마이크로소프트 트랜잭션 서버의 이용 (I)

제대로 된 확장성을 고려한 클라이언트/서버 시스템을 구현하는 것은 매우 어렵다. 이런 형태의 시스템을 제작하는 과정에서는 몇 가지 어려운 난관에 봉착하게 된다. 쓰레드가 여러 개 사용되는 형태의 복잡한 어플리케이션에서는 쓰레드를 안전하게 관리하고, 동시에 세션의 통신을 효과적으로 유지하면서 동시에 네트워크 트래픽을 최소한으로 줄이고, 서버와 데이터베이스 리소스를 적게 잡아먹게 작성하여야 한다.
분산 환경의 확장성이 뛰어난 클라이언트/서버 어플리케이션을 제작하는데 있어 이런 문제를 해결하기 위해 제공되는 라이브러리로 대표적인 것을 마이크로소프트 트랜잭션 서버(MTS, Microsoft Transaction Server)를 들 수 있다.
이번 장에서는 마이크로소프트 트랜잭션 서버의 사용 방법과 개념, 그리고 델파이에서 이를 활용하는 방법에 대한 기초적인 내용에 대해서 다루고자 한다.

마이크로소프트 트랜잭션 서버의 기능

- 분산 컴퓨팅 (Distributed Computing)과 멀티 tier 환경

분산 컴퓨팅 환경이 필요한 이유가 뭘까 ? 그것은 최근의 어플리케이션 요구 사항이 하나의 기계로는 감당할 수 없을 만큼 커진 것이 하나의 원인이고, 또 하나는 하드웨어를 교체하기 보다는 추가로 하드웨어를 구입해서 전체적인 성능을 높이는 것이 여러가지로 효율적이기 때문일 것이다.
이렇게 네트워크 상의 각 컴퓨터를 하나의 노드로 보고, 이들 노드에 컴퓨팅 로드를 분산시키는 것이 분산 컴퓨팅 환경이라고 할 수 있다. 여기서 반드시 고려해야 하는 것은 각 노드를 연결하고 있는 네트워크 환경이 컴퓨터 내부의 프로세서와 메모리의 연결을 연결하는 구조에 비해 훨씬 느리다는 점이다.
여기서 확장성의 문제가 가장 많이 발생하게 된다.
전통적인 2 tier 클라이언트/서버 시스템에서 이런 문제를 극복하기 위해서 제안된 것이 3 tire 모델이다. 이 모델에서 클라이언트 어플리케이션은 presentation tier 에 해당되고, 조작하는 데이터가 data tier, 그리고 그 중간에서 전체적인 흐름을 조율하는 workflow tier 로 나누어볼 수 있다.
일반적으로 이런 workflow tier 에 비즈니스 규칙(business rule)을 위치시키고, 여기에서 데이터의 무결성을 강화할 수 있다. 이 부분에 중요한 내용을 위치시키게 되면, 이후에 확장을 하게 될 때 workflow tier 의 수정만으로 쉽게 확장성을 담보할 수 있게 된다.

MTS 에서 제공되는 런타임 환경에서 동작하는 컴포넌트가 바로 이런 workflow tier 가 된다.

- MTS 구성 요소

MTS 에 대한 기능을 알아보기 전에, 먼저 MTS 를 구성하는 주요 요소에 어떤 것들이 있는지에 대해서 알아보도록 하자.

1. 기초 프로세스 (Base Process)

데스크탑 컴퓨터에서 동작하는 클라이언트 응용 프로그램이나 웹 브라우저가 여기에 해당한다. 서버에 어떤 작업을 처리하도록 요청함으로써 트랜잭션을 시작하는 동기를 제공한다.

2. 어플리케이션 컴포넌트 (Application Components), MTS 실행부 (MTS Executive)

비즈니스 로직이 구현되어 있는 부분으로 COM 객체로 작성되어 있다. MTS 에 설치되어 MTS 실행부에 의해 수행된다. MTS 실행부는 응용 프로그램 컴포넌트를 위한 런타임 서비스 환경을 제공한다.

3. 리소스 디스펜서 (Resource Dispenser), 리소스 관리자 (Resource Manager)

리소스 디스펜서는 한 프로세스 내에서 공유되는 상태(데이터베이스의 연결 상태 등) 데이터를 관리한다. 그에 비해 리소스 관리자는 지속적으로 유지되는 데이터를 관리해주는 시스템 서비스를 말하는 것으로 이러한 리소스 관리자로는 마이크로소프트의 SQL 서버, 메시지 큐, 트랜잭션을 지원하는 파일 시스템 등이 있다.

- MTS 서비스

그러면, MTS 에 의해 제공되는 서비스에 대해서 알아보도록 하자.

1. 세션 관리 (Session Management)

언제나 클라이언트와 서버 사이의 연결을 관리하는데에는 세션이 관계하게 된다. 세션은 그 상태를 저장하기 위해 메모리 리소스를 요구하며, 동시에 이를 업데이트하고 관리하기 위해서

는 프로세서 리소스를 요구한다. 그런데, 이렇게 세션을 관리하는데 들어가는 리소스는 보통 서버 측에서 주로 부담하게 된다. 또한 전통적인 2-tier 방식의 경우에 일반적으로 n 개의 서버에 m 개의 클라이언트가 접속한다고 할 경우에 실제로 필요한 연결의 수는 m * n 이 된다. 예를 들어 100,000 개의 클라이언트가 100 개의 서버에 접속한다고 할 때 실제로 관리하게 되는 분리된 세션의 수는 10,000,000 개나 된다.
Workflow tier 의 가장 핵심적인 역할 중의 하나가 이러한 전체 세션을 관리하는 것이다. 예를 들어, 10 개의 새로운 workflow 컨트롤러를 클라이언트와 서버 사이에 위치시키고 각각의 컨트롤러가 클라이언트의 10%를 관리한다고 할 때, 앞의 예에서의 전체 100,000 개의 클라이언트 중 각 컨트롤러가 담당하는 클라이언트는 10,000 개가 되며 세션은 10,000 * 1 인 10,000 개를 관리하게 된다. 각각의 workflow 컨트롤러는 또한, 모든 서버 들과의 연결을 담당하므로 100 개의 서버에 대해 이들은 100 개의 세션을 관리하게 된다. 그러므로 관리하는 총 세션의 수는 각 컨트롤러 당 10,100 개가 되며, 총 10 * 10,100 = 101,000 개의 세션이 된다. 이 숫자는 2-tier 모델에서의 10,000,000 개의 세션에 비해 약 1/100 정도 밖에 되지 않는다.

2. 서버 리소스 관리 (Server Resource Management)

데이터베이스 연결은 꽤나 리소스를 많이 잡아 먹는다. 그렇기 때문에 많은 수의 클라이언트 접속이 필요하고, 또한 이 숫자가 더욱 늘어나게 된다면 이들의 연결을 유지하기 위해 많은 양의 서버 리소스가 필요하게 된다.
이때 workflow tier 에 커넥션 풀링(connection pooling)이라는 스키마를 이용해 데이터베이스 연결을 효과적으로 공유하는 로직을 사용하면 리소스를 많이 절약할 수 있다.
MTS 에는 ODBC 리소스 디스펜서(ODBC resource dispenser)라는 컴포넌트가 포함되어 있는데, 이 컴포넌트는 자동으로 ODBC 연결을 풀링한다. 풀링을 이용하면 ODBC 데이터베이스로 작업을 할 때 ODBC 연결을 생성하거나 이를 파괴하는 오버헤드를 줄여줄 수 있기 때문에, 상당한 수행성능의 향상을 기대할 수 있다.

3. 데이터 컨텐션 (Data Contention)

다중 사용자 데이터베이스 어플리케이션의 경우, 확장성에 있어 가장 커다란 장애를 들라고 하면 데이터베이스 레코드 들에 대한 컨텐션(contention)을 들 수 있다. 보통 여러 벤더 들은 레코드의 잠금(locking) 기술을 최적화는데 많은 시간과 돈을 투자한다. 그렇지만, 실제로 데이터베이스 어플리케이션을 개발하는 개발자들이 데이터베이스 레코드 들의 컨텐션을 최소화할 책임을 다소는 지고 있다.

개발자는 2 가지 요소를 조절할 수 있다. 어플리케이션에 의해 좌우되는 데이터베이스 잠금의 수와 어플리케이션이 한번 잠금을 할 때 사용하는 시간이 그것이다. 어플리케이션 코드를 다시 디자인하면 어플리케이션에 의해 사용되는 잠금의 수를 최소화할 수 있다. 또한, 데이터를 관리하는 코드를 데이터베이스에 보다 가깝게 위치시킬 수 있다면 아마도 데이터베이스 잠금에 걸리는 시간을 줄일 수 있을 것이다.
전통적인 2-tier 클라이언트/서버 어플리케이션은 SQL 저장 프로시저(SQL stored procedure)를 이용하여 개발자가 잠금을 할 때 걸리는 시간을 최소화한다. 그렇지만, SQL 문장은 기본적으로 OOP 언어와 맞지 않는다.
MTS 객체들은 기본적으로 3-tier 클라이언트/서버 시스템에서 미들 tier 에 위치하며, 제대로 디자인되면 데이터베이스에 접근하는 어플리케이션의 로직을 중앙집중적으로 관리할 수 있도록 해준다. 이렇게 데이터로의 접근을 컴포넌트로 중앙집중함으로써 커넥션 풀링과 데이터베이스 서버로의 연결 속도를 높일 수 있으며, 데이터베이스 잠금에 걸리는 소요 시간을 줄여줄 수 있다.

