XML 반복부 데이터의 변경 협상 방법

이 은 정†

요 약

모바일 환경에서 여러 사용자가 XML 트리를 공유하는 응용이 점차 많아지고 있다. XML 트리를 낙관적인 중복 방식으로 모바일 환경에서 공유하려면 동시에 수정된 데이터를 협상하여 하나의 버전을 만들 수 있어야 한다. 특히 XML 트리의 부분 트리를 삽입/삭제한 구조 변경 행위들을 협상하기 위해서는 트리의 노드 매핑과 비교 과정이 필요한데, 이것은 $O(n^2)$의 복잡도를 가지는 문제로 알려져 있다. 또한 응용에 따라 달라지는 의미 기반의 충돌 해결 방식이 도입되어야 할 필요성이 제기되고 있다. 본 연구에서는 모바일 환경에서 XML 트리의 편집 기록(edit script)을 협상하기 위한 효율적인 협상 방법을 제시하였다. 본 논문에서는 이러한 요구를 만족시킬 수 있는 XML 데이터 공유 모델로서 문서 타입에서 반복부에 해당하는 부분만 삽입과 삭제를 허용하고 반복부의 부분 트리들이 형제들과 구분되는 키 값을 가지는 리스트 공유 모델을 이용한다. 이러한 리스트 공유 모델은 구조 변경 행위가 항상 문서의 유효성을 보장할 수 있다는 장점을 가지며, 키 기반의 리스트 협상을 하게 되므로 효율적인 협상 알고리즘을 얻는 것이 가능하다. 본 논문에서는 리스트 공유 모델에서 서로 키의 충돌이 없는 편집 기록에 대해 편집 기록의 길이를 m이라 할 때 $O(m)$ 복잡도를 가지는 협상 알고리즘을 제안하였다. 기존 트리 협상 방법이 키의 충돌이 없는 경우 트리 크기에 선형 비례하는 시간 복잡도를 가지므로 제안된 방법은 모바일 환경에서 효과적인 협상 방법으로 사용될 수 있을 것이다.

Change Reconciliation on XML Repetitive Data

Eunjung Lee†

ABSTRACT

Sharing XML trees on mobile devices has become more and more popular. Optimistic replication of XML trees for mobile devices raises the need for reconciliation of concurrently modified data. Especially for reconciling the modified tree structures, we have to compare trees by node mapping which takes $O(n^2)$ time. Also, using semantic based conflict resolving policy is often discussed in the literature. In this research, we focused on an efficient reconciliation method for mobile environments, using edit scripts of XML data sent from each device. To get a simple model for mobile devices, we use the XML list data sharing model, which allows inserting/deleting subtrees only for the repetitive parts of the tree, based on the document type. Also, we use keys for repetitive part subtrees ; keys are unique between nodes with a same parent. This model not only guarantees that the edit action always results a valid tree but also allows a linear time reconciliation algorithm due to key based list reconciliation. The algorithm proposed in this paper takes linear time to the length of edit scripts, if we can assume that there is no insertion key conflict. Since the previous methods take a linear time to the size of the tree, the proposed method is expected to provide a more efficient reconciliation model in the mobile environment.

키워드 : 협상(Reconciliation), XML, DTD, 낙관적 중복(Optimistic Replication), 모바일 데이터 공유(Mobile Data Sharing)

1. 서 론

무선 컴퓨팅 장치가 광범위하게 보급되면서 단말 간에 XML 데이터를 공유하거나 또는 무선 데이터베이스를 사용하는 응용이 많아지고 있다. 모바일 환경에서의 낮은 대역폭과 잦은 접속 단절로 인해 모바일 단말 간에 데이터를 공유하기 위하여 낙관적인 중복 방식이 많이 도입되고 있다. 낙관적인 중복 방식을 도입하면 공유 데이터가 비연결 상태에서도 수정될 수 있으며, 여러 개의 중복 데이터(replica)가 독립적으로 수정된 후 통합되어야 할 필요가 생긴다. 여러 버전의 중복 데이터를 하나의 통합 데이터로 만드는 과정을 흔히 데이터 협상(reconciliation)이라고 한다.

무선 데이터베이스에서 비연결 상태의 수정을 협상하는 문제가 많이 연구되었다. Phatak 등은 트랜잭션 로그를 이용한 중복 협상 방법을 제시하였고[11], Kermarrec 등은 IceCube 플랫폼에서 의미적인 충돌 해결 규칙이나 정책을 사용자가 지정할 수 있게 하여 수정 로그를 협상하는 방법을 소개하였다[8]. 한편 Amza 등은 다중 중복을 허용하기 위한 분산 버전 관리 방식을 제안하였다[3].

낙관적 중복을 위해 XML 문서는 관계형 데이터와 달리 수정된 트리 구조를 통합하기 위해 새로운 문제들이 해결되어야 한다. 우선 두 트리를 비교하여 달라진 부분을 찾아내야 하며, 데이터 값이 다르거나 구조가 달라진 경우 이를 협

※ 한국과학재단의 우수여성과학자 지원연구(R04-2002-000-20153-0)의 지원을 받았음.
† 정 회 원 : 경기대학교 정보과학부 교수
논문접수 : 2004년 7월 20일, 심사완료 : 2004년 10월 11일

상하여 하나의 결과 트리를 생성하여야 한다. 일반적으로 문헌에서는 트리 비교에 의해 편집 기록의 형태로 차이를 구하는 방법이 많이 연구되었고[5] XML 문서의 협상에 대한 최근의 연구 결과로는 XMIDDLE[11]과 TMA[10]가 있다.

XMIDDLE은 XML 기반의 피어투피어 환경의 모바일 데이터 공유 미들웨어인데, XML 데이터의 동기화와 협상 기능을 제공한다. 협상은 버전 기반으로 이루어지며, 새로운 문서와 원래 버전의 문서를 비교하여 찾아진 차이를 가지고 다른 새 버전의 문서와 협상하게 된다. 이 때 차이는 집합의 형태로 표현되며, 충돌이 일어난 경우는 어플리케이션에 의해 지정된 협상 규칙을 이용하여 협상을 하게 된다.

한편 Lindholm 등이 제안한 TMA(Three-way merging algorithm)은 원래 문서에 대해 동시 수정된 두개의 XML 문서를 이용하여 three-way로 통합하는 방법을 제시하였다. 이들은 XML 문서의 차이를 구하는 단계와 협상 단계로 나누어 three-way 통합을 위해 노드 매핑 및 차이를 계산하고, 그 결과를 통해 협상을 하는 $O(n^2)$ 알고리즘을 제시하였다. XML 요소들이 모두 ID를 가진 경우 $O(n)$ 계산이 가능하다. 이들의 방법은 XML 문서에 대한 제약 없이 일반적인 TMA가 가능하고 더구나 부분 트리의 이동과 같은 기존의 다른 협상 방법에서는 지원하지 못하던 편집 행위를 허용한다. 그리고 어플리케이션에서 적절한 협상 규칙(policy)를 제공하게 하여 TMA를 다른 시스템에 포함시키기가 쉽다고 한다.

한편 DeltaXML[7]은 상용화된 XML 문서 공유 도구로 본 논문의 접근과 유사한 방법으로 키와 부분집합 방식에 의한 통합(merge)를 수행한다. 실제 XML 문서 간의 동기화나 three-way 통합을 위해 이 도구를 이용할 수 있으며, 충돌이 있는 경우 두 문서의 차이를 보여주어 사용자가 개입된 통합을 지원한다. 이들은 또한 본 연구에서와 같이 편집 기록(edit script)에 의한 통합 방법을 이용한다.

협상에 관한 기존 연구에서는 노드의 매핑을 통해 데이터 변경과 트리 구조 변경 행위를 같이 취급하고 있으나 데이터 변경의 협상은 모바일 데이터베이스에서 이미 다루어졌던 문제로 어플리케이션에 따라 최소값, 최대값, 또는 합을 취하거나 사용자 기반의 우선순위를 협상 규칙으로 지정하는 등의 방법이 많이 사용되고 있다. 그러나 구조 변경 행위인 부분 트리의 삽입이나 삭제를 협상에서 어떻게 다루어야 할 것인가에 대해서는 아직 효과적인 해결책이 없는 상황이다. 더구나 트리 구조 차이를 계산하는 것이 $O(n^2)$을 요구하는 노드 매핑을 포함하고 있는데, 이것은 트리 전체를 다른 트리의 모든 부분과 비교하는 것으로 상당한 메모리와 계산 양을 요구한다.

