XML 데이타 갱신에 대한 효율적인 유효 검증 기법

(An Efficient Validation Method for XML Data Updates)

이 지 현 † 박 명 제 † 정 진 완 ††
(Ji-Hyun Lee) (Myung-Jae Park) (Chin-Wan Chung)

요 약 XML은 웹 데이타 표현 및 교환을 위한 표준으로 많은 응용 분야에서 널리 이용되고 있다. XML Schema는 표준 XML 스키마로, 정의된 XML Schema에 의거하여 작성된 XML 문서를 '유효한 XML 문서'라고 하며 이러한 XML 문서는 갱신이 된 후에도 XML Schema에 대해 유효함이 보장되어야 한다. 본 논문은 갱신 이전에 갱신과 관련된 부분에 대한 유효 검증을 수행하여 불필요한 유효 검증을 제거한 XML Schema 유효 검증 메커니즘인 '예상 갱신 부분 유효 검증 기법'과 갱신 별 유효 검증 알고리즘을 제안한다. 또한 XML 데이타와 XML Schema 간의 매핑 방법과 XML Schema 유효 검증을 지원하기 위해 효율적인 스키마 정보 추출을 제공하는 XML Schema 저장 방법을 제안한다. 마지막으로 실험을 통해 스키마 저장 방법에 따른 갱신 별 유효 검증 성능을 비교한다.

키워드 : XML, XML Schema, 갱신, 유효 검증

Abstract XML is widely used in various applications as the standard for representing and exchanging data on the Web. XML Schema is the standard schema for XML and an XML document generated based on the XML Schema is called 'Valid XML document'. The XML Schema validity should be guaranteed after the XML document is updated. In this paper, we design an efficient method that verifies XML Schema validity before update, and so eliminates unnecessary validations. Also, we propose validation algorithms for each update. In addition, we propose the mapping between XML data and XML Schema and a storage method for XML Schema in order to efficiently extract the schema information for the validation. Finally, we compare the performance of the validation according to the storage methods.

Key words : XML, XML Schema, update, validation

1. 서 론

웹 상의 데이타 표현 및 교환의 표준인 XML(eXtensible Markup Language)[1]은 사용자 정의가 가능한 비 정형적인 구조를 가지는 준 구조적(semi-structured) 데이타[2]이다. XML은 구조적으로 유연함과 동시에 스키마(예, DTD[3], XML Schema[4])를 통해 데이타 구조를 정의할 수 있는 특징을 바탕으로 많은 응용 분야에서 이용되고 있다. XML 문서에 대해 XML Schema가 정의되어 있다면 그 문서는 XML Schema에 정의된 타입 및 구조적 제약 조건에 의거하여 생성되어야 한다. 이렇게 생성된 XML 문서를 '유효한 문서'라고 하며, 유효한 XML 문서는 갱신(Update)이 된 후에도 XML Schema에 대해 유효함이 보장되어야 한다. 즉, XML 데이타에 대한 갱신을 지원하는 시스템들은 효율적인 XML 데이타 갱신과 더불어 갱신 결과가 XML Schema에 대해 유효함을 보장해야 한다. 따라서 본 논문에서는 XML 데이타 갱신에 대한 효율적인 XML Schema 유효 검증 방법 및 이를 지원하기 위한 XML Schema 저장 방법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 XML 저장 시스템 별 갱신 방법 및 전통적인XML 유효 검증 기법에 대해 간단히 소개한다. 3장에서는 본 논문에서 다루는 XML 데이타와 XML Schema 모델에 대해 설명하고 XML 데이타와 XML Schema 간의 매

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성·지원 사업(IITA-2006-C1090-0603-0031)의 연구결과로 수행되었음
† 학생회원 : 한국과학기술원 전산학과
hyunlee@islab.kaist.ac.kr
jpark@islab.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전산학과 교수
chungcw@islab.kaist.ac.kr
논문접수 : 2004년 10월 12일
심사완료 : 2006년 11월 27일

핑에 대해 다룬다. 4장에서는 XML Schema에 대해 유효한 XML 데이타가 만족해야 할 XML Schema 유효 제약 조건에 대해 설명하고, 5장에서는 XML 데이타 갱신에 대한 효율적인 XML Schema 유효 검증 기법 및 갱신 별 유효 검증 알고리즘을 제안한다. 또한 6장에서는 유효 검증을 위한 스키마 정보를 효율적으로 추출하기 위한 XML Schema 저장 방법을 제안한다. 마지막으로 7장에서는 스키마 저장 방법에 따른 유효 검증의 성능을 실험을 통해 보이고 결과를 분석한다.

2. 관련연구

XML 데이타 갱신 방법은 저장 및 질의 처리 방법과 마찬가지로 저장 시스템과 저장 스키마에 의존적이다. 파일 시스템의 경우, XML 문서 전체를 메모리에서 파싱하고 갱신 위치에 접근하여 갱신을 수행한 후 다시 파일로 변환하여 파일 시스템에 재저장해야 한다. 갱신의 종류에 따라서는(예, 이동(Move) 갱신) 문서 전체 혹은 일부를 특정 데이타 구조에 유지해야 한다. 따라서 갱신 성능은 데이타 크기 혹은 데이타를 유지하기 위한 데이타 모델에 따라 좌우된다. 또한 XML Schema와 같은 스키마를 동반한 경우 갱신 후 일단 파일 시스템에 저장되면 스키마에 대한 유효성 보장을 위해서는 XML 문서 전체에 대한 유효 검증이 불가피하다.

