COM의 기초 개념

(Basic Concepts of Component Object Model)

COM (Component Object Model)은 OLE와 액티브X 기술의 기초가 되는 개념으로 인터페이스라는 미리 정의된 루틴의 세트를 통해 각 객체들간의 상호운용을 가능하게 해주는 객체 기반의 프로그래밍 specification 이다. COM 은 기본적으로 소스 코드 수준의 표준이 아니라 바이너리 표준이다. 이 때문에 여러가지 다른 언어로 객체를 구현할 수 있으며, 서로 다른 플랫폼과 다른 주소 공간에서 실행과 통신이 가능하다. 또한, COM 객체들은 유일한 identifier 가 있어서 생성과 인터페이스로의 접근이 쉬우므로 확장, 수정, 업데이트가 용이하다. COM에는 COM 객체의 핵심적인 기능을 정의하고, COM 객체들의 생성과 관리를 가능하게 하는 API 함수의 세트를 정의하고 있는 표준 인터페이스들의 세트를 포함한 라이브러리를 가지고 있다.
또한, COM 은 OLE와 액티브X의 기초가 되는데, 이들 기술들은 운영체제의 확장부분처럼 동작하여 이들 type 의 객체들을 생성하고 조작할 수 있는 라이브러리들을 제공한다. COM 을 기반으로 하여 개발자들은 다른 COM 에 기반한 기술들과 상호작용할 수 있는 그들만의 확장부분을 생성할 수 있다.
델파이에는 이런 COM, OLE, 액티브X 응용프로그램의 핵심적인 요소를 쉽게 생성할 수 있는 클래스들과 마법사들이 준비되어 있다. Delphi object framework 에는 하나의 어플리케이션에서 쓰이는 간단한 COM-호환 클래스들로부터 OLE 자동화 서버와 액티브X 컨트롤에서 쓰일 수 있는 클래스들까지 지원하고 있다. 델파이를 COM-기반 어플리케이션을 개발하는데 사용하여 어플리케이션의 내부적으로 인터페이스를 기반으로한 세련된 소프트웨어 디자인을 할 수 있으며, 윈도우 95 쉘 확장이나 DirectX 와 같은 COM 에 기초한 여러가지 API 객체 들과 상호작용이 가능한 객체들을 생성해낼 수 있다.
델파이 4가 지원하는 위저드로 생성할 수 있는 것으로는 다음과 같은 것들이 있다.

- 단순한 COM 객체
- 자동화 서버
- 자동화 컨트롤러
- 액티브X 서버와 액티브 폼
- 마이크로소프트 트랜잭션 서버(MTS) 객체

자주 쓰이는 용어 정의부터 하겠다.

인터페이스: 명확하게 정의된 목적이 있는 메소드 프로토타입들의 세트

COM 객체: CoClass 라고 불리는 클래스의 인스턴스로 COM 인터페이스들의 메소드들을 실제로 구현한다.

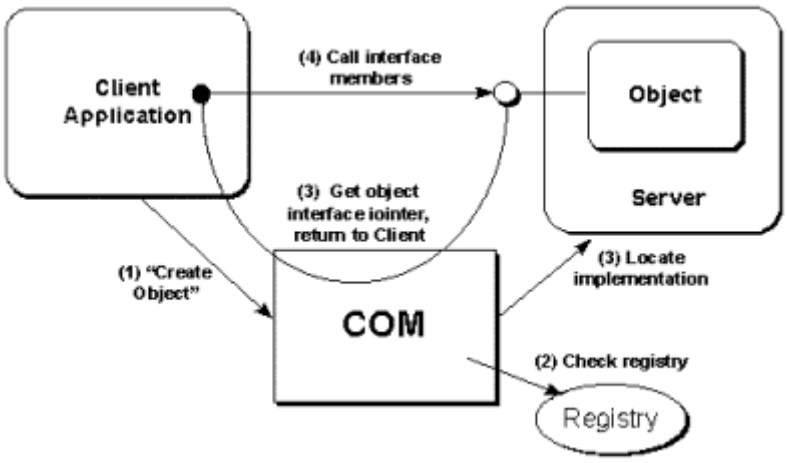
COM/액티브 X 서버: COM 이나 액티브 X 객체들을 포함한 모듈 (EXE, DLL, OCX).

클래스 팩토리: 지정된 CoClass 로부터 COM 객체를 생성할 수 있는 객체

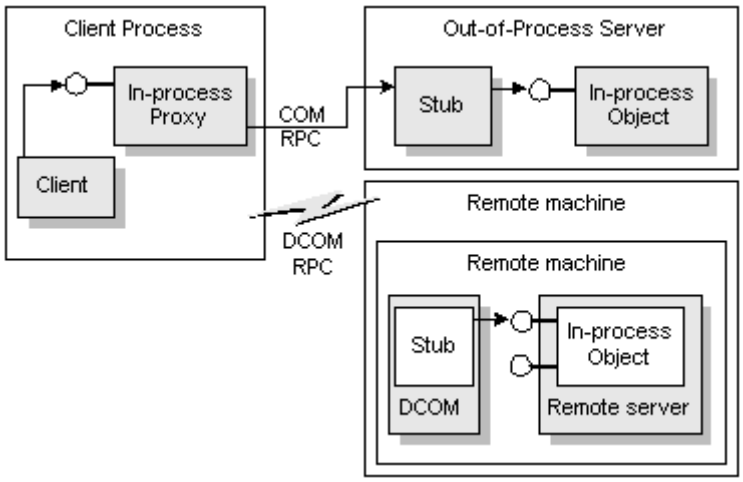
타입 라이브러리: 리소스로 저장이 가능한 이진 심볼 파일로, COM 이나 액티브 X 서버에 대한 데이터 형 정보를 담고 있다.