4. 네트워크 문제

어플리케이션의 확장성에 제한점이 될 수 있는 또 하나의 요소로는 네트워크 트래픽을 들 수 있다. 최근의 OOP 스타일의 클라이언트/서버 어플리케이션의 경우 보통 전형적으로 서버 컴퓨터에 인스턴스를 생성하고, 객체에서 하나 이상의 메소드를 실행하게 하며, 실행이 끝나면 서버 컴퓨터에서 생성된 인스턴스를 파괴하게 된다.
보통 데이터베이스 어플리케이션에서 가장 커다란 비중을 차지하는 것이 데이터베이스 연결과 레코드 잠금이다. 그러므로, 어플리케이션의 확장성을 좋게 하려면 이런 리소스를 보존하는 것이 중요하다. 이를 위해서 클라이언트 어플리케이션 프로그래머는 전통적으로 이들을 필요할 때만 생성해서 사용한 뒤에 이를 파괴하는 형태로 프로그래밍을 하게 된다.
이런 전술은 리소스를 절약하는데에는 도움이 되겠지만, 서버 측의 객체를 생성하고 실제 메소드를 실행하고 파괴하는 등 3 차례 네트워크에 접근해서 작업을 해야하므로 부하를 걸게 만든다.
MTS 는 컨텍스트 wrapper 객체(context wrapper object)라는 것을 이용하여 이런 네트워크 부하를 최소화하는 역할을 한다. 컨텍스트 wrapper 객체는 연결된 컨텍스트 객체와 함께 객체 런타임 환경(object runtime environment)을 형성하는데, 이 환경은 클라이언트 어플리케이션에서 바라볼 때 개념상 하나의 객체로 간주하게 된다.
MTS 에서 클라이언트가 MTS 객체를 생성하고, 파괴할 필요가 없게 하기 위해서, 클라이언트 어플리케이션은 MTS 객체를 그때 그때 생성하고 파괴하는 것이 아니라, 최대한 빨리 MTS 객

체에 대한 레퍼런스를 얻어다가 최대한 늦게 이를 해제한다. 이렇게 함으로써 MTS 는 객체를 사용하는 메소드 호출이 있을 때마다 MTS 객체가 인스턴스를 생성하도록 한다. 객체가 MTS 에 메소드 호출이 끝났음을 알리면 MTS 는 이를 파괴하지만, 컨텍스트 wrapper 는 다음 번 요구에 대비하기 위해서 살아있게 된다. 이렇게 함으로써 MTS 가 객체의 생성과 파괴에 대한 오버헤드를 줄임으로써 전체적인 수행 성능을 높이게 된다.
MTS 의 네트워크 트래픽을 줄이기 위한 이러한 최적화 방법을 just-in-time 활성화(activation)이라고 하는데, 여기에서 중요한 역할을 하는 것은 실제 객체에 의해 구현된 인터페이스에 대한 메소드를 구현한 컨텍스트 wrapper 객체이다. 이 객체는 MTS 에 의해 런타임에서 동적으로 생성되며, 클라이언트 어플리케이션과 실제 MTS 객체 사이에 삽입된다.

5. 트랜잭션 (Transactions)

분산 어플리케이션을 제작할 때에는 확장성 이외에도 컴퓨터 시스템에 복잡성이 증가하기 때문에, 여기에 따른 시스템 failure 가능성이 높아질 수 밖에 없어 이를 고려해야 한다. 잘 디자인된 분산 어플리케이션은 확장성도 좋지만, 이런 failure 가 있을 때 이를 자동으로 복구할 수 있는 고려가 들어가 있다.
대부분의 분산 환경에서 이런 자동 복구를 구현하기 위해서 트랜잭션을 사용하고 있다. 트랜잭션이란 여러 가지 기능의 복잡한 세트를 하나의 단위로 간주하여 사용하는 것을 말한다. 트랜잭션은 클라이언트 어플리케이션과 서버에서 실행되는 코드 사이의 일종의 계약과도 같아서, 일단 서버의 코드가 클라이언트에게 트랜잭션이 성공했음을 알리면, 그것은 클라이언트에서 요구한 내용이 완전히 저장되었다는 의미이며, 반대로 실패했음을 알리면 클라이언트에서 요구한 내용이 전혀 수행되지 않는다.
트랜잭션은 에러 처리의 가능성을 많이 줄여 준다. 트랜잭션을 이용하면 전부 실행(commit)되거나, 아예 실행되지 않으므로(rollback) 에러 복구를 위해 일부 실행된 내용들을 undoing 할 필요가 없다.
트랜잭션을 보다 효과적으로 사용하려면, 하나 이상 중첩된 트랜잭션을 사용할 수 있다. 즉, 하나의 커다란 트랜잭션을 마치 트리와 같은 형태로 여러 개의 중첩된 트랜잭션이 포함된 트랜잭션으로 재구성하는 것이다. 이렇게 하면, 중첩된 트랜잭션이 실패할 경우 실패한 트랜잭션만 복구하거나 전체 트랜잭션을 취소하는 것을 선택할 수 있다.

6. Data Tier 의 확장

트랜잭션을 분산 컴퓨팅 환경에서 유용하게 사용하려면, 분산 환경을 구축할 때 병목 구간이

될 수 있는 data tier 를 확장할 수 있어야 한다. 즉, 아무리 여러 클라이언트와 workflow tier 에서의 비즈니스 처리를 분산 환경에서 할 수 있더라도, data tier 인 실제 데이터베이스 서버를 한 군데에서 처리해야 한다면 데이터베이스에 접근하는 부분에서 병목이 될 수 밖에 없다. Data tier 를 확장하기 위해서는 하나 이상의 데이터베이스를 MTS 트랜잭션에서 사용할 수 있어야 한다. 즉, 논리적인 데이터베이스를 하나 이상의 데이터베이스 서버 노드에 분산하는 것이다. 이를 위해서 MTS 는 마이크로소프트의 DTC(Distributed Transaction Coordinator)를 이용하여 트랜잭션이 하나 이상의 데이터베이스 서버에서 이루어질 수 있도록 허용하고 있다. 이때 DTC 는 산업 표준인 X/Open XA 프로토콜, IBM 의 LU6.2 프로토콜, 마이크로소프트의 OLE 트랜잭션 프로토콜 등을 이용하게 된다.

7. 보안 (Security)

보안 프로그래밍과 연관된 세부적인 내용을 해결하기 위해 MTS 는 role 에 기반을 둔 보안 모델을 제공한다. MTS 의 보안을 이용하기 위해서는 일단 MTS 컴포넌트를 패키지로 그룹화하고, 각각의 패키지에 role 을 부여해야 한다. 예를 들어, Manager 와 Data Entry Clerk 이라는 role 을 미리 정의했다고 하면, 배포할 때 윈도우 NT 관리자가 실제 사용자에게 개발 당시에 미리 정의한 role 을 부여할 수 있다. 런타임에서 사용자들은 자신에게 부여된 role 에 따라 MTS 패키지에서 특정 기능에 접근할 수도 있고, 접근이 거부될 수도 있다.
Role 에 기반을 둔 보안을 이용하여 MTS 는 자동으로 보안 검사를 해주는 셈이다. 클라이언트 어플리케이션이 MTS 객체를 호출하게 되면, MTS 는 클라이언트에 대한 보안 검사를 수행한다. 이런 보안 검사는 인증(authentication)과 접근 제어(access control)와 관련이 있다. 인증은 사용자 자체에 대한 보안 검사인데, MTS 는 기존의 윈도우 NT 보안 인프라를 그대로 유지하면서 이를 보다 쉽게 만들어 준다. 접근 제어는 사용자가 특정 작업을 하고자 할 때 이를 허용할 것인지 여부를 결정하는 것으로 MTS 는 사용자에게 부여된 role 을 가지고 사용자의 요구를 수행할 것인지 여부를 결정한다.
기억해야 할 것은 MTS 패키지 내의 모든 MTS 객체 들은 동일한 보안 컨텍스트를 공유한다는 점이다. 그러므로, 보안 검사는 패키지 범위 내에서 수행되는 것이지 같은 패키지 내의 컴포넌트들 사이의 보안 검사는 모두 동일하다.

- 자주 사용되는 용어

MTS 에 대해 더 알아보기 전에, 엔터프라이즈 환경에서 사용되는 몇 가지 용어에 대해 다음에 정리해 보았다.

용 어	설 명
네트워크 리시버 (network receiver)	네트워크를 통한 클라이언트로부터 호출을 받아들이고 병목 현상을 관리하는 역할을 하는 단위
커넥션 관리자 (connection manager)	각 클라이언트에 대한 작업과 시스템 리소스를 추적, 관리한다.
쓰레드 풀 (thread pool)	한 쓰레드가 각 사용자에게 전유되는 것을 막하 시스템 리소스를 효율적으로 사용할 수 있도록 한다.
컨텍스트 관리자 (context manager)	각 사용자에 대해 현재 사용자를 확인하고 그 상태를 추적한다.