객체 지향형 데이타베이스 관리 시스템의 경우, XML 데이타는 객체 노드들로 구성된 트리 형태로 저장된다. 그리고 객체 식별자(Object Identifier) 기반 탐색을 통해 갱신이 적용될 위치를 찾아 새로운 객체를 삽입(Insert)하거나 기존의 객체를 삭제(Delete)하게 된다. 실제 객체 지향형 데이타베이스 관리 시스템을 기반으로 만들어진 XML 전용 저장소인 eXcelon[5]에서는 XML 데이타를 Portal Server를 통해 영구적인 트리 형태로 저장하며 각 객체 노드에 대한 삽입, 삭제, 이동 등의 갱신을 지원하고 있다. 그리고 eXcelon에서는 스키마로서 DTD만을 지원하며 XML 문서 전체에 대한 유효 검증만을 제공한다.

관계형 데이타베이스 관리 시스템을 통해 XML 문서를 저장하는 경우에는 저장 스키마에 따라 XML 갱신 질의를 적절한 SQL의 갱신 질의로 변환하여 수행하게 된다. [6]에서는 삽입, 삭제, 새 이름 할당(Rename), 수정(Modify) 등의 기본 갱신을 지원하도록 XML 표준 질의 언어인 XQuery[7]를 확장하였다. 그리고 삭제 갱신 시 연쇄적인 삭제 성능을 향상 시키기 위해 트리거(Trigger)를 사용하는 등 갱신의 종류 별로 데이타베이스 관리 시스템이 제공하는 다양한 기능들을 이용한 방법들을 제시하고 그 성능을 비교 분석 하였다. 그러나 갱신된 문서의 유효성 검증에 대한 언급은 없다.

대표적인 관계형 및 객체 관계형 데이타베이스 시스템을 기반으로 한 XML 저장 시스템인 Oracle XML DB[8,9]에서는 XML Schema로부터 관계형 스키마를 생성하여 XML 문서를 저장하기 때문에 새로운 문서를 저장할 때나 삽입 갱신이 발생했을 때에는 XML Schema에 정의되지 않은 엘리먼트(Element)들이나 에트리뷰트(Attribute)들을 검출해내는 간단한 구조적 제약 조건 검증은 제공한다. 그러나 XML Schema에 정의된 모든 제약 조건들을 만족하는지 확인하기 위해서는 갱신 후 문서 전체에 대하여 유효 검증을 수행해야 한다. 또한 XML Schema를 파일 시스템에 저장하고 Oracle XML DB에 등록하는 형식으로 XML Schema를 유지하기 때문에 특정 요소의 구조 정보를 알기 위해서는 XML Schema 전체를 탐색해야 한다.

DTD는 정규 표현(regular expression)으로 엘리먼트 컨텐츠 모델(content model)을 정의한다. 따라서 S를 엘리먼트의 자식 엘리먼트들의 이름들로 이루어진 문자열이라 하고, E를 DTD에 선언된 엘리먼트의 컨텐츠 모델, L(E)는 E를 만족하는 문자열의 집합(즉, Language)이라 할 때, DTD 유효 검증은 S가 L(E)에 속하는지를 검증한다[1]. 하지만 이 방법을 XML 데이타 갱신에 대한 유효 검증에 적용하기 위해서는 갱신의 종류와는 상관 없이 DTD를 파싱하여 필요한 정규 표현을 추출하고, 그에 대응하는 오토마타를 만들어, 갱신 후 문자열 S'에 대해 상태 전이(state transition)을 수행해야 한다. [10]은 앞서 언급한 오토마타를 통한 유효 검증 메커니즘을 제안하고 있는 가장 최근 연구이다. [10]은 DTD의 유형에 따른 오토마타 기반 유효 검증 방법 제안하고, 유효 검증을 위해 필요한 정규 표현 정보를 접근하는데 유용한 데이타 구조(예, B-tree)를 언급하고 있다. 하지만, 일반적인 스키마(즉, non-conflict-free DTD)의 경우 유효 검증 시 문자열 S를 구성하는 엘리먼트 수만큼의 반복적인 상태 전이 수행은 피할 수 없다. 또한, 갱신의 종류로 엘리먼트와 에트리뷰트의 삽입, 삭제만을 고려하고 있다.

본 논문에서는 효율적인 유효 검증을 위해 XML Schema를 적절히 쪼개어 저장하고, 갱신에 따라 필요한 스키마 정보만을 효율적으로 추출 할 수 있도록 하며, 갱신과 직접적으로 관련된 부분에 대한 유효 검증을 수행함으로써 불필요한 검증은 제거하고, 갱신 이전에 갱신의 유효 여부를 검증 함으로서 위의 번거로운 과정을 피할 수 있다. 또한 DTD와 XML Schema는 XML 데이타와의 스키마 간의 매핑 메커니즘에 차이가 있고, 엘리먼트 카디널리티와 같이 DTD가 포함하지 않는 제약 조건들 때문에 XML Schema에 대한 유효 검증에는 단순히 기존의 방법을 그대로 이용할 수가 없으므로 본

논문에서는 XML Schema을 고려하여 유효 검증 알고리즘과 스키마 저장 방법을 고안하였다. 또한, 본 연구에서 엘리먼트와 에트리뷰트의 삽입 및 삭제 뿐만 아니라 이동, 새 이름 할당 그리고 텍스트 데이타 수정에 대한 유효 검증 메커니즘 및 갱신 알고리즘을 설계, 구현, 실험하였다.