본 논문에서는 모바일 환경의 특성을 고려하여 트리 비교를 하는 대신 단말에서 데이터를 수정한 편집 기록을 이용하여 다른 버전의 XML 트리 간의 협상을 하는 방법에 대해 살펴본다. 본 논문에서는 데이터 값의 변경은 기존의 방법을 그대로 사용하고 XML 트리의 구조적인 변경 만을 협상의 대상으로 한다. 구조 변경의 협상을 위한 효과적인 알고리즘을 찾기 위하여 XML 데이터에서 문서 타입 정보에 의해 반복부에 대해서만 삽입/삭제가 허용되는 리스트 데이터 공유 모델을 도입하고자 한다. 그 결과 트리의 협상 문제는 리스트 부분의 협상으로 바뀔 수 있다. 반복부 리스트의 협상은 키의 매핑을 기반으로 하므로 편집 기록의 협상은 트리 크기에 대해 선형 시간에 가능하다. 본 모델에서 사용하는 키는 트리 전체에서 유일한 ID 값과 달리 반복부의 형제들 간에 유일한 값으로 관계형 데이터베이스 테이블의 키와 유사한 역할을 한다. 만약 수정된 버전들 사이에 동일한 키로 삽입된 부분 트리의 충돌이 없다면 협상은 편집 기록의 길이에 대해 선형시간에 가능하다.

본 논문의 구성은 다음과 같다. 2장에서는 리스트 기반의 XML 데이터 공유 모델을 소개하고, 3장에서는 이 모델에 기반한 편집 기록을 정의하고 몇 가지 성질을 소개한다. 4장에서는 편집 기록의 협상 알고리즘을 소개하고 기존 연구와 비교한다. 5장에서 결론을 맺는다.

2. 리스트 기반의 XML 데이터 공유 모델

여기서는 XML 데이터 공유 모델로서 XML 문서의 반복부에 대한 구조 변경만 허용하는 리스트 기반의 공유 모델을 소개한다. 본 연구팀은 한번 이상의 삽입 삭제가 반복적으로 적용 가능한 대상이 DTD 상의 * 반복부 부분임에 착안하여 삽입 삭제 행위의 정형화된 표현을 위한 모델로 반복부를 다른 부분과 구별하여 독립적인 구조를 가지게 하는 XML 데이터 모델을 제안하여 동기화 프로토콜 문제의 해결에 도입하였다[1]. 본 논문에서는 동일한 모델을 도입하여 편집 기록의 협상 문제에 적용하고자 한다. 본 절에서는 먼저 데이터 공유 시나리오를 통해 리스트 공유 모델을 소개하고 이 모델을 편집 기록의 협상 문제에 이용하기 위하여 *-factoring 방법 및 성질을 간단히 소개한다.

2.1 리스트 기반 XML 공유 시나리오

이 절에서는 반복부 데이터에 대한 구조 변경 행위를 공유하는 응용 사례를 통해 이 논문에서 사용하는 모델을 소개한다. 아래 예에서는 배송 내역과 처리 기록을 가지는 XML 문서의 사례를 살펴본다.

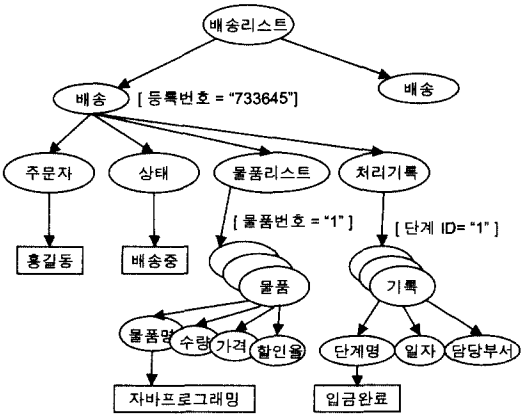
```
<배송리스트> := (<배송>)*
<배송> := <등록번호><주문자><상태><주문일><결재액>
          <주문내역><처리기록>
<주문내역> := (<물품>)*
<처리기록> := (<기록>)*
<물품> := <물품명><수량><가격><할인율>
<기록> := <단계명><일자><담당부서>
   이하 생략
```

(리스트 2-1) 배송리스트 문서의 DTD

```
<배송리스트>
    <배송 등록번호 = "733654">
        <주문자>홍길동</주문자>
        <상태>배송중</상태>
        <주문일>2001-08-01</주문일>
        <결재액>43600</결재액>
        <주문내역>
                <물품 물품번호 = "1">
                  <물품명>자바 프로그래밍</상품명>
                    <수량>1</수량>
                    <가격>25600</가격>
                    <할인율>20</할인율>
                </물품>
                <물품 물품번호 = "2">
                    <물품명>JSP 따라하기</상품명>
                    <수량>1</수량>
                    <가격>18000</가격>
                    <할인율>10</할인율>
                </물품>
        </주문내역>
        <처리기록>
                <기록 단계ID = "1">
                    <단계명>입금완료</단계명>
                    <일자>2001-08-01</일자>
                  <담당부서>온라인관리팀</담당부서>
                </기록>
                <기록 단계ID = "2">
                    <단계명>발송</단계명>
                    <일자>2001-08-02</일자>
                  <담당부서>본부물류팀</담당부서>
                </기록>
        </처리기록>
</배송>
<배송>

                이하 생략
```

(리스트 2-2) 배송리스트 문서 예



(그림 1) 배송리스트 문서의 XML 트리

2.2 XML 데이터 모델 - *-factored XML

XML 트리에서 각 노드가 가지는 자식노드의 형식은 DTD 규칙에 의해 정의되는데, DTD 규칙에서 자식 노드들은 필수, 생략 가능부, 선택부, 그리고 반복부로 나눌 수 있다. 본 논문에서는 반복부가 XML 데이터 공유에서 특히 구조 변경 행위의 대상으로 중요한 역할을 함에 착안하여, 다음 절의 편집 기록 협상을 위해 반복부 부분을 별도의 생성 규칙으로 독립시키는 형태의 DTD 변환을 소개한다. 이 모델은 본 연구팀이 공유 데이터의 동기화를 위한 잠금 프로토콜을 제안하기 위하여 도입되었다[1].

본 논문에서는 DTD 규칙을 표현하기 위하여 사용되는 심볼의 집합을 V, 심볼들을 이용한 정규 표현을 RE(V)라고 표현한다. 그리고 심볼을 나타내는 기호로 $A, B, C, \cdots \in V$를 사용하고 규칙의 우변에 나타나는 정규표현을 나타내는 기호로 $\alpha, \beta, \gamma, \cdots \in$, RE(V)를 사용한다. 정규표현이 * 를 포함하지 않은 경우 *-free RE(V)라고 한다. 예를 들어 α, β, γ가 *-free 정규표현이라면 $(\alpha \mid \beta)\gamma$는 *-free 정규표현이다.

DTD에서 반복되는 부분이 독립된 부분 트리로 묶여지고 반복부를 묶어주는 부모 노드가 다른 노드들과 분리되어 있는 DTD를 *-factored DTD라고 정의한다.

[정의 2.1] *주어진 DTD 규칙 $A ::= \delta$가 아래와 같은 성질을 만족하면 그 규칙을 *-factored라고 한다.*

(i) δ *가 *-free RE(V)이거나*

(ii) $\delta = \alpha(\beta)^*\gamma$ *의 형태를 가질 때* $\alpha = \varepsilon, \gamma = \varepsilon, \beta \in V$*를 만족한다. 여기서* $A, B, C, \cdots \in V$,

$$\alpha, \beta, \gamma, \cdots, \in RE(V).$$

DTD의 모든 규칙이 *-factored라면 반복부 factored DTD라고 한다.

