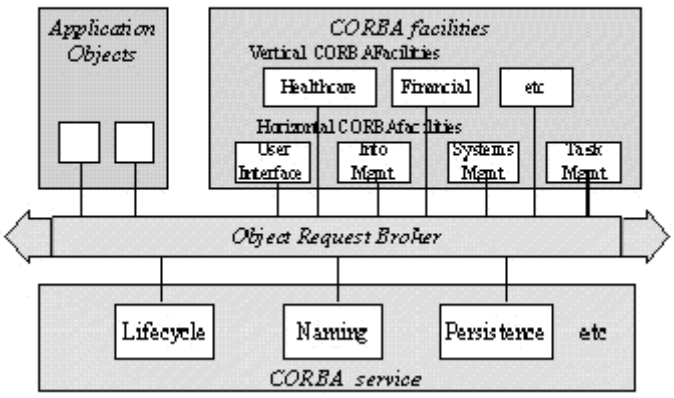
CORBA의 개념과 활용 (II)

OMG(Object Management Group)라는 비영리 단체는 1989 년 4 월에 설립되었다. 이 단체는 현재 존재하는 객체지향 기술을 밑바탕으로 하여 프로그램들을 결합하기위한 산업 표준안을 제정하기 위해 600 개 이상의 컴퓨터 단련 단체 및 기업의 연합체로 구성되어 있다. 여기에서 객체지향 기술을 기반으로 이기종의 분산 환경을 지원하기위한 표준 기술을 제정하였는데 이 표준이 OMA(Object Management Architecture)이다.
이 기능 중에서 CORBA 는 컴퓨터 H/W 의 내부 버스처럼 프로그램 사이에서도 서로의 위치에 관계없이 서로를 사용할 수 있는 기능을 제공한다. 그러므로, CORBA 는 OMA 중에서도 가장 중요한 요소이다.
이번 장에서는 CORBA에 대해 더 깊이 알아볼 것이다.

OMA 와 ORB (Object Request Broker)

CORBA 에서는 ORB 가 가장 중요하다. 바로 이 ORB 때문에 만들어진 CORBA 간의 호환성 문제가 있다. 델파이 4 에서는 비지제닉에서 제작한 비지 브로커(Visibroker)를 ORB 로 사용한다. 대표적인 ORB 로는 비지제닉의 비지 브로커 이외에도 아이오나의 오빅스가 있다. 그 밖에도 CORBAPLUS, FNORB, OMNI-ORB2, JTRADER, MICO, JOE, JYLU, JACORB, ELECTRA, ILU, OAK, DOME, CHORUS/COOL ORB 등의 여러 회사에서도 ORB 를 제작하고 있다.
OMA 는 객체를 작성하는 구조인데, 이 OMA 에는 분산환경에서 통신을 담당하는 CORBA 와 객체를 조작하는 기본기능을 정의한 COSS(Common Object Service Specification), 그리고 기타 기본 기능과 개발자가 작성한 각종 응용 객체들로 구성된다.
이 중에서 CORBA 서비스는 객체들이 연결하기 위한 기본기능을 제공하는데 이 기능에는 객체의 생명주기 서비스(Object Lifecycle Service), 명명 서비스(Naming Service), 디렉토리 서비스(Directory Service)등이 있고, 객체지향의 접근 방법인 온라인 트랜잭션 처리(On-line Transaction Processing)과 트레이더 서비스(Trader Service)등을 포함하고 있다. 이러한 CORBA 서비스는 객체의 통신과 연결에 필요한 서비스를 제공하고, CORBA Facilities 는 필요한 서비스를 제공한다. 대표적인 복합문서 관리(Compound Document Management) CORBA Facility 는 복합문서의 컴포넌트를 접근할 때 표준화된 방법을 지원한다. 이러한 CORBA Facilties 는 크게 두가지로 구분될 수 있는데, 하나는 Horizintal Facilties 로 복합문서 서비스가 이에 준하는 것으로 전반적인 영역에서 사용할 수 있는 서비스이고, 또 다른 하나는 Vertical Facilties 로 특정한 곳에만 사용하는 서비스를 말한다.
다음의 그림은 OMA 의 구조를 나타내고있다.

