한 내용은 다음과 같다.

(1) 위치 투명성 : DCOM은 위치 투명성을 제공하여 객체가 네트워크 상 어디에든지 존재할 수 있게 한다. DCOM이 원격 객체를 지원하기 위해 사용하는 프로토콜은 Object Remote Procedure Call (ORPC)이다. ORPC는 DCE의 RPC를 확장하여 객체 참조자의 자료형과 객체에 대한 호출에 매개변수를 추가하였다. 클라이언트가 원격 객체를 요구하면 로컬 SCM이 원격 기계에 있는 SCM을 호출하여 객체를 생성하고 인터페이스 포인터를 반환하여 클라이언트가 원격 객체를 같은 기계에 있는 것처럼 호출할 수 있다(그림 2 참조). Lightweight RPC(LRPC) 프로토콜은 프로세스간의 통신을 효과적으로 하도록 RPC의 기능을 축소시킨 것으로 같은 기계에 있는 서버에게 사용하는 프로토콜이다.

(2) 복수 스레드의 서버 : 윈도우 NT 4.0에 자유 스레딩(free threading)이란 병행 모델이 도입되어 한 객체에 대한 복수개의 호출이 동시에 별개의 스레드에 할당되어 처리될 수 있다.

(3) 보안 : DCOM은 관리자나 개발자가 선택할 수 있는 다단계의 보안을 제공한다.

(4) 효율적인 ping : 클라이언트가 Release() 메소드를 호출하지 못하고 비정상적으로 종료하는 문제를 해결하기 위해 COM은 각 인터페이스 별로 ping 메시지를 보내었으나 DCOM은 객체 별로 ping 메시지를 클라이언트에 보내어 네트워크 트래픽을 줄인다.

3. CORBA와 DCOM의 비교

CORBA와 DCOM의 중요한 내용을 비교[3, 4, 5, 6]하여 CORBA와 DCOM의 차이점을 표 1에 정리하였다.

4. 시스템 통합

CORBA 응용 프로그램과 COM/오토메이션 응용 프로그램간의 상호 작용을 가능케 하려면 두 시스템의 객체 형식을 모두 지원하는 브리지 프로그램을 이용하여 CORBA-COM/오토메이션 변환을 지원하거나, 프로그래머가 OMG IDL 인터페이스에 정의된 속성과 메소드에 대한 정보를 COM/오토메이션 형태로 타입을 변환하여 객체 서비스를 제공하는 방법이 있다. 후자는 응용 프로그래머가 직접 프로그래밍 언어를 이용하여 구현해야 하므로 어려운 일이기 때문에 전자의 경우에 대해서 설명한다.

표 1 CORBA와 DCOM의 비교

항 목	CORBA	DCOM
벤더지원	OMG가 인증한 산업체 표준 명세, 구현은 소프트웨어 벤더들 담당	Microsoft가 만든 명세이고 구현하였음
플랫폼 지원	다양한 이질 플랫폼들 지원	윈도우와 NT, 몇 개의 플랫폼 지원
객체지향 특성	전통적인 객체 모델에 근거	전통적인 객체 모델에 근거하지 않음 (인터페이스는 관련된 함수의 집합)
재사용	inheritance	containment, aggregation
객체상태	연결간에 상태 유지	상태 유지하지 않음—인터넷 상에 잘못된 연결이 있으면 문제 야기
배치 특성	ORB가 별도로 설치되어야 함	DCOM은 윈도우와 NT에 포함
개발 도구	Java, C++ 등 다수의 언어 Delphi 지원 예정	다수의 언어 및 4GL 언어(VisualBasic, Visual C++, Delphi 등)
다중 상속	인터페이스가 다중 상속 지원	인터페이스는 다중 상속을 지원하지 않으나 객체가 지원
동적 호출(DII)	동일한 서버가 정적/동적 호출 지원	오토메이션 객체만 지원
DSI	Dynamic Skeleton Interface 지원	지원하지 않음
보안	보안 서비스가 담당	COM 객체가 2진화되어 있으므로 부당한 변경(바이러스 등의) 가능
인터페이스	OMA의 IDL로 정의 Interface Repository에 보관	DCE의 IDL을 확장한 MIDL로 정의 Type Library에 보관
예외 사항	인터페이스에 정의	인터페이스에 정의할 수 없다.
프로토콜	IIOP	Object RPC
객체 참조	객체 참조자(object reference)	인터페이스 포인터
객체 찾기와 기동	Object adapter	Service Control Manager
구현서버 등록	Implementation Repository	System Registry(Class Store)
스텁 이름	stub/skeleton(클라이언트/서버)	proxy/stub(클라이언트/서버)