*-factored DTD에서는 생성 규칙의 우측에 반복부가 나타나는 경우 $A ::= B^*$ 와 같은 형식으로 단일 심볼의 반복부만 존재한다. 위 (그림 1)의 예에서 <배송리스트> ::= <배송> * 이고 <주문내역> ::= <물품>*, <처리 기록> ::= <처리> * 와 같이 반복부는 모두 독립 생성 규칙을 가지고 생성 규칙의 우측이 단일 심볼의 반복으로 표시된다. 아래 규칙은 *-factored되어 있지 않은 예이다.

```
<경매리스트> ::== <분류> (<경매>) * (<취소사유><경매> * ) *
                  (<종료일><경매> * )) *
```

이 규칙에 대응하는 *-factored 된 DTD 규칙은 다음과 같다.

```
<경매리스트> ::== <분류> <경매리스트> <취소경매리스트>
                  <종료경매리스트>
<경매리스트> ::= <경매> *
<취소경매리스트> ::= <취소경매> *
<취소경매> ::= <취소사유> <경매리스트>
<종료경매리스트> : := <종료경매> *
<종료경매> : := <종료일> <경매리스트>
```

위의 예에서 살펴보았듯이 *-factored 되어 있지 않은 DTD 규칙을 새로운 심볼을 추가하여 *-factored로 변환하는 것이 가능하다. 이 변환 알고리즘은 [1]에 소개되어 있다. *-factored인 DTD에 대해 유효한 XML 문서에서 반복부 생성규칙의 좌변에 해당하는 노드를 리스트 부모 노드라 하고 반복부 자식 노드들은 리스트 노드라고 한다. 그러면 리스트 부모 노드의 자식들은 모두 동일한 이름을 가지며, 리스트 부모 노드에 새로운 자식 트리를 추가하거나 삭제하는 것은 문서의 유효성을 해치지 않는 유효한 행위가 된다.

반복부 자식 트리들 간의 구별을 위하여 본 논문에서는 형제 간에 구분 가능한 키를 가진다고 가정한다. 이하의 논

의에서는 XML 문서가 *-factored이면서 리스트 노드들이 형제 간에 유일한 키 값을 가지는 문서라고 가정한다. 형제 간에 구분 가능한 키 값은 ID 속성과는 달리 트리 전체에서 유일해야 하는 것이 아니라 동일한 부모 노드를 가지는 리스트 노드들 간에서 구별될 수 있으면 된다.

노드 v에 대해 v의 경로는 path(v), 키는 key(v), 이름은 name(v)로 표시하며, v의 이름이 A이고 키가 k라면 경로에서 A[k] 형태로 표시할 수 있다. 예를 들어 위의 그림에서 첫 번째 배송의 물품리스트 중 세 번째 "자바프로그래밍"이라는 물품명 노드의 경로는 "배송리스트/배송[등록번호="733654"]/주문내역/물품[물품번호="3"]/물품명"으로 나타낼 수 있다.

2.3 유효한 반복부 편집 행위

XML 트리 T의 수정을 표현하는 방법으로 본 논문에서는 편집 행위를 이용한다. 편집 행위는 주로 XML 문서 변경을 검출하는 기존의 연구에서 많이 사용되었다.

XML 트리 T에 대해 편집 행위 σ는 T에 σ를 적용한 결과 트리 T'에 의해 다음과 같이 정의된다.

i. Delete(π) : $\pi = \pi_0 A[k]$라 하면, 트리 T에서 경로 π_0에 의해 표시되는 노드의 자식 중 키가 k인 부분 트리를 삭제한 트리가 결과 트리 T'이 된다.

ii. Insert(π, T_{new}) : 트리 T에서 경로 π에 의해 표시되는 노드의 자식으로 T_{new}를 삽입한다.

iii. Update(π, val_{new}) : 트리 T에서 경로 π에 의해 표시되는 터미널 노드의 데이터 값을 val_{new}로 바꾼다.

$\sigma(T)$는 T에 σ를 적용한 결과 트리를 표시한다. XML 트리에서 임의의 부분 트리를 삭제하거나 삽입하는 것은 문서의 유효성을 해칠 수 있다. 본 저자는 부분 트리의 삽입과 삭제를 DTD에 의한 문서의 반복부에 한정하면 문서의 유효성을 보장할 수 있음을 증명하였다[1]. 부분 트리의 삽입과 삭제 행위를 반복부 리스트 노드에 한정하여 리스트 구조 변경 행위를 아래와 같이 정의할 수 있다.

[정의 2.2] *XML 트리 T에 대해 리스트 구조 변경 행위 s는 T에 s를 적용한 결과 트리 T'에 의해 다음과 같이 정의된다.*

(i) *ListInsert(p, Tnew) : Insert 행위 중에서 경로 p가 표시하는 노드가 리스트 부모 노드이고 T_{new}의 키가 다른 형제 노드와 구분되는 유일한 값일 때만 허용된다.*

(ii) *ListDelete(p) : Delete 행위 중에서 경로 p가 표시하는 노드가 리스트 노드일 때만 허용되는 행위이다.*

[정리 2.1] *리스트 구조 변경 행위는 항상 유효한 결과 트리를 생성한다.*

이 정리는 [1]에서 증명되었으며 이를 이용하면 리스트 구조 변경 행위와 데이터 수정 행위의 연속에 의해 얻어진 결과 트리는 항상 유효하다는 것을 쉽게 증명할 수 있다.

3. 리스트 기반 XML 공유를 위한 편집 기록

이 장에서는 협상의 대상의 되는 편집 기록에 대해 살펴본다. 본 논문에서는 XML 문서의 협상을 위해 XML 트리를 이용하지 않고 수정 로그인 편집 기록을 이용하므로 협상을 시작하기 전에 편집 기록의 중복이나 행위간 포함 관계를 제거하여 협상알고리즘의 논의를 간단하게 하고자 한다. 먼저 편집 행위의 포함 관계와 순서 관계에 대한 성질을 살펴보고 XML 트리의 구조 적합 관계를 정의한 후 편집 기록을 다듬어 협상할 수 있는 상태로 변환하는 알고리즘을 소개한다.

3.1 편집 행위의 성질

편집 행위는 위에서 정의된 리스트 변경 행위인 Update, ListDelete 또는 ListInsert이다.

삽입, 삭제, 수정 행위로 구성되는 리스트 구조 변경 행위는 한 트리를 다른 트리로 변환할 수 있는 연산이다. 계층적 구조를 가지는 트리에 대한 편집 행위는 서로 포함 관계를 가질 수 있는데, 행위의 포함관계(Containment)는 질의 최적화 분야에서 많이 연구되었으며, 트리 구조의 데이터에 대한 질의 최적화 방법도 많이 연구되었다[6]. 본 연구에서는 경로 기반의 포함 관계를 아래와 같이 정의한다.

[정의 3.1] *두개의 편집 행위 σ_1과 σ_2에 대해 path(σ_1)이 path(σ_2)의 자손 노드인 경우 σ_1이 σ_2를 포함된다고 하며 $\sigma_1 \ll \sigma_2$로 표시한다.*

포함관계를 가지는 경우 두 개 편집 행위는 적용 결과가 순서에 따라 달라지게 되며, 하나의 편집 행위가 다른 하나의 적용 효과를 포함하기도 한다. 예를 들어 터미널 노드의 값을 수정한 후 그 노드를 포함한 부분 트리가 삭제되어 버린다면 값의 수정 행위는 적용되지 않는 것과 같다. 또 부분 트리를 삽입한 후 그 부분 트리를 포함하는 조상 노드의 부분 트리가 삭제되었다면 삽입 행위는 무효화된다. 두 편집 행위가 서로 포함관계가 없는 경우 $\sigma_1 \sim \sigma_2$라고 표시한다.

이하에서 편집 기록의 협상을 다루기 위해 여기서는 포함 관계를 가지는 편집 행위의 제거가 가능함을 살펴본다.

[보조정리 3.1] *두 개의 편집 행위 σ_1과 σ_2에 대해 $\sigma_1 \ll \sigma_2$라면 $\sigma_1'(T) = \sigma_1\sigma_2(T)$이면서 σ_1과 동일한 경로를 가지는 σ_1'이 반드시 존재한다.*

[증명] $\sigma_1 = ListInsert(\pi_1, T_{new1}, k_1)$이고 $\sigma_2 = ListInsert(\pi_2, T_{new2}, k_2)$라 하자.

이 때 $\sigma_1 \ll \sigma_2$이면 π_2는 σ_1에 의해 추가된 부분 트리의 자손 중에 하나인 리스트 노드를 가리키게 되고 π_1는 π_2의 머리부(prefix)가 된다. π_2/π_1을 π_2에서 π_1의 경로 만큼을 떼어낸 꼬리부라고 하면 T_{new1}에 경로 π_2/π_1인 노드가 존재한다. ListInsert(π_2/π_1, T_{new2}, k_2)를 T_{new1}에 적용한 결과 트리를 T_{new1}'이라 하면 편집 행위 $\sigma_1' = (\pi_1, T_{new1}', k_1)$은

위의 조건을 만족한다.

σ_2가 ListDelete이거나 Update 행위인 경우에 대해서도 동일한 방식으로 σ_1의 삽입 트리에 σ_2를 미리 적용하여 하나의 편집 행위 σ_1'을 만들 수 있다.