COM 아키텍처의 이해

이들 COM 객체가 동작하는 방식은 아래 그림과 같다. 클라이언트 어플리케이션이 ClassID 를 가지고 COM 객체를 생성하는 요구를 하면, OS 에서 제공하는 COM 라이브러리 (윈도우 95 의 경우 COMOBJ.DLL, OLE32.DLL 에 구현되어 있다.)가 시스템 레지스트리에서 해당되는 CLSID 를 가진 COM 객체를 찾게 된다. 이때 이 COM 객체는 클라이언트 어플리케이션의 요구를 서비스하는 COM 서버로 동작하게 되는 것이다. 이렇게 찾은 COM 객체의 구현 부분을 찾아서 해당 인터페이스 멤버 함수의 포인터를 클라이언트의 어플리케이션의 해당 부분에 매핑 시켜준다.



그런데 이때 COM 객체의 위치는 하나의 프로세스 공간에 있을 수도 있고, 다른 작업 공간에 있을 수도 있으며, 심지어는 다른 컴퓨터 안에 있을 수도 있다. 이때 COM 객체의 인터페이스 포인터의 위치를 하나의 프로세스 공간에 있는 것처럼 포장하는 과정을 마샬링이라고 하며, 이때 COM 서버 측의 포인터를 포장하는 객체를 스텁(stub), 클라이언트 어플리케이션 측의 객체를 프록시(proxy)라고 한다. 이를 그림으로 정리해보면 다음과 같다.



만약 클라이언트 어플리케이션에서 크래쉬가 일어나게 되면(예를 들어 클라이언트 어플리케이션이 동작하는 컴퓨터가 불의의 사고로 다운된 경우) 클라이언트의 프록시 객체는 파괴되며, 이와 연관된 서버 측의 스텁 객체가 커넥션이 단절된 것을 감지하여 클라이언트 객체와 연결된 수만큼 레퍼런스 카운트를 감소시키며, 이것이 0 이 되면 서버 객체가 파괴된다. 반대로, 서버 객체가 비정상 종료되면 클라이언트 어플리케이션은 프록시 객체에서 에러 코드를 받을 수 있게 된다 (DLL 의 경우에는 클라이언트 어플리케이션에도 크래쉬가 일어나면서 비정상 종료하게 된다.).

COM 어플리케이션의 구성

COM 어플리케이션을 구현할 때에는 다음과 같은 부분을 제공해야 한다.

- COM 인터페이스
  객체가 자신의 서비스를 클라이언트에게 노출시키는 방법이다. COM 객체는 인터페이스를 연관된 메소드와 프로퍼티의 세트로 제공한다.

- COM 서버
  EXE, DLL, OCX 형태의 모듈로 COM 객체를 위한 코드를 포함하고 있다. 객체 들은 이 서버 안에 있으며, COM 서버는 하나 이상의 인터페이스를 구현한다.

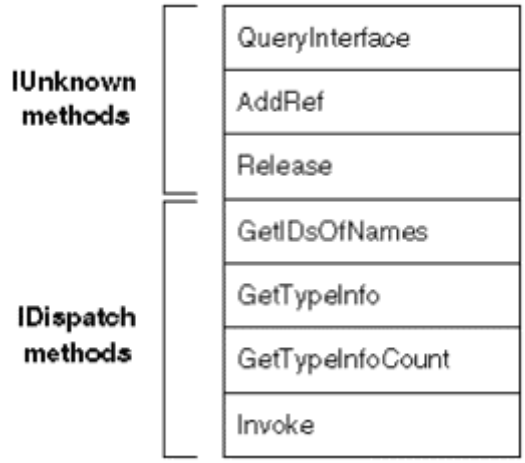
- COM 클라이언트
  서버에서 요구한 서비스를 얻기 위해 인터페이스를 호출하는 코드이다. 클라이언트는 서버에서 제공하는 서비스가 어떻게 구체적으로 동작하는지는 모른다. COM 클라이언트 중 가장 흔한 것은 자동화 컨트롤러이다.

