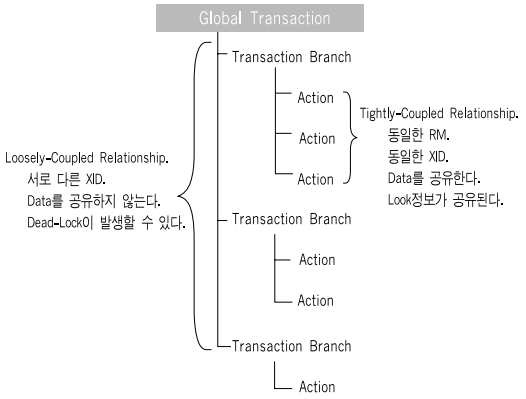
단일 도메인 환경과 복수 도메인 환경에서의 트랜잭션 비교

단일 도메인 환경과 복수 도메인 환경에서의 트랜잭션 비교

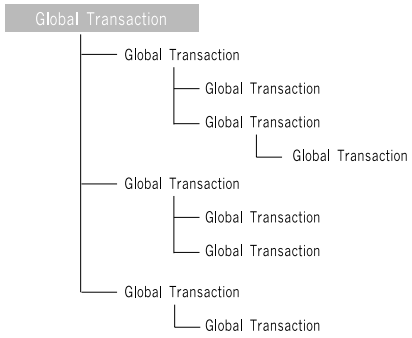
X/OPEN DTP MODEL에서 Transaction Manager(TM)는 Resource Manager(RM)가 제공하는 XA Interface에서 사용되는 Transaction Identifier(XID)의 해석에 따라 강한 연결(Tightly-Coupled Relationship)과 약한 연결(Loosely-Coupled Relationship), 두 가지 형태에 의해 Transaction Tree를 형성한다. 강한 연결은 동일한 GlobalTransaction에 속해 있는 프로세스들이 동일한 RM을 사용하고 동일한XID를 사용하는 경우이다. 강한 연결에서 RM은 프로세스들이 요청한 작업에 대해 전체를 하나의 Transaction으로 취급하기 때문에 서로 Data와 Lock을 공유한다. 약한 연결에서는 각각의 작업에 대해 Transaction Branch를 만든다. 즉, 서로 다른 XID를 사용한다. 약한 연결에서는 서로 다른 XID를 사용하므로 RM은 이들을 서로 별개의 Transaction으로 취급하고 서로 Lock을 공유할 수 없다. 그러므로 Transaction Branch간에 Deadlock이 발생할 수 있다. TUXEDO에서는 동일한 자원을 사용하는 프로세스들을 하나 의 GROUP으로 묶어서 GROUP안에서는 강한 연결을 사용하고 서로 다른 GROUP간의 연결은 약한 연결을 사용한다. TUXEDO에서 동일한 Database를 사용하는 프로세스를 하나의 GROUP으로 묶는 이유는 Deadlock 을 최소로 하기 위함이다.



하나의 Global Tranaction안에서는 모두 동일한 Global Transaction Indentifier(GTRID)를 사용한다. TUXEDO에서는 하나의 GROUP내에서 발생하는 모든 작업은 동일한 Transaction Branch안에 포함되며 이들 은 동일한 RM을 사용하며 동일한 XID를 사용한다. 모든 Transaction Branch는 Root Node 바로 아래 Node에 위치하여 Flat Transaction Tree(2-level Tree)를 형성하게 된다.

Domain이 서로 다르면 TM이 관리하는 Global Transaction의 Boundary를 넘는 것이므로 Global Transaction Identifier (GTRID)가 공유 되지 않는다. TUXEDO에서는 /Domain을 사용하여 Domain간의 Global Transaction들을 연결하여 2-Phase Commit을 구축한다. 하나의 Domain내에서는 하나의 GTRID만을 사용하므로 모든 Transaction Branch가 Transaction Tree에서 동일한 Level에 있게 되므로 Transaction Tree가 평평하다.(Flat Transaction Tree - 2 Level Tree)