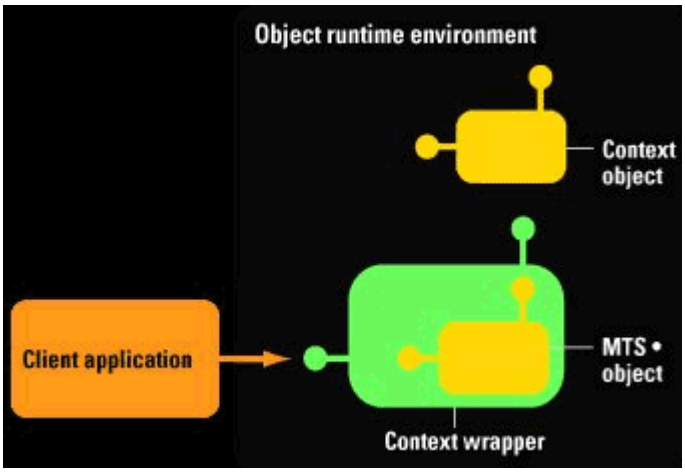
- MTS 프로그래밍 모델

MTS 는 COM 이 제공하는 컴포넌트 인프라 구조로 사용할 수 있도록 디자인되었다. MTS 는 COM 객체 들에 대한 런타임 인프라 구조를 제공한다. COM 객체가 MTS 객체로 사용되기 위해서는 MTS 런타임 지원을 받기 위한 몇 가지 요구 사항을 지켜 주어야 한다.
MTS 객체는 반드시 32 비트 in-process COM 서버로 제공되어야 한다. MTS 는 MTS 객체에 의해 구현된 모든 인터페이스의 파라미터 데이터 형과 함수 이름을 알고 있어야 한다. 그러므로, MTS 객체로 구현하기 위해서는 인터페이스 정의에 대한 정보가 제공되어야 한다. 이를 위해서 MTS 는 동적으로 컨텍스트 wrapper 객체를 생성한다. MTS 객체의 인터페이스는 반드시 자동화에 호환되거나 다른 표준 마샬링 인터페이스를 지원해야 한다.

1. 컨텍스트 wrapper

컨텍스트 wrapper 객체는 실제 MTS 객체로 외부 클라이언트에서 보일 수 있도록 하는 객체이다. 클라이언트 어플리케이션이 MTS 객체에 대한 레퍼런스를 얻을 때, 실제로는 MTS 에 의해 런타임에서 생성되는 컨텍스트 wrapper 객체를 참조하게 된다. 컨텍스트 wrapper 객체는 just-in-time 활성화를 구현하며, 자동으로 보안 검사를 한다.
클라이언트 어플리케이션이 MTS 객체의 메소드를 호출할 때에는 MTS 가 먼저 앞서 설명한 방법으로 보안 검사를 수행한다. 클라이언트 어플리케이션이 보안 검사를 통과하면 컨텍스트 wrapper 객체는 객체의 인스턴스에 대한 레퍼런스를 가지고 있는지 검사하게 된다. 대부분의 MTS 객체의 경우에는 컨텍스트 wrapper 객체는 반드시 객체의 새로운 인스턴스를 생성하고, 객체의 실제 메소드를 호출하게 된다. 메소드가 컨트롤을 컨텍스트 wrapper 객체에 돌려 주면, SetAbort 나 SetComplete 메소드를 호출했는지 확인하고 그렇다면 컨텍스트 wrapper 는 컨트롤을 클라이언트 어플리케이션에 반환하기 전에 객체를 파괴한다.

MTS 객채와 컨텍스트 wrapper, 그리고 컨텍스트 객체는 모두 MTS 의 객체 런타임 환경(object runtime environment)에 포함된다. 다음 그림과 같이 MTS 객체는 컨텍스트 wrapper에 의해 캡슐화되고, 각각의 MTS 객체마다 하나의 컨텍스트 객체를 가지게 된다.



2. 컨텍스트 객체 (Context Objects)

MTS 객체의 각각의 인스턴스는 유일한 컨텍스트 객체를 가지게 된다. 이를 사용하기 위해서는 MTS API 함수인 GetObjectContext 함수를 이용하여 MTS 객체의 컨텍스트 객체에 대한 레퍼런스를 가져올 수 있다. 컨텍스트 객체는 IObjectContext 인터페이스를 구현하는 COM 객체이다.
MTS 프로그래머는 IObjectContext 인터페이스의 SetAbort, SetComplete 메소드를 자주 이용하는데, 이렇게 함으로써 MTS 에게 객체에 대한 처리를 마쳤으므로 함수를 빠져나갈 때 객체를 파괴해도 된다는 것을 알려야 한다.
이 과정을 오브젝트 파스칼 코드로 간단하게 설명하면 다음과 같다.

```
var
  Context: IObjectContext;
begin
  Context := GetObjectContext;
  try
    ... 여러가지 처리
    Context.SetComplete;
  except
```

```
    Context.SetAbort;
  end;
end;
```

또한 트랜잭션에 추가적인 MTS 객체 들을 나열하고, 프로그램으로 보안을 구현하기 위해서 컨텍스트 객체의 메소드를 이용할 수도 있다. IsCallerInRole 메소드를 이용하면 호출자의 접근 토큰이 MTS 탐색기(MTS Explorer)에 의해 부여된 role 중에서 일치하는 것이 있는지 검사할 수 있다. 또한, CreateInstance 메소드를 이용하면 MTS 객체의 인스턴스를 새로 생성해서 이를 현재의 트랜잭션 범위에서 사용할 수 있도록 추가하는 것이 가능하다.

- IObjectContext 인터페이스

MTS 어플리케이션은 하나 이상의 MTS 객체 들로 구성되어 있다. 이들은 in-process COM 서버로 컴파일 되어 위치한다. 클라이언트는 MTS 객체에 다른 COM 객체에 접근하는 것과 마찬가지로 접근할 수 있다. MTS 자체는 클라이언트에서 볼 수 없으며, MTS 에 특이한 코드가 따로 존재하는 것은 아니지만 실제로 클라이언트가 객체의 메소드를 호출할 때 MTS 객체에 접근하는 경우에 실제로는 MTS 객체가 중간에 이를 가로채서 관리하게 된다.
MTS 객체가 생성될 때, 일반 COM 객체와 다른 점으로는 MTS 는 항상 컨텍스트 객체를 관리한다는 점이다. 이런 컨텍스트 객체 역시 IObjectContext 인터페이스를 지원하는 COM 객체이다. 이 인터페이스의 역할은 MTS 객체가 MTS 실행부(MTS Executive)와의 통신을 담당한다. 이들을 호출하기 위해서는 MTS 객체가 해당 컨텍스트 객체의 IObjectContext 인터페이스에 대한 포인터를 얻어야 한다. 이를 위해 사용하는 API 함수가 바로 GetObjectContext 함수이다.
IObjectContext 인터페이스는 몇 가지 메소드를 포함하는데, 그 중에서도 가장 중요한 것이 SetComplete 와 SetAbort 메소드이다.
앞에서도 간단히 설명한 바 있지만, 이들 메소드를 MTS 객체가 호출하는 것은 MTS 실행부에 더 이상 관리할 데이터가 없다는 것을 선언하는 것임과 동시에 트랜잭션의 사용 여부와 상관없이 SetComplete 일 경우 객체가 수행한 기능을 commit 하고, SetAbort 인 경우에는 rollback 하는 역할을 한다. 다시 말해 이들 메소드가 호출된다는 것은 더 이상 객체가 존재할 필요가 없다는 것을 선언하는 것이다.
이를 가능하게 하려면, MTS 실행부가 모든 MTS 객체를 잘 포장하고 MTS 객체가 SetComplete 나 SetAbort 메소드를 호출할 때 MTS 는 객체가 가지고 있는 모든 인터페이스를 해제해야 한다. 그렇지만 이때 클라이언트는 객체의 인터페이스 포인터라고 믿는 포인터를 가

지고 있게 되는데, 실제로 이 포인터는 MTS 에서 제공하는 wrapper 에 대한 포인터이다.
이렇게 함으로써 클라이언트가 MTS 객체를 사용할 때의 지속성을 유지하면서, 동시에 실제 MTS 객체에 의해 사용되던 리소스를 다른 객체 들이 사용할 수 있도록 할 수 있다.
클라이언트가 MTS 객체의 다른 메소드를 다시 호출하면, MTS 는 클래스 팩토리를 이용해서 그 객체의 새로운 인스턴스를 생성한 후 이를 클라이언트에게 넘겨준다.

- MTS 객체의 사용 예

이 구조의 장점을 예를 들어 설명해보자. Add, Remove, Submit 이라는 3 개의 메소드를 가지는 IOrderEntry 인터페이스를 구현한 MTS 객체가 있다고 하자. 클라이언트는 이를 구현한 MTS 객체를 이용하여 주문서에 제품을 추가, 삭제, 발송한다고 하자. 여기서 인터페이스의 Add 메소드가 다음과 같은 형태로 구현되어 있다.

```
procedure TOrderEntry.Add(Item);
begin
  If (NoOrderExists) then CreateOrder;
  Lookup(Item);
  If (Available) then
  begin
     AddItemToOrder(Item);
     DecrementInventory(Item);
  end
  else
    ErrorHandler;
end;
```

다시 말해 현재 주문이 존재하지 않으면 일단 하나를 생성한 뒤에, 데이터베이스에서 요구한 아이템이 있는지 검사한 뒤에 그렇다면 아이템을 현재의 주문에 추가하고, 이 아이템에 대한 재고를 하나 감소시킨다. 이 메소드를 구현함에 있어서 SetComplete 나 SetAbort 를 호출하지 않았기 때문에, 객체는 비활성화되지 않고 메모리에 남아 있는 주문이 생성되며 이를 다른 메소드 들이 이용할 수 있다.
Remove 메소드는 다음과 같은 형태로 구현하면 될 것이다.

```
procedure TOrderEntry.Remove(Item);
begin
  If (ItemInOrder) then
  begin
      RemoveItemFromOrder(Item);
      IncrementInventory(Item);
  end
  else
    ErrorHandler;
 end;
```