3. XML 데이타와 XML Schema간의 매핑

본 논문에서 다루는 XML 데이타와 XML Schema 모델을 설명하고, XML 문서를 구성하는 요소(예. 엘리먼트)와 XML Schema 상에 선언되어 있는 각 요소에 대한 선언(예. 엘리먼트 선언) 간의 매핑을 정의한다.

정의 1. XML 문서는 XML 트리 X = (V, E)로 나타낸다. V는 XML 문서를 구성하는 요소인 엘리먼트와 에트리뷰트들에 대응되는 노드들의 집합이다. E는 노드들 간의 부모-자식 관계를 나타내는 간선들의 집합이다. 형제(sibling) 엘리먼트 노드들 사이에는 순서가 존재하며 말단(leaf) 노드들은 데이타 값을 갖는다. 모든 v ∈ V는 다음과 같은 정보를 갖는다.

v.name	v에 대응되는 엘리먼트 혹은 에트리뷰트의 이름
v.type	노드의 종류로 v.type ∈{'element', 'attribute'}
v.value	v가 말단 노드인 경우 데이타 값
parent(v) /children(v)	v의 부모 노드/v의 자식 노드들의 리스트
leftSibling(v) /rightSibling(v)	v의 좌·우 형제 노드
cardinality(parent(v), v.name)	parent(v)가 부모 노드이며 v.name이 이름인 노드들의 수

XML 표준 스키마인 XML Schema는 XML 문서에 나타나는 요소(즉, 엘리먼트, 에트리뷰트)들의 선언과 그들이 가지는 타입(Type)의 정의로 구성된다. 요소의 선언에서는 해당 요소가 어떤 타입을 가지는지 선언한다. XML Schema에서는 타입으로 String과 같은 원시(primitive) 타입뿐만 아니라 사용자 정의 타입을 제공한다. 사용자 정의 타입 정의(즉, complexType)에서는 그 타입을 갖는 엘리먼트의 자식 엘리먼트들과 에트리뷰트들의 선언을 포함하고 자식 엘리먼트들 간의 순서와 카디널리티(cardinality)와 같은 구조적 제약을 정의한다. 본 논문에서는 다루는 XML Schema의 범위를 XML Schema의 가장 기본적인 기능을 표현 할 수 있는 최소 범위의 엘리먼트와 에트리뷰트들로 한정하였으며 아래와 같다. 또한 그림 1은 XML Schema의 예이다.

정의 2. XML Schema XS = (T, D)는 타입 정의(Type Definition)들의 집합 T, D는 선언(Declaration)들의 집합이다. D는 엘리먼트 선언(Element Declaration)들의 집합 ED, 그리고 에트리뷰트 선언(Attribute Declaration)들의 집합 AD을 포함한다(D = ED ∪ AD). 모든 d ∈ D와 t ∈ T는 다음과 같은 정보를 가지고 있다.

d.name	d로 선언된 엘리먼트 혹은 에트리뷰트의 이름
d.minC/d.maxC	d로 선언된 엘리먼트가 같은 부모 밑에 나타날 수 있는 최소/최대 수
d.order	d ∈ ED, complexType정의 안에서 d의 순서
d.type	엘리먼트 혹은 에트리뷰트의 타입으로 말단 엘리먼트와 에트리뷰트의 경우에는 xsd:String, 그 외의 엘리먼트들은 특정 complexType을 갖는다고 가정
d.attrType	d ∈ AD, d.attrType ∈ {'ID', 'IDREF'}
d.use	d ∈ AD, d.use ∈ {'require', 'optional'} d에 대응하는 에트리뷰트가 문서상에 반드시 나타나야 하는지 여부를 나타냄
superType(d)	d를 포함하는 complexType
t.name	타입 t 의 이름

complexType 정의 안에서 엘리먼트 선언들은 특정 그룹(group) ∈ {'sequence', 'choice'}에 속한다. 하나의 그룹은 다른 그룹을 포함할 수 있기 때문에 하나의 엘리먼트 선언은 중첩된 여러 그룹에 동시에 포함될 수 있다. 그러므로 두 형제 엘리먼트 노드들은 하나 이상의 공통된 그룹에 속할 수 있다.

정의 3. 두 형제 엘리먼트 노드들이 속한 공통된 그룹들 중 가장 내부에 있는 그룹을 '최하위 공통 그룹(The Lowest Common Group)'이라 하고 형제 노드 d1, d2 ∈ ED의 최하위 공통 그룹을 LCG(d1, d2)이라 표기한다.

XML Schema는 이름 영역(Name scope)으로 전역 이름뿐만 아니라 지역 이름도 허용한다. complexType와 전역으로 선언된 엘리먼트·에트리뷰트의 이름은 전역적으로 유일하지만 특정 complexType 안에서 지역적으로 선언된 엘리먼트·트리뷰트의 이름은 자신을 포함하는 complexType 안에서만 유일하다. 즉, DTD와는 다르게 XML Schema에서는 엘리먼트·에트리뷰트 자신의 이름만 가지고서는 자신의 엘리먼트·에트리뷰트 선언을 유일하게 찾아낼 수 없고, 자신의 이름과 자신의 선언을 포함하는 타입의 이름을 알아야 한다.