4.1 상호작용 구조

최상위의 관점에서 보면, COM과 CORBA는 상당히 유사하다. COM 인터페이스는 CORBA 인터페이스와, COM 인터페이스 포인터는 CORBA 객체 참조자와 유사하다. 보다 하위의 설계 세부 사양(호출 기법, 데이터 타입 등등)들은 의미적으로 비교적 대칭 관계를 갖기 때문에 적절한 단계에서 변환이 제공되면 두 객체 시스템간의 상호작용은 가능하다[1].

OMA 상호작용 구조에 제시된 네 가지 모델은 CORBA 서버와 COM 클라이언트, CORBA 클라이언트와 COM 서버, CORBA 서버와 오토메이션 클라이언트, CORBA 클라이언트와 오토메이션 서버간의 통합이다 그림 5는 이 중 두 가지 경우를 보이고 있으며 각 모델이 제공하는 연동 방법들이 모두 다르지만 지면 관계상 그림 5의 b)의 경우를 설명한다. CORBA 클라이언트는 오토메이션 서버에게 대상 COM 객체에 대한 요청을 하는데, 이에 대한 투명한 변환과 요청의 전달이 브리지의 목표이다. 이를 위해 브리지는 뷰(view)라는 객체를 클라이언트에게 제공한다. 뷰 객체는 서버에 존재하는 대상 객체를 클라이언트에서 대리하는 객체를 의미한다. 뷰는 서버 측의 대상 인터페이스에 대한 대칭적 인터페이스인 뷰 인터페이스를 제시한다. 뷰 인터페이스의 메소드는 CORBA 클라이언트로부터의 요청을 COM 서버의 대상 객체에 대한 요청으로 변환한다. 뷰는 브리지의 한 요소이며, 브리지는 다

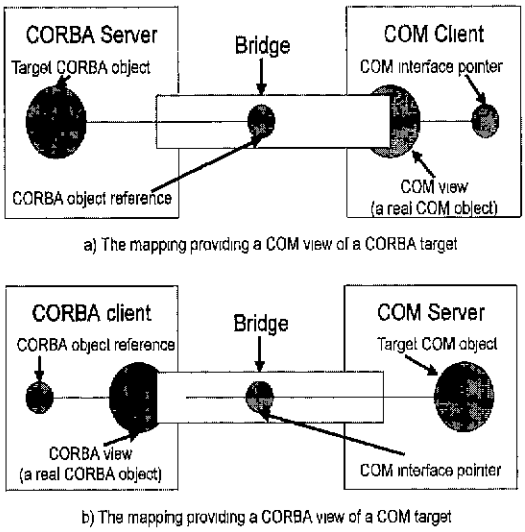


그림 5 상호 작용 구조

수의 뷰들로 구성된다[1].

클라이언트는 뷰에게 자신의 시스템 형식으로 요청을 한다. 뷰는 클라이언트의 요청을 서버 시스템 객체의 형식으로 전환하고 대상 객체에 전달한다.

4.2 상호작용 구조 구현

OMG의 COM-CORBA 또는 오토메이션-CORBA 상호작용 명세는 브리지의 실제 구현 방법은 다루고 있지 않으므로 구현 기법은 브리지 제품에 따라 다르지만 대체로 그림 6과 같이 정적 브리지(static bridge)와 동적 브리지(dynamic bridge)로 나눌 수 있다[10, 11].

원격지에 존재하는 두 시스템간의 통신은 일반적으로 버퍼 지향적이다. OMG의 상호작용 구조 명세에서 정의된 뷰를 구현할 때 프록시를 이용한다. 클라이언트 측의 프록시는 매개변수들을 버퍼에 마셜링(marshaling)하고, 서버는 버퍼로부터 전달받은 매개 변수들을 언마셜링(unmarshaling)한다. 이것은 실제로 어떤 형태의 브리지가 사용되더라도 동일하다. 일반적으로 서버에게 전달된 최초의 버퍼를 전송 버퍼(send buffer)라 하고 명령 수행 결과와 함께 반송되는 버퍼를 응답 버퍼(reply buffer)라 한다[11].