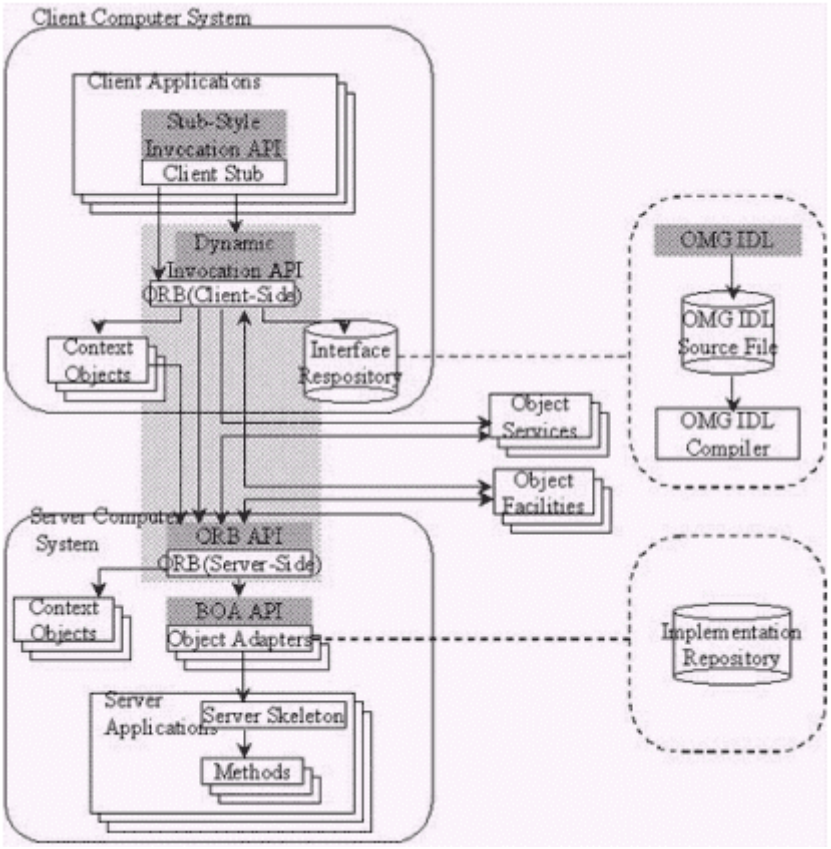
OMA



OMG 는 1990 년 OMA 를 발표한 이래, 1997 년에 CORBA 스펙 2.1 을 발표했으며 현재 CORBA 3 규격까지 나와있다.

CORBA 의 동작 원리

이번에는 CORBA 의 동작 원리에 대해서 자세히 살펴보자. 먼저 CORBA 의 구조는 전체적으로 다음 그림과 같이 표현할 수 있다.



해당 그림을 살펴보면서 CORBA 로 구현된 시스템의 호출 순서에 대해 알아보자.

- 클라이언트 프로그램은 OMG IDL 로 정의된 객체들의 동작 들에 대한 요청을 ORB 를 통하여 호출한다. 이때 호출하는 형태는 Stub-style invocation(스텁 호출), Dynamic invocation(동적 호출)이 있는데, 2 가지 방식을 혼용할 수 도 있다.

1. 스텁 호출 (Stub style)

클라이언트 프로그램이 미리 만들어진 인터페이스를 통해 정의된 오퍼레이션에 대하여 링크하는 방법이다.

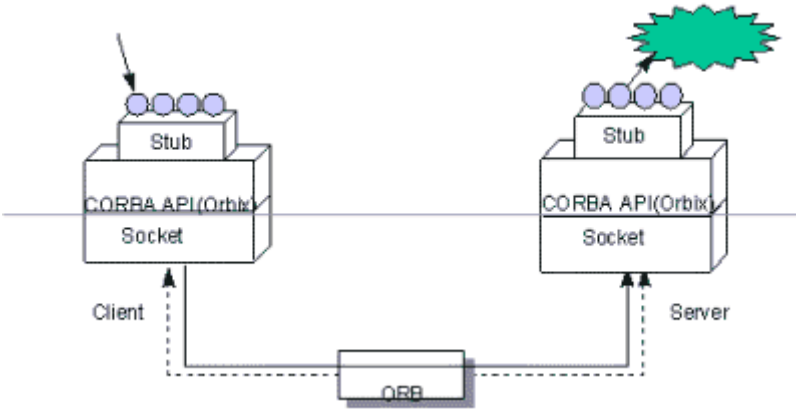
2. 동적 호출 (Dynamic style)

클라이언트 프로그램이 인터페이스 저장소를 통하여 동작하도록 실행 시간에 만들어질 것을 요구한다. 클라이언트와 서버는 ORB(Object Request Broker)를 사용하여 통신하므로 서로간의 정보는 필요가 없게 된다. 클라이언트는 ORB 에서 필요한 요청을 하고 ORB 는 필요한 요청을 서버의 구현에서 선택해서 해당 되는 메소드를 구현부에 보낸다.