OLE 와 액티브 X 객체

OLE, 액티브 X 객체들은 비주얼할 수도 논비주얼할 수도 있으며, 클라이언트에서 같은 작업공간에서 돌아갈 수도 있고, 다른 작업공간에서 돌아갈 수도 있다. 다른 작업공간에서 돌아가게 만들거나 remote-procedure call 을 쓸 수 있게 하려면 필요한 인자들을 package 하고, unpackage 할 수 있어야 한다. 위에서도 설명했듯이 이와 같은 과정을 마샬링이라고 하는데, 이렇게 RPC 를 쓰거나 프로세스 공간을 초월한 어플리케이션 작업을 할 때에는 여러가지 방법으로 여러가지 인자들이 마샬링 작업을 거쳐야 하며, 여러가지 종류의 OLE 와 액티브 X 객체들이 서로 다른 방법의 마샬링 작업을 하게 된다.
다음 표에 생성이 가능한 OLE 객체의 type 들을 나열해 보았다.

OLE 객체	비주얼	작업 공간	통 신	타입 라이브러리
OLE/Active 문서	거의	In-process, cross-process	Verbs for marshaling	No
OLE 자동화	일부	In-process, cross-process, or remote	IDispatch auto-marshaling	Recommended
액티브 X 컨트롤	거의	In-process	IDispatch auto-marshaling	Required
사용자 정의 인터페이스 객체	옵션	In-process, cross-process, remote	Manual marshaling	Recommended

OLE 자동화 서버들은 IDispatch 인터페이스를 구현한 객체들이다. IDispatch 는 IUnknown 에서 파생되었기 때문에 IUnknown 인터페이스도 구현하고 있다. IUnknown 인터페이스는 IDispatch 인터페이스의 메소드들이 OLE 자동화 객체의 메소드와 프로퍼티에 접근하는 동안 서버 객체가 지원하는 다른 모든 인터페이스들을 관리한다. 다음 그림은 IUnknown 과 IDispatch 인터페이스를 지원하는 객체의 IDispatch 인터페이스 vtable 의 모식도이다.



OLE 자동화 컨트롤러는 OLE IDispatch 인터페이스를 이용해 이 인터페이스를 구현한 OLE 서버 객체에 접근하는 클라이언트이다. 컨트롤러는 반드시 먼저 객체를 생성하고, 그 IDispatch 인터페이스 객체의 포인터에 해당하는 IUnknown 인터페이스를 질의(query)로 알아낸다. OLE 자동화 컨트롤러는 OLE 자동화 객체에 다음의 두가지 다른 방법으로 접근이 가능하다.

- IDispatch 인터페이스를 이용한다.
- 만약 듀얼 인터페이스 (dual interface)가 정의된 경우라면 객체의 가상함수 테이블 (virtual function table, vtable)이 멤버 함수를 직접 호출한다.

인터페이스 라이브러리의 이용

특별한 type 의 객체를 제작하거나 사용할 때에는 그 type 에 대한 해당되는 라이브러리나, 표준 객체에 대한 규칙을 참조해야 한다. 다음에 인터페이스 라이브러리의 type 에 대한 요약과 가능한 확장성을 기술하였다.

1. COM 인터페이스

COM 은 인터페이스의 표준 라이브러리를 정의하고 있다. 여기에는 인터페이스의 기본적인 기능을 정의하는 IUnknown, 외부에서 COM 객체를 인스턴스화할 때 필요한 클래스 factory 객체를 정의하는 IClassFactory 인터페이스 등을 포함한다.
COM 인터페이스를 구현하는 클래스의 제작에 관심이 있는 개발자들은 델파이에서 제공하는 TInterfacedObject, TComObject, TTypedComObject, TComObjectFactory, TTypedComObjectFactory 클래스를 참조하면 도움이 된다.

2. OLE/액티브 X 확장 부분

OLE 라이브러리 확장 부분은 많은 수의 객체 type 을 지원하며, 각각의 객체는 지원할 수 있거나 반드시 지원해야 하는 인터페이스들의 정의를 가지고 있다. OLE 그 자체는 아주 일반적인 목적의 인터페이스들을 많이 제공하는데, 이런 인터페이스는 IOle 로 시작한다.
OLE 처럼 액티브 X 역시 COM 의 구현 부분이다. 액티브 X 기술을 이용한 공통적인 어플리케이션에는 Active 문서, 액티브 X 컨트롤 등이 있다. 이 기술은 COM 에 대한 확장 부분으로 개발된 것으로 크기와 속도를 최적화한 기술이다.

3. 액티브 도큐먼트 (Active document)

액티브 도큐먼트는 문서 객체와 컨테이너에 대한 인터페이스의 라이브러리이다. 이 라이브러리에서는 그 자신의 인터페이스를 이용해서 컨테이너와 객체 어플리케이션간의 링킹과 임베딩을 위한 통신 등의 다양한 서비스를 제공한다. 액티브 도큐먼트 컨테이너를 제작하는데 관심이 있는 개발자는 TOleContainer 컴포넌트를 이용하면 된다.

4. OLE 자동화

OLE 자동화는 IDispatch 인터페이스의 구현을 필요로 한다. OLE 자동화는 서로 다른 작업 공간에서의 파라미터의 전송과 패키징, 객체의 저장과 객체 이름의 번역 등을 지원하는 서비스를 포함한다. OLE 자동화에 관심이 있는 개발자는 IDispatch 인터페이스를 구현하고 있는 TAutoObject 클래스를 이용하면 된다.