이 역시 매우 간단하게 구현 했는데, 먼저 현재 주문이 있는지 확인하고 주문에서 아이템을 삭제한 뒤에 재고를 하나 증가시킨다. 역시 여기에서도 SetComplete 나 SetAbort 를 호출하지 않기 때문에 주문에 대한 상태는 여전히 메모리에 남아 있게 된다. 결국에는 Submit 함수에서 이들을 최종적으로 처리하는 것이다.

```
procedure TOrderEntry.Submit;
begin
  Context = GetObjectContext;
  If (EverythingIsOK) then
  begin
    SubmitOrder;
    Context.SetComplete;
  end
  else
    Context.SetAbort;
end;
```

실제 MTS 객체에 대한 가장 핵심적인 기능을 하는 메소드이다. 먼저 GetObjectContext API 함수를 호출하여 적절한 컨텍스트 객체의 포인터를 얻는다. 특별한 문제가 없으면 주문을 전송하고 SetComplete 를 호출한다. 그러나, 실패한 부분이 있다면 SetAbort 를 호출하게 된다. 이들 메소드에 의해 MTS 는 객체가 가지고 있는 모든 인터페이스 포인터의 Release 메소드를 호출하며, 객체가 사용하던 모든 리소스와 메모리 상의 데이터는 해제된다. 만약 클라이언트가

다시 이 객체에 대한 Add 메소드를 호출하면 MTS 는 이 객체에 대한 새로운 인스턴스를 생성한다.
이런 식으로 객체를 구현하면 클라이언트가 MTS 객체를 생성, 파괴하는 작업이 없이 마치 계속 객체의 인스턴스를 메모리에 띄워 놓은 것처럼 여러 아이템을 그때 그때 알아서 추가, 삭제하고 전송할 수 있다. 그렇지만, 서버 상에서는 실제로 주문이 진행되는 동안에만 객체를 관리해도 되므로 여기에 들어가는 오버헤드를 많이 줄일 수 있다.
물론 Add, Remove, Submit 메소드를 구현할 때 이들 각각에 대해 SetComplete, SetAbort 를 실행하도록 구현할 수도 있다. 그렇지만, 이렇게 할 경우에는 메모리 상에 있는 데이터의 지속성을 보장할 수 없으므로 주문의 내용을 디스크에 저장하고, 이를 읽는 형태를 가져야 한다. 이렇게 하면 물론 확장성은 더 뛰어나다고 할 수 있지만, 디스크 공간이 더 필요하고 클라이언트의 수에 따라서는 수행 성능도 떨어질 수도 있다.
그러므로, 앞으로의 확장성과 수행 성능의 균형을 고려하여 전체적인 형태를 디자인해야 한다.

- 서버와 리소스

데이터베이스에 접속하기 위해서는 상당한 리소스를 소모하게 된다. 또한, ODBC 연결의 경우 한정된 리소스만 가지고 있기 때문에 계속적인 ODBC 연결이 생성되고, 이 연결이 해제되지 않으면 리소스가 부족하게 된다. 그렇다고, 연결을 필요할 때마다 생성하고 사용한 뒤에 해제하는 것은 수행 속도에 문제를 일으키는 경우가 많다.
MTS 에서는 리소스 디스펜서(resource dispenser)를 이용하여 시스템의 공유할 수 있는 비지속성 리소스(RAM 등)를 효율적으로 관리하는 방법을 제공한다. 가장 중요한 리소스 중의 하나가 ODBC 연결에 대한 관리이다. MTS 객체가 데이터베이스에 연결을 요구할 때, 리소스 디스펜서가 이를 풀(pool)에서 가져다가 이용한 뒤, 객체가 연결을 해제하면 이를 다시 풀로 반환한다. 이때 데이터베이스에 연결하는 것이 실제 리소스를 할당하거나 해제하는 것이 아니기 때문에, 이들을 얻고 해제하는 과정이 훨씬 빠르게 이루어지며 MTS 객체 들이 리소스를 공유하기 때문에 리소스도 많이 절약할 수 있다.

- 서버의 역할과 보안

클라이언트가 COM 객체의 메소드를 호출할 때 클라이언트가 메소드를 실행할 권한이 있는지 검사해야 한다. 메소드가 파일 등의 다른 리소스에 접근할 경우에는 그런 리소스에 대해서도 접근 권한이 있는지 검사한다. COM 과 DCOM 서버는 IServerSecutiry 인터페이스의 메소드를 이용하여 클라이언트에 권한 설정을 할 수 있다. 참고로 운영체제가 윈도우 NT 일 경우 각

리소스의 ACL 을 자동으로 검사하기 때문에 리소스에 대한 설정을 따로 해 줄 필요가 없다.
이런 형태의 보안은 권한을 얻는 과정이 복잡하기 때문에 다소 확장성이 떨어진다는 단점이 있다. 그러므로 각 리소스 별로 접근 권한을 설정하는 방법 이외에 role 에 기반을 둔 보안을 MTS 가 지원한다. Role 이란 윈도우 NT 사용자 들과 특정 문자열 이름을 부여 받은 그룹 들의 컬렉션이다. Role 은 MTS 컴포넌트의 모임인 패키지 당 하나씩 부여할 수 있다.
MTS 관리자는 MTS 탐색기를 이용하여 각 사용자와 그룹에 role 을 부여할 수 있고, 정확히 어떤 role 들이 각각의 MTS 컴포넌트에 접근할 수 있는지를 지정할 수 있다.
MTS 는 인터페이스 호출을 한 호출자의 role 을 검사하여 여기에 접근할 수 없다면 호출을 거부한다. 이런 role 에 기반을 둔 보안은 MTS 관리자가 직접 나름대로 변경해서 이용할 수 있기 때문에 장점이 많다.
물론 프로그램에서도 role 을 검사하는 루틴을 삽입할 수 있는데, IsCallerInRole 메소드가 그 역할을 담당한다. 이 메소드를 이용하면 MTS 가 현재 호출자의 role 을 검사하여 role 에 해당하는 코드를 작성하여 사용하게 구현할 수도 있다.

- 서버 트랜잭션

사실상 MTS 의 이름에 들어있는 내용을 보면 마치 MTS 가 트랜잭션을 전담해서 관리하는 서버 처럼 들리는 것이 사실이다. 그렇지만, MTS 가 제공하는 대부분의 기능은 COM 서버를 쉽게 확장할 수 있도록 하는 런타임 환경을 제공하는 것이지 트랜잭션 처리는 주된 기능이 아니다. 트랜잭션 자체는 사실 데이터베이스 서버에서 기본적으로 제공하고 있으며, ODBC 에서도 트랜잭션을 사용할 수 있다. 그렇다면, MTS 가 트랜잭션을 처리하는 것에 대해 지원하는 것은 무엇일까 ? 문제는 하나 이상의 DBMS 를 이용한 트랜잭션을 처리할 경우에 있다. 이런 경우에는 MTS 를 이용하여 트랜잭션을 관리하게 하면 된다.
전통적인 트랜잭션 시스템과는 달리 MTS 는 클라이언트에게 트랜잭션의 경계를 숨긴다. 이런 방식의 장점은 같은 컴포넌트가 내부 트랜잭션에서 실행될 수도 있고, 다른 컴포넌트와 합쳐서 커다란 트랜잭션을 만들 수 있다는 것이다. MTS 가 자동으로 클라이언트의 요구가 있을 때마다 트랜잭션을 생성하기 때문에 여러 컴포넌트를 다양한 방법으로 결합해서 사용할 수 있다.
예를 들어, 앞에서 설명한 제품을 주문하는 시스템 컴포넌트와 은행 계좌간에 현금을 전송하는 컴포넌트가 각각 존재할 때 이들은 각각의 트랜잭션을 처리하지만, 동시에 주문과 현금 거래를 묶어서 실행하는 더 큰 트랜잭션을 묶어서 사용할 수 있다.
예를 들어 클라이언트가 주문장(order entry) MTS 컴포넌트를 생성하고, MTS 가 이 요구를 가로채서 이 컴포넌트가 트랜잭션을 요구하면 자동으로 트랜잭션을 시작한다. 동시에 이 컴포넌트가 IObjectContext 인터페이스의 CreateInstance 메소드를 이용하여 현금 전송 컴포넌트

의 인스턴스를 생성하고, MTS 가 이 컴포넌트를 로드하면 이 컴포넌트에 의해 트랜잭션이 요구된다. 이때 컴포넌트를 생성한 생성자가 이미 트랜잭션의 일부이기 때문에 현금 전송 컴포넌트의 인스턴스는 트랜잭션의 일부가 된다. 현금 전송 컴포넌트가 작업을 끝내면 SetComplete 또는 SetAbort 를 호출하게 되며, 이것으로 트랜잭션이 끝나는 것이 아니라 주문장 컴포넌트가 SetComplete 또는 SetAbort 를 호출해야 전체 트랜잭션이 완료된다. 여기서 이들 컴포넌트가 모두 SetComplete 를 호출한 경우에는 컴포넌트에 의해 변경된 모든 사항이 commit 되거, 하나라도 SetAbort 가 호출되면 전체 변경 사항이 rollback 된다.
이처럼 MTS 를 이용한 트랜잭션 처리는 지금까지 알려진 트랜잭션 처리 방법에 비해 대단히 유연한 것이 장점이다.