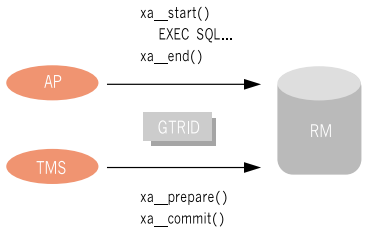
그러나 여러개의 Domain에 걸쳐서 Transaction이 구성되면 여러개의 GTRID가 사용되고 이들은 계층적인 Transaction Tree(Hierarchical Transaction Tree)를 형성하게 된다. 단일 Domain에서는 하나의 GTRID만이 있으며 하나의 Global Transaction Table(GTT)만이 관련 되어 있다. 그러므로 TM이 Global Transaction의 완료를 요청 받았을 때 TM은 이 Global Transaction에 관련된 모든 Participant를 GTT를 통하여 알 수 있고 Transaction Tree가 평평하기 때문에 실제 Service 호출 순서와는 무관하게 Global Transaction의 완료를 실행할 수 있다. 그러나 복수 Domain에서는 여러개의 GTRID가 관여하게 되고 GTT도 여러 개가 있게되므로 이들 GTRID간의 상호 관계는 Service 호출 순서에 영향 을 받는다. 즉, 동일한 Service들을 호출하더라도 호출 순서에 따라 Transaction Branch가 발생하는 순서가 다르므로 Transaction Tree 가 달라지게 된다.



각각의 Global Transaction은 서로 독립적인 GTRID을 갖는다. 이들이 하나의 Transaction으로 처리되어 2-Phase Commit을 달성하기 위해 서는 GTRID를 서로 연관시켜주는 작업이 필요하며, 이는 Domain Gateway의 역할이다. 각각의 GTRID는 기본적으로 서로 다른 GTT에 등록 되므로 하나의 GTT가 전체 Transaction Tree를 완전히 파악하고 있을 수 없다. 그러므로 Transaction Tree가 평평한 구조를 가질 수 없고 계층적인 Tree(Hierarchical Transaction Tree)를 형성할 수 밖에 없다.

단일 Domain환경에서 2-Phase Commit의 구현

TUXEDO에서는 RM을 사용하는 부분과 이와 관련된 Global Transaction 을 관리하는 부분을 분리하여 처리한다. 즉 Application Program (AP)이 RM이 제공하는 Native Interface(Embedded SQL, Call Level Interface, etc)를 사용하여 필요로 하는 작업을 하고, 이와 관련된 Global Transaction은 Transaction Management Server(TMS)가 처리하도록 구성되어 있다. 이 둘간의 Global Transaction에 대한 정보 교류는 GTRID를 통하여 이루어지며, 이 GTRID는 Client가 Global Transaction의 시작을 요청할 때 부여되어 GTT에 등록된다.



GTRID(Global Transaction Identifier)
Global Transaction들을 구분 하기 사용되는 고유한 값. Application Program(AP)과 Transaction Management Server(TMS)는 이 GTRID를 서로 공유하여 Transaction을 공유한다.

xa_start() XA Session의 시작을 알린다.
서비스로 진입하기 전에 호출된다.

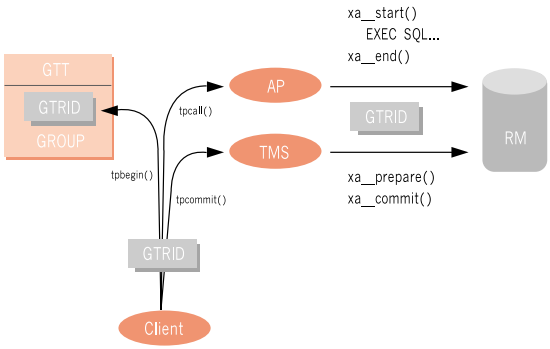
EXEC SQL RM이 제공하는 Native Interface임을 의미한다. Database의 경우 Embedded SQL, Call Level Interface 등 여러 가지의 형태로 제공되고 있다.

xa end() XA Session의 종료를 알린다.
서비스에서 tpreturn()를 호출하면 tpreturn() 내부에서 호출된다.

xa_prepare() 2-Phase Commit의 Phase I 의 시작을 알린다. 이 단계에서는 RM은 내부적으로 정보를 저장하고 그 정보에 대한 Lock을 유지한다.

xa_commit() 2-Phase Commit의 Phase II 의 시작을 알린다. 이 단계에서 RM은 Phase I 에서 유지한 Lock을 해제 한다.