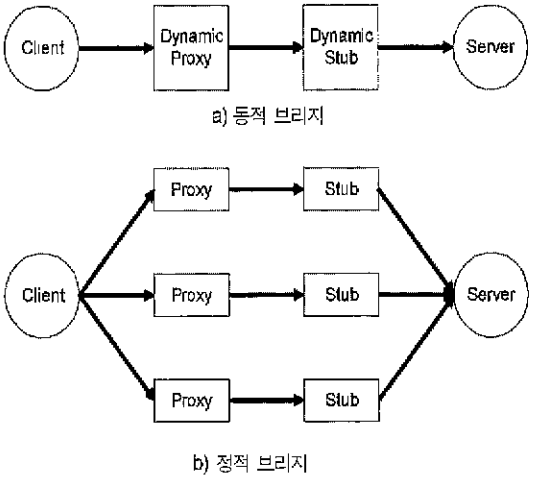


그림 6 브리지 타입

(1) 정적 브리지(Static Bridge)

IDL 번역기는 인터페이스로부터 정적 프록시와 각 인터페이스에 대한 스텁 구현 코드를 생성한다. 프록시는 인터페이스의 API를 구현한 일부 코드로써 매개 변수를 전달받고 전송 버퍼에 마셜링한다. 이 때 버퍼가 원격 서버 프로세스에게 전송되면, 서버 측의 스텁이 전송 버퍼를 언마셜링하여 서버에 존재하는 구현 객체의 해당 메소드를 호출한다. 이후 스텁이 명령 수행의 결과를 응답 버퍼에 마셜링하여 클라이언트 프록시에게 다시 전달한다. 프록시는 반환 결과를 버퍼로부터 언마셜링하고 최초로 명령을 호출한 클라이언트에게 값을 전달한다.

정적 브리지의 최대 장점은 각 명령 호출을 처리하는 특정한 코드가 존재하기 때문에 신속한 처리가 이루어진다는 점이다. 이러한 타입의 브리지는 시스템에 의해 사용되는 인터페이스들이 정형화되어 있고 인터페이스의 형태 변화가 드문 경우에 가장 적합하다.

단점은 유지보수가 어렵다는 점이다. 응용 프로그램의 개발에서 IDL 정의는 시작 단계에 불과하기 때문에 프로젝트가 진행될수록 인터페이스가 변화되며, 이에 따라 브리지 코드들도 수정해야 한다.

(2) 동적 브리지(Dynamic Bridge)

동적 브리지는 모든 객체에 대한 임의의 메

소드 호출을 하나의 프록시와 스텁이 처리한다. 이를 위해 요청 객체(request object)의 동적 생성, 요청 객체의 원격지 전송, 요청 객체의 총체적 수용 등이 도입된다. 동적 브리지는 자료에 대한 런타임 타입 정보가 필요하므로 DCOM에서는 시스템 레지스트리와 타입 라이브러리를 이용하고 CORBA에서는 인터페이스 저장소를 이용한다. 모든 동적 브리지의 기본 요소인 동적 마셜기(dynamic marshaller)가 런타임 정보를 이용하여 동적 마셜링을 수행한다.

동적 브리지의 장점은 브리지 소프트웨어를 수정할 필요가 없고, 브리지를 유연하게 배치할 수 있으며 인터페이스에 대한 정적 프록시/스텁 코드가 없기 때문에 자원의 사용량을 현저하게 절약할 수 있다[11].

4.3 브리지를 이용한 시스템 구성

브리지를 이용한 시스템 구성 방안은 세 가지가 있다. 첫째 브리지를 클라이언트와 동일한 호스트에 배치한다. 둘째 브리지를 서버와 동일한 호스트에 배치한다. 셋째 브리지를 독립된 호스트에 배치한다. 브리지가 배치된 위치에 따라 시스템간의 통신 방법이 결정된다. 예를 들어 첫 번째 방법을 이용해서 CORBA 클라이언트와 오토메이션 서버를 구성하면, 클라이언트와 브리지는 동일 호스트 내에서 IIOP를 이용해서 통신하지만 네트워킹은 발생하지 않는다. 브리지와 서버는 서로 물리적으로 격리되어 있으므로 네트워크를 통해 DCOM 통신 즉, ORPC 프로토콜로 상호 작용을 한다. 두 번째 방법을 이용해 시스템을 구성하면 클라이언트와 브리지간에는 네트워크 상에서 IIOP를 통해 통신하고 브리지와 서버는 LRPC 프로토콜로 통신한다.