컨텍스트 객체(Context Object)는 해당 행위로 전달되지 않는 정보나 서버에 대한 선택, 환경 등의 정보를 가지고 있다. 보통 클라이언트 어플리케이션에서 서버를 호출할 경우에 컨텍스트 객체를 참조하게 된다. 인터페이스 저장소는 네트워크 상에 있는 프로그램들과 데이터 객체에 대한 인터페이스를 모두 가지고 있다. 이 인터페이스 저장소는 동적 호출시에 필요한 정보를 제공한다.

- CORBA 의 서버 프로그램 구조

그 구조는 클라이언트 객체에서 요구를 할 경우, 이 요구를 받아들여 동작하기 위한 구현부(Implementation)를 하나 이상 가지고 있다. 이러한 요구를 클라이언트는 ORB 를 통해서 하게 되므로, ORB 는 BOA(Basic Object Adapter)를 사용하여 필요한 구현 내용을 선택하고 해당 구현 내용을 요구한다. 이때 BOA 는 해당 구현부 내부의 메소드 들을 호출하기 위해 서버 스켈레톤(Server Skeleton)을 사용한다.
객체 어댑터(Object Adapter)는 서버의 객체를 관리하는 중요한 도구인데, 이 역할은 구현부를 ORB 에서 호출할 수 있도록 만들어 주는 것이다. 일반적으로 이러한 객체 어댑터를 BOA 라고 부른다. 그러므로, 모든 CORBA 제작사는 그들의 시스템 일부분에서 BOA 를 지원해야 한다. 그러므로, 서버는 이러한 BOA 와 객체의 작업에 대한 메소드 들과의 연결을 제공하며 각각의 연결에 필요한 정보를 가지고 있는 것이다. 이상의 설명을 다음 그림과 같이 표현할 수 있다.



IDL 에 대하여

IDL 은 Interface Definition Language 의 약자인데, 말 그대로 객체의 인터페이스를 정의하는 언어라는 뜻이다. 쉽게 이야기해서 델파이에서 만들어진 클래스를 CORBA 에서 호출할 수 있도록 중간에서 연계해주는 언어이다.
델파이에서는 타입 라이브러리 에디터를 이용해서 이러한 IDL 을 간단하게 만들 수 있다.

다만, 다른 언어에서 사용한다면 이러한 IDL 의 코딩 방법 쯤에 대해서는 알고 있어야한다.

- IDL 문법

IDL 의 문법은 C++, 자바와 유사하다. 직접 에디터를 사용해서 만들 수 있으며, 이렇게 직접 제작한 텍스트 파일을 이용하여 스텁과 스켈레톤을 생성하려면 델파이 4 의 BIN 디렉토리의 TLIBIMP.EXE 파일을 이용하여 IDL 코드를 컴파일할 수 있다.

1. 주석 (Comment)

기본주석은 //, /* .. */로 한 줄 주석과 여러 줄 주석을 사용할 수 있다.

2. 예약어

보통은 알고 있는 일반적인 C, 자바 등의 예약어와 동일하지만 약간 다른 부분만 설명하도록 하겠다.

예약어	내 용
any	C 나 C++의 void* 나 오브젝트 파스칼에서의 Variant 데이터 형처럼 데이터 타입만 받아들이는 것이 아니라 객체를 포함한 모든 타입의 변환이 가능한 데이터 형을 말한다.
in	클라이언트에서 구현부로만 전달되는 매개변수를 지칭한다.
out	구현부에서 클라이언트로만 전달되는 매개변수를 지칭한다.
inout	양방향 전달이 가능한 매개 변수를 지칭한다.
oneway	보통 메소드가 동기적으로 수행되는 반면에 비동기적으로 수행할 수 있는 메소드를 지칭하는 것으로 결과를 기다리지 않고 나중에 처리할 수 있는 메소드이다.

- 데이터 형 (data type)

IDL 에서 만들어내는 인터페이스에서 해당 작업의 매개 변수나 반환되는 값의 데이터 형태를 지정하는데 사용한다.