Client가 Global Transaction의 시작을 요청(tpbegin())하면 TUXEDO 는 이에 고유한 GTRID를 할당하고 이 Global Transaction을 Global Transaction Table(GTT)에 등록한다. 이후 Client가 필요로 하는 Service를 호출(tpcall(), tpacall(), etc)하면, TUXEDO는 GTRID를 사용 하여 Client가 요청한 Service의 정보(Service가 속해있는 GROUP 정보)를 GTT에 기록한다. Service로 요청이 들어가기 직전에 TUXEDO는 Database에 XA Session의 시작을 알린다.(xa_start()) Service는 필요 로 하는 작업을 수행(EXEC SQL ...)하고, Client에 처리결과를 반환(tpreturn()) 하면 TUXEDO는 Database에게 XA Session의 종료를 알린다.(xa_end()) 이때 Service가 RM에 수행한 작업은 RM의 Log에 남게 되고, 나중에 Transaction 처리 요청이 RM에 전달(xa_prepare(), xacommit(), etc)되면 RM은 이 Log를 바탕으로 Transaction처리를 하게 된다.



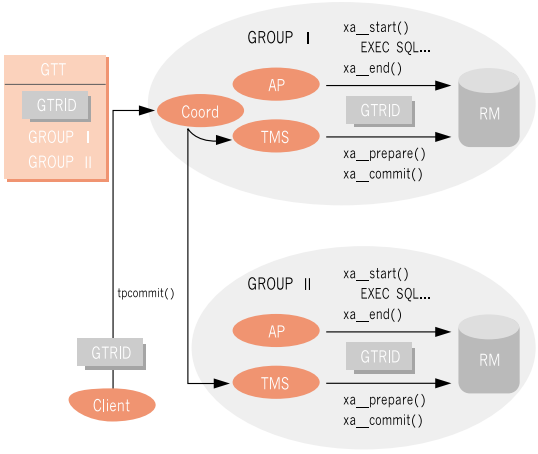
GTT(Global Transaction Table)
현재 사용되고 있는 GTRID와 이와 관련된 GROUP의 정보를 가지고 있다.

단일 도메인 환경과 복수 도메인 환경에서의 트랜잭션 비교

tpbegin()	Global Transaction의 시작을 알린다. 이 단계에서 TUXEDO는 GTRID를 생성하여 이를 GTT 에 등록하고 이를 Client에 건네준다. 이후에 발생하는 모든 호출에서 이 GTRID가 사용된다.
tpcall()	tpbegin()이 호출된 이후의 모든 서비스 호출은 TPNOTRAN이 설정되어 있지 않으면 이 Global Transaction에 포함된다. 또한 모든 메시지에 GTRID를 같이 전달하여 동일한 Global Transaction에 속해 있음을 알게 한다.
tpcommit()	Global Transaction의 종료를 지시한다.

이제 이 Server Process는 Client가 기동한 Global Transaction과 분리 되어 있으므로 즉시 다른 Service 요청을 실행할 수 있게 된다. 이와 같이 AP를 Global Transaction으로부터 분리시킴으로써 높은 가용성을 달성 할 수 있다. 만일 Global Transaction이 종료할 때까지 AP가 대기해야 한다면 동시에 Global Transaction이 발생하는 만큼의 AP가 실행되어야 할 것이다. TUXEDO에서는 이와 같은 지연 시간을 제거함으로써 필요한 Server Process의 수를 최소로 한다.