트랜잭션이 동작하는 방법

트랜잭션이 동작하는 방법을 알기 위해서 앞에서 간단히 예를 들어 설명한 OrderEntry 를 이용하도록 하겠다. 앞에서 아이템을 주문에 추가하기 위해서는 데이터베이스의 재고 물량을 감소시켜야 했다. 이때 주문에 있는 여러 아이템 들에 대한 재고 물량 정보가 서로 다른 2 개의 데이터베이스에 저장되어 있다고 가정하자. 주문이 완료되면 이들 데이터베이스의 재고 정보는 새롭게 갱신되거나 또는 원래의 상태를 유지하여야 한다.
여기에 대해서 설명하기 전에 몇 가지 설명하고 넘어가야할 부분이 있다.
보통 데이터베이스 시스템이 TP 모니터를 사용한다면 보통 XA 라고 불리는 X/Open 에서 정의한 인터페이스를 지원한다. MTS 는 XA 를 지원하는 데이터베이스와 함께 사용할 수 있다. 마이크로소프트는 TP 모니터와 OLE 트랜잭션이라고 불리는 데이터베이스 사이의 상호작용을 위한 인터페이스를 정의하였다. MS SQL Server 는 현재 이 인터페이스를 지원하고 있으며, 다른 DBMS 도 이를 지원하게 될 것으로 보인다. 어쨌든 트랜잭션이 수행되는 방법은 XA 와 OLE 트랜잭션 중 어느 것을 사용하느냐 여부에 따라 다르다. 여기서는 XA 를 사용하는 방법을 기준으로 설명하겠다.
MTS 객체가 처음 생성되면 MTS 실행부가 이를 감지하게 되고, 이 컴포넌트의 트랜잭션 속성(transaction attribute)을 검사한다. 컴포넌트가 트랜잭션을 필요로 하므로 이 속성이 설정된 것으로 간주하겠다. 이 사실은 새롭게 생성된 MTS 객체와 연관된 컨텍스트 객체에 트랜잭션에 대한 식별자와 함께 저장된다.
트랜잭션을 시작하기 위해서는 MTS 실행부가 DTC(Distributed Transaction Coordinator)에 접근하여 트랜잭션이 시작된다는 것을 기록한다. DTC 는 분리된 프로세스에서 실행되는 윈도우 NT 서비스로 트랜잭션 들을 조화시킨다.
이제 클라이언트가 Add 메소드를 호출하여 아이템을 주문에 추가하려고 하면, 만약 Add 가 처

음으로 호출된 경우에는 메모리에 주문 상태(order state)가 생성된다. Add 메소드는 ODBC 호출을 이용해 적절한 데이터베이스에 연결하게 된다.
MTS 객체가 데이터베이스와의 연결을 요구하면 ODBC 드라이버는 객체의 컨텍스트 객체를 질의해서 이 객체가 트랜잭션에 속해있는지를 알아보게 된다. 만약 트랜잭션에 속해 있다면, 이 경우에 드라이버는 DTC 에서 해당 데이터베이스를 트랜잭션에 추가하고, 데이터베이스에게는 객체의 요구가 트랜잭션의 일부분이라는 것을 알리게 된다.
이 같은 과정이 MTS 객체가 접근하는 각각의 데이터베이스에 대해서 일어나게 된다.
마지막으로 클라이언트는 주문이 Submit 함수를 호출하여 완료하게 되는데, 컨텍스트 객체가 SetComplete 호출을 받으면 DTC 에게 트랜잭션을 commit 하게 된다. 이때 XA 가 사용되면 DTC 는 ODBC 드라이버와 통신을 해서 트랜잭션을 commit 하도록 각각의 데이터베이스에게 요구하게 된다. 이때 XA 대신 OLE 트랜잭션이 사용된다면 DTC 는 ODBC 를 거치지 않고 직접 데이터베이스에 접근할 수 있다.
지금까지 설명한 내용은 MTS 객체를 생성하고, 이를 개발하는데에는 그렇게 중요한 내용이 아니다. 하지만, 트랜잭션이 실제로 어떻게 처리되는지에 대해서 이해하는 것은 향후의 문제 해결이나 개발하는데 기반 지식으로서 큰 도움을 주게될 것이다.

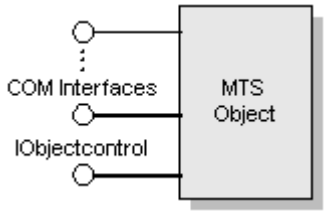
MTS 와 COM

COM 을 분산 어플리케이션 개발 과정에 적용할 경우에는 분산 에러 복구와 concurrency 관리, 로드 밸런싱(load balancing), 데이터 cosistency 등의 여러 가지 문제점을 해결해야 한다. 불행하게도 COM 은 객체가 어떻게 이런 난점 들을 해결해야 하는지에 대해서는 특별한 해결 방안을 제시하지 않고 있다.
그러므로, 이를 해결하기 위해서 개발자가 나름대로의 스키마를 이용하여 작성을 해야 했다.
여기에 대해서 공통적인 인프라 구조로서 제시된 것이 바로 MTS 이다. COM 프로그래밍 모델은 전통적인 OOP 모델을 나름대로 표준화하고 확장한 것이라고 한다면, MTS 프로그래밍 모델은 COM 프로그래밍 모델에 기본적인 상태(state)와 행동(behavior)에 대한 여러가지 관계를 추가하여 확장한 것이라고 생각하면 된다.
MTS 의 기본적인 원칙은 객체가 논리적으로 상태와 행동을 모델링하는 것이다. 그렇지만, 물리적인 구현 방법은 2 가지로 다시 나누어 볼 수 있다.
어플리케이션 개발자는 MTS 를 이용하여 객체의 상태를 관리한다면 잠금 관리나 에러 복구, 로드 밸런싱과 데이터 consistency 등을 지원하기 위한 짐을 상당히 덜 수 있다. 이는 객체의 대부분의 상태가 가상 함수 포인터로 대별되는 객체의 행동(behavior)와 같이 저장되어서는 안된다는 것을 의미한다. 대신 MTS 가 객체의 상태를 RAM 과 같은 비지속성(non-persistent)

저장소나 테이블이나 파일과 같은 지속성(persistent) 저장소에 저장하게 된다. 이렇게 저장된 상태는 MTS 런타임 환경의 제어 하에서 잠금 관리나 데이터 consistency 등에 관계 없이 객체의 메소드에 의해 안전하게 접근이 가능하다. 그러므로, 객체의 상태는 기계가 다운되거나 프로그램이 비정상적으로 종료되더라도 유지될 수 있다.

델파이 4 와 마이크로소프트 트랜잭션 서버 컴포넌트

MTS 컴포넌트는 DLL 의 형태로 구성되는 COM in-process 서버 컴포넌트이다. 이들은 다른 COM 컴포넌트와는 달리 MTS 런타임 환경에서 실행된다. 이들 컴포넌트를 델파이 4 에서 생성하고 구현할 수 있으며, 다른 액티브 X 호환 개발 툴에서도 사용이 가능하다.
MTS 컴포넌트는 델파이에서 클래스로 구현되었다. 참고로 MTS 에서 일컫는 컴포넌트라는 의미는 COM 객체를 구현하는 코드를 의미하므로 델파이의 컴포넌트와 혼돈하지 않기 바란다. 그러므로, 앞으로 언급하는 MTS 컴포넌트란 MTS 클래스를 의미하는 것으로 받아들이기 바란다.
MTS 서버 객체는 작으면서도 괜찮은 성능을 보여 준다. 예를 들어, MTS 컴포넌트는 비즈니스 룰과 어플리케이션 상태에 대한 뷰와 변환 등을 구현할 수 있다. 구체적인 예를 들어, 의학 어플리케이션에서 의료 기록(메디칼 레코드, medical record)은 여러 개의 데이터베이스에 저장되어 관리될 수 있다. 이때 클라이언트 어플리케이션에서는 환자의 그동안의 상태에 대한 정보를 계속 받아본다고 하자. 이럴 때 MTS 컴포넌트는 새로운 환자에 대한 정보, 혈액 검사 결과, 방사선 촬영 결과 등의 상태를 계속 업데이트 해준다.
다음 그림과 같이 MTS 객체는 다른 COM 객체처럼 사용된다. MTS 객체는 COM 인터페이스 이외에 MTS 인터페이스를 지원한다. IUnknown 인 COM 객체에서 공통적인 인터페이스인 것처럼, IObjectControl 은 모든 MTS 객체에서 공통적으로 지원하는 인터페이스 이다. IObjectControl 인터페이스는 MTS 객체를 활성화하거나, 비활성화하는 메소드와 데이터베이스 연결과 같은 리소스를 다루는 메소드를 제공한다,



클라이언트 측에서 보면, MTS 환경 내에 있는 COM 객체는 다른 COM 객체와 별 차이가 없어 보인다. 이럴 때 MTS 는 클라이언트의 요구를 서비스하는 프록시 역할을 하게 된다. 보통

MTS 컴포넌트는 MTS 실행파일(.EXE)에 담겨 있는 in-process 서버이다. MTS 실행파일을 실행하면 MTS 객체는 리소스 관리, 트랜잭션 지원 등의 MTS 런타임 환경의 혜택을 누릴 수 있다.
연결 정보는 MTS 프록시에 의해 관리된다. MTS 클라이언트와 프록시의 연결은 클라이언트가 서버에게 연결을 요구하는 동안 계속 유지되며, 그렇기 때문에 클라이언트는 마치 서버와의 연결이 계속 유지되는 것으로 생각하게 된다.

MTS 컴포넌트의 요구 사항

MTS 는 COM 에 비해 요구사항이 많다. 우선 MTS 컴포넌트는 반드시 DLL 이어야 한다. 또한, 다음의 몇 가지 요구사항을 반드시 만족시켜야 한다.

