UDDI 3.0에서의 복제 연산의 설계 및 구현

김동민[0], 진주용, 이석호

서울대학교 전기컴퓨터공학부

{kimdm[0], dansun}@db.snu.ac.kr, shlee@cse.snu.ac.kr

Design and Implementation of Replication Operation in UDDI 3.0

Dongmin Kim[0], Juyong Jin, Sukho Lee

School of Electrical Engineering and Computer Science, Seoul National University

요약

하나의 UDDI 노드에 저장된 웹 서비스 정보는 복제 연산을 통해 같은 레지스트리에 속한 다른 노드에도 똑같이 저장된다. 이러한 복제 연산은 changeRecord의 생성, 전송, 처리를 통해 수행된다. 그런데, 한 노드에서 성공적으로 처리된 연산이라도 복제 과정에서는 올바르게 동작하지 않을 수가 있다. 본 논문에서는 기본적인 복제 과정 외에, 복제 연산 중에 발생한 에러의 발견 및 처리, 저널의 효율적인 관리, 키 생성 등의 문제를 해결하기 위한 방법을 제안하고 있다.

1. 서론

UDDI(Universal Description Discovery & Integration)[1]는 웹 서비스 정보에 관한 자료구조와 연산을 정의하는 표준이다. 이러한 표준을 따르는 UDDI 노드는 businessEntity, businessService, bindingTemplate, tModel 등을 등록하고 검색할 수 있는 서버이다.

서비스 제공자는 이러한 UDDI 노드에 자신이 제공할 서비스를 등록(publish)하고, 서비스 이용자는 등록된 정보를 검색(inquiry)할 수 있다. 그런데 UDDI 노드가 자신의 노드에 직접 등록된 정보만 저장하고 있다고 하면, 각 노드의 사용자는 제한된 정보만을 얻을 수 있다.

따라서 UDDI 3.0 명세에서는 동일한 내용을 갖는 여러 개의 UDDI 노드로 하나의 UDDI 레지스트리(registry)를 구성하도록 한다. 이렇게 하면, 하나의 노드에만 접근해도 레지스트리의 모든 정보를 얻을 수 있고, 하나의 노드에 실패가 일어나더라도 동일한 내용의 다른 노드를 이용할 수 있게 된다.

UDDI 레지스트리가 논리적으로 하나의 개체임을 보장하기 위해서는 레지스트리를 구성하는 노드들 간에 주기적으로 서로의 내용을 복제해야 하며, 복제 과정 중에 에러가 발생했을 때, 그 사실을 감지하고 적절하게 복구할 수 있어야 한다. 또한, 레지스트리에 새롭게 추가되는 노드는 레지스트리의 전체 정보를 제공받음으로써 다른 노드들과 동일한 내용을 가질 수 있어야 한다. 본 논문에서는 이러한 요구 사항을 만족시키는 복제 연산의 설계와 구현을 목표로 한다.

논문의 구성은 다음과 같다. 2장에서는 복제가 수행되는 과정을 설명하고, 3장에서는 복제 서브시스템의 구현을 위해 필요한 고려사항에 대해 다룬다. 4장에서는 결론을 맺는다.

2. 복제 연산

UDDI의 복제 연산을 위해서는 changeRecord라는 자료구조를 사용한다. changeRecord는 UDDI 노드에서 한 번 업데이트(삽입, 삭제, 갱신)가 일어날 때마다 하나씩 생성되는 XML 메시지로서 변화에 관한 모든 정보를 포함한다. 한 노드에서 변화가 일어나면 changeRecord가 생성되어(2.1절) 다른 노드에게 전송된다.(2.2절) changeRecord를 받은 노드는 이 내용에 맞춰 자신의 데이타베이스를 변경함으로써(2.3절) 두 노드의 내용은 동일하게 유지될 것이다. 그림 1은 복제 연산이 일어나는 과정을 보여주고 있다.