한편 σ_1이 ListDelete라면 σ_2는 삭제된 트리의 자손을 접근한 것이므로 σ_1이후에 σ_2를 적용하는 것은 가능하지 않다. □

위와 같은 성질을 만족하는 두 편집 행위의 결합 $\sigma_1' = \sigma_1\sigma_2$라고 표시한다.

[보조정리 3.2] *두 개의 편집행위 σ_1과 σ_2에 대해 $\sigma_1 \sim \sigma_2$라면 $\sigma_1\sigma_2(T) = \sigma_2\sigma_1(T)$를 만족한다.*

[증명] 두 편집 행위의 적용 대상은 서로 조상이나 자손 관계를 갖지 않는 부분 트리이다. 그러므로 적용 순서를 바꾸어도 결과 트리는 동일하다. □

3.2 트리의 구조 적합 관계

다음으로 트리의 협상에 필요한 트리의 구조 적합 관계를 정의한다. 본 논문에서 사용하는 공유 모델이 반복부에 대한 구조 변경 행위만 허용하므로 사용자가 임의로 삽입한 트리가 충돌을 일으킨 경우 구조 적합 관계를 만족하여야만 협상이 가능하게 된다.

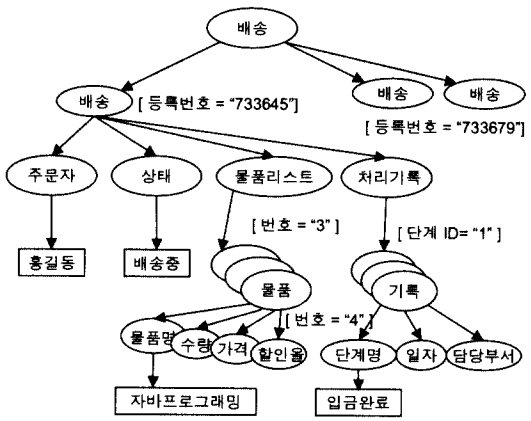
[정의 3.2] *두 개의 XML 트리 T_1, T_2에 대해 다음의 조건을 만족하면 트리 간의 구조 적합 관계 $T_1 \approx T_2$를 만족한다고 한다.*

1. $label(root(T_1)) = label(root(T_2))$
2. *T_1의 자식 트리는 $T_{11}, T_{12}, \cdots, T_{1k}$이고 T_2의 자식 트리는 $T_{21}, T_{22}, \cdots, T_{2m}$이라고 하자.*
   - 2.1 *$root(T_1)$이 둘다 리스트 부모 노드가 아니라면 $k=m$이고 $T_{1i} \approx T_{2i}, 1 \le i \le k$를 만족한다.*
   - 2.2 *$root(T_1)$과 $root(T_2)$가 둘다 리스트 부모 노드라면 $T_{1i} \approx T_{2j}$, if $\exists 1 \le j \le m, key(root(T_{1i})) = key(root(T_{2j}))$를 만족한다.*

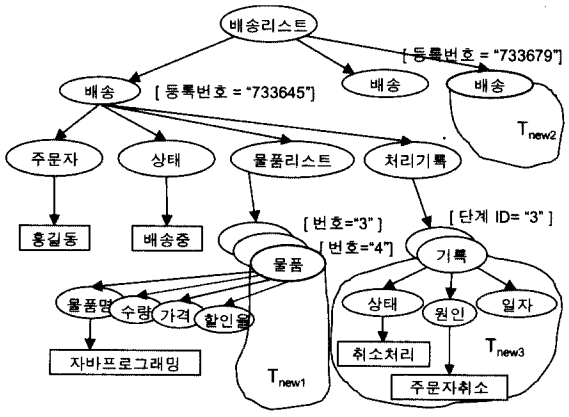
즉 구조 적합 관계가 만족하려면 반복부가 아닌 부분은 동일한 구조를 가지고 반복부에서 키가 같은 부분 트리끼리는 역시 구조 적합 관계를 만족하여야 한다.

주어진 트리 T_1에 대한 변환이 T_1과 구조 적합 관계를 만족하는 결과 트리 T_2를 생성한다면 이 편집 기록에 의한 변환은 구조 적합 관계를 만족한다고 한다. 예를 들어 반복부 배송 정보의 추가나 물품리스트의 추가는 구조 적합 관계를 만족하는 변환이다. 그러나 선택부나 생략부의 부분 트리를 교체하거나 삭제/삽입하는 경우는 구조 적합 관계를 만족하지 못한다. 또한 동일한 키에 대해 트리를 삭제 후 삽입하였다면 새 부분트리가 원래 있던 것과 구조적합 관계를 만족해야 한다. 아래 그림은 (그림 1)의 배송리스트 문서에 대한 구조 적합 변환과 그렇지 않은 변환의 예를 보여준다.

(그림 2)(a)는 ListDelete("배송리스트/배송[733654]/물품리스트/물품[번호='1']"), ListInsert("배송리스트/배송[733654]/물품리스트", T_{new1}), ListInsert("배송리스트", T_{new2})의 세 가지 편집 행위를 적용한 경우이다. 여기서 $key(root(T_{new1})) = 4$이고 다른 물품의 번호와 구분된다. 또 $key(root(T_{new2}) =$ "733679"로 다른 등록번호와 구별된다. 이것은 원래의 배송리스트 문서에 대해 구조 적합 변환이다. 그러나 (그림 2)(b)는 여기에 추가로 Delete("배송리스트/배송[733654]/처리기록/기록[단계ID='3']"), Insert("배송리스트/배송[733654]/처리기록", T_{new3})를 적용하였는데, 삽입한 트리 T_{new3}이 키로 단계ID="3"을 가지므로 동일한 키에 대해 삭제 삽입한 경우이다. 그런데 원래 "입금완료" 트리와 T_{new3}이 구조 적합관계가 아니므로 이것은 구조 적합 변환이 아니다.



(a) 구조 적합 변환



(b) 구조 적합 변환이 아닌 예

(그림 2) (a) 구조 적합 변환과 (b) 구조 적합 변환이 아닌 예

이러한 성질은 아래에서 편집 기록의 협상 알고리즘을 구하기 위하여 트리의 변환에 대한 제약으로 사용된다.

한편 2장에서 소개된 유효한 행위는 반복부에 대한 추가와 삭제만 허용하므로 항상 구조 적합 변환이다. 단 삽입에서 원래 있던 부분 트리와 동일한 키의 부분트리를 삭제 후 삽입하였다면 새 트리가 원래 트리와 구조 적합 관계를 만족하지 않으면 위의 T_{new3}의 경우처럼 구조 적합 관계가 만족되지 않는다.

[보조정리 3.3] *하나 이상의 리스트 구조 변경과 Udpate 행위의 결합으로 이루어진 변환은 동일한 키의 부분트리의 삭제 후 삽입의 경우가 아니면 항상 구조 적합 변환이다.*

3.3 다듬어진(trimmed) 편집 기록

이 장에서는 편집 기록에서 중복되거나 무효화되는 편집 행위를 제거하여 협상 단계에서 편집 행위들 간의 관계를 보다 쉽게 분류할 수 있도록 하는 편집 기록의 변환에 대해 살펴본다. 편집 기록 $\Sigma=(\sigma_1,\sigma_2,\cdots,s_n)$는 편집 행위의 연속으로 표시되는 트리의 수정기록이다.

트리 T에 대해 편집 기록을 적용한 결과는 $\Sigma(T)$라고 표현하며, $\Sigma(T)=\sigma_1(\sigma_2(\cdots,\sigma_n(T)\cdots))$가 된다. 두 개의 편집 기록 Σ_1과 Σ_2에 대해서 두 편집 기록을 동일한 트리에 적용하였을 때 결과 트리가 동일하면, 즉 $\Sigma_1(T)=\Sigma_2(T)$이면 두 편집 기록이 동치라고 하고 $\Sigma_1\equiv\Sigma_2$로 표시한다.

한편 주어진 편집 기록에서 포함관계를 가지는 편집 행위가 없고 동일한 키의 부분트리를 삭제 후 삽입하는 편집 행위 쌍이 없다면 이를 다듬어졌다고 한다.