- 기본 데이터 타입 (char, short, long, float)
- 구조체 타입 (struct, union, enumeration)
- 템플릿 타입 (sequence, array, string)

물론 앞에서 설명한 any 데이터 형도 가능한데, 이를 사용하면 수행 속도가 느려지는 것은

두말할 나위도 없다.

1. 구조체 데이터 형 (Constructed Types)

구조체 데이터 형에는 3 가지가 있다. struct, union, enumeration 의 3 가지가 있다.

- structure

일반적인 구조체와 동일하다.

```
struct 구조체명 {
데이터타입 변수명;
데이터타입 변수명;
}
```

- union

일반적인 공용체와 동일하다. 서로 다른 데이터 형과 크기를 공유하기 위한 방법이 있을 경우에 사용한다. C(++)와 다르게 각 데이터 멤버는 case 라벨과 함께 선언한다.

```
union token switch (long) {
case1: char 변수명;
case2: float 변수명;
default: long 변수명;
};
```

- enumeration

데이터 멤버를 순서적으로 나열할 경우에 사용한다.

```
enum workday {Monday, Tuesday, Wednesday, Thursday, Friday};
```

2. 템플릿 데이터 형 (Template types)

enumeration 과 string 두 종류의 템플릿 타입을 제공한다.

- enumeration

일차원 배열을 선언하는 데이터 형이며 최대 크기가 지정되어 있으면 바운드(bounded) 시퀀스라 하고, 최대 크기가 없으면 언바운드(unbounded) 시퀀스라고 한다. 해당 길이는 동적으로 변경되며 선언할 수 있는 데이터 형은 IDL 의 모든 데이터 형이 가능하다.

- string

시퀀스와 동일한 방법으로 사용하는데, 시퀀스와 다른점은 문자형 만을 사용할 수 있다는 점이다.

3. 배열 (array)

다차원 배열을 선언하는 경우에 사용한다.

4. 상수형 (constant type)

일반적으로 사용되는 상수와 동일하다. 사용할 수 있는 데이터 형으로는 boolean, short, char, string, double, float, long, unsigned long, unsigned short 등이 있다.

5. 인터페이스 (interface)

클라이언트에서 서버 객체에 서비스를 요청할 때에 처리해주는 속성과 작업을 정의한 세트를 말한다.

- 작업 (operation)

인터페이스의 몸체에 선언되는 것으로 구성은 작업 매개 변수, 예외처리(exception), 반환값의 형태, 매개변수 전달방향 등으로 구성된다.

1. 예외 처리

서비스 요청 시에 예외상황이 발생할 경우 클라이언트에 전달하는 자료구조로 보통은 시스템이 미리 정한 예외처리와 개발자가 설정한 예외처리의 두 가지 종류가 있다. 주의할 점은 이 예외처리에는 사용자가 정한 작업의 매개변수나 데이터 형이 될 수 없다. 그리고 예외처리를 위한 raises 라는 키워드가 있다.

2. 컨텍스트

구현 객체의(implemantation object)의 작업에 영향을 줄 수 있는 클라이언트의 환경 요소와 관련된 리스트를 컨텍스트라 한다. 이 컨텍스트를 사용하기 위해서는 raises 문장 바로 뒤에 context(context1, context2, ...) 형태로 선언하여 주기만 하면 된다. 그러나 이 문법은 가능한 사용하지 않는 것이 좋다.

3. Oneway 작업(operation)

Oneway 란 작업을 호출하고 결과를 기다리지 않는 것을 선언하는 것으로, 클라이언트에서 호출만 하고 결과에 대해서는 책임지지 않는 호출방법을 의미한다. 다만, 이 Oneway 작업은 다른 작업보다다 먼저 선언되어야 하는 점만 주의하자.

4. 속성 (Attribute)

속성은 클라이언트가 어떤 변수에 값을 설정하거나 검색할 때에 쉽게 IDL 로 표현하기 위해서 만들어진 형식이다. 속성은 readonly 를 사용할 수 있고, 해당 값을 설정하고 읽어오는 두가지 형태의 일만 수행한다. 간단히 보면 변수 선언하는 것과 비슷하나 실제로는 일종의 작업(operation)이다.

- 상속 (Inheritance)

일반적인 객체지향 언어(?)라면 상속이 지원되는 것이 당연할 것이고, 이 상속개념을 사용하면 한번 만들어진 IDL 을 사용하여 새로운 파생 인터페이스를 만들어 낼 수 있다.
이때 파생되는 인터페이스는 간접(indirect)과 직접(direct)기반 인터페이스로 구분된다.