Client는 이와 같은 Service 호출을 반복할 수 있고 모든 Service 호출이 정상적으로 반환되면 Global Transaction의 종료를 요청한다. (tpcommit()) TUXEDO는 GTRID를 이용하여 이 Global Transaction 에 관련된 모든 GROUP을 GTT에서 알 수 있고, 이 중의 하나 (주로 맨 처음 Service를 요청받은 GROUP)를 Coordinator로 지정하여 Global Transaction의 처리를 요청한다. Coordinator는 자기 자신과 모든 Subordinate에게 xa_prepare()를 호출하도록 지시한다.(Phase I의 시작) xa_prepare()를 호출하도록 요청받은 TMS Server들은 xa_prepare()를 호출하고 xa_prepare()의 결과를 Coordinator에게 반환한다. 모든 TMS Server로부터 정상적으로 호출이 반환되면 Coordinator는 TLOG에 Phase I의 완료에 대한 Log를 남기고(Phase I의 종료), 역시 자기 자신과 모든 Subordinate에게 xa_commit()을 호출하도록 지시한다.(Phase II의 시작) xa_commit()을 호출하도록 요청받은 TMS Server들은 xacommit() 을 호출하고 xacommit()의 결과를 Coordinator에게 반환한다. 모든 TMS Server로부터 정상적으로 호출이 반환되면 Coordinator는 TLOG 의 Log를 지우고 클라이언트에게 Global Transaction처리의 완료를 반환 한다.(2-Phase Commit의 종료) Global Transaction의 처리 중에 비 정상적인 상태가 발생하면 TLOG의 Log와 RM의 Log를 이용하여 복구가 된다.



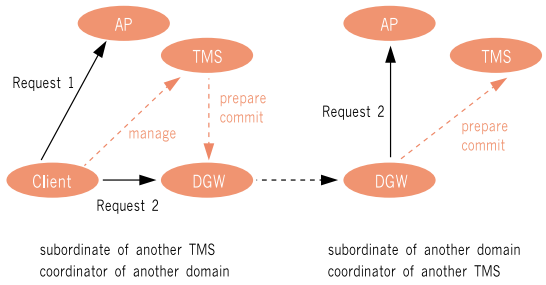
Coordinator	Client로부터 요청을 받아 Global Transaction을 완료한다. 주로 맨 처음 서비스를 수행한 GROUP의 TMS가 이 역할을 한다.
Subordinate	Global Transaction에 포함된 GROUP의 TMS가 Coordinator로부터 메시지를 받아 단계별로 xa_prepare()와 xa_commit()를 호출한다.

이와 같이 GTT에 Global Transaction에 포함된 모든 GROUP의 정보를 가지고 있기 때문에 Client가 서비스를 호출하는 방식(동기, 비동기, 대화형 등)이나 서비스에서 다른 서비스로 Forwarding하는 것과는 무관하게 동일한 방법으로 Global Transaction의 처리가 이루어진다. 다시 말해서 서비스가 호출되는 세부 경로와는 무관하게 Global Transaction의 처리는 이루어진다.

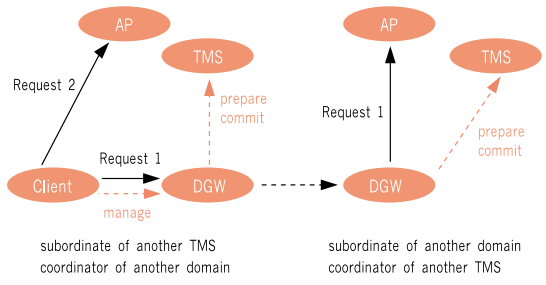
복수 Domain환경에서 2-Phase Commit의 구현

/Domain을 사용하는 환경에서는 2-Phase Commit의 구현은 다소 복잡 하다. 하나의 Domain내에서 발생하는 Transaction에 대해서는 하나의 GTRID를 이용하여 처리를 하기 때문에 Coordinator를 제외한 나머지는 전부 Subordinate가 되어서 평평한 Transaction 구조(Flat Trans-action Tree-2 Level Tree)를 형성하지만 여러 개의 Domain을 사용할 때는 Domain을 넘을 때마다 새로운 GTRID를 부여 받게되고(각 Domain 은 독자적인

Application Boundary를 형성하고 서로 독립적인 GTT를 가지고 있으므로 각기 고유의 GTRID를 생성한다.) 이들 GTRID간의 상호 연결을 Domain Gateway가 처리하게 된다. Domain Gateway는 또한 TMS의 역할을 수행하며 Domain간의 통신을 중계하는 Network Bridge 의 역할도 수행한다.