정의 4. 스키마 매핑 *m*: V -> D 은 ∃v ∈ V와 그에 대응되는 ∃d ∈ D를 매핑하며 다음과 같은 조건을 만족한다.

(1) 만약 v가 루트라면 v.name = d.name

(2) 만약 v가 루트가 아니라면 *m*(parent(v)).type.name = superType(d).name과 e.name = d.name을 만

```
<schema  targetNamespace = anyURI>
      ((import)*, (complexType| element)*)
</schema>
```

```
<import  namespace=anyURI  schemaLocation = anyURI/>
```

```
<element  maxOccurs = (nonNegativeInteger| unbounded) : 1
          minOccurs = nonNegativeInteger : 1
          name = NCName
          ref = Qname
          type = Qname
      (complexType)?
</element>
```

```
<complexType  name = NCName>
      simpleContent| complexContent| ((choice| sequence)?, (attribute)*)
</complexType>
```

```
<simpleContent>  extension  </simpleContent)
```

```
<complexContent>  extension  </complexConent>
```

```
<extension  base = QName>  (choice| sequence)?, (attribute)*  </extension>
```

```
<choice>  (element| choice| sequence)*  </choice>
```

```
<sequence>  (element| choice| sequence)*  </sequence>
```

```
<attribute  name = NCName
            type = Qname
            use = (optional| required)/>
```

```
<schema  targetNamespace = 'http"//www.example.com/Address'
         xmlns = 'http://www.w3.org/2001/XMLSchema'
         xmlns:add = 'http://www.example.com/Address'
         xmlns:ipo = 'http://www.example.com/IPO'>
   <import  namespace = 'http://www.example.com/IPO'/>
   <element  name = 'purchaseOrder'  type = 'add:PurchaseOrderType'/>
   <complexType  name = 'PurchaseOrderType'>
      <sequence>
         <element  name = 'shipTo'  type = 'add:USAddress'/>
         <element  name = 'billTo'  type = 'add:KRAddress'/>
         <element  name = 'items'  type = 'add:Item'/>
      </sequence>
      <attribute  name = 'orderDate'  type = 'date'/>
   </complexType>
   <complexType  name = 'USAddress'>
      <complexContent  base = 'ipo:Address'>
         <sequence>
            <element  name = 'state'  type = 'ipo:USState'/>
            <element  name = 'zip'  type = 'positiveInteger'/>
         </sequence>
      </complexContent>
   </complexType>
   <complexType  name = 'KRAddress'>
```

그림 1 order.xsd

족해야 한다.

엘리먼트·에트리뷰트 노드의 타입을 알기 위해서는 정의 4에 따라 엘리먼트·에트리뷰트의 선언을 추출 해야 한다. 그리고 그 선언을 찾기 위해서는 그 선언을 포함하고 있는 complexType을 알아야 하는데 이 타입은 부모 엘리먼트 노드의 타입이다. 그리고 부모 노드의 타입 역시 그의 부모 노드의 타입을 알아야 찾아낼 수 있다.

정리 1. 노드 타입은 노드 경로에 대해 유일하므로 특정 노드에 대한 선언은 그 노드의 루트로부터의 노드 경로(Path)를 통해 찾아낼 수 있다. (정의 4와 수학적 귀납법에 의해 쉽게 증명가능)

4. XML Schema 유효 제약 조건

XML Schema는 타입 정의와 XML 문서 상의 요소들에 대한 선언들을 포함하고 있으며 이들 정의와 선언들은 유효한 XML 문서상의 요소들이 만족해야 할 조건들을 내포하고 있다. 하지만 갱신과 관련된 구체적인 제약 조건들과 그들에 대한 적절한 검증 방법을 찾기 위해서는 XML Schema와 갱신에 대한 분석이 필요하

다. 따라서 XML Schema 스펙과 갱신을 분석하여 갱신과 관련된 XML Schema 유효 제약 조건들을 조사, 분석, 분류하였으며 다음과 같다.

엘리먼트 유효 제약 조건: XML 문서 내의 엘리먼트들이 지켜야 할 제약 조건으로 e ∈ V이고 e.type = 'element'인 모든 e에 대하여 다음 조건들을 만족 해야 한다.

(1) d = m(e)인 d ∈ ED가 존재해야 한다.

(2) children(e)는 d.type을 만족해야 한다. (엘리먼트 타입 유효 제약조건)

엘리먼트 타입 유효 제약 조건: 엘리먼트의 자식 노드들이 따라야 하는 유효 제약 조건으로 '엘리먼트 그룹 유효 제약 조건'과 '에트리뷰트 유효 제약 조건'을 포함한다.

엘리먼트 그룹 유효 제약 조건: 엘리먼트들로 구성된 그룹이 만족해야 할 유효 제약 조건으로 다음과 같은 조건들을 포함한다. e, p ∈ V, e.type = p.type = 'element', 그리고 e ∈ children(p) 인 모든 e에 대하여

(1) e는 엘리먼트 유효 제약 조건을 만족 해야 한다.

(2) d∈D이고 d=m(e)인 d에 대하여 d.minC ≤ cardinality(parent(e),e.name) ≤ d.maxC 를 만족해야 한다.