1. 간접 기반 인터페이스

만들어진 파생 인터페이스의 기반 인터페이스가 다른 기반 인터페이스에서 파생된 인터페이스일 경우를 말한다.

2. 직접 기반 인터페이스

만들어진 파생 인터페이스의 기반 인터페이스가 기본적으로 만들어진 인터페이스일 경우를 말한다.

파생 인터페이스를 만들 경우에 동일한 작업이나 속성 이름을 가지고 있는 기반 인터페이스로부터 상속 받을 수 없고, 또한 상속받은 작업이나 속성의 이름을 재정의 할 수 없다. 이 점이 일반적인 상속과 다른 점이다.

- 모듈(Module)

오브젝트 파스칼의의 유닛 개념과 동일한 것으로, namespace 를 제공하기 위해 IDL 의 범위를 설정하는 것이다. 이때 외부의 식별자를 사용하기 위해서는 '::'(name resolution operator)앞에 모듈 이름을 붙여 사용한다.

비지브로커(Visibroker)

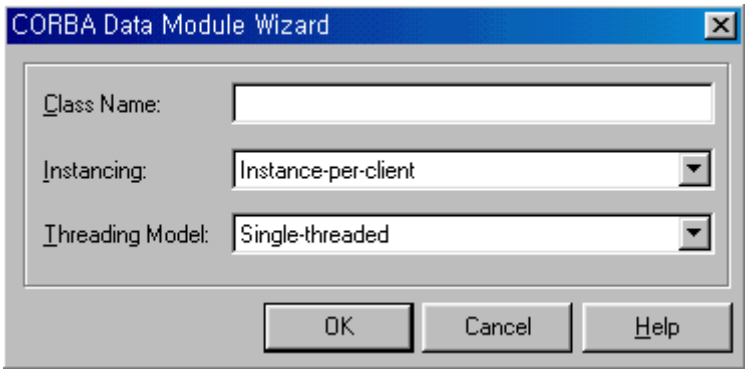
델파이 4 에서 지원되는 CORBA 의 ORB 는 비지제닉사의 비지브로커이다. 현재 개발된 CORBA 의 ORB 중 다양한 지원과 강력한 성능을 기대할 수 있는 ORB 이다.
현재 지원되는 언어는 자바와 C++, 그리고 델파이이다. 그리고 자바 표준의 JDBC 와 연결하기 위한 Visichannel for JDBC 도 지원한다. 그리고 여러가지 서버도 지원하는데 VisiBroker Event Service, VisiBroker Naming Service, VisiBroker Productivity Tools 등의 다양한 서비스도 지원하며 새롭게 VisiBroker ITS (트랜잭션 서비스) 도 지원한다.
넷스케이프사에서는 인터넷에서의 강력한 C/S 기반 어플리케이션을 위해 CORBA 자바 기술 이용에 관한 인증 체결에 대한 기술을 지원한다. 그리고, 오라클사에서는 비지제닉의 CORBA 관련 분산객체기술 인증을 체결하여, 이 기술과 함께 객체, 자바 기술, 인트라넷에 적용하여 어플리케이션을 개발하고 있으며, 노벨사에서는 인트라넷웨어 서버 플랫폼에서 비지브로커 사용에 관한 인증을 체결하였다. CORBA, IIOP 관련 개발자 지원과 인트라넷웨어에서 분산 어플리케이션 배치를 가능하게 하고 있다. 사이베이스에서는 비지제닉사의 비지브로커와 관련 기술에 관한 인증을 체결하여, 사이베이스의 새로운 트랜잭션 서버에서 자바, C++ 관련 비지브로커 제품군을 지원한다.
델파이를 설치하면 다음의 4 가지 유틸리티가 설치 된다.

프로그램 명	내 용
Object Activation Daemon	클라이언트에서 서버를 호출할 경우 자동으로 동작시켜 주는 프로그램
Visibroker Reg-Edit tool	ORB 의 설정 및 자바 머신의 환경 등을 등록한다.
Visibroker SmartAgent	실제 ORB 를 구성하는 메인 모듈, 이 모듈이 동작해야 CORBA 의 ORB 가 동작한다.
Visibroker SmartFinder	연결된 HOST 및 네트워크에 존재하는 CORBA 모듈을 검색한다.