[정의 3.3] *주어진 XML 트리 T에 대한 편집 기록 $\Sigma=\sigma_1,\sigma_2,\cdots,\sigma_v$는 다음의 조건을 만족하면 다듬어졌다고 한다.*

(i) *$i<j$인 임의의 σ_i,σ_j에 대해 $\sigma_i\sim\sigma_j$를 만족하고,*

(ii) $1\le k<l\le n, \sigma_k=ListDelete(\pi_A[k])$, $\sigma_l=ListInsert(\pi_2, T_{new})$이고 $label(root(T_{new}))=A, key(root(T_{new}))=k$인 σ_k와 σ_l은 *존재하지 않는다.*

서로 포함관계를 가지는 편집 행위 쌍이 없어야 다듬어진 편집 기록이 되는데, 편집 기록의 적용 결과를 동일하게 유지하면서 주어진 편집 기록에서 포함관계인 편집 행위 쌍을 제거하는 것이 가능하다. 위의 [보조정리 3.1]의 결과에 의해 포함관계를 가지는 두개의 편집 행위를 하나로 결합하는 것이 가능함을 보였다. 한편 [정의 3.3]의 두번째 조건은 동일한 트리를 삭제 후 삽입한 경우가 없어야 함을 나타내는데, 두 트리가 구조 적합 관계를 만족하면 아래 [보조정리 3.4]에 의해 정의의 두 번째 조건을 만족하도록 부분 트리의 변경 편집 기록으로 대치할 수 있다.

[보조정리 3.4] *두 XML 트리 T_1과 T_2가 구조적합 관계라면 구조 적합 변환 행위의 결합인 편집 기록 Σ가 존재하여 $\Sigma(T_1)=T_2$를 만족한다.* 이러한 성질을 이용하여 임의의 편집 기록을 받아 다듬어진 편집 기록을 생성하는 알고리즘을 살펴본다.

입력 : 리스트 편집 기록 $\Sigma=(\sigma_1,\sigma_2,\cdots,\sigma_n)$.
출력 : 다듬어진 편집 기록 Σ. Σ가 다듬어진 편집 기록이 될 수 없는 경우는 fail을 출력함.
Σ에서 $i<j$인 모든 편집 행위 쌍 σ_i,σ_j에 대해,
① *$\sigma_j\ll\sigma_i$이면 σ_i를 Σ에서 제거한다.*
② *$\sigma_j\ll\sigma_i$이면 σ_j를 제거하고 σ_i를 $\sigma_i'=\sigma_i\sigma_j$로 대치한다.*
③ *σ_i,σ_j가 같은 키의 부분 트리 삽입/삭제 행위 쌍이라면 삭제 부분 트리를 T_1, 삽입 부분 트리를 T_2라 할 때*
A. $T_1\approx T_2$가 아니면 fail,
B. $T_1\approx T_2$이면 σ_i, σ_j를 제거하고 $\Omega(T_1)=(T_2)$인 편집 기록 Ω에 경로 π_1/π_2를 확장한 Ω'을 σ_i자리에 끼워 넣는다.

(알고리즘 3.1) 편집기록 다듬기 알고리즘

[보조정리 3.5] *위의 (알고리즘 3.1)의 결과 생성된 편집 기록 Σ'은 원래의 편집 기록 Σ와 동치이다.*

[증명] $i<j$를 만족하는 σ_i와 σ_j에 대해서

① $\sigma_j\ll\sigma_i$인 경우,

A. σ_j = ListInsert라면 조상 부분 트리가 삽입되기 전에 그 하부의 부분 트리가 삭제/삽입/수정되는 것은 가능하지 않음.

B. σ_j = ListDelete라면 σ_i의 적용 효과가 두 번째 조상 부분 트리의 삭제 행위에 의해 무효화되었으므로 σ_j를 제거해도 편집 기록은 동치임.

② $\sigma_i\ll\sigma_j$인 경우 [보조정리 3.2]에 의해 $\sigma_i'=\sigma_i\sigma_j$인 편집 행위 σ_1'이 존재하므로 그것으로 대치한 결과 편집 기록은 동치이다.

③ σ_i, σ_j가 부분 트리 교체 행위 쌍이고 교체 트리가 원래 트리와 구조 적합 관계라면 [보조정리 3.4]에 의해 정제된 변환 편집 기록 Ω가 반드시 존재한다. $\Sigma=\Sigma_1\sigma_1\Sigma_2\sigma_2\Sigma_3$라 하자. 부분 트리에 대한 변환 편집 기록 Ω에 경로 π_1/π_2를 확장한 Ω'을 Σ에 끼워넣은 $\Sigma\Omega'\Sigma_2\Sigma_3$는 Σ와 동치이면서 정제된 편집 기록이다.

그러므로 (알고리즘 3.1)에서 생성된 정제된 편집 기록이 있다면 이것은 반드시 Σ와 동치이다. □

위 (알고리즘 3.1)은 $|\Sigma|=m$이라 할 때 정렬하면서 비교할 수 있으므로 정렬 알고리즘과 동일한 복잡도 $O(mlogm)$을 가진다. 단 ③ 단계에서 부분트리 비교에 의한 편집기록 계산 부가 $O(|T_1|+|T_2|)$가 걸린다. 구조적합 트리 간의 변환 편집 기록 계산 알고리즘은 여기서는 생략한다.

[보조정리 3.6] *Σ가 다듬어진 편집기록 $\Sigma=(\sigma_1,\sigma_2,\cdots,\sigma_n)$이라면 σ_i와 σ_k를 교환한 편집기록 $\Sigma=\sigma_1\sigma_2,\cdots,\sigma_{i-1}\sigma_k\sigma_{k+1},\cdots,\sigma_n$에 대해 $\Sigma(T)=\Sigma(T)$를 만족한다.*

[증명] 위의 [보조정리 3.5]에 의해 경로 독립적인 두 편집 행위 σ_1과 σ_2는 순서를 바꾸어 적용할 수 있다. 그러므로 이를 편집 기록으로 확장하면 편집 기록 중 두 개의 편집 행위의 순서를 바꾸었을 때 두 편집 기록은 동일한 결과 트리를 만든다. 예를 들어 위의 (그림 2)(b)에서 부분트리 T_{new1}과 T_{new2}의 삽입은 순서가 바뀌어도 서로 영향을 미치지 않으므로 결과는 같다. 마찬가지로 번호="1"의 물품 삭제를 삽입과 순서를 바꾸어도 결과는 같다.

[보조정리 3.7] *Σ가 다듬어진 편집기록이면, Σ에서 편집기록의 순서를 바꾼 임의의 편집 기록 Σ'에 대해서 $\Sigma(T)=\Sigma'(T)$를 만족한다.*

(증명 생략)

위의 [보조정리 3.6]에서 정제된 편집 기록은 편집 행위의 순서를 바꾸는 것이 결과에 영향을 미치지 않음을 보였으므로 이하에서는 정제된 편집기록 Σ를 아래와 같이 편집 행위의 집합으로 $\Sigma=\{\sigma_1,\sigma_2,\cdots,\sigma_v\}$와 같이 표시한다. 순서에 관계없는 집합으로 표현하면 위 알고리즘 1번에 의해 생성된 다듬어진 편집 기록은 유일하게 존재함을 증명할 수 있다.

[정리 3.1] *임의의편집 기록 Σ에 대해 동치인 다듬어진 편집 기록이 항상 유일하게 존재한다.*

4장에서 편집 기록 협상의 입력이 되는 편집 기록은 다듬어진 편집 기록이면서 preorder 경로에 의해 정렬되어 있다고 가정한다. 또한 동일한 경로에 대해서는 삭제 행위가 먼저 나오고 삽입 행위가 나중에 나오도록 순서를 가지는 편집 기록이라고 본다.

4. 편집 기록 협상 방법

4.1 편집 행위의 충돌과 구문적 충돌 해결 규칙

일반적인 XML 트리 협상에서는 모든 충돌의 경우를 고려하여야 되지만 리스트 기반 공유 모델의 XML 수정은 가능한 경우의 수가 몇 가지로 줄어든다. 아래에서는 행위의 종류와 경로에 따른 충돌의 경우를 자세히 살펴본다. 두 편집 행위 σ_i와 ρ_j의 결합을 고려해보자.

① $\sigma_i=\text{ListDelete}(\pi_1)$, $\rho_j=\text{ListDelete}(\pi_2)$라면 이 때 두 경로는 $\pi_1\sim\pi_2$이거나 $\pi_1=\pi_2$, $\pi_1\ll\pi_2$, $\pi_2\ll\pi_1$의 세 가지 중 한 가지 관계를 가지게 된다.