5. 액티브 X 컨트롤

액티브 X 컨트롤은, 과거에 OLE 컨트롤이나 OCX 로 불리던 것으로, 액티브 X 라이브러리에 있는 인터페이스들을 구현한 객체를 가리키는 말이다. 이들 인터페이스들은 객체가 이벤트를 발생시키고, 데이터 소스에 바인딩이 가능하고, 라이센싱을 지원하는 인터페이스를 제공한다. 델파이는 TActiveXControl 과 TActiveForm 클래스를 제공하며 동시에 쉽게 액티브 X 객체를 제작할 수 있도록 위저드를 제공한다.

6. 커스텀/서드파티 인터페이스의 구현

자기 자신의 라이브러리와 규칙 들을 정의할 수 있다. 커스텀 또는 서드파티 인터페이스를 구현하는 객체를 제작할 때에는 TComObject, TTypedComObject, TAutoObject 를 기초 클래스로 하면 된다.

클래스 팩토리

COM 객체는 하나 이상의 COM 인터페이스를 구현한 CoClass 의 인스턴스이며, COM 객체는 CoClass 인터페이스에 의해 정의된 서비스를 제공한다.
클래스 팩토리는 COM 객체를 제작하고 등록하는데 사용하는 객체이다. 클래스 팩토리는 CoClass 를 인스턴스화하는 표준 메커니즘을 제공한다. 클래스 팩토리를 이용하여 객체를 인스턴스화하면 클라이언트가 유일한 identifier 만 알게 되면 CoClass 에 대해 다른 것들은 알 필요가 없어진다. 클래스 팩토리는 생성자(constructor) 메소드와 argument 등을 다룬다. 각각의 CoClass 정의에는 클래스 팩토리가 있다. 클래스 팩토리는 하나 이상의 CoClass 와 관련이 있을 수 있으며, 이들 각각이 인스턴스화될 수 있다. 어쨌든 일단 클래스 팩토리 자신이 인스턴스화되면 이것은 특정 클래스 identifier(CLSID)와 연관이 되며, 이것이 생성자(constructor)에 넘겨진다. 이 파라미터는 어떤 CoClass 가 클래스 팩토리에 의해 인스턴스화 되는지를 나타낸다. 그래서, 클래스 팩토리 클래스가 하나 이상의 CoClass 에 적용되면 각각의 클래스 팩토리 인스턴스는 오직 하나의 CoClass 의 인스턴스를 생성할 수 있다.

1. CoClass 의 인스턴스화

CoClass 는 CoGetClassObject 라는 글로벌 윈도우 API 함수를 호출하여 인스턴스화될 수 있다. 이 함수는 레지스트리에서 CLSID 를 찾고, 서버의 패스를 알아내어 서버를 로드하고 클래스 팩토리 인터페이스(보통은 IClassFactory)에다가 포인터를 넘긴다. IClassFactory 포인터는 CoClass 의 인스턴스를 생성할 때 쓰이는 클래스 팩토리의 CreateInstance 메소드를 호출할 때 쓰인다. 위의 과정을 밟지 않고, API 함수인 CoCreateInstance 를 호출하면 위의 모든 과정을 한 번에 해준다. 일단 서버를 로딩하면 CoCreateInstance 는 자동으로 CoGetClassObject 함수를 호출하고, IClassFactory 포인터를 통해서 CreateInstance 메소드를 호출한다. 어쨌든 클래스의 여러 개의 인스턴스를 생성할 때에는 하나의 클래스 팩토리를 이용해서 CreateInstance 메소드를 직접 호출하는 것이 빠르다. CoGetClassObject 만이 IClassFactory 인터페이스 포인터를 제공할 수 있기 때문에, CoCreateInstance 를 호출하기 보다는 CoGetClassObject 를 호출하는 것이 빠르다.

2. 델파이의 클래스 팩토리 클래스

객체가 클래스 팩토리가 되려면 IClassFactory 인터페이스를 지원해야 한다. 델파이의 CoClass 들은 그에 해당하는 클래스 팩토리 클래스들을 가지고 있다. 이들은 IClassFactory 인터페이스를 구현하거나 IClassFactory2 인터페이스를 지원한다.

IClassFactory2 인터페이스는 라이센싱이 필요한 액티브 X 객체를 지원한다.
델파이의 클래스 팩토리 객체들은 서버를 포함한 유닛의 initialization 섹션에서 생성되어야 한다. 이로 인해 서버가 일단 로드되면 클래스 팩토리는 자동적으로 사용이 가능해 진다. 델파이의 COM 서버 클래스는 COM 또는 액티브 X 서버의 제작에 쓰이는데, 여기에는 클래스 팩토리를 관리하는 메소드들을 포함하고 있다. 클래스 팩토리에는 reference count 를 하는 기능이 있기 때문에 서버는 언제 unload 되어야 하는지 알 수 있다.