(3) s∈D이고 s∈ children(p)인 s에 대해, 만약 LCG(e,s)='sequence'이고 m(e).order < m(s).order 이면 e.order < s.order 를 만족해야 한다. 만약 LCG(e,s)='choice'이라면 e와 s중 하나만 존재해야 한다.

에트리뷰트 유효 제약조건: 에트리뷰트들이 만족해야 할 제약 조건으로 다음과 같은 조건들을 포함한다. a ∈ V 이고 a.type = 'attribute'인 모든 a에 대하여

(1) d = m(a)인 d ∈ AD가 존재해야 한다.

(2) 만약 d.use = 'require'라면 a는 반드시 문서상에 존재해야 한다.

(3) parent(a) = parent(s)이고 a.name = s.name인 s는 문서상에 존재할 수 없다.

(4) 만약 d.type = 'ID'이면 d.value는 문서 전체에서 유일해야 한다.

(5) 만약 d.type = 'IDREF'이면 a.value를 갖는 ID가 반드시 문서에 존재해야 한다.

ID 에트리뷰트가 삭제되면 삭제된 ID 값에 대한 IDREF 에트리뷰트는 존재하지 않는 ID 값을 참조하게 되며 이는 유효하지 않은 상태(즉, dangling reference)로 분류 된다. 따라서 에트리뷰트 유효 제약 조건 (5)가 필요하다. 추가적으로 ID 에트리뷰트의 삭제로 유효하지 않은 상태가 되면 참조 무결성(Reference integrity)를 보장하기 위해 ID 에트리뷰트가 삭제될 때 이를 참조하는 모든 IDREF 에트리뷰트를 삭제(Cascading deletion)하거나 ID 에트리뷰트 삭제를 무효화 시키게 된다. 갱신에 대한 유효 검증 결과, 이와 같은 유효하지 않은 상태에 따른 처리 방법은 데이타베이스 설계자의 판단에 따라 결정된다.

유효한 XML 문서는 갱신이 된 후에도 앞서 언급한 모든 XML Schema 유효 제약 조건들을 만족해야 한다. 즉, 특정 엘리먼트 혹은 에트리뷰트의 삽입·삭제로 XML 문서의 구조가 바뀌었을 때, 갱신된 XML 문서는 언급한 제약 조건들을 모두 만족해야 한다. 그러므로 갱신 후 유효 검증은 필수적이다. 본 논문에서는 이 장에서 명시한 모든 제약 조건들에 대한 유효 검증을 지원한다.

5. XML 데이타 갱신에 대한 효율적인 유효 검증 기법

원시적인 유효 검증 방법은 갱신 후 XML 문서 전체에 대한 유효 검증을 수행하는 것이다. 그러나 이 방법은 두 가지 큰 문제점을 가지고 있다. 첫째, 갱신과 관계없는 부분에 대해서도 불필요한 유효 검증을 수행해야 하며 협소한 범위의 갱신의 빈도가 높은 경우 넓은 범위의 불필요한 유효 검증의 반복을 초래한다. 둘째, 유효 검증 결과 유효하지 않은 갱신이라면 갱신을 취소시키고 갱신 이전의 상태로 복구 시켜야 한다. 유효한 XML 문서에 대한 갱신은 갱신에 의해 직접적으로 변경된 영역과 갱신으로 인해 유효하지 않은 부분으로 바뀔 수 있는 영역에 대해서만 유효 검증을 수행하면 된다. 또한 유효 검증을 갱신 이전에 수행하여 유효하지 않은 갱신은 사전에 차단하면 불필요한 갱신과 갱신 이전으로의 복구를 피할 수 있다. 본 논문은 이를 바탕으로 XML 데이타 갱신에 대한 효율적인 XML Schema 유효 검증 기법 및 갱신 별 유효 검증 알고리즘을 제안한다.

5.1 예상 갱신 부분 유효 검증 기법

그림 2는 XML 데이타 갱신에 대한 효율적인 XML Schema 유효 검증 과정의 개괄적인 그림이다.

Query Processor는 갱신 질의를 분석하고 데이타베

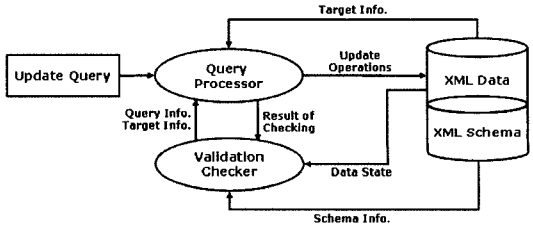

그림 2 XML 데이타 갱신 메커니즘

이스로부터 갱신될 타겟(target) 노드 정보를 추출 해 온 후, Validation Checker를 호출하며 갱신 질의 정보와 타겟 노드 정보를 인자(Parameter)로 넘긴다. Validation Checker는 갱신 질의 종류에 따른 XML Schema 유효 제약 조건과 타겟 노드에 따라 갱신과 관련된 데이타의 상태 정보 및 스키마 정보를 데이타베이스로부터 가져와 갱신 이후의 XML Schema 유효 여부를 예측하고 그 결과를 Query Processor에게 반환한다. 예측 결과가 유효라면 Query Processor는 적절한 SQL 갱신 질의를 생성하여 데이타베이스 상의 XML 데이타를 갱신한다. 이 방법은 갱신 후 갱신된 XML 문서 전체에 대하여 XML Schema 유효 검증을 수행하는 것이 아니라 갱신이 발생하기 이전에 갱신의 종류와 갱신과 관련된 정보를 근거로 '예상 갱신 부분 유효 검증'을 수행함으로써 XML Schema에 유효하지 않은 갱신을 사전에 막고, 불필요하고 중복되는 유효 검증 범위를 없앴다. 또한 갱신에 대한 유효 검증과 관련된 데이타 상태 정보(예, 카디널리티) 및 스키마 정보(예, 타입) 만을 필요한 순간에 추출함으로써 효율성을 높였다.