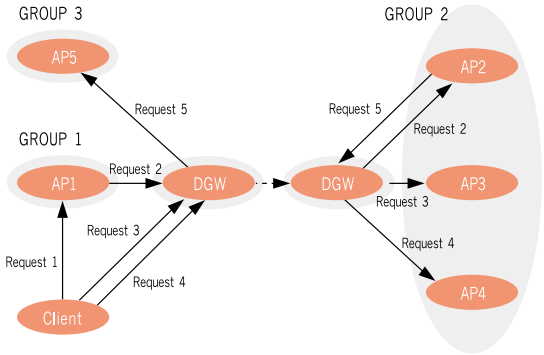
위 그림에서 왼쪽의 Domain Gateway는 TMS의 Subordinate로서의 역할과 다른 Domain의 Coordinator로서의 역할을 수행한다. 오른쪽의 Domain Gateway는 다른 Domain의 Subordinate로서의 역할과 TMS 의 Coordinator로서의 역할을 수행한다.



위 그림에서 왼쪽의 Domain Gateway는 TMS의 Coordinator로서의 역할과 다른 Domain의 Coordinator로서의 역할을 수행한다. 오른쪽의 Domain Gateway는 다른 Domain의 Subordinate로서의 역할과 TMS 의 Coordinator로서의 역할을 수행한다.

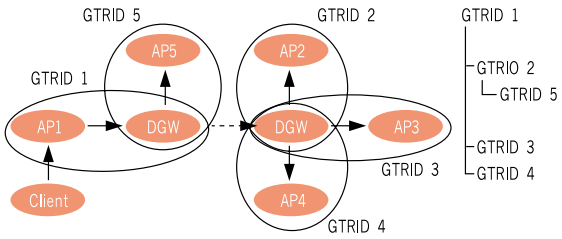
이와 같이 GTRID가 여러개 관련되고 동일한 Service들이라 할지라도 호출 순서에 따라 Transaction Branch가 생성되는 순서가 달라질 수 있으므로 기본적으로 Transaction 구조가 계층적(Hierarchical Transaction Tree)일 수 밖에 없다. /OSITP Instance에서는 Transaction Branch가 Domain과 Domain사이의 호출이 발생하는 만큼 생성되게 되어있다. 그러나 /TDOMAIN Instance에서는 Domain과 Domain 사이의 Domain Gateway쌍이 주어지면 이 Domain Gateway쌍을 이용하는 모든 호출은 하나의 Transaction Branch를 공유하도록 하기 때문에 /OSITP Instance에 비해 Transaction Tree가 단순하다. 이러한 각각의 Transaction Branch에는 각각 하나의 GTRID가 부여된다. /OSITP Instance이건 /TDOMAIN Instance이건 하나의 Transaction Branch내에서는 앞에서의 단일 Domain환경에서의 Global Transaction처리 과정을 따르게 된다.

예를들어 아래 그림과 같은 호출이 발생한다고 가정하자. Client에서는 AP 1, AP 3, AP 4를 차례로 호출하고, AP 1에서 AP 2를 호출한다. AP 2에서는 AP 5를 호출하며, AP 1과 AP 5는 서로 다른 GROUP에 속해 있다고 가정 하자. 물론 Domain Gateway들은 별도의 GROUP으로 구성되어 있다.