1. 컴포넌트는 반드시 표준 클래스 팩토리를 가져야 한다. 델파이 4 의 MTS 자동화 위저드를 사용하면 이것이 자동으로 제공된다.
2. 모든 컴포넌트 인터페이스와 클래스는 반드시 타입 라이브러리에 기술되어야 한다. 이 역시 델파이 4 의 위저드를 이용하면 자동으로 실행해 준다. 타입 라이브러리에 메소드와 프로퍼티를 추가할 때에는 타입 라이브러리 에디터를 사용한다. 타입 라이브러리의 정보는 MTS 탐색기(MTS Explorer)에 의해 설치된 컴포넌트에 대한 정보를 런타임에서 조회할 수 있다.
3. 컴포넌트는 반드시 표준 COM 마샬링을 사용하는 인터페이스만 export 해야 한다. 이 역시 위저드에 의해 자동으로 제공된다.
4. 델파이의 MTS 는 사용자 정의 인터페이스에 대한 수동 마샬링을 지원하지 않는다. 모든 인터페이스는 COM 의 자동 마샬링을 이용하는 듀얼 인터페이스를 사용하여야 한다.
5. 컴포넌트는 반드시 DllRegisterServer 함수를 export 해야 하며, 이를 통해 CLSID, ProgID, 인터페이스와 타입 라이브러리의 Self-registration 을 지원해야 한다. 이것도 마찬가지로 위저드에 의해 제공되어야 한다.

리소스 pooling 과 관리

- Just-in-time 활성화

클라이언트 레퍼런스를 가지고 있는 동안 객체가 비활성화 되었다가 활성화 되는 것을 just-in-time 활성화라고 한다. 클라이언트 측에서 보면 클라이언트가 객체의 인스턴스를 생성한

시점에서 마지막으로 해제할 때까지 단지 하나의 인스턴스가 존재한다. 그러나, 실제로 객체는 여러 차례 활성화, 비활성화 될 수 있다. 객체가 비활성화 되면 MTS 는 데이터베이스 연결 등의 객체의 모든 리소스를 해제한다.
COM 객체가 MTS 환경의 한 파트로 생성되면, 여기에 해당되는 컨텍스트 객체(context object)도 같이 생성된다. 이 컨텍스트 객체는 하나 이상의 재활성화 사이클(reactivation cycle)에서 계속해서 존재하게 된다. MTS 는 객체 컨텍스트를 이용해서 비활성화된 동안 객체를 추적한다. 이런 컨텍스트 객체는 IObjectContext 인터페이스를 통해 접근할 수 있으며, 트랜잭션을 관리한다. 비활성화 상태에서 생성된 COM 객체는 클라이언트의 요구를 받으면 활성화된다.
MTS 객체는 다음의 경우에 비활성화 된다.

1. SetComplete 나 SetAbort 에 의해 객체가 비활성화를 요구하는 경우:
   객체는 객체가 모든 작업을 끝내고, 클라이언트의 다음 호출이 있을 때 객체의 내부 상태(internal object state)를 저장할 필요가 없을 때, IObjectContext 인터페이스의 SetComplete 메소드를 호출한다. 또한, 작업을 정상적으로 끝내지 못하고, 내부 상태를 저장할 필요가 없을 때에는 SetAbort 를 호출한다. 이때 객체의 상태는 트랜잭션이 있기 전의 상태로 돌아간다. 가끔 객체가 상태(state)를 가지지 않도록 디자인 된 경우도 있는데, 이럴 때에는 모든 메소드에서 반환될 때 객체가 비활성화된다.
2. 트랜잭션이 commit 되거나 취소된 경우:
   객체의 트랜잭션이 commit 되거나 취소된 경우 객체는 비활성화 된다. 이런 비활성화된 객체들에서 계속 존재하는 것은 트랜잭션의 외부에 있는 클라이언트로부터의 레퍼런스이다. 이들을 재활성화하는 호출은 다음 트랜잭션을 실행하도록 한다.
3. 마지막 클라이언트가 객체를 해제한 경우:
   클라이언트가 객체를 해제하면, 객체는 비활성화되며 객체 컨텍스트도 해제된다.

- 리소스 pooling

MTS 는 비활성화된 동안 놀고 있는 시스템 리소스를 해제하고, 이렇게 해제된 리소스는 다른 서버 객체에 의해 사용될 수 있게 된다. 예를 들어, 데이터베이스 연결이 서버 객체에 의해 더 이상 사용되지 않으면, 다른 클라이언트에 의해 사용될 수 있다. 이런 것들을 리소스 pooling 이라고 한다.
데이터베이스에 연결을 하고, 끊는 작업은 꽤 시간이 걸리는 작업이다. MTS 는 리소스 디스펜서(resource dispenser)를 이용하여 새로운 데이터베이스 연결을 생성하기 보다는, 현재 존재

하고 있는 데이터베이스 연결을 재사용할 수 있도록 해준다. 리소스 디스펜서는 데이터베이스 연결 등의 리소스를 캐쉬에 저장하기 때문에, 하나의 패키지에 들어있는 컴포넌트 들은 리소스를 공유할 수 있다. 예를 들어, 하나의 어플리케이션에 데이터베이스 검색과 데이터베이스 업데이트 컴포넌트를 같이 가지고 있다고 하자. 이때 이 컴포넌트들을 하나의 패키지로 묶으면 데이터베이스 연결을 공유하게 할 수 있다.

- 객체 pooling

MTS 는 리소스 뿐만 아니라 객체를 관리하는 데에도 유용하게 쓸 수 있다. MTS 가 비활성화 메소드를 호출하고 나면, CanBePooled 메소드를 호출하는데 이 메소드를 통해 재사용을 위해 객체가 pooling 될 수 있는지 여부를 알 수 있다. CanBePooled 가 True 로 설정되면 비활성화시 객체를 파괴하기 보다, 객체 pool 로 객체를 이동한다. 객체 pool 의 객체들은 다른 클라이언트가 이를 요구하면 즉시 사용될 수 있다. 객체 pool 이 비어있을 때에만 MTS 가 새로운 객체 인스턴스를 생성한다.
그런데, 현재 까지의 MTS 버전은 객체 pooling 과 recycling 을 지원하지 않는다. MTS 는 CanBePooled 메소드를 호출하지만 실제 pooling 은 일어나지 않는 것이다. 그렇지만 앞으로의 버전에서 이를 지원하게 될 것이다. 델파이 4 는 CanBePooled 를 False 로 초기화하기 때문에, 기본적으로 객체 pooling 이 지원되지 않는다. 앞으로 지원이 된다면 이를 True 로 초기화해서 사용해야 할 것이다.

- 객체 컨텍스트로의 접근

MTS 를 사용하는 COM 객체는 사용되기 전에 생성된다. COM 클라이언트는 COM 라이브러리 함수인 CoCreateInstance 를 호출하여 객체를 생성한다.
MTS 에서 동작하는 COM 객체는 반드시 컨텍스트 객체를 가져야 한다. 컨텍스트 객체는 MTS 에 의해 자동으로 구현되며, MTS 컴포넌트와 트랜잭션을 관리하게 된다. 컨텍스트 객체의 인터페이스는 IObjectContext 이다. 객체 컨텍스트의 대부분의 메소드에 접근하려면 TMtsAutoObject 객체의 ObjectContext 프로퍼티를 이용하거나, TMtsAutoObject 의 메소드를 직접 이용하면 된다.

- 리소스의 해제

개발자는 객체의 리소스를 해제할 의무가 있다. 보통 이럴 때 클라이언트의 요구를 서비스하

고 나서 SetComplete, SetAbort 메소드를 이용한다. 이들 메소드는 MTS 리소스 디스펜서에 의해 할당된 리소스를 해제한다. 동시에 개발자는 MTS 객체나 컨텍스트 객체를 포함한 다른 객체에 대한 레퍼런스를 비롯한, 다른 모든 리소스에 대한 레퍼런스를 해제해야 한다.
이런 메소드를 호출하지 않는 경우는 클라이언트 호출 사이에 상태(state)를 유지하고자 하는 경우이다.

리소스 디스펜서 (Resource dispensers)

리소스 디스펜서는 비영속적인(nondurable) 어플리케이션 컴포넌트의 상태(state)를 관리한다. SQL 서버와 같은 리소스 관리자와 비슷하지만, 영속성에 대한 보장을 하지 않는다는 점이 다르다. 델파이 4 에서는 MTS 에 대해서 2 개의 리소스 디스펜서를 제공한다.

- BDE 리소스 디스펜서
- 공유 프로퍼티 관리자 (Shared Property Manager)

BDE 리소스 디스펜서는 표준 데이터베이스 인터페이스를 사용하는 MTS 컴포넌트에 대한 데이터베이스 연결의 pool 을 관리한다. 즉, 객체들에 대한 데이터베이스 연결을 빠르면서 효과적으로 할당한다. 원격 MTS 데이터 모듈의 경우 가능한 연결 들이 객체의 트랜젝션에 나열되며, 리소스 디스펜서는 자동으로 이들 연결을 재사용한다.
MTS 자동화 객체에 자동으로 객체 트랜잭션을 나열하고, 연결을 재사용하려면 uses 절에 BDEMTS 유닛을 추가한다.