5.2 갱신 별 유효 검증 알고리즘

4장에서 언급한 유효 제약 조건을 바탕으로 갱신 별 유효 검증 알고리즘을 설계하였다.

5.2.1 삽입 갱신

• 엘리먼트 삽입: InsertBefore/After(target, content)

target 엘리먼트의 앞/뒤에 content를 삽입하는 갱신이다. content가 target 엘리먼트가 속한 엘리먼트 그룹에 삽입되면서 그 그룹만 갱신되므로 엘리먼트 삽입 유효 검증 시에는 XML Schema 유효 제약 조건 중 그룹 유효 제약 조건만 체크하면 된다. 알고리즘 1은 Insert-After()에 대한 유효 검증 알고리즘이다. 라인 1~3는 유효 검증과 관련된 정보들을 추출한다. 라인 4는 그룹 유효 제약 조건 1, 라인 5은 그룹 유효 제약 조건 2, 그리고 라인 6~12는 그룹 유효 제약 조건 3을 체크하여 엘리먼트 n이 엘리먼트 s 뒤에 삽입 가능한가를 확인한다. 이를 위해 알고리즘 1에서는 LCG 값을 계산한다. sd와 nd는 각각 자신을 포함하는 그룹들의 리스트 정보를 포함하며 (6.2장의 XML Schema의 타입 별 저장 스키마 참조) 두 리스트를 비교하여 LCG를 찾는다.

알고리즘 1 Validation_InsertAfter(s, n)
1: Access s, and extract s' declaration sd ∈ ED, sd = *m*(s) 2: Access the immediate right sibling r = rightSibling(s), and extract rd = *m*(r) ∈ ED 3: Extract nd = *m*(n) ∈ ED, given the node path of s 4: IF nd = null THEN Return False 5: IF nd.maxC = cardinality(parent(s), n) THEN Return False 6: IF s.name = n.name OR r.name = n.name THEN Return False 7: ELSE 8: IF LCG(sd, nd) = 'sequence' AND sd.order < nd.order 9: THEN 10: IF LCG(rd, sd) = 'sequence' AND rd.order > sd.order THEN Return True 11: ELSE Return False 12: ELSE Return False

content가 well-formed XML fragment라면 깊이 우선 검색 방식(Depth First Traversing)으로 content 상의 각 노드의 삽입에 대한 유효 여부를 검증하면서 갱신을 수행한다.

• 에트리뷰트 삽입: Insert_attribute(target, attrName, attrValue)

attrName을 이름으로 갖고, attrValue를 값으로 갖는 에트리뷰트를 target의 에트리뷰트로 삽입하는 갱신이다. 유효 검증에서 검증 해야 할 유효 제약 조건들은 에트리뷰트 유효 제약 조건 1, 3, 4, 5이며, 알고리즘 2는 에트리뷰트 삽입 유효 검증 알고리즘이다. 라인 1~2는 유효 검증과 관련된 정보를 추출한다. 라인 3은 에트리뷰트 유효 제약조건 1을, 라인 4와 5는 각각 유효 제약 조건 4와 5를, 라인 6은 유효 제약조건 3을 체크 한다.

알고리즘 2 Validation_InsertAttribute(p, a)
1: Access p, and extract p.type 2: Extract the declaration of a, ad = *m*(a)∈ AD 3: IF ad = null THEN Return False 4: IF ad.type = 'ID' THEN IF a.value is in the document Then Return False 5: ELSE IF ad.type = 'IDREF' THEN IF a.value is not in the document THEN Return False 6: ELSE IF the attribute with a.name already exists in children(p) THEN Return False 7: Return True

5.2.2 삭제 갱신

• 엘리먼트 삭제: Delete_element(target)

target 엘리먼트를 삭제하는 갱신으로 target을 포함한 모든 자손(descendant) 엘리먼트들과 에트리뷰트들을 삭제한다. 유효 검증에서는 삭제되는 엘리먼트 target이 속해 있는 엘리먼트 그룹에 변화가 생기므로 그룹 유효 제약 조건 2를 체크해야 한다. 알고리즘 3은 엘리먼트 삭제 유효 검증 알고리즘이다. 라인 1은 유효 검증과 관련된 정보를 추출하며 라인 2는 그룹 유효 제약 조건 2를 체크한다.

알고리즘 3 Validation_DeleteElement(n)
1: Access n, and extract the declaration of n, nd = *m*(n) ∈ ED 2: IF nd.minC = cardinality(parent(n), n.name) THEN Return False 3: ELSE Return True

• 에트리뷰트 삭제: Delete_attribute(target)

target 에트리뷰트를 삭제하는 갱신으로, 유효 검증에서는 target에 대한 에트리뷰트 유효 제약 조건 2와 5를 체크한다. 알고리즘 4는 에트리뷰트 삭제 유효 검증 알고리즘이다. 라인 1은 유효 검증과 관련된 정보를 추출하며, 라인 2는 유효 제약조건 2를, 라인 3~6은 유효 제약조건 5를 체크한다.