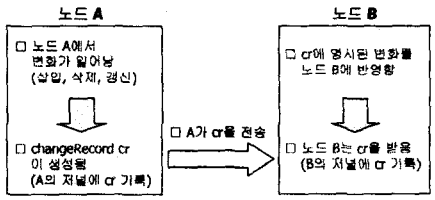


그림 1. 복제 연산의 과정

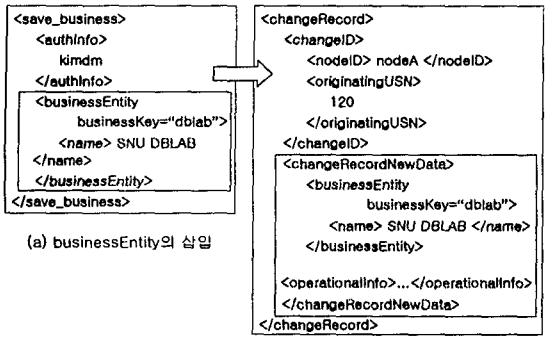
2.1. changeRecord의 생성

삽입, 삭제, 갱신 등의 변화가 노드 A에 발생하면, 노드 A는 변화의 내용에 맞게 changeRecord cr을 생성한다. 이 때 changeRecord는 업데이트의 내용 외에도 그 업데이트가 어느 노드에서 일어났는지를 의미하는 nodeID와 그 노드에서 일어난 몇 번째 업데이트인지를 나타내는 originatingUSN 정보를 포함한다.

그림 2(a)는 노드 A에 businessKey가 "dblab"이고 name이 "SNU DBLAB"인 새로운 businessEntity가 삽입되는 예를 나타낸다. 이에 대응되는 changeRecord는 그림 2(b)에서 확인할 수 있는데, changeRecord의 changeRecordNewData 태그는 변화의 종류가 새로운 데이타의 삽입임을 나타내고 있으며, businessEntity 엘리먼트는 새로 삽입되는 businessEntity의 내용을 포함하고 있다. nodeID, originatingUSN 엘리먼트

* 본 연구는 정보통신부의 대학 IT연구센터(ITRC) 지원을 받아 수행되었습니다.

는 이 변화가 노드 nodeA의 120번째 업데이트임을 나타낸다.



(a) businessEntity의 삽입

(b) changeRecord 메시지

그림 2. changeRecord의 생성

이렇게 생성된 changeRecord는 각 노드의 저널에 기록된다. 저널(journal)은 노드 자신이 생성하거나 다른 노드에게 받은 changeRecord를 기록해 두는 데이타베이스로서, 다른 노드에게 changeRecord들을 전송할 때 사용된다.

2.2. changeRecord의 전송

다음으로 UDDI 노드는 저널에 저장된 changeRecord들을 일정한 주기마다 다른 노드에게 전송한다. 이 때 특별한 정보가 없다면, 노드는 자신이 저장하고 있는 모든 정보를 보낼 것이다.

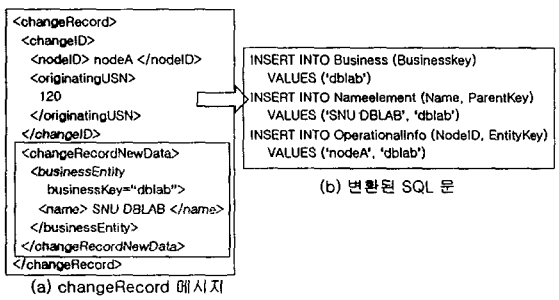
각 노드는 꼭 필요한 changeRecord들만을 전송하기 위해, 다른 노드의 몇 번째 업데이트까지를 성공적으로 처리했는지에 대한 최고점 정보를 유지한다. 각 노드는 최고점 정보를 주고 받은 다음, 아직 받지 못한 changeRecord들만을 요청한다.

2.3. changeRecord의 처리

changeRecord들을 넘겨 받은 UDDI 노드는 다음의 두 단계의 과정으로 changeRecord들을 처리한다.

1) 노드는 changeRecord가 도착하면 이를 나중에 다른 노드들에게 전송되어야 하기 때문에 이를 저널에 기록한다.

2) 노드는 changeRecord에 명시되어 있는 내용에 맞게 자신의 데이타베이스를 갱신한다.



(a) changeRecord 메시지

(b) 변환된 SQL 문

그림 3. changeRecord의 처리

그림 3은 changeRecord를 받은 노드에서 changeRecord의 내용에 따라 업데이트를 반영하는 과정을 보여주고 있는데, (a)는 businessEntity 삽입에 대한 changeRecord를, (b)는 삽입 연산에 대응되는 SQL문들을 각각 나타낸다. 이 예에서 확인할 수 있듯이, changeRecord에는 업데이트에 관한 모든 정보가 있기 때문에 이것만으로도 업데이트 연산을 충분히 수행할 수 있다.

이처럼 한 노드의 변화가 다른 노드에 반영되는 과정은 changeRecord의 생성, 전송, 처리 과정으로 설명될 수 있다.

3. 복제 서브시스템의 구현

이 장에서는 복제 과정을 실제로 구현할 때 고려해야 할 에러 처리, 저널 관리, 키 생성에 대해서 다루도록 하겠다.