COM, 액티브 X 서버

COM 서버는 클라이언트 어플리케이션이나 라이브러리에 서비스를 제공하는 어플리케이션이나 라이브러리를 말한다. COM 서버는 클라이언트와 같은 작업 공간에서 동작하는 DLL인 in-process 서버일 수도 있고, 클라이언트와 다른 작업 공간이되 같은 기계 안에서 동작하는 EXE 파일인 로컬 서버일 수도 있으며, 클라이언트와 다른 기계에서 동작하는 리모트 서버일 수도 있다. COM 서버는 COM 객체가 존재하는 모듈이다. COM 서버는 OLE 자동화 객체, 액티브 X 컨트롤, 액티브 폼에 대한 코드를 담고 있다. 델파이는 액티브 X 서버가 가지는 기본적인 기능들을 캡슐화한 클래스를 가지고 있는데, 그 기능에는 다음과 같은 것들이 포함되어 있다.

1. 서버의 등록, 클래스의 등록, 서버를 로딩하거나 언로딩하거나 객체의 인스턴스화를 담당하는데 필요한 루틴들을 export 한다.
2. 서버에 객체를 구현하는데 필요한 클래스 factory 의 생성과 관리
3. 서버의 type, help 파일, 서버의 이름, 타입 라이브러리 등의 서버에 대한 핵심적인 정보를 제공한다.

델파이의 서버 클래스는 타입 라이브러리를 필수로 한다. 여기에는 서버에서 사용이 가능한 객체와 인터페이스에 대한 정보를 담고 있다.

마샬링(marshaling) 기전

마샬링은 클라이언트가 다른 프로세스나 기계에 있는 원격 객체의 인터페이스 함수를 호출할 수 있도록 해주는 기전을 말한다. 다른 말로 하면, 서버 프로세스의 인터페이스 포인터를 클라이언트 프로세스에서 사용할 수 있는 포인터로 바꾸고, 클라이언트의 파라미터를 원격 객체의 프로세스 공간으로 옮겨야 한다.
어느 인터페이스 호출에서도 클라이언트는 argument 들을 스택에 밀어넣고, 인터페이스 포인터를 호출하여 함수를 실행한다. 만약 이때 객체가 in-process 가 아니면 호출된 함수는 프록시에 전달된다. 프록시는 argument 들을 마샬링 패킷에 포장하고, 이를 원격 객체에

전달한다. 원격 객체의 스텁은 이 패킷을 풀어서, argument 들을 스택에 집어 넣고, 객체의 구현 부분을 호출한다. 즉, 객체는 클라이언트의 함수 호출하는 방식을 자신의 프로세스에서 자신의 주소를 이용하여 재생성하는 것이다.
어떤 형태의 마샬링을 사용할 것인가 하는 문제는 COM 객체가 구현된 방법에 달려있다. IDispatch 인터페이스가 제공되는 경우에는 표준 마샬링 기법을 이용하게 된다. 이 방법은 시스템 표준 RPC 를 통해 통신하게 된다.

자동화 (Automation)

자동화란 어플리케이션이 다른 어플리케이션의 객체를 제어할 수 있는 능력을 말한다. 자동화 객체의 클라이언트는 자동화 컨트롤러(automation controller)라고 하며, 서버 객체를 자동화 객체(automation object)라고 한다. 자동화는 in-process, 로컬, 원격 서버에서 모두 사용이 가능하다.
자동화의 가장 큰 특징은 다음의 2 가지 이다.

1. 자동화 객체는 반드시 객체의 인터페이스와 인터페이스 메소드, 파라미터 등의 정보를 기술해야 한다. 이런 정보는 보통 타입 라이브러리에 기록되며, 이를 이용해 여러가지 작업을 할 수 있게 된다. 델파이 4 에서 제작한 자동화 서버에도 타입 정보가 포함된다.
2. 자동화 객체는 반드시 메소드 들을 다른 어플리케이션이 접근할 수 있도록 해야 한다. 이를 위해 반드시 IDispatch 인터페이스를 구현해야 하며, 이를 통해 인터페이스의 메소드와 프로퍼티에 접근할 수 있게 된다.

액티브 X 컨트롤

액티브 X 컨트롤은 in-process 서버에서만 돌아가는 비주얼 컨트롤로 OLE 컨테이너 어플리케이션에 플러그인 될 수 있다. 액티브 X 컨트롤은 자체로 완전한 어플리케이션이 될 수는 없지만, 여러가지 어플리케이션에서 사용될 수 있는 재사용 가능한 컨트롤이다. 액티브 X 컨트롤의 프로퍼티, 메소드, 이벤트를 사용하려면 자동화 기법을 이용해야 한다.
액티브 X 컨트롤은 웹 페이지에서 많이 사용되고 있다. 그렇기 때문에 액티브 X 는 현재 웹에서 interactive 한 컨텐트를 제공하는 표준으로 점점 자리를 잡아가고 있다.
델파이 4 에서는 이런 액티브 X 컨트롤을 쉽게 제작할 수 있는 위저드를 제공한다.