알고리즘 4 Validation_DeleteAttribute(a)
1: Access a, and extract the declaration of a, ad = m(a) ∈ ED
2: IF ad.use = 'Require' THEN Return False
3: IF ad.type = 'ID' THEN
4: count := the number of references to the element having a.value as 'ID'
5: IF count > 0 Then Return False
6: Return True

5.2.3 그 밖의 갱신

본 논문에서는 앞서 언급한 삽입, 삭제 갱신 이외에도 엘리먼트 및 에트리뷰트 이동, 엘리먼트 및 에트리뷰트의 새 이름 할당, 엘리먼트, 에트리뷰트 및 텍스트 값 수정과 같은 여러 갱신들에 대한 유효 검증 메커니즘 및 데이타베이스 갱신 프로세스를 설계, 구현, 실험하였다. 이들 갱신들의 유효 검증 및 갱신 처리 과정은 앞서 설명한 삽입·삭제 갱신과 거의 유사하다.

6. 효율적인 스키마 정보 추출을 위한 XML Schema 저장 방법

특정 데이타 노드의 스키마 정보를 추출하는 방법은 XML Schema를 저장하는 방법에 따라 결정된다. 본 논문에서는 XML Schema를 XML 문서와 함께 객체 관계형 데이타베이스 시스템에 저장하였고 특정 노드에 대한 스키마 정보를 효율적으로 추출하기 위한 XML Schema 저장 방법을 제안한다.

6.1 네임스페이스 별 저장 방법

XML Schema 상의 엘리먼트와 에트리뷰트의 선언은 특정 네임스페이스[11]에 종속된다. 즉, 그들의 선언을 포함한 XML Schema의 타겟 네임스페이스가 그에 대응하는 엘리먼트와 에트리뷰트의 유일성을 보장하는 영역이다. 네임스페이스 별 저장 방법은 XML Schema를 타겟 네임스페이스 별로 파일 형태로 저장하는 방법으로 가장 원시적인 방법이다. 정리 1에서 언급한 바와 같이 XML 문서 상의 특정 데이타 노드에 대응되는 선언은 노드 경로를 통해 찾아낼 수 있다. 따라서 특정 노드의 스키마 정보를 추출하기 위해서는 노드 경로상의 노드들에 대한 선언들을 포함하는 스키마 문서들을 차례로 메모리로 가져와서 파스트리(parse tree)를 만들고 이 파스 트리를 전체를 검색함으로써 노드에 대응하는 선언을 추출할 수 있다. 이 방법은 관련된 모든 XML Schema를 모두 메모리에 유지시켜야 하므로 실제 본 논문에서 제안하는 '예상 갱신 부분 유효 검증'에는 불필요한 내용 -갱신과 관련 없는 부분에 대한 스키마 정보들- 까지 메모리에 유지시켜야 하므로 메모리 이용 효율이 낮으며 더불어 불필요한 검색 영역이 크다는 단점이 있다. 일반적으로 스키마의 크기는 데이타에 비해 크지 않다. 또한, 참조하는 스키마가 적은 경우에는 스키마 전체를 메모리에 올려 검색하고 필요한 스키마를 추출하는 성능은 그다지 나쁘지 않다. 하지만 갱신과 관련된 XML Schema의 크기가 커지고 참조하는 XML Schema의 수가 많은 경우, 그리고 다양한 스키마를 기반으로한 데이타에 대한 다수의 갱신 질의가 발생하는 경우에는 갱신 성능과 메모리 이용 효율성은 저하될 것이다.

6.2 타입 별 저장 방법

수많은 종류의 XML 데이타를 동시에 유지하는 XML 데이타 관리 시스템은 그만큼 다양한 종류의 XML Schema를 유지해야 한다. 따라서 언제 접근할지 알 수 없는 XML Schema들을 시스템이 구동 될 때부터 메모리에 유지하는 것은 비효율적이다. 따라서 유효 검증을 위해 필요한 XML Schema만 메모리에 읽어와야 하는데 6.1에서 언급한 바와 같이 XML Schema내에서도 필요한 일부분 만을 가져오는 것이 더 효율적이다. 따라서 이를 위해 스키마 정보를 적절히 테이블로 분할하여 데이타베이스에 저장하고 필요한 순간에 필요한 정보만을 추출해오는 기법이 필요하다.