기초 클라이언트(Base clients)와 MTS 컴포넌트

MTS 런타임 환경에서는 클라이언트와 객체 사이의 차이점을 이해하는 것이 중요하다. 클라이언트(기초 클라이언트)는 기본적으로는 MTS 하에서 돌아가는 것이 아니다. 기초 클라이언트는 MTS 객체의 주된 사용자이다. 전형적으로 클라이언트는 어플리케이션의 사용자 인터페이스를 제공하거나 최종 사용자의 요구를 MTS 서버 객체에 정의된 비즈니스 함수에 매핑한다. 클라이언트는 트랜잭션 지원이나 리소스 디스펜서 등의 MTS 에서 지원하는 여러가지 혜택을 누리지 못한다.

MTS 자동화 위저드의 이용

MTS 자동화 객체를 만들려면 다음과 같은 과정을 밟는다.

1. File|New 메뉴에서 ActiveX 탭을 선택한다.
2. MTS Automation Object 아이콘을 더블 클릭한다.
3. 자동화 객체의 이름을 적는다.
4. 쓰레딩 모델과 트랜잭션에 대한 옵션을 선택하고 OK 를 클릭한다.

이런 과정을 완료하면, 현재의 프로젝트에 자동화 객체의 정의를 포함한 새로운 유닛에 추가된다. 또한, 위저드는 타입 라이브러리 프로젝트를 추가하게 되는데, 여기에서 자동화 객체에서 사용하게 될 프로퍼티와 메소드를 노출시킨다. 이렇게 생성된 자동화 객체는 듀얼 인터페이스를 지원한다.
MTS 자동화 위저드는 기본적인 IObjectControl 인터페이스의 Activate, Deactivate, CanBePooled 메소드를 구현한다.

MTS activities

MTS 는 activity 를 통해 concurrency 를 지원한다. 각각의 MTS 객체는 하나의 activity 에 속하며, 이들은 객체의 컨텍스트에 기록된다. 객체와 activity 의 관계는 변할 수 없다. Activity 에는 기초 클라이언트에서 생성된 MTS 객체와 이런 객체에 의해 생성된 MTS 객체와 그 자손들이 포함된다. 이런 객체 들은 하나 이상의 프로세스에 분산되며, 하나 이상의 컴퓨터에서 실행된다.
예를 들어, 의료 어플리케이션에서 여러 종류의 의료 데이터베이스에 레코드를 업데이트하고, 제거하는 MTS 객체가 있다고 하자. 이들은 각각 다른 객체에 의해 작동하게 된다. 이들 객체도 트랜잭션을 기록하기 위한 객체 등의 다른 객체를 사용할 수 있다. 결국 여러 개의 MTS 객체 들이 직간접적으로 기초 클라이언트의 아래에 놓이게 되며, 이들은 모두 동일한 activity 에 속한다.
MTS 는 각각의 activity 를 통해 실행되며, 어플리케이션 상태를 망가뜨릴 수 있는 여러가지 상황을 방지한다. 이런 형태는 결국 하나의 논리적인 쓰레드가 여러 객체 들이 분산되어 있음에도 잘 실행될 수 있도록 하는 것이다. 하나의 논리적 쓰레드를 가진 어플리케이션은 작성하기가 쉽다.
새로운 MTS 객체가 생성되면, 새로운 activity 가 생성된다. MTS 객체가 현재 존재하는 컨텍스트에서 생성된다면 새로운 객체는 같은 activity 의 멤버가 된다. 이럴 때 컨텍스트는 트랜잭션 컨텍스트 객체이거나 MTS 객체 컨텍스트일 수 있다.

MTS 는 하나의 activity 내에는 단지 하나의 논리적 실행 쓰레드 만을 허용한다. 이것은 객체들이 여러 프로세스에 분산될 수 있다는 것을 제외하면 COM apartment 와 유사하다. 기초 클라이언트가 activity 를 호출하면 첫번째 실행 쓰레드가 클라이언트로 돌아올 때까지 activity 내의 다른 모든 요구(다른 클라이언트 쓰레드의 요구 등)는 중지된다.

객체 레퍼런스(Object references)와 콜백

객체 레퍼런스를 넘겨줄 때에는 CoCreateInstance, IObjectContext.CreateInstance, ITransactionContext.CreateInstance 등의 객체 생성후 반환되는 레퍼런스를 이용하거나, QueryInterface 의 호출, 객체 레퍼런스를 얻기 위한 SafeRef 호출 등의 방법을 사용하게 된다. 이런 방법으로 획득한 객체 레퍼런스를 세이프 레퍼런스(safe reference)라고 한다. MTS 는 세이프 레퍼런스에 의해 호출되는 메소드는 적절한 컨텍스트에서 실행된다고 간주한다. 또한, 이런 호출은 MTS 런타임 환경을 이용한다. 이렇게 함으로써 MTS 가 컨텍스트 스위치를 관리하고, MTS 객체가 클라이언트 레퍼런스에 독립적으로 존재할 수 있게 된다.

- SafeRef 메소드의 이용

객체는 SafeRef 함수를 이용해서 레퍼런스를 얻고, 이를 컨텍스트의 외부로 안전하게 넘겨줄 수 있다. SafeRef 는 다음을 입력으로 받을 수 있다.

- 현재 객체가 다른 객체나 클라이언트에게 넘겨주기를 바라는 인터페이스 ID 의 레퍼런스 (RIID)
- 현재 객체의 인터페이스에 대한 레퍼런스

SafeRef 는 현재 객체의 컨텍스트 외부로 안전하게 넘길 수 있는 RIID 파라미터에 지정된 인터페이스의 포인터를 반환한다. 이때 nil 이 반환되면, 객체가 자신이 아닌 객체에서 세이프 레퍼런스를 요구했거나 RIID 파라미터에 의해 요구된 인터페이스가 구현되지 않은 경우이다.
MTS 객체가 셀프-레퍼런스(self-reference)를 클라이언트나 다른 객체에게 넘기길 원할 때(예를 들어 콜백을 사용할 경우) 언제나 SafeRef 를 먼저 호출하고, 이를 통해 반환된 레퍼런스를 넘겨준다. 객체는 셀프 포인터나 QueryInterface 의 내부 호출을 통해 얻은 셀프 레퍼런스를 클라이언트나 다른 객체에 넘겨서는 안된다. 이런 레퍼런스가 객체 컨텍스트 외부로 전달되면, 더 이상 유효한 레퍼런스가 되지 못한다.

- 콜백 (Callbacks)

객체는 클라이언트나 다른 MTS 컴포넌트에 대해 콜백을 만들 수 있다. 예를 들어 다른 객체를 생성한 객체를 가진 경우, 객체의 생성은 자신의 레퍼런스를 생성된 객체에게 넘겨주게 된다. 생성된 객체는 이 레퍼런스를 이용하여 자신을 생성한 객체를 호출한다.
콜백을 사용할 때에는 다음과 같은 제약이 있다.

- 기초 클라이언트나 다른 패키지으로의 콜백은 클라이언트에 접근-레벨 보안(access-level security)을 필요로 한다. 또한, 클라이언트는 반드시 DCOM 서버이어야 한다.
- 방화벽(firewall)을 간섭할 경우 클라이언트로의 콜백이 중단된다.
- 콜백은 같은 트랙잰션이 될 수도 있고, 다른 트랜잭션이 될 수도 있으며 트랜잭션이 아닐 수도 있다.
- 객체의 생성에는 반드시 SafeRef 의 호출이 필요하며, 여기서 반환된 레퍼런스를 콜백을 위해 생성된 객체에 넘겨주어야 한다.

MTS 객체의 MTS 패키지로의 설치

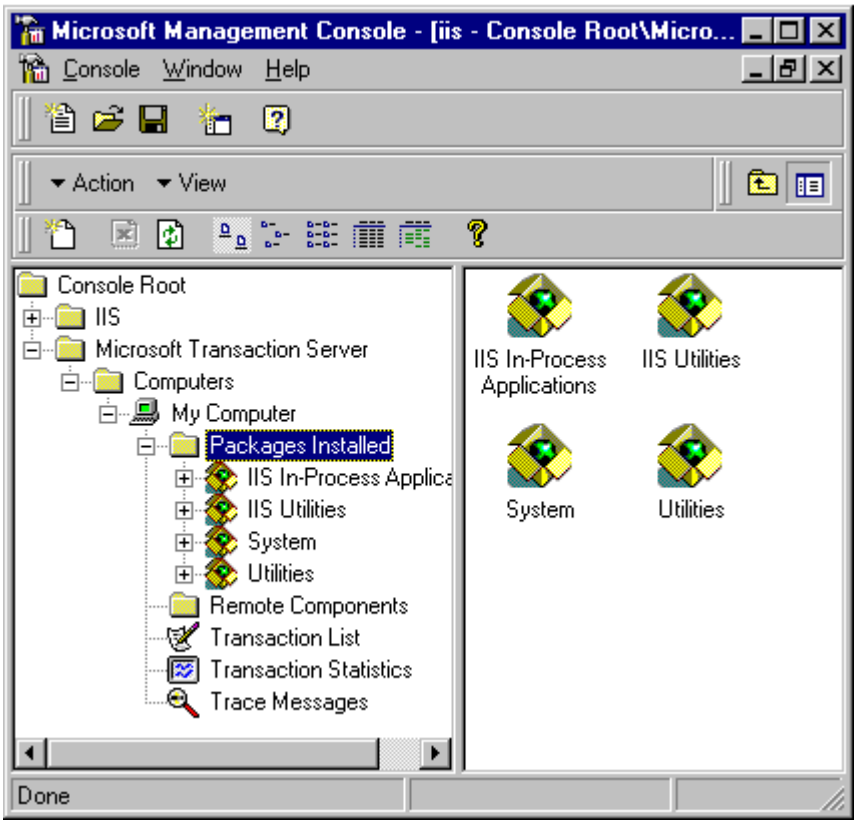
MTS 어플리케이션은 하나의 MTS 실행파일 인스턴스에서 동작하는 in-process MTS 자동화 객체들 또는 MTS 리모트 데이터 모듈의 그룹으로 구성되어 있다. 같은 프로세스에서 동작하는 COM 객체 들의 그룹을 패키지라고 하는데, 하나의 기계는 여러 개의 다른 패키지에서 실행될 수 있으며, 각각의 패키지는 여러 개의 MTS 실행 파일에서 실행된다.
개발자는 어플리케이션 컴포넌트들을 하나의 프로세스에서 실행하기 위해 하나의 패키지로 그룹을 짓는다. 그러나 어떤 경우에는 컴포넌트들을 서로 다른 패키지에 분산시켜, 어플리케이션이 여러 개의 프로세스나 기계에서 실행되기를 원할 수 있다.
MTS 객체 들을 하나의 패키지로 설치하려면 다음과 같이 한다.