이중에서 가장 중요한 Smart Agents 는 분산환경을 지원하기 위한 ORB 의 기본적인 구성을 지원한다. 실제 구동은 point-to-point 의 UDP 프로토콜을 사용하므로 일반적인 TCP/IP 환경이 지원되는 곳에서 사용한다.

CORBA 서버의 제작과 구동

ORB 에서 클라이언트가 서버를 호출할 경우에 서버가 자동으로 동작하게 하는 유틸리티는 비지브로커에서 OAD 를 사용하여 구동한다. 이 OAD 는 네트워크의 한 서버에서만 동작하고 있으면 자동적으로 동작한다.
델파이에서 CORBA 서버를 만드는 방법은 CORBA 데이터 모듈 위저드를 이용하거나, CORBA 객체 위저드를 사용하여 만드는 2 가지 방법이 있다.
File|New 메뉴의 Multitier 탭에서 CORBA DataModule 을 더블 클릭하면 다음의 화면이 나타난다.



여기서 Class Name 에 데이터 모듈의 이름을 입력한다음 'OK'를 선택하면 기본적인 CORBA 데이터 모듈을 만들 수 있다. 여기서는 'Test'로 이름을 입력하도록 하자. 그리고, 만들어진 데이터 모듈의 유닛을 각각 U_ExamSvr1.pas, ExamSvr1_TLB.pas, U_ExamSvrImpl1.pas 로 저장하고, 프로젝트 파일을 ExamSvr1.dpr 로 저장하도록 하자.
이중에서 U_ExamSvr1 은 일반적인 어플리케이션을 수행하기 위한 빈 유닛이다. 이것은 별다른 점이 없다.
ExamSvr1_TLB.pas 유닛은 CORBA IDL 에 해당되는 파스칼 소스 코드이다. 실제 코드를 자동으로 생성시켜 주므로 개발자는 타입 라이브러리 에디터에서 필요한 메소드나 모듈, 속성 등을 만들면 자동적으로 IDL 로 전환되며, 스텁과 스켈레톤을 지원하기 위한 코드를 생성한다. U_ExamSvrImpl1 는 타입 라이브러리 소스 코드에 해당되는 클래스를 구현하는 유닛이다. 아마도 다음과 같은 유닛이 생성되었을 것이다.

```
unit U_ExamSvrImpl1;
```

```
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ComObj, VCLCom, StdVcl, BdeProv, DataBkr, CorbaRdm, CorbaObj,
  ExamSvr1_TLB;

type
  TTest = class(TCorbaDataModule, ITest)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Test: TTest;

implementation

{$R *.DFM}

uses CorbInit, CorbaVcl;

initialization
  TCorbaVclComponentFactory.Create('TestFactory', 'Test', 'IDL:ExamSvr1/TestFactory:1.0', ITest,
    TTest, iMultiInstance, tmSingleThread);
end.
```

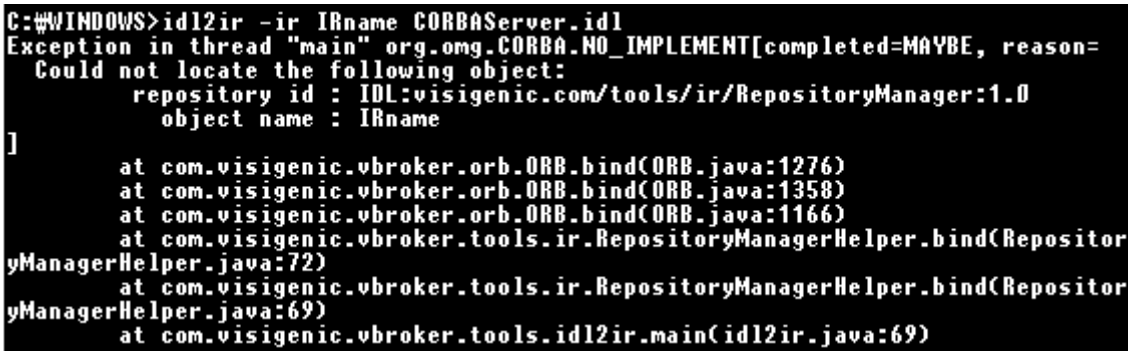
만들어진 TTest 클래스는 TCorbaDataModule 를 상속받아 개발자가 선언한 ITest 인터페이스를 구현한 CORBA 데이터 모듈이다.
이렇게 만들어진 인터페이스는 인터페이스 저장소에 등록되며 보통은 해당 서버 어플리케이션을 한번 실행하면 해당 내용이 자동으로 등록된다.
이때 만들어진 CORBA 서버가 클라이언트에서 호출할 때에 자동으로 동작하게 하려면 OAD 를 사용하여 등록하여야 한다.