타입 라이브러리 (type libraries)

타입 라이브러리는 COM 객체에 어떤 인터페이스가 존재하는지, 그리고 각 인터페이스 메

소드 argument 들의 수와 데이터 형 등의 정보를 담고 있다. 또한, CoClass 의 identifier(CLSIDs)와 인터페이스의 identifier(IIDs) 그리고, 자동화 인터페이스 메소드에 대한 디스패치 identifier(dispIDs) 등도 가진다.
타입 라이브러리에는 그 밖에도 다음과 같은 정보를 담고 있다.

- 사용자 정의 인터페이스와 연관된 사용자 정의 데이터 형 정보
- 인터페이스 메소드가 아니면서도 자동화나 액티브 X 서버에 의해 export 되는 루틴
- 열거형(enumeration), 구조체(structure), 공용체(union), 앨리어스, 모듈 데이터 형 등에 대한 정보
- 다른 타입 라이브러리의 타입 기술에 대한 레퍼런스

● 타입 라이브러리의 생성 방법과 사용

지금까지의 개발 도구에서는 IDL(Interface Description Language) 또는 ODL(Object Description Language)를 이용하여 스크립트를 작성함으로써 타입 라이브러리를 작성했다. 그리고, 이렇게 작성한 스크립트를 컴파일러를 동작시켜 헤더 파일을 만들어 사용해왔다. 델파이 4 는 자동화 서버나 액티브 X 컨트롤 위저드를 사용할 때 타입 라이브러리를 자동으로 만들어 준다. 그리고, 타입 라이브러리 에디터를 이용해서 타입 라이브러리의 내용을 보면서 쉽게 정보를 편집할 수 있게 해주며, 델파이는 해당되는 소스를 자동으로 업데이트 한다.
타입 라이브러리 에디터는 자동으로 표준 타입 라이브러리를 생성하고, 델파이용 인터페이스 파일(.pas 파일)에 오브젝트 파스칼 문법으로 인터페이스 정의를 한다.
외부 사용자에게 노출될 객체 들의 타입 라이브러리를 작성하는 것은 매우 중요한 작업이다. 예를 들면 다음과 같은 것들이 있다.

- 액티브 X 컨트롤은 액티브 X 컨트롤이 포함된 DLL 에 타입 라이브러리가 리소스로 함께 포함되어 있기를 요구한다.
- 자동화 서버를 구현한 어플리케이션은 반드시 타입 라이브러리를 제공해서, 클라이언트가 early 바인딩을 사용할 수 있도록 해야 한다.
- Vtable 바인딩을 지원하는 노출된 객체 들은 반드시 타입 라이브러리 기술되어야 한다. 이는 vtable 레퍼런스가 컴파일 시에 사용되기 때문이다.
- IProvideClassInfo 인터페이스를 지원하는 클래스에서 인스턴스화된 객체들은 반드시 타입 라이브러리를 가져야 하는데, 델파이의 VCL 중에서는 TTypedComObjectClass 를 상속한 클래스 들이 해당된다.
- Drag-and-drop 에는 타입 라이브러리가 반드시 필요한 것은 아니지만, 제공되면 객체를 확인할 때 유용하다.

만약 인터페이스를 단지 어플리케이션 내부에서만 사용할 것이라면 타입 라이브러리는 생성할 필요가 없다.

● 타입 라이브러리의 접근 방법과 활용

이진 타입 라이브러리는 보통 리소스 파일(.res)의 일부이거나 .tlb 파일 확장자를 가진 독자적인 파일로 존재한다. 일단 타입 라이브러리가 생성되면, 오브젝트 브라우저(object browser), 컴파일러 등의 도구가 특별한 타입 인터페이스를 통해 타입 라이브러리에 접근할 수 있다. 이러한 타입 인터페이스에는 다음과 같은 것들이 있다.

인터페이스	설 명
ITypeLib	타입 라이브러리에 접근할 수 있는 메소드를 제공한다.
ITypeInfo	타입 라이브러리에 포함된 각 객체에 대한 설명을 제공한다. 예를 들어, 브라우저는 이 인터페이스를 이용하여 타입 라이브러리에서 각 객체에 대한 정보를 뽑아낸다.
ITypeComp	컴파일러가 인터페이스에 바인드할 때 필요한 접근 정보를 빨리 뽑아낼 수 있다.

델파이는 다른 어플리케이션에서 타입 라이브러리를 import 하거나, 이를 사용할 수 있다. COM 어플리케이션을 위해 사용된 대부분의 VCL 클래스는 타입 라이브러리와 실행 중인 객체의 인스턴스에서 타입 정보를 불러오거나 저장할 때 사용되는 핵심 인터페이스를 제공한다. TTypedComObject VCL 클래스는 타입 정보를 제공하는 인터페이스를 지원하며, 액티브 X 객체 프레임웍에서 사용되는 기반 클래스이다.
어플리케이션에서 타입 라이브러리를 요구하지 않아도, 타입 라이브러리를 사용하면 다음과 같은 잇점을 얻을 수 있다.