4.4 브리지를 이용한 시스템 통합

다음 예는 오토메이션 Visual Basic 클라이언트가 CORBA 서버를 접근하며 정적 브리지를 사용하는데 그림 5의 브리지 형태 중 a)번의 경우에 해당한다[12].

(1) CORBA IDL 작성

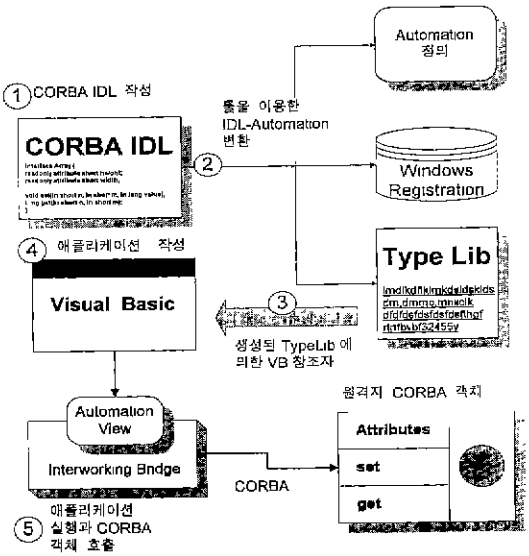


그림 7 통합 과정 개요

[리스트 1]은 "height"와 "width"라는 속성과, "set"과 "get"이라는 메소드를 갖는 "Array"라는 객체를 선언한 CORBA 인터페이스이다.

[리스트 1] Array CORBA IDL

```
interface Array {
readonly attribute short height;
readonly attribute short width;
void set(in short n,
      in short m, in long value);
long get(in short n, in short m);
};
```

(2) 툴을 이용한 IDL-오토메이션 변환

① CORBA 인터페이스를 MIDL 인터페이스로 변역한다.

② 번역된 MIDL이 시스템 레지스트리에 자동으로 등록된다.

③ CORBA IDL을 인터페이스 저장소에 등록한다.

아래의 [리스트 2]는 IONA의 Orbix를 사용하여 Array 인터페이스를 MIDL로 변환한 결과이다.

[리스트 2] 변환된 Array MIDL

```
interface IArray : IUnknown
  {
   HRESULT _get_height
           ([out] short *val);
   HRESULT _get_width
           [out] short *val);
   HRESULT get ([in] short n,
           [in] short m, [out] long *val);
   HRESULT set([in] short n,
           [in] short m, [in] long value);
  };
```

(3) 생성된 타입 라이브러리에 의한 Visual Basic 참조자 생성

① Visual Basic 창에서 "프로젝트" 메뉴 중 "참조자"를 선택한다. "가용 참조자"나 타입 라이브러리 목록이 활성화된다.

② 앞의 (2)번 단계에 의해 생성된 타입 라이브러리를 선택한다.

③ "DIArray"가 가용 참조자 리스트에 표시된다.

(4) Visual Basic 클라이언트 작성

① CORBA 객체에 대한 변수를 선언한다.

② Array 객체의 인스턴스를 생성한다.

③ 호출에 필요한 매개변수 설정한다.

④ set이나 get 메소드를 호출한다.

(5) 통상적인 CORBA 서버 개발

5. 제품 소개

현재 국내외에 보급되고 있는 COM-CORBA 브리지는, 브리지 전문 업체인 Visual Edge의 ObjectBridge와 상용 ORB 업체 중 유일하게 제공하고 있는 IONA의 OrbixCOMet이 있다.

5.1 ObjectBridge

ObjectBridge를 만든 Visual Edge는 COM-CORBA 브리지뿐만 아니라 SAP R/3-CORBA 브리지도 제공하는 특성화된 기업이다. 이러한 업체의 특징 때문에 ObjectBridge는 상용 ORB인 Orbix, VisiBroker, PowerBroker와 NEO 등을 모두 지원한다. Inprise는 "Enterprise Client ObjectBridge"라는 이름으로 ObjectBridge를 판매하고 있다. 이 제품의 기능적 특성을 살펴보면 다음과 같다[13].