1. OAD 를 사용하여 등록하기

irep IRname 을 DOS 커맨드에서 실행해 보자. 이 프로그램은 인터페이스 저장소 역할을 한다. 이곳에서 IDL 을 로드하여 필요한 인터페이스를 인터페이스 저장소에 저장할 수 있다. 지원되는 것은 일반 IDL 과 자바, C++을 지원한다. 단순히 커맨드 모드에서 사용하려면 irep –console IRname [file.idl]을 사용하여도 무방하다.
델파이 4 의 Demo 중에 CORBA 서브 디렉토리의 DataModule 서브 디레토리에 있는 프로젝트를 읽은 다음 CORBAServer_TLB.pas 를 CORBAServer.idl 로 출력하고, 해당 IDL 을 등록해 보자.
irep IRname CORBAServer.idl 이라고 DOS 명령을 사용하면 인터페이스 저장소에 인터페이스가 등록된다. 이렇게 만들어진 IDL 을 다시 배치하려면 idl2ir 유틸리티를 사용하게 된다. 사용법은 다음과 같다.

idl2ir –ir IRname –replace file.idl

그러므로 idl2ir –ir IRname CORBARServer.idl 을 수행하면 다음의 화면과 같이 인터페이스 저장소에 등록될 것이다.



이제 OAD 를 살펴보자. OAD [옵션]의 형태로 수행하는데, 대표적인 옵션으로 –C 옵션을 사용하면 NT 서비스로 등록하여 사용할 수 있다. 다음은 OAD 에서 사용 가능한 옵션이다.

파라미터	내 용
-v	verbose 모드로 동작한다.
-f	다른 호스트로 OAD 가 있는지 확인하여 동작한다.
-t<n>	해당 n 초만큼 지연된 다음에 수행한다.
-C	NT 서비스로 구동된다.
-k	연결된 프로세스 들의 객체를 해제한다.
-?	도움말을 살펴본다.

그러면, OAD 에 해당 서버 인터페이스를 등록하는 방법에 대해서 알아 보자. oadutil 이라는 프로그램을 사용하는데 다음과 같은 형태로 실행하면 된다.

oadutil reg –r IDL:내서버/내객체:1.0 – o 내객체 –cpp 내서버.exe –p unshared

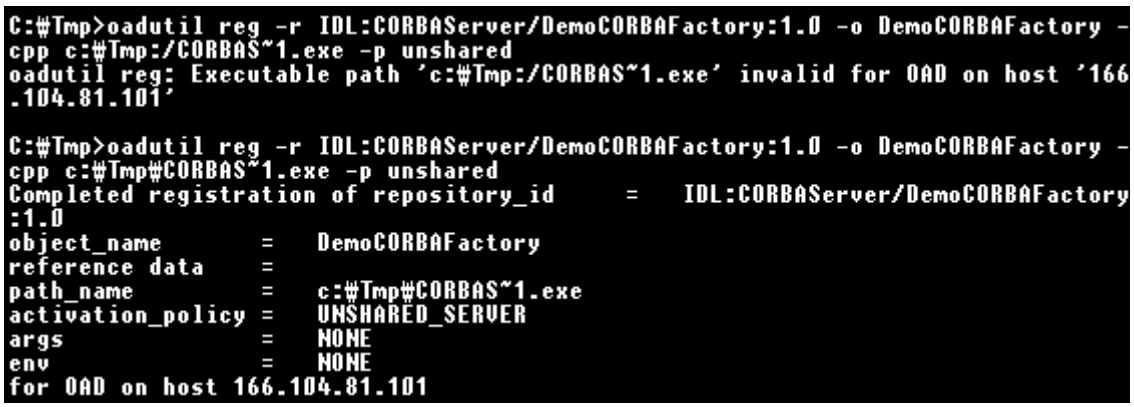
입력하는 내용은 CORBAServer_TLB.Pas 의 initialization 부분을 참고하면 된다. 앞서 등록한 CORBA 서브 디렉토리의 DataModule 서브 디렉토리에 있는 CORBAServer 프로젝트의 initialization 섹션은 다음과 같다.

```
initialization
  TCorbaVclComponentFactory.Create('DemoCORBAFactory', 'DemoCORBA',
   'IDL:CORBAServer/DemoCORBAFactory:1.0', IDemoCORBA, TDemoCORBA, iMultiInstance,
   tmSingleThread);
```