- 데이터 형 검사를 컴파일 시에 할 수 있다.
- 자동화에서 early 바이딩을 사용할 수 있으며, vtable 이나 듀얼 인터페이스를 지원하지 않는 자동화 컨트롤러가 dispIDs 를 컴파일 시에 인코드할 수 있으며, 이로 인한 런타임 수행 성능의 향상을 얻을 수 있다.
- 타입 브라우저(Type browsers)로 라이브러리를 스캔할 수 있으므로, 클라이언트가 개발자가 만든 객체의 특징을 볼 수 있게 된다.
- RegisterTypeLib 함수를 이용하여 노출된 객체 들을 레지스트리 데이터베이스에 등록하는데 사용할 수 있다.
- 시스템 레지스트리에서 객체를 제거할 때에도 UnRegisterTypeLib 함수를 사용할 수 있다.
- 자동화 작업이 타입 라이브러리에서 얻은 정보를 이용하여 파라미터를 포장하게 되므로,

로컬 서버에 접근하는 성능이 향상된다.

액티브 도큐먼트 (Active Documents)

과거에 OLE 도큐먼트로 불리던 액티브 도큐먼트는 연결(linking)과 임베딩(embedding), drag-and-drop, 비주얼 편집(visual editing)을 지원하는 COM 서비스의 세트이다.
액티브 도큐먼트는 사운드 클립, 스프레드 시트, 텍스트와 비트맵과 같은 다른 포맷의 데이터와 객체를 통합할 수 있다. 액티브 X 컨트롤과는 달리 액티브 도큐먼트는 in-process 서버로 한정되지 않고, 프로세스의 경계를 넘어서 사용될 수 있다. 참고로, 액티브 도큐먼트의 스펙을 살펴보면 프로세스의 경계를 넘어설 수 있도록 마샬링이 지원되지만, 사실상 액티브 도큐먼트는 동일 기계 상의 다른 프로세스에서는 사용될 수 있지만 원격 서버에서는 동작하지 않는다. 이는 액티브 도큐먼트가 사용하는 데이터 형이 윈도우 핸들, 메뉴 핸들과 같이 주어진 기계 상의 시스템에 특정한 것들을 사용하기 때문이다.
대부분이 비주얼하지 않은 자동화 객체와는 달리, 액티브 도큐먼트 객체는 다른 어플리케이션에서도 비주얼하다. 그러므로, 액티브 도큐먼트 객체는 디스플레이나 출력 장치에 비주얼 하게 보여지는데 사용되는 프리젠테이션 데이터(presentation data)와 객체를 편집할 때 사용되는 원시 데이터(native data)의 2 가지 데이터 종류를 가지게 된다.
액티브 도큐먼트 객체는 도큐먼트 컨테이너이거나 도큐먼트 서버일 수 있다. 델파이 4 는 액티브 도큐먼트를 생성하는 자동화 위저드를 제공하지 않지만, TOleContainer VCL 클래스를 이용하여 이미 존재하는 액티브 도큐먼트를 연결하거나 임베딩할 수 있다. 또한, TOleContainer 는 액티브 도큐먼트 컨테이너로 사용할 수도 있다. 액티브 도큐먼트 서버에 대한 객체를 생성하려면 COM 기초 클래스 하나를 사용하고, 지원하고자 하는 객체 데이터 형에 대한 적절한 인터페이스를 구현하여야 하며, 위저드는 제공되지 않는다.

COM 과 VCL 클래스

델파이에서 COM 기술을 구현한 유닛을 살펴보면, 우선 COM 기술의 API 세트와 인터페이스 정의를 오브젝트 파스칼의 형식에 맞도록 선언해 놓은 부분이 ActiveX.pas 와 Ole2.pas, OleCtl.pas, OldDlg.pas 유닛에 기록되어 있다. ActiveX.pas unit 은 델파이에서 지원하는 인터페이스 선언문을 사용하여 COM 인터페이스를 재선언하고, COM 라이브러리 DLL 에서 지원하는 API 세트를 정의해 놓은 유닛이다. 그러므로, COM 기술을 위한 모든 VCL 클래스들은 기본적으로 이 유닛을 사용해야만 한다. 이렇게 델파이가 액티브 X 기술을 지원하기 위해서 제공하는 클래스를 DAX(Delphi ActiveX Framework)라고 한다.
실제로 COM 객체 클래스에서 가장 기초 클래스가 되는 TComObject, TAutoObject, TComClassFactory 등의 클래스는 ComObj.pas unit 에 구현되어 있다. 델파이의 모든 COM 객체는 TComObject 클래스에서 상속받은 객체라고 할 수 있는데, 그 선언부분은 다