1. Run|Install MTS Objects 메뉴를 선택한다.
2. 설치될 CoClass 들을 체크한다.
3. MTS 카타로그를 refresh 하려면 OK 를 선택한다.
4. 현재 존재하는 패키지에 컴포넌트를 설치하려면 Into Existing Package 탭을 선택하고 컴포넌트를 설치할 패키지의 이름을 고른다. 그렇지 않으면, Into New Package 탭을 선택하고 새롭게 생성할 패키지의 이름을 적는다.

패키지는 여러 DLL 의 컴포넌트를 포함할 수 있으며, 하나의 DLL 의 컴포넌트를 여러 개의 패키지로 나누어 설치할 수도 있다. 그러나, 하나의 컴포넌트는 여러 패키지로 분산될 수 없다.

MTS 객체들과 MTS 탐색기(Explorer)

일단 MTS 객체들을 MTS 런타임 환경에 설치하면, 이런 런타임 객체 들을 MTS 탐색기를 통해 관리할 수 있다. MTS 탐색기는 MTS 컴포넌트를 관리하고, 배포하는 기능을 담당하는 GUI 이다.
다음 그림은 실제 MTS 탐색기를 띄워서 패키지를 보여주는 화면이다.

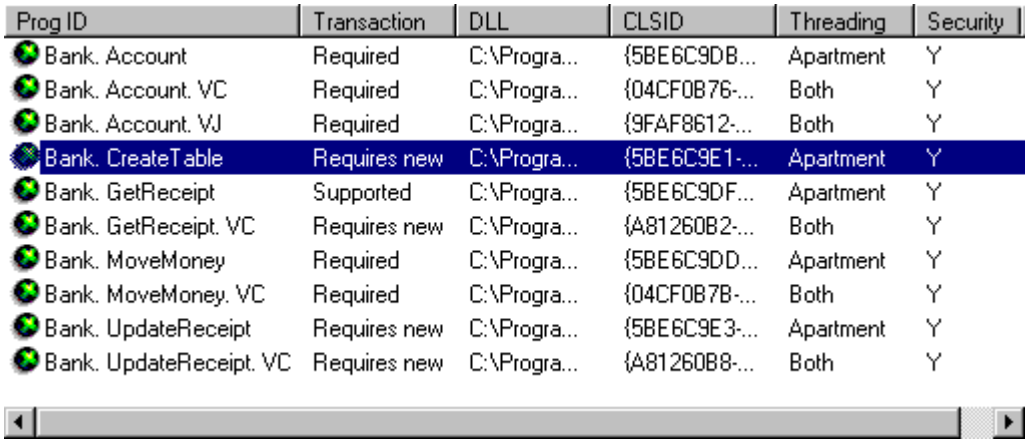


MTS 탐색기의 역할은 다음과 같다.

1. MTS 객체, 패키지, role 의 환경 설정
2. 패키지 내의 컴포넌트의 프로퍼티와 컴퓨터에 설치된 패키지를 보여준다.
3. 트랜잭션을 구성하는 MTS 컴포넌트 들에 대한 트랜잭션을 모니터, 관리한다.

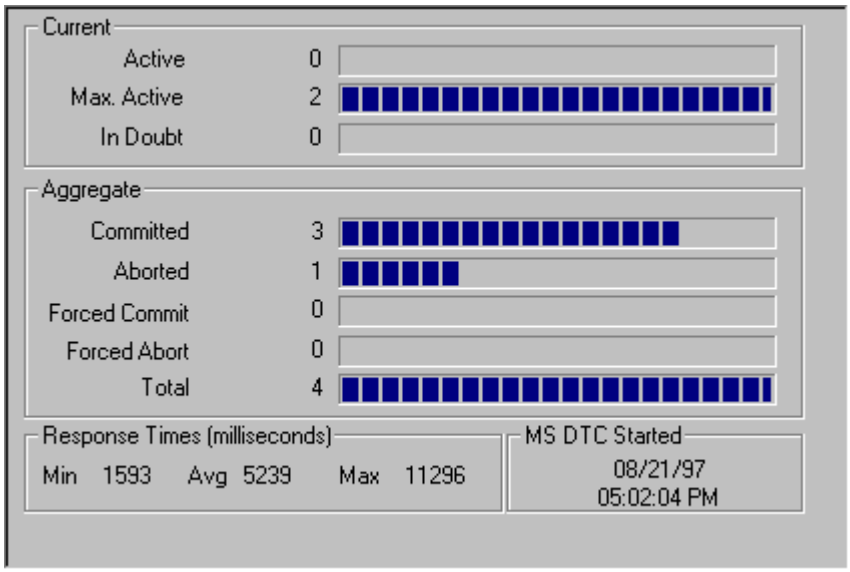
4. 컴퓨터 사이에 패키지를 이동한다.
5. 원격 MTS 객체를 로컬 클라이언트에서 쓸 수 있게 한다.

MTS 탐색기는 개발자 뿐만 아니라, 시스템과 웹 관리자 들이 사용하여 패키지를 설치, 배포, 테스트, 관리할 수 있다. 개발자 들은 MTS 탐색기를 이용하여 컴포넌트를 패키지로 모으고, MTS 환경에서 컴포넌트를 분산시키고 이를 테스트하는데 사용하게 되며, 시스템이나 웹 관리자 들은 MTS 탐색기를 이용하여 컴포넌트와 패키지를 설치, 배포, 관리하는데 사용하게 된다. 다음과 같이 MTS 탐색기의 프로퍼티 윈도우(properties window)를 이용하면 패키지에 포함된 컴포넌트의 프로퍼티를 보거나 컴퓨터에 설치된 패키지를 살펴볼 수 있다.



Prog ID	Transaction	DLL	CLSID	Threading	Security
Bank. Account	Required	C:\Progra...	{5BE6C9DB...	Apartment	Y
Bank. Account. VC	Required	C:\Progra...	{04CF0B76-...	Both	Y
Bank. Account. VJ	Required	C:\Progra...	{9FAF8612-...	Both	Y
Bank. CreateTable	Requires new	C:\Progra...	{5BE6C9E1-...	Apartment	Y
Bank. GetReceipt	Supported	C:\Progra...	{5BE6C9DF...	Apartment	Y
Bank. GetReceipt. VC	Requires new	C:\Progra...	{A81260B2-...	Both	Y
Bank. MoveMoney	Required	C:\Progra...	{5BE6C9DD...	Apartment	Y
Bank. MoveMoney. VC	Required	C:\Progra...	{04CF0B7B-...	Both	Y
Bank. UpdateReceipt	Requires new	C:\Progra...	{5BE6C9E3-...	Apartment	Y
Bank. UpdateReceipt. VC	Requires new	C:\Progra...	{A81260B8-...	Both	Y

그리고, 트랜잭션 통계 윈도우(Transaction Statistics window)를 이용하면 현재의 트랜잭션에 대한 요약된 통계를 다음과 같이 볼 수 있다.



MTS 탐색기 윈도우의 좌측 pane 에는 객체의 계층도를 표시하고, 우측 pane 에는 좌측 pane 에서 선택된 아이템을 표시하는 역할을 한다.
각 아이템에 대한 기본적인 정보는 아이템의 프로퍼티 시트(property sheet)를 통해 확인할 수 있는데, 예를 들어 컴퓨터 아이템에는 컴퓨터의 이름과 로그 파일의 위치와 업데이트 설정 등이 담겨 있고, 패키지 아이템에 대해서는 보안과 다른 여러가지 설정에 대한 정보를 담게 된다. 프로퍼티 시트를 보기 위해서는 아이템을 선택하고 Action 메뉴에서 Properties 명령을 선택하거나 오른쪽 버튼을 클릭하고 Properties 메뉴를 선택하면 된다.
MTS 의 런타임 환경 중에서 트랜잭션 처리에 대한 부분은 MS DTC(Distributed Transaction Coordinator)가 주로 담당하게 된다. MS DTC 는 MTS 가 사용하는 윈도우 NT 서비스이다.
MTS 탐색기에 대한 더 자세한 내용은 도움말을 참고하기 바란다.

정 리 (Summary)

이번 장에서는 MTS 가 등장하게 된 배경과 MTS 가 지원하는 여러가지 개념에 대한 총론적인 내용과 델파이 4 에서 지원하는 MTS 관련 기능에 대해 전체적으로 알아 보았다.
다음 장에서는 이를 바탕으로 실제 MTS 패키지를 만들고, 이를 설치하여 실행하는 방법에 대해서 알아볼 것이다.