① OMG의 COM-CORBA 상호작용 구조 지원

② Microsoft가 COM과 DCOM에 제공하는 공식 인터페이스를 충실히 지원

③ 런타임 객체 구성에 필요한 Just-in-Time IDL 컴파일러 제공

④ 제품 설치시 선택한 주 ORB (master ORB)로 설정된 CORBA 제품의 고유 API들 지원

⑤ COM/오토메이션 객체와 CORBA 객체 간의 양 방향 호환 제공

⑥ CA의 Unicenter, HP의 Open View나 Tivoli의 TME 등을 통한 폭넓은 관리 기능 제공

다른 제품에 비하여 예외 사항 처리 등에 있어 다소 복잡하고, 브리지의 별도 구매로 인한 비용이 추가되는 단점이 있다.

5.2 OrbixCOMet

IONA는 Orbix 구 버전에서 내장하여 제공하였던 정적 OLE-CORBA 브리지를 Microsoft와의 기술 협약 이후 동적 브리지 기능 등을 보강하여 OrbixCOMet으로 별도 제품화했다. 처음 출시되었을 때에는 단독 제품이었지만 현재는 다른 제품들과 함께 Orbix 3에 포함되어 있다. ObjectBridge처럼 OLE(COM, ActiveX) 프로그래밍을 지원하는 모든 프로그래밍 언어를 지원하고 OMG 표준을 만족하며 그 특징은 다음과 같다[14].

① OMG의 COM-CORBA 상호작용 구조 지원

② 기존 COM, OLE 및 CORBA 오브젝트들과 통합

③ COM/오토메이션 객체와 CORBA 객체 간의 양 방향 호환 제공

④ OMG의 공식 API와 Orbix의 다양한 API 모두 제공

⑤ 성능 저하가 유발되지 않으면서도 로컬 캐쉬 기능 제공
⑥ OMG 상호작용 구조가 발표되기 전에 제공되었던 IONA의 정적 브리지를 이용해 작성된 레거시 코드들과 95%이상의 호환성 제공

6. 결 론

DCOM에 대한 기술적인 고찰과 CORBA와 DCOM을 비교하였으며 또한 둘간의 통합을 하는 상호작용 구조와 통합을 지원하는 제품을 살펴보았다. CORBA가 객체 지향형 분산 처리를 표방했다면 COM은 초기에 데스크 톱 컴퓨터에서 단지 다른 프로그램과의 자원 공유를 목표로 하여 그 기원과 기본적인 플랫폼의 성격이 다르지만 이제는 두 기술의 역할과 기능이 서로 유사하게 되었다.

CORBA는 이론적으로 지금까지 정의된 가장 최선의 미들웨어이지만 사용자가 얻는 실제 혜택은 CORBA를 구현하는 제품의 질에 달려 있다. 특히 CORBA 제품간에 상호 운용성을 보장하기 위해 각 제품의 독자적인 기능 추가를 억제하고 CORBA의 표준을 충실하게 구현하여야 한다. 또한 사용을 쉽게 하기 위한 개발 도구의 지원이 요구된다.

데스크 톱에서 실질적인 표준인 DCOM이 다른 플랫폼에 확산되기 시작하였는데 윈도우에서와 동일한 지원이 요구된다. 다른 플랫폼에서 DCOM의 수용 정도가 DCOM의 경쟁력을 결정할 것이다.

또한 시장을 누가 먼저 선점하느냐 하는 것이 앞으로의 분산 객체 미들웨어의 장래를 결정하는 중요한 변수이다. 그러나 서론에서 지적하였듯이 두 기술은 각자의 장점으로 인해 상호 보완적인 성격이 강하므로 둘 사이의 통합으로 인한 공존이 큰 흐름이 될 것이다. 우리의 경험에 의하면 CORBA와 DCOM의 많은 차이점에도 불구하고 5장에서 소개한 도구들을 사용하면 실제 통합하는 일의 복잡성은 CORBA 또는 DCOM만을 이용한 업무 개발과 큰 차이가 없다. 따라서 양 기술의 장점을 사용하여 상승 효과를 얻는 분산 객체 시스템을 개발하는 것이 앞으로 방향이 될 것이다.