A. $\pi_1=\pi_2$인 경우는 양쪽에서 동일하게 삭제된 부분이므로 충돌이 아니다. 마찬가지로 $\pi_1\sim\pi_2$인 경우도 서로 독립적이므로 충돌이 아니다.

B. 경로가 서로 포함관계인 경우 충돌을 일으킨 두 삭제 행위에 대해 공통되는 하부 부분 트리를 삭제할 수도 있고 최대 트리를 삭제할 수도 있다. 이에 대해서는 사용자의 우선순위 규칙을 따른다.

② $\sigma_i=\text{ListDelete}(\pi_1)$, $\rho_j=\text{ListInsert}(\pi_2,T_{new})$라면 $\pi_1\ll\pi_2$거나 $\pi_1\sim\pi_2$, $\pi_1=\pi_2$가 만족한다.

A. $\pi_1=\pi_2$인 경우는 $key(node(\pi_1))\neq k$가 만족된다. 이것은 편집 행위 Σ_2에서 동일한 경로의 동일한 키 에 대해 삽입이 가능하려면 원래 존재하던 해당 노드가 삭제되었어야 하는데, 한 부분트리가 삭제된 후 동일한 키로 부분 트리를 다시 삽입하는 것은 허용되지 않는다([정의 3.3] (ii)번).

B. $\pi_2\ll\pi_1$인 경우는 발생하지 않는다. 왜냐하면 Σ_1에서 삽입한 트리의 자손을 Σ_2에서 접근하는 것은 가능하지 않기 때문이다.

C. $\pi_1\ll\pi_2$의 경우 σ_i에 의해 삭제된 부분 트리의 하부 노드를 Σ_2에서 삽입하거나 수정한 경우 충돌을 일으킨 행위 중 한 가지를 선택하여야 된다.

③ $\sigma_i=\text{ListInsert}(\pi_1)$, $\rho_j=\text{ListInsert}(p_2)$라면, $\pi_1\sim\pi_2$이거나 $\pi_1=\pi_2$이 만족된다.

A. $\pi_1\ll\pi_2$또는 $\pi_2\ll\pi_1$는 가능하지 않은데, 왜냐하면 한 쪽 편집 기록에서 삽입한 트리의 자손을 다른 부분 트리에서 접근할 수 없기 때문이다.

B. 동일한 경로의 자손으로 부분 트리를 삽입하는 경우 키가 다르다면 두 부분 트리가 충돌 없이 삽입될 수 있다. 그러나 키가 같다면 두 부분 트리를 모두 삽입할 수 없으므로 충돌이다.

④ $\sigma_i=\text{ListDelete}(\pi_1)$, $\rho_j=\text{update}(\pi_2,x)$, $\pi_1\ll\pi_2$인 경우만 충돌이 발생한다.

A. $\pi_1=\pi_2$이거나 $\pi_1\gg\pi_2$인 경우는 발생하지 않는다. π_2가 터미널 노드를 가리키므로.

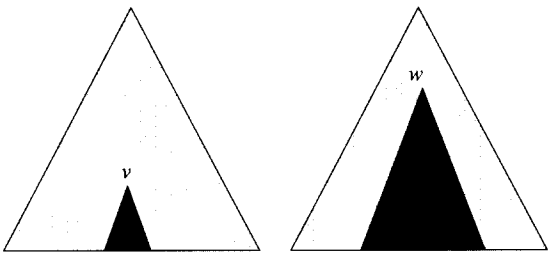
⑤ $\sigma_i=\text{ListInsert}(p_1)$, $\rho_j=\text{update}(\pi_2.x)$, ④와 동일함

⑥ $\sigma_i=\text{update}(\pi_1,x_1)$, $\rho_j=\text{update}(\pi_2,x_2)$, $\pi_1=\pi_2$인 경우만 충돌이 발생한다.

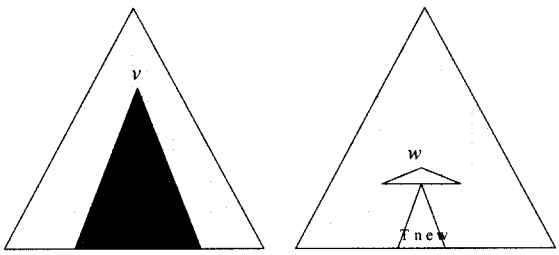
위의 관찰 결과를 표로 나타내면 다음과 같다.

〈표 1〉 행위와 경로에 따른 충돌 관계표

	$\pi_1=\pi_2$	$\pi_1\ll\pi_2$	$\pi_2\ll\pi_1$
$\sigma_i=\text{ListDelete}(\pi_1)$, $\rho_j=\text{ListDelete}(\pi_2)$	OK	Conflict	Conflict(*)
$\sigma_i=\text{ListDelete}(\pi_1)$, $\rho_j=\text{ListInsert}(\pi_2,T_{new})$	NA	Conflict(**)	NA
$\sigma_i=\text{ListDelete}(\pi_1,T_{new1})$, $\rho_j=\text{ListInsert}(\pi_2,T_{new2})$	키가 같으면 충돌	NA	NA
$\sigma_i=\text{ListDelete}(\pi_1)$, $\rho_j=\text{update}(\pi_2,x)$	NA	Conflict	NA
$\sigma_i=\text{ListInsert}(\pi_1)$, $\rho_j=\text{update}(\pi_2,x)$	NA	NA	NA
$\sigma_i=\text{update}(\pi_1,x_1)$, $\rho_j=\text{update}(\pi_2,x_2)$	데이터 충돌	NA	NA



(*) ListDelete(v)와 ListDelete(w), path(v)≫path(w)



(**) ListDelete(v)와 ListInsert(w, Thew), path(v)≫path(w)

(그림 3) 경로에 의한 충돌 예 (<표 1>에서 *과 ** 표시된 경우)

협상을 위해서 편집 행위가 충돌한 경우 그것을 해결한 하나의 버전을 만들어야 한다. 이를 위하여 적용할 수 있는 규칙을 충돌 해결 규칙이라고 하는데, 일반적으로 XML 문서 협상에 관한 기존 문헌에서는 의미적 충돌 해결 규칙이 많이 언급되었다. 의미적 충돌 해결 규칙은 데이터의 특성이나 사용자의 권한 등 응용에 따라 달라지는 규칙을 사용자가 협상 엔진에 제공하여 충돌을 해결하도록 하는 방법이다. 한편 구문적인 규칙을 적용하여 충돌을 해결하려는 노력도 있었는데, 그 예로 래티스 방식에 따라LUB(Least Upper Bound) 통합을 한 [13]의 경우와 부분집합 방식에 의해 협상을 한 DeltaXML의 예가 있다[7].

본 연구에서는 List 데이터에 대한 삽입 삭제만 허용하므로 행위에 따라 또는 경로 포함 규칙에 의해 구문적인 충돌 해결 규칙을 지정하는 것이 가능하다. 이것은 yacc 등의 파서 생성기에서 사용하는 연산자 우선 순위 방식과 유사하다고 할 수 있다[14].

[정의 4.1] *리스트 기반의 XML 데이터 공유 환경에서 사용되는 구문적 협상 규칙(RP : Reconciliation policy)은 다음과 같이 정의된다.*

(i) *행위의 종류에 따른 우선 순위 규칙*
(iii) *동일한 행위에 대해 경로의 포함관계에 따른 우선순위*
(iv) *데이터 협상 규칙 : 요소 이름에 따라 (max, min, sum, left, right) 등의 구문적 규칙 중에서 선택하게 함*

구문적 충돌 해결 규칙의 예로 아래와 같은 세 가지 규칙을 고려해 보자.

① ListInsert > ListDelete
② Update > ListDelete
③ ListDelete π_1 > ListDelete π_2 if $\pi_1 \ll \pi_2$

우선 ①과 ②의 규칙은 삭제된 부분 트리의 아래 쪽에 삽입이나 데이터 수정이 있을 때는 삽입과 수정을 우선한다는 규칙이다. 세 번째 규칙은 삭제 충돌이 일어난 경우 작은 부분 트리의 삭제를 우선한다는 것이다. 결과적으로 충돌이 일어난 경우 삭제는 우선순위가 낮아지고 결과 트리는 가능한 한 삭제가 아닌 행위를 모두 포함하고 있는 것으로 선택하게 된다. 배송리스트 문서의 예에서 (그림 2)(a)의 변환과 충돌을 가지는 다음의 편집 기록을 고려해보자.