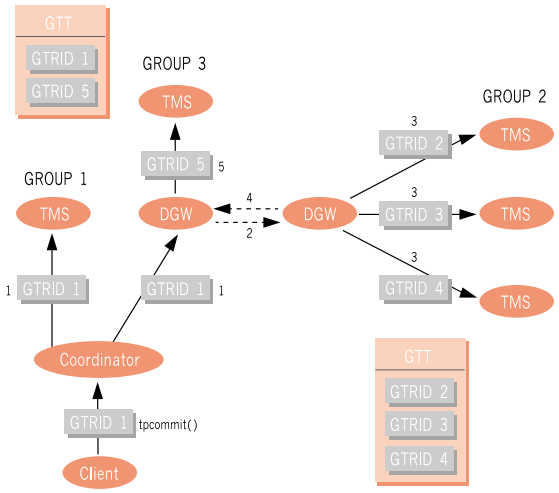
아래의 그림은 위와 같은 호출이 발생했을 때 /OSITP Instance에서의 Transaction Tree의 구조를 나타낸 것이다. 왼쪽은 각 Global Transaction의 영역을 나타낸 것이며 오른쪽은 Transaction Branch의 연결 상태를 나타낸 것이다. 동일한 Domain Gateway를 통하여 동일한 GROUP에 Service 호출을 하더라도 이들은 서로 다른 GTRID를 부여받게 되고(Loosely-Coupled Relationship) 서로 간에 Lock을 공유할 수 없게 되어 Deadlock이 발생할 가능성이 높다. 또한 Transaction Branch가 복잡하게 되어 전체 Transaction을 완료하는 시간이 길어지게 된다.

단일 도메인 환경과 복수 도메인 환경에서의 트랜잭션 비교

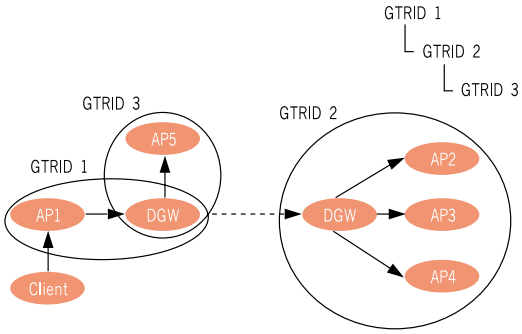


/OSITP Instance

아래의 그림은 /OSITP Instance에서 Client가 tpcommit()을 호출할 때 Transaction 완료 메시지(prepare, commit)의 전파 순서와 경로를 나타낸 것이다. 물론 GTRID간의 연결은 Domain Gateway가 담당한다. 아래 그림 에서 GTRID 1에서는 GROUP 1의 TMS가 Coordinator로서 역할하며 Domain Gateway는 GTRID 1의 Subordinate이며 GTRID 5의 Coordinator이기도 하다. 오른쪽 Domain Gateway는 GTRID 2, GTRID 3, GTRID 4의 Coordinator가 된다.

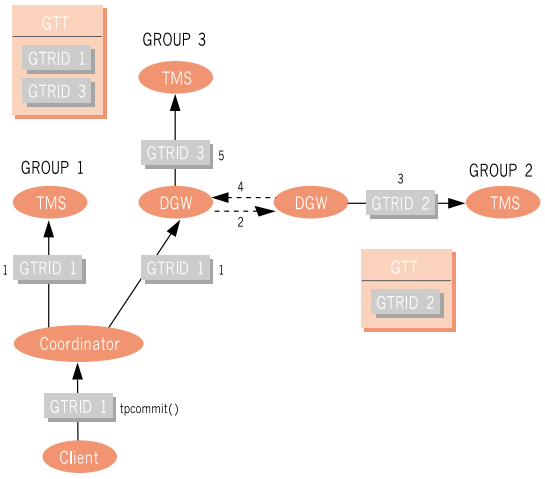


아래의 그림은 같은 호출이 발생했을 때 /TDOMAIN Instance에서의 Transaction Tree의 구조를 나타낸 것이다. /OSITP Instance의 경우 와는 달리 동일한 Domain Gateway를 통하여 Service 호출을 하게 되면 이들은 하나의 GTRID를 부여받게 되고(Tightly-Coupled Relationship) 서로간에 Lock을 공유하기 때문에 Deadlock이 발생할 가능성이 줄어 든다. 또한 Transaction Branch가 /OSITP Instance에 비해 단순하므로 전체 Transaction을 완료하는 시간이 상대적으로 짧다. 그러나 /TDOMAIN Instance의 경우에도 만일 AP2, AP3, AP4가 서로 다른 GROUP에 속해 있다면 /OSITP Instance의 경우와 같아진다.