3.1. 에러 처리

복제 과정에서는 기본적인 복제 연산 외에, 에러를 감지하고 유형에 따라 적절히 복구하는 절차가 반드시 필요하다. UDDI 노드에서 발생할 수 있는 에러의 종류에는 네트워크 실패, 노드 내부의 에러, 참조 관계에 의한 에러 등이 있다.

네트워크 실패의 경우 changeRecord들을 받지 못하게 된 노드는 자신의 최고점 벡터를 더 이상 변경하지 않기 때문에, 복구된 이후에 다른 노드들과 같은 내용을 가질 수 있게 된다. 따라서 이러한 에러에 대한 특별한 절차는 필요하지 않다.

다음으로, 잘못된 데이타의 삽입이나 존재하지 않는 데이타의 삭제 등 노드 내부의 에러를 생각해 볼 수 있다. 그런데 실패한 연산에 대해서는 changeRecord가 생성되지 않으므로 이러한 에러가 복제를 통해 전파되지 않는다.

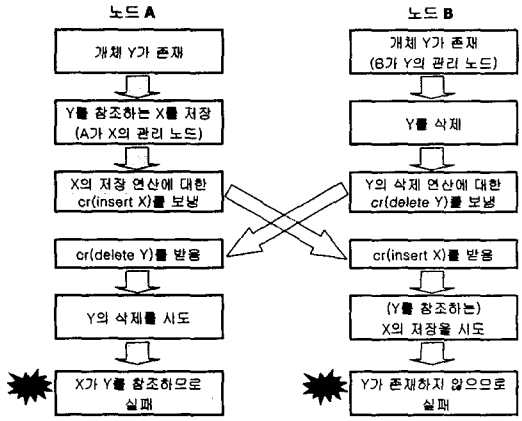


그림 4. 데이타의 참조 관계로 인해 발생하는 에러

또 다른 종류의 에러가 참조 관계에 있는 두 개체에 의해 발생하는 에러이다. 개체 X가 개체 Y를 참조하는 상황에서, X와 Y는 서로 다른 노드를 관리 노드로 가진다고 가정하겠다. 이 때 그림 4와 같이, 한 노드에서는 X가 삽입되고, 동시에 다른 노드에서는 Y가 삭제된다고 하면 복제 과정에서 에러가 발생한다.

그림 4에서 노드 A에는 X와 Y가 둘 다 남아 있지만, B에는 X와 Y가 둘 다 존재하지 않는 불일치(inconsistency) 상황이 발생한다. 여기서 주목할 것은, 두 노드 모두 각자의 연산

을 제대로 완료했음에도 불구하고 참조 관계에 있는 데이타의 특성상 복제 과정에서 에러가 발생했다는 사실이다.

위의 예에서는 Y의 참조에 관한 모순된 두 연산이 서로 다른 노드에서 동시에 일어난다. 즉, 한 노드에서는 Y를 삭제하고, 동시에 다른 노드에서는 Y를 참조하는 새로운 개체를 저장하고 있는 것이다. 만약 한 연산이 완전히 처리된 다음에 다른 연산이 일어났다면 결과는 달라질 것이다.

우선, 삽입 후에 삭제 연산이 일어나는 경우를 생각할 수 있다. 노드 A에서 X를 저장한 다음, 복제의 결과로 노드 B에도 X가 성공적으로 저장될 것이다. 그런 다음 B에서 Y를 삭제하려고 시도할 때, Y는 X에 의해 참조되고 있으므로 삭제 연산이 아예 일어나지 않을 것이다. 반대로 삭제 연산 다음에 삽입 연산이 일어나는 경우도 고려해야 한다. 노드 B에서 Y를 삭제하고, 이러한 삭제 연산이 노드 A에도 복제되어 Y가 A에서도 삭제될 것이다. 이 상황에서 A가 X를 등록하려 했다면, 이미 존재하지 않는 Y를 참조하려 하는 것이 되기 때문에 X를 등록하지 못했을 것이다.

둘 중 어느 경우라도 각 연산을 처음 시작하는 에러가 발생하는 순간, 그 연산을 취소한다. 그런데 실패한 연산에 대해서는 changeRecord가 생성되지 않기 때문에 에러가 다른 노드에게 전파되지 않는다. 따라서 모순되는 두 연산의 경우, 한 연산이 완료된 다음에 다른 연산을 진행할 수 있도록 강제한다면 참조 관계 때문에 생기는 에러를 방지할 수 있을 것이다.