Σ_1 = { σ_1 = ListDelete("배송리스트/배송[733654]/물품리스트/물품[번호 = '1']"), σ_2 = ListInsert("배송리스트/배송[733654]/물품리스트", T_{new1}), σ_3 = ListInsert("배송리스트", T_{new2})}

Σ_2 = { ρ_1 = ListInsert("배송리스트/배송[733423]/물품리스트", T_{new5}), ⋯, ρ_m = ListDelete("배송리스트/배송[733654])}

여기서 σ_1와 σ_2은 ρ_m과 충돌을 일으킨다. 이 경우 위의 우선순위 규칙을 적용하면 ListInsert가 ListDelete에 우선한다는 규칙에 의해 σ_1이 ρ_m보다 우선하고, 같은 ListDelete에 대해서는 더 작은 트리를 가지는 쪽이 우선한다는 규칙에 의해 σ_2가 ρ_m보다 우선한다.

한편 삽입과 삽입 충돌은 같은 리스트 부모 노드에 대한 삽입이라도 키가 다르면 충돌이 아니고 두 가지 다 삽입할 수 있다. 키가 같은 경우의 충돌 해결은 키 값을 고유하게 지정할 수 있도록 바꾸거나(postprocessing)[7, 10] 또는 부분 트리 협상에 의해 통합 트리를 삽입하는 방법이 가능하다. 아래에서는 부분 트리가 구조적합 관계인가를 검사하면서 구조 적합인 경우 협상 트리를 생성하는 부분 트리 협상 알고리즘을 소개한다.

4.2 편집 기록 협상 알고리즘

편집 기록은 부모 선방문(preorder) 방식에 의해 경로로 소팅되어 있고, 중복과 포함 행위는 제거된 정제된 편집 기록이며, 동일한 경로에 대한 행위는 삭제가 먼저 나오고 나서 삽입이 나온다고 가정한다.

$$\Sigma_1 = \{\sigma_1, \sigma_2, \cdots, \sigma_k\}$$
$$\Sigma_2 = \{\rho_1, \rho_2, \cdots, \rho_n\}$$

선언적 해결 규칙의 집합인 RP(Reconciliation Policy Set)가 있어서 충돌 해결을 위한 정책을 사용자가 지정한다고 가정한다. 이 때 RP는 구문적인 충돌 해결 규칙인 경우 위의 [정의 4.1]에서 살펴본 세 개의 규칙으로 표현된다. 데이터 update의 충돌은 기존의 방법대로 의미적인 해결 규칙을 사용한다고 본다.

입력 : $\Sigma_1 = \{\sigma_1, \sigma_2, \cdots, \sigma_k\}$, $\Sigma_2 = \{\rho_1, \rho_2, \cdots, \rho_n\}$, 해결 규칙 RP
출력 : 통합 편집 기록 Λ
1 $i \leftarrow 1$, $j \leftarrow 1$
2 σ_i, ρ_j에 대해서 $i < k$, $j < n$일 동안 계속한다
2-1 $\sigma_i \sim \rho_j$이면서 preorder(σ_i) > preorder(ρ_j)이면 $\Lambda = \Lambda + \rho_j$, $j = j+1$

2-2 $\sigma_i \sim \rho_j$이면서 $preorder(\sigma_i) < preorder(\rho_j)$이면 $\Lambda = \Lambda + \sigma_i$, $i \leftarrow i+1$
2-3 $path(\sigma_i) \ll path(\rho_j)$이면 $j' = \max_{j \geq j}(path(\sigma_i) \ll path(\rho_j))$, $j \leftarrow j'+1$
2-3-1 ρ_j와 $\rho_j, \cdots, \rho_{j'}$까지의 편집 행위의 우선순위를 RP에 의해 비교함
2-3-2 σ_i가 우선 순위가 높으면 $\Lambda = \Lambda + \sigma_i$
2-3-3 $\rho_j, \cdots, \rho_{j'}$ 중에 σ_i보다 우선순위가 높은 것이 있으면
$\Lambda = \Lambda + \rho_j, \cdots, \rho_{j'}$으로 바꿈
2-3-4 $i \leftarrow i+1$, $j \leftarrow j'+1$
2-4 $path(\sigma_i) \gg path(\rho_j)$이면 $[\sigma_i, \cdots, \sigma_I]$에 대해 C와 동일한 방식으로 우선순위가 높은 것을 추가, 낮은 것을 삭제함
2-5 $path(\sigma_i) = path(\rho_j)$이면
2-5-1 둘다 LiistDelete이면 하나만 추가함. $\Lambda = \Lambda + \sigma_i$, $i \leftarrow i+1$, $j \leftarrow j+1$
2-5-2 둘다 Update이면 Data 협상 방법을 따라 하나의 Update만 Λ에 추가함
2-5-3 둘다 Insert이면
2-5-3-1 $key(\sigma_i) \neq key(\rho_j)$이면 두 삽입을 모두 에 추가함
2-5-3-2 아니면, 아래 부분 트리 협상에 의한 통합 트리를 삽입한다. 단 $T_i \approx T_j$가 아니면 fail함.
$\Lambda = \Lambda + ListInsert(path(\sigma_i), T_i \oplus T_j)$
2-5-3-3 두 경우 모두 $i \leftarrow i+1$, $j \leftarrow j+1$

(알고리즘 4.1) 편집 기록 협상 알고리즘

이 알고리즘은 단계 2를 $O(|\Sigma_1|+|\Sigma_2|)$번돌게 된다. 부분 트리의 삽입 시 키 충돌이 없다면 각 협상 단계가 $O(1)$이므로 전체 복잡도는 $O(|\Sigma_1|+|\Sigma_2|)$이 된다. 그러나 2-5-3-2 단계의 부분 트리의 협상이 필요한 경우 아래 (알고리즘 4.2)를 이용하여야 한다.

입력 : 입력 트리 T_1, T_2
출력 : 통합 트리 $T_m = T_1 \oplus T_2$ $T_i \approx T_j$ 가 아니라면 fail을 출력함.
1. $name(root(T_1)) \neq name(root(T_2))$이면 $T_1 \approx T_2$가 아니므로 fail
2. 노드 v가 $name(root(T_1))$을 가지는 T_m의 루트 노드라 하자.
3. $T_{11}, T_{12}, \cdots, T_{1n1}$과 $T_{21}, T_{22}, \cdots, T_{2n2}$가 $root(T_1)$과 $root(T_2)$의 자식 트리들이라면,
i. $root(T_1)$이 리스트 부모 노드가 아니면
1. $n_1 = n_2$여야 하고
2. $T_{1i} \approx T_{2i}, 1 < i < n_1$을 만족하여야 하며,
3. $T_{1i} \oplus T_{2i}$를 v의 자식트리로 추가한다.
ii $root(T_1)$이 리스트 부모 노드라면
1. T_{1i}에 대해서 $key(T_{1i}) = key(T_{2j})$인 j가 존재하지 않는다면 T_{1i}를 v의 자식트리로 추가한다.
2. 마찬가지로 T_{2j}도 대응하는 동일키의 자식트리가 T_1에 존재하지 않으면 v의 자식트리로 추가한다.
3. $key(T_{1i}) = key(T_{2j})$인 모든 부분 트리 T_{1i}와 T_{2j}에 대해서 $T_{1i} \oplus T_{2j}$를 v의 자식트리로 추가한다. 단 $T_{1i} \approx T_{2j}$가 아니면 fail임.

(알고리즘 4.2) 부분 트리 협상 알고리즘

(알고리즘 4.2)는 리스트 기반의 협상으로 트리의 크기에 대해 선형 시간 복잡도를 가진다. 위에서 살펴보았던 예를 통해 편집 기록 협상의 결과를 보면 아래와 같다.

$\Sigma_1 = \{ \sigma_1 = $ ListDelete("배송리스트/배송[733654]/물품리스트/물품[번호='1']"), $\sigma_2 =$ ListInsert("배송리스트/배송[733654]/물품리스트", T_{new1}), $\sigma_3 =$ ListInsert("배송리스트", T_{new2})},

$\Sigma_1 = \{ \rho_1 =$ ListInsert("배송리스트/배송[733423]/물품리스트," T_{new5}), $\cdots, \rho_m, =$ ListDelete("배송리스트/배송[733654]")}에 대해 σ_1, σ_2와 ρ_m의 충돌 이외에 다른 충돌이 없다고 가정하면 $\Sigma_1 \oplus \Sigma_2 = \{\sigma_1, \sigma_2, \sigma_3, \rho_1, \rho_2, \cdots, \rho_{m-1}\}$이 된다.