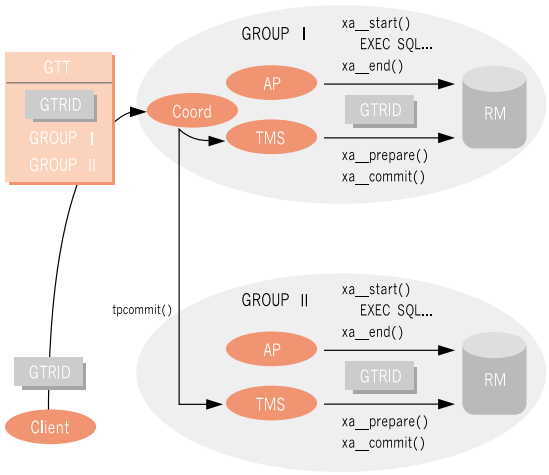


/TDOMAIN Instance

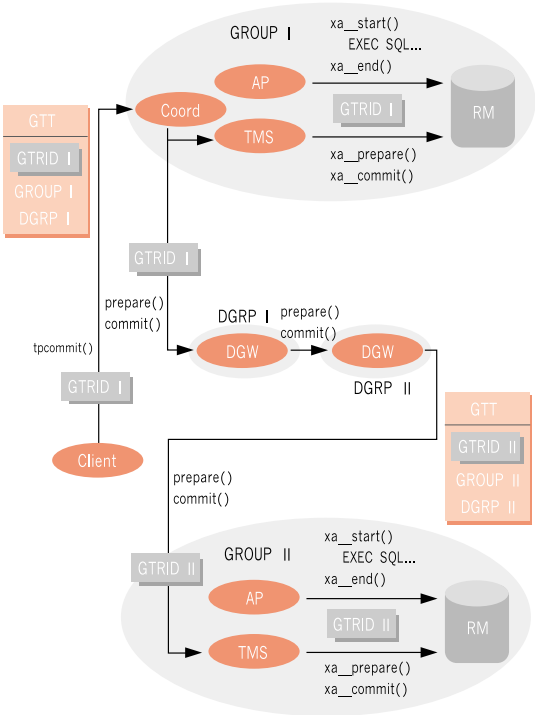
아래의 그림은 /TDOMAIN Instance에서 Client가 tpcommit()을 호출할 때 Transaction 완료 메시지(prepare, commit)의 전파 순서와 경로를 나타낸 것이다. Transaction Tree 자체가 /OSITP Instance에 비해 단순 하기 때문에 Transaction 완료 메시지의 전파 경로도 상대적으로 단순하다.



단일 Domain의 경우와 비교하기 위해 앞에서의 경우(단일 Domain에서 2-Phase Commit)를 /Domain을 사용하는 경우로 변경하여 생각하여 보자. 이 경우는 앞의 단일 Domain의 경우의 2개의 GROUP을 서로 다른 Domain으로 분리한 경우이다. 비교를 하기 위해 앞에서의 그림을 다시 사용 하였다.



▲ 단일 Domain



▲ 복수 Domain

위와 같이 X/OPEN DTP MODEL에 따른 2-Phase Commit의 실행은 단일 Domain 구성 환경이건 복수 Domain 구성 환경이건 실제 사용자가 필요로 하는 Service 호출 뿐만 아니라 데이터 일치를 위해 여러 가지 작업을 하게 되어있다.(위에서 언급한 모든 호출들은 전부 Service 호출에 부가적으로 들어가는 Transaction 관리를 위하여 사용되는 것들이다. 즉, 2-Phase Commit을 사용하지 않는다면 전혀 사용되지 않는 것들이다.) 이러한 부가적 인 작업은 분산 환경에서 데이터 일치를 위해서는 피할 수 없지만 분산 데이터 변경이 발생하지 않는 작업에서는(Data 조회만 한다거나 하나의 Database에서만 변경이 발생하는 경우) 사용하지않는 것이 시스템의 자원을 절약하고 성능을 높일 것이다. 특히 복수 Domain 구성 환경에서의 2-Phase Commit의 과다한 사용은 Domain Gateway에 많은 부하를 주게 된다.