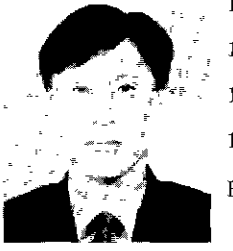
참고 문헌

[1] OMG, The Common Object Request Broker: Architecture and Specification 2.2, OMG, MA, 1998, ftp://ftp.omg.org/pub/docs/formal/98-07-01.pdf.
[2] DCOM architecture - White Paper, Microsoft Corporation, 1998. URL:http://www.microsoft.com/com/wpaper/default.asp#DCOMpapers.
[3] Robert Orfali and Dan Harkey, Client/Server Programming with JAVA and CORBA 2nd edition, John Wiely & Sons, Inc., 1998.
[4] A Detailed Comparison of CORBA, DCOM and Java/RMI, Gopalan Suresh Raj, URL: http://www.execpc.com/~gopalan/misc/Compare.html.
[5] DCOM and CORBA Side by Side, Step by Step, and Layer by Layer, P. Emerald Chung et al, http://www.research.att.com/~ymwang/papers/HTML/DCOMnCORBA/S.html.
[6] White Paper - Middleware: CORBA and DCOM, Ronald J. Norman, URL : http://www-rohan.sdsu.edu/faculty/rnorman/papers/Whitepaper.zip.
[7] Corry et al, COM/DCOM Primer Plus, SAMS, 1999.
[8] Hoque and Sharma, Programming Web Components, McGrawHill, 1998.
[9] William Rubin and Marshall Brain, Understanding DCOM, Prentice Hall, 1999.
[10] Ronan Geraghty and Sean Joyce, COM-CORBA Interoperability, Prentice Hall, 1999.
[11] Michael Rosen and David Curtis, Integrating CORBA and COM Applica-

tion, John Wiely & Sons, 1998.
[12] Orbix ActiveX Integration Guide, IONA.
[13] ObjectBridge Programming Guide, Visual Edge.
[14] OrbixCOMet Programming Guide, IONA.

금 영 옥

1978 서울대학교 수학과(이학사)
1978~1981 (주) 대우 프로그레머
1982~1993 IBM 시스템 엔지니어
1992 서강대학교 대학원 정보처리학과(이학석사)
1997 서강대학교 대학원 전자계산학과 박사
1997~현재 성결대학교 컴퓨터학부 교수
E-mail:ywkeum@hana.sungkyul.ac.kr



장 연 세

1993 서울산업대학교 전자계산학과(공학사)
1995 수원대학교 대학원 전자계산학과(이학석사)
1997 아주대학교 대학원 컴퓨터공학과 박사 과정 수료
1998~1999. 2 (주)필컴 전임 컨설턴트
E-mail : cgys@ce.ajou.ac.kr

'99 정례회의 및 편집위원회 연간일정표

월 별	정 례 회 의		편 집 위 원 회	
	상임이사회	정례이사회	논 문 지	학 회 지
1월	8일(금) 17:00		29일(금) 16:00, 전체회의	20일(금) 16:30
2월	5일(금) 16:00	26일(금) 18:00	26일(금) 16:00	19일(금) 16:30
3월	5일(금) 16:00		26일(금) 16:00, 전체회의	19일(금) 16:30
4월	9일(금) 16:00	16일(금) 17:00	30일(금) 16:00	23일(금) 19:00
5월	7일(금) 16:00		28일(금) 16:00, 전체회의	21일(금) 16:30
6월	11일(금) 16:00	25일(금) 17:00		18일(금) 16:30
7월	9일(금) 16:00		2일(금) 16:00 30일(금) 16:00, 전체회의	16일(금) 16:30
8월	6일(금) 16:00	10일(금) 17:00	27일(금) 16:00	20일(금) 16:30
9월	3일(금) 16:00		17일(금) 16:00, 전체회의	17일(금) 16:30
10월	8일(금) 16:00	15일(금) 17:00	29일(금) 16:00	22일(금) 19:00
11월	5일(금) 16:00		26일(금) 16:00, 전체회의	19일(금) 16:30
12월	3일(금) 16:00	17일(금) 17:00	24일(금) 16:00	17일(금) 16:30

※ 회의일정은 사정에 따라 변경될 수 있음.