3.2. 저널 관리

두 번째로 언급할 고려 사항은 저널을 위한 저장 공간을 최적화하는 방법에 관한 것이다. 한 노드에서 업데이트가 발생하면 changeRecord가 생성된 다음 저널에 저장된다. 그런데 만약 changeRecord가 생성될 때마다 저널에 쌓이기만 한다면, 결국 저널을 위한 저장 공간이 부족한 상황에 다다르게 될 것이다.

따라서 어떤 changeRecord들이 저장될 필요가 없는지 판단한 다음, 이들 중 일부를 저널에서 제거해야 한다. 그런데, 이미 성공적으로 처리된 changeRecord들이라고 해서 모두 저널에서 지울 수는 없다. 레지스트리에 새로 노드가 추가될 때, 현재 레지스트리의 내용을 제공해야 하기 때문이다. 따라서 저널에 한 개체에 대한 changeRecord들이 여러 개 있으면, 최종 버전에 필요한 것만 남기고 제거하면 된다.

changeRecord 처리가 성공했는지 여부를 판단하기 위해서 changeRecord의 ackRequested 속성을 이용한다. 한 노드가 ackRequested 속성이 'true'인 changeRecord를 받으면, 자신이 이 changeRecord와 그 이전의 모든 changeRecord들을 제대로 처리했음을 알리기 위해 changeRecord를 생성한 노드에게 ack으로 응답해야 한다.

그림 5는 노드 A에서 ackRequested 속성이 'true'인 changeRecord cr을 생성하고, cr이 B, C, D에 차례로 전파되는 예를 보여주고 있다.

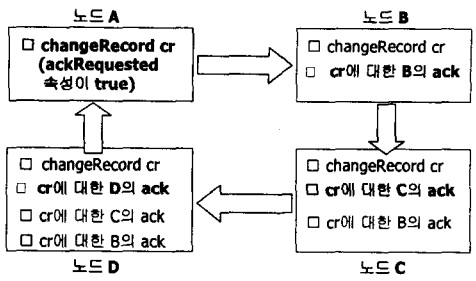


그림 5. ackRequested 속성과 ack

그림 5에서 노드 A가 자신을 제외한 모든 노드의 ack을 받았다는 것은 cr 이전의 모든 changeRecord가 각 노드에서 제대로 처리되었음을 의미한다. 따라서 노드 A는 cr 이전의 changeRecord들 중 현재 버전을 반영하고 있는 것들을 제외한 나머지를 저널에서 삭제할 수 있다.

3.3. 키 생성

마지막으로 고려해야 할 사항은 UDDI 노드에 저장될 개체의 키를 부여하는 문제이다. UDDI에서 키는 각 개체를 유일하게 식별하기 위해 이용되는 값이다. 그런데, 복제 개념이 도입되면 한 노드에서 유일한 키값이 반드시 레지스트리 전체에서 유일하다는 보장이 없게 되므로, 같은 키를 동시에 부여하는 문제가 생길 수도 있다. 따라서 키의 공간을 겹치지 않는 계층적인 영역으로 분할한 다음, 각 영역에 대한 권한을 노드 또는 사용자에게 지정해 줄 수 있다.

예를 들어 'uddi:lily.snu.ac.kr:keyGenerator'라는 tModel을 성공적으로 등록한 사용자는 'uddi:lily.snu.ac.kr'이라는 접두사(prefix)를 갖는 키를 부여할 수 있는 권한을 가지게 된다. 또한 자신이 관리하는 영역의 권한을 다른 사용자에게 양도할 수도 있으며, 그 영역의 하위 영역에 대한 권한을 위임할 수도 있다.

4. 결론

본 논문에서는 changeRecord를 통해 한 노드에서 일어난 변화가 다른 노드에 반영됨으로써 레지스트리에서 복제가 일어나는 과정에 대하여 기술하였다. 또한 실제 복제 서브시스템을 구현하기 위해 필요한 에러 처리, 저널 관리, 키 생성에 대해 언급하였다. 특히, 한 노드에서 성공적으로 수행했더라도 레지스트리 단계에서 실패할 가능성이 있는 연산들에 대한 부가적인 작업들을 제안하였다.

참고문헌

[1] UDDI 3.0 Specification, http://uddi.org/pubs/uddi-v3.00-published-20020719.pdf, uddi.org, 2002

[2] 유수진, 박송희, 김영선, 이경하, 이규철, "관계 DBMS에 기반한 UDDI 3.0 레지스트리 서버의 개발", 한국정보과학회 추계 학술 발표논문집 2003

[3] Andrew S. Tanenbaum, Maarten van Steen, "Distributed System", Prentice Hall, 2002

[4] Heather Kreger, "Web Service Conceptual Architecture (WSCA 1.0)", IBM Software Group, 2002