4.3 비 교

모바일 환경의 XML 데이터 협상에 관한 최근의 연구 결과와 본 논문에서 제안된 방법을 비교한다.

XMIDDLE의 협상은 two-way 결합만 고려하였으며, 대신 버전 관리 방식을 통해 모바일 단말간의 차이를 표시하도록 하였다. 이들의 방법은 트리 비교 알고리즘을 사용하여 $O(n^2)$시간이 요구된다[11]. 한편 TMA 알고리즘은 노드 매핑 단계와 통합 단계로 나누어서 노드 매핑이 $O(n^2)$인 새로운 알고리즘을 소개하였다[10]. 이들은 ID 기반의 매핑이 가능한 경우 $O(n)$이라고 하였다.

본 논문에서는 반복부 형제간에 구별되는 키 값을 이용한 $O(n^2)$협상 알고리즘을 소개하였다. 키의 충돌이 없다고 가정하는 경우 편집기록의 길이를 m이라 할 때 $O(m)$복잡도를 가지는 협상 알고리즘을 제시하였다.

아래 <표 2>는 기존의 방법과 본 논문에서 제시된 방법의 차이를 보여준다.

〈표 2〉 협상 방법의 비교

	XMIDDLE / TMA	본 논문에서 제안된 방법
협상 알고리즘 시간 복잡도	$O(n^2)$ id 충돌이 없는 경우 $O(n)$	$O(n^2)$ 키 충돌이 없는 경우 $O(m)$
통신량	$O(n)$	$O(m)$
해결규칙	구문적 또는 의미적 해결규칙	구문적 해결규칙을 적용
문서의 가정	일반 XML 데이터	반복부 정규형 XML 데이터
행위의 제약	임의의 수정 행위	반복부 편집 행위

n : XML 트리의 노드 개수, m : 편집 기록의 길이 (편집 행위의 개수)

모바일 환경에서는 편집 행위의 충돌을 방지하기 위해 서버에서 키를 후 배정하는 방법을 사용하는 것이 적합할 것으로 기대된다. 그 경우 위의 <표 2>에서 보여지는 바와 같이 제안된 방법은 통신량이나 협상 시간 복잡도에서 $O(m)$을 보여, $O(n)$인 기존의 방법에 비해 우수한 성능을 보인다. XML 트리의 노드 개수 n이 보통 100~500개 정도인 데 비해, 편집 기록의 길이 m은 5~10 정도이다.

이러한 성능의 향상은 다음의 두 가지 원인에 의해 가능해 진다. 우선 본 논문에서 사용하는 편집 기록 기반의 협상은 기존의 XML 트리 비교 방법에 비해 노드 매핑의 단계를 생략할 수 있어 협상의 시간복잡도를 낮출 수 있다. 한편 제안된 방법이 가지는 XML 문서와 편집 행위에 대한 제약으로 일반적인 경우를 모두 다루는 기존의 방법에 비해 훨씬 효율적인 알고리즘이 가능하였다. 통신량과 협상을 위한 계산 속도의 효율이 중요한 모바일 환경에서는 제안된 방법이 효과적으로 사용될 수 있을 것으로 기대된다.

한편 본 논문에서 제안된 구문적 협상 방법은 충돌 시 삽입이나 수정을 삭제보다 우선하는 규칙을 적용하면 자연스럽게 구조적 충돌을 해결할 수 있다. 간단한 규칙은 사용자들이 이해하고 사용하면 결과를 예측할 수 있게 해 주며, 또한 삭제를 최소화하므로 추후 복구가 가능한 결과를 생성한다. 기존에 구문적인 협상 방법을 이용한 예로는 Lattice 방식에 의해 LUB 통합을 제시한 [13]와 부분집합에 의한 통합 방법을 소개한 [7] 등이 있다. 본 논문에서 제안된 구문적 충돌 해결 규칙은 yacc 등에서 사용된 파서 충돌해결 규칙[14]과 유사한 방법으로 반복부 공유 모델에 적합할 것으로 기대된다.

5. 결 론

본 연구에서는 모바일 환경에서 XML 문서의 편집 기록(edit script)을 협상하기 위한 효율적인 협상 방법을 제시하였다. 본 논문에서는 모바일 환경의 요구를 만족시킬 수 있는 XML 데이터 공유 모델로서 문서 타입에서 반복부에 해당하는 부분만 삽입과 삭제를 허용하고 반복부의 부분 트리들이 형제들과 구분되는 키 값을 가지는 리스트 공유 모델을 이용한다. 이러한 리스트 공유 모델은 구조 변경 행위가 항상 문서의 유효성을 보장할 수 있다는 장점을 가지며, 키 기반의 리스트 협상을 하게 되므로 매우 효율적인 협상 알고리즘을 얻는 것이 가능하다. 본 논문에서 제시되는 알고리즘은 트리 크기를 n이라 할 때 일반적으로는 $O(n^2)$복잡도를 가지지만, 여러 사용자 간에 키의 충돌이 없다는 가정을 한다면 편집 기록의 길이를 m이라 할 때 $O(m)$복잡도를 가진다. 또한 통신량의 측면에서도 문서 전체가 아닌 편집 기록만 전송하여 $O(m)$의 통신량을 가져, ID 충돌이 없는 경우 $O(n)$시간복잡도 및 통신량을 보이는 기존 알고리즘[7, 11]에 비해 모바일 환경에서 효과적으로 사용될 수 있을 것으로 기대된다.

본 논문에서 제안된 협상 방법은 현재 서버 환경에서 구현 중이며, 이를 모바일 환경에서 협상을 통한 데이터 공유 플랫폼으로 확장할 예정이다. 또한 제안된 방법과 기존의 방법을 성능 면에서 비교 평가할 계획이다.

참 고 문 헌

[1] 저자, "XML 문서 공유를 위한 리스트 잠금 프로토콜", 정보처리학회논문지D, 제11-D권 제7호, 게재예정, 2004.

[2] S.Agarwal, et al., "On the scalability of data synchronization protocols for PDAs and mobile devices," IEEE Network, Vol.16, No.4, July, 2002.

[3] C. Amza, A. Cox, W. Zwaenepoel. "Distributed Versioning : Consistent Replication for Scaling Back-end Databases of Dynamic Content Web Sites," Proceedings of the ACM/IFIP/Usenix Middleware Conference, 2003.

[4] Giacomo Cabri, Letizia Leonardi, Franco Zambonelli, "XML dataspaces for mobile agent coordinatio," Proceedings of the 2000 ACM symposium on Applied computing, 2000.

[5] S.Chawathe, et al., "Change detection in hierarchically structured information," ACM SIGMOD, pp.493-504, 1996.

[6] G. Miklau, D. Suciu, "Containment and equivalence for an Xpath fragment," Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp.65-76, 2002.

[7] R. Fontaine, "Merging XML files : a new approach providing intelligent merge of XML data sets," In Proceedings of XML Europe 2002, May, 2002.

[8] Anne-Marie Kermarrec, et al., "The IceCube approach to reconciliation of divergent replicas," 20th Symp., on Principles of Dist. Comp. (PODC), Newport RI (USA), Aug., 2001.

[9] Franky Lam, Nicole Lam, Raymond Wong, "Efficient synchronization for mobile XML data," Proceedings of the eleventh international conference on Information and knowledge management, pp.153-160, 2002.

[10] Tancred Lindholm, "Consistency and replication : XML three-way merge as a reconciliation engine for mobile data," Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access, pp.93-97, 2003.

[11] C. Mascolo, L. Capra, S. Zachariadis and W. Emmerich. XMIDDLE : A Data-Sharing Middleware for Mobile Computing. *Personal and Wireless Communications*, April, 2002.

[12] Shirish H. Phatak and B.R. Badrinath, "Transaction-centric Reconciliation in Disconnected Databases," *ACM Monet Journal*, Vol.53, 1999.

[13] Kristin Tufte, David Maier, "Merge as a Lattice-Join of XML Documents," Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002.

[14] S.Johnson, "Yacc : Yet Another compiler compiler,"AT&T Bell lab Technical report No.32, 1975.

이 은 정

e-mail : ejlee@kyonggi.ac.kr

1988년 서울대학교 계산통계학과(학사)

1990년 한국과학기술원 전산학과 (공학석사)

1994년 한국과학기술원 전산학과 (공학박사)

1994년~2000년 한국전자통신연구원 선임연구원

2001년~현재 경기대학교 정보과학부 조교수

관심분야 : 컴파일러 이론, XML 처리 기술 등