일단 컴파일을 하고 만들어진 CORBAServer.exe 파일을 적당한 디렉토리로 이동한 뒤에, 다음과 같이 등록하도록 하자.

oadutil reg –r IDL:CORBAServer/DemoCORBAFactory:1.0 –o DemoCORBAFactory –cpp 파일의 위치 디렉토리₩CORBAServer.exe –p unshared

제대로 등록되었으면 다음 화면과 같은 내용이 나타날 것이다. 주의할 점은 이와 같은 등록을 할 때 비지브로커 스마트 에이전트와 OAD 가 동작하는 상태에서 등록하여 주어야 한다.



CORBA 클라이언트

일단 CORBA 서버를 만들었으면, 이를 인터페이스에 저장하여야 한다. 그리고, 이를 클라이언트에서 사용하는 방법은 CORBA 서버를 만들 때 생성된 _TLB.pas 파일을 uses 절에 추가해서 사용하면 된다.
예를 들기 위해서 앞서 사용한 CORBA 서버 예제의 클라이언트 프로젝트를 열어보도록 하자. 그러면, CORBAServer_TLB.pas 유닛이 서버에서와 같이 사용되었다는 것을 알 수 있는데, 개발자는 이와 같이 필요한 인터페이스를 만든 다음 간단하게 해당 프로그램에 등록하여 사용하기만 하면 되는 것이다.
실제 CORBAServer_TLB.pas 의 initialization 부분을 살펴보자.

```
initialization
  CorbaStubManager.RegisterStub(IDemoCORBA, TDemoCORBAStub);
  CorbaInterfaceIDManager.RegisterInterface(IDemoCORBA, 'IDL:DemoCORBA:1.0');
  CorbaSkeletonManager.RegisterSkeleton(IDemoCORBA, TDemoCORBASkeleton);
```

해당 코드는 스텁과 스켈레톤을 정의하여 주며 IDManager 에 해당 CORBA 데이터 모듈을 등록하는 코드를 자동으로 만들어 준다.
그렇다면 CORBA 의 호출 방법중에 동적으로 호출하는 방법에 대해서 알아보자.

- 동적호출

DII(dynamic interface invocation)를 사용하는 방법은 해당 서버의 객체를 호출하기 위한 스텁 클래스를 마샬링한 인터페이스에서 찾아와야 한다. 이때 찾을 인터페이스는 앞에서 설명한데로 인터페이스가 인터페이스 저장소에 미리 등록되어 있어야 한다.
보통 호출하는 경우에는 Any 를 사용하여 호출하는 것이 안전하다.

```
var
  InCall : TAny;
begin
  InCall := CorbaBind( 'IDL:MyServer/MyServerObject:1.0' );
```

이렇게 코딩을 하면 필요한 인터페이스의 스텁을 알 수 있다.
실제 TAny 형을 찾아보면 델파이의 Variant 데이터 형임을 알 수 있다. 델파이 도움말에서 CorbaBind 에 대한 예제를 살펴보면, CorbaBind 를 사용해서 동적으로 호출하는 DII 를 잘 알 수 있다.

```
var
  HR, Emp, Payroll, Salary: TAny;
```

```
begin
  HR := CorbaBind('IDL:CompanyInfo/HR:1.0');
  Emp := HR.LookupEmployee(Edit1.Text);
  Payroll := CorbaBind('IDL:CompanyInfo/Payroll:1.0');
  Salary := Payroll.GetEmployeeSalary(Emp);
  Payroll.SetEmployeeSalary(Emp, Salary + (Salary * StrToInt(Edit2.Text) / 100));
end;
```

CorbaBind 를 통하여 원하는 IDL 인터페이스를 찾아서 해당 메소드를 사용할 수 있다.

정 리 (Summary)

이번 장에서는 CORBA 의 동작 원리에 대해서 더 깊숙히 알아보고, IDL 문법과 실제 CORBA 서버를 제작한 뒤에 어떻게 인터페이스를 등록하고 사용할 수 있는지에 대해서 알아보았다. 이것으로 제 5 부의 내용을 마치게 된다. 다음에 이어지는 제 6 부에서는 인터넷과 통신에 대해서 알아볼 것이다.