음과 같으며, 이해를 돕기 위해 필자가 약간의 주석을 달았다.

```
TComObject = class(TObject, IUnknown, ISupportErrorInfo)
  {액티브X.pas unit에 선언된 IUnknown, ISupportErrorInfo 인터페이스를 구현한 클래스임을 나타내는 코드}

  private
    FRefCount: Integer;                    //내부적인 참조계수로 사용되는 변수
    FFactory: TComObjectFactory;   //COM 객체의 클래스 factory
    FController: Pointer;                  //Aggregation 된 경우의 IUnknown의 구현부분
    function GetController: IUnknown;

  protected

    //아래의 메소드는 IUnknown 자체의 구현이다.
    function IUnknown.QueryInterface = ObjQueryInterface;
          //QueryInterface를 TComObject.ObjQueryInterface가 구현한다는 의미
    function IUnknown._AddRef = ObjAddRef;
    function IUnknown._Release = ObjRelease;
    //다른 인터페이스들에 대한 IUnknown 메소드
    function QueryInterface(const IID: TGUID; out Obj): Integer; stdcall;
          //해당 IID를 받아서 IUnknown의 QueryInterface를 실행
    function _AddRef: Integer; stdcall;
    function _Release: Integer; stdcall;

    //아래의 메소드는 ISupportErrorInfo 인터페이스를 구현한다.
    function InterfaceSupportsErrorInfo(const iid: TIID): HResult; stdcall;

  public
    constructor Create;   //Aggregate의 부분이 아닌 COM 객체를 생성하는 메소드
    constructor CreateAggregated(const Controller: IUnknown);
//Aggregate 된 부분으로서의 COM 객체의 생성을 담당하는 메소드
    constructor CreateFromFactory(Factory: TComObjectFactory;
      const Controller: IUnknown);
        //실제 constructor 역할을 하는 메소드로 Create, CreateAggregated 메소드에 의해 호출된다.
COM 객체에 메모리를 할당하고 프로퍼티 들을 설정한다.
```

```
    destructor Destroy; override;                    //COM 객체를 파괴하는 메소드
    procedure Initialize; virtual;                   //가상함수로 초기화에 쓰인다.
    function ObjAddRef: Integer; virtual; stdcall;
    function ObjQueryInterface(const IID: TGUID; out Obj): Integer; virtual; stdcall;
    function ObjRelease: Integer; virtual; stdcall;
    function SafeCallException(ExceptObject: TObject;
      ExceptAddr: Pointer): HResult; override;
    property Controller: IUnknown read GetController;
  //컨트롤러가 되는 IUnknown 인터페이스
    property Factory: TComObjectFactory read FFactory;
    property RefCount: Integer read FRefCount;
  end;

{COM class }
  TComClass = class of TComObject;   //TComClass 클래스의 정의
```

TComObject 는 클래스 identifier 를 가지는 COM 객체에 대한 기초 클래스이다. 이때 CLSID 를 사용하는데 이것은 클래스를 데이터베이스 레지스트리에 등록하고 클래스 팩토리를 통해 외부에 인스턴스화 하는데 사용된다. TComObject 의 클래스 팩토리는 TComObjectFactory 가 담당하며, 클래스 팩토리의 프로퍼티에서 이름, 설명, CLSID 등의 TComObject 클래스의 여러가지 정보를 제공한다. 또한, TComObjectFactory 메소드를 사용하여 TComObject 클래스를 레지스트리에 등록하고 인스턴스화 할 수 있다.
TComObject 는 하나의 COM 객체로 인스턴스화될 수도 있고, aggregate 의 한 부분으로 인스턴스화될 수도 있다. 만약 인스턴스화된 COM 객체가 aggregate 의 내부 객체라면 그 IUnknown 인터페이스 메소드가 적절한 컨트롤러 인터페이스에 의해 동작하여 aggregation 을 지원하게 된다. 결과적으로 내부 COM 객체에 의해 구현된 모든 인터페이스들은 인터페이스 참조가 생길 때마다 참조계수(reference count)가 직접적으로 영향받지 않고, 컨트롤러 인터페이스에 의해 영향 받는다.
마지막으로 TComObject 는 ISupportErrorInfo, InterfaceSupportsErrorInfo 를 구현한다. 그러므로, OLE 예외처리와 safecall 호출규칙을 지원한다.
델파이의 COM 지원은 이렇게 TComObject 를 기초 클래스로 해서 풍부한 VCL 클래스로 이루어져 있다. 이들 중 가장 중요한 클래스들은 TTypedComObject, TAutoObject, TActiveXControl 등을 들 수 있다. 이들중 TTypedComObject 와 TAutoObject 는 ComObj.pas unit 에 구현되어 있으며, TActiveXControl 의 AxCtrls.pas unit 에 구현되어 있다. AxCtrls.pas unit 에서는 VCL 클래스를 액티브 X 컨트롤로 전환하기 위한 TActiveXControl 클래스의 많은 인터페이스를 구현하고 있다.

정 리 (Summary)

이번 장에서는 델파이 4 와 COM 에 대한 기초적인 개념 설명과 내용을 알아 보았다. COM 과 CORBA 에 대해서는 델파이가 많은 일을 해 주지만, 기초적인 개념을 이해하고 있어야 제대로 된 어플리케이션을 만들 수 있다.
다음 장에서는 이번 장의 개념을 바탕으로 실제 COM 인터페이스를 활용하고, COM 객체를 만드는 방법에 대해서 알아보도록 할 것이다.