XML Schema 역시 일종의 XML 문서이므로 지금까지 연구되어온 관계형 혹은 객체 관계형 데이타베이스 관리 시스템을 기반으로한 XML 데이타 저장 방법으로 저장할 수 있다. 그러나 XML Schema는 스키마로서 한정된 엘리먼트와 에트리뷰트만을 사용하여 작성되기 때문에 저장 효율성을 높이고 동시에 효율적으로 특정 데이타 노드에 대한 스키마 정보를 추출하기 위해서는 순수한 XML 데이타 저장 방법으로는 한계가 있다. 따라서 본 논문에서는 '타입 별 저장 방법'을 고안하였다. 그림 3은 타입 별 저장 방법의 저장 스키마 전체를 보여준다. 기본적으로는 엘리먼트 종류에 따라 테이블을 생성하고 각 엘리먼트 별로 종속되어 있는 일부 엘리먼트들과 에트리뷰트들을 해당 엘리먼트 테이블에 인라이닝 시킨다. 예를 들어 complexType 엘리먼트에 대응되는 테이블이 생성되고 name, simpleContent, complexContent와 같은 종속된 엘리먼트는 테이블 에트리뷰트로 포함된다. 그리고 스키마 문서 파싱 과정에서 생성된 효율적인 유효 검증을 위해 필요한 정보(예, 그림 3의 XSLocalElement 테이블 상의 order, GroupList)를 추

```
Table XSNamespaceTable{ NSID Integer; NSURI String; }
Table XSGlobalElement{ NSID Integer; Name String; Type String; }
Table XSLocalElement{ NSID Integer; superType String; Name String; rminC Integer; maxC Integer;
                      order Integer; RefInfo String; Type String; GroupList(String); }
Table XSAttribute{ NSID Integer; superType String; Name String; Type String; UseType Integer; }
Table XSComplexType{ NSID Integer; Name String; IsGlobal Boolean; ContentType Integer; BaseType String; }
Table XSTypeHierarchy{ Type String; BaseTypes List(String); }
```

그림 3 타입 별 저장 방법의 저장 스키마

출하여 별도의 테이블 에트리뷰트에 저장한다. 네임스페이스 정보는 XSNamespaceTable에 저장하며 NSID는 Namespace 식별자, NSURI는 XML Schema의 target Namespace이다. 그리고 스키마 상의 element, complexType, attribute 엘리먼트 별로 테이블(XSGlobalElement, XSLocalElement, XSComplexType, XSAttribute)를 만들었다.

XSLocalElement 테이블에는 특정 complexType에 종속된 element 엘리먼트 선언 정보를 유지하며 테이블 에트리뷰트들을 살펴보면 다음과 같다. superType는 엘리먼트의 선언을 포함하는 상위 ComplexType 정보이며 그 타입의 NSID와 타입의 이름으로 구성된다. Name은 엘리먼트 선언의 name 값, minC/maxC는 mimOccurs/maxOccurs의 값, Type는 type 값(값에 해당되는 타입의 NSID와 이름으로 구성), 그리고 RefInfo는 ref의 값(값에 해당되는 엘리먼트의 NSID와 이름으로 구성)을 갖는다. 또한 order는 스키마 상에서의 형제 엘리먼트들 간의 순서 정보를 유지하며, GroupList는 상위 ComplexType 노드로부터 현재 엘리먼트까지 이르는 그룹 경로를 나타내는 것으로 두 엘리먼트간의 LCG를 체크할 때 유용하게 사용된다. XSGlobalElement 테이블에는 전역 element 엘리먼트 선언 정보를 저장하며 superTypeInfo와 name은 XSLocalElement 테이블과 동일하다.

XSAttribute 테이블에는 특정 complexType에 종속되어 선언된 attribute 엘리먼트 정보를 유지하며 UseType은 에트리뷰트 선언에 포함된 use의 값을 유지하며 나머지 테이블 에트리뷰트들은 XSLocalElement와 동일한 정보를 유지한다.

complexType 엘리먼트 정보는 XSComplexType 테이블에 유지한다. ISGlobal는 complexType 선언이 전역을 선언된 타입인지, 특정 엘리먼트에 종속된 타입인지를 나타낸다. 그리고 contentType은 complexType 엘리먼트가 포함하는 컨텐츠의 속성(simpleContent 혹은 complexContent) 정보가 포함된 것이다. 그리고 BaseType은 상속받은 상위 타입을 나타내는 것으로 타입의 NSID와 이름으로 구성된다.

마지막으로 XML Schema에서는 상속을 지원하므로 타입의 상속 관계를 효율적으로 관리하기 위해 XSTypeHierarchy 테이블을 두었다. 이는 XML Schema 문서 저장 시 유용하게 이용된다. 그림 4는 타입 별 저장 방법에 따라 그림 1의 order.xsd를 저장한 예이다.

6.2.1 타입 별 저장 방법에서의 스키마 추출 방법

타입 별 저장 방법에서의 스키마 추출 역시 정의 4와 정리 1에 의거하여 이루어진다. 일반적으로 관계형 데이타베이스 시스템을 기반으로 한 XML 저장소들은 주어진 노드 경로에 대응하는 엘리먼트 혹은 에트리뷰트를 효율적으로 추출하기 위해 노드 경로와 그에 대응하는 노드의 매핑 정보들을 별도의 테이블을 두어 저장한다. 본 논문에서는 이 테이블을 PATHTABLE이라 하며 기본적으로 노드 경로에 대응하는 Path와 노드 경로 식별자 PathID를 유지한다.

특정 노드의 스키마 정보를 추출하기 위해 PATHTABLE에 ParentTypeInfo라는 에트리뷰트를 첨가한다. ParentTypeInfo은 Path에 대응되는 스키마 컴포넌트를 포함하는 타입에 대한 정보로 타입이 선언된 XML Schema의 NSID와 타입의 이름으로 구성되어 있다. XML 문서가 저장될 때 노드 경로를 추출하여 PATHTABLE에 저장하는 과정에서 노드 경로를 통해 XML Schema를 검색하여 ParentTypeInfo 정보를 생성하고 이를 Path 정보와 함께 PATHTABLE에 저장한다. 따라서 특정 노드에 대한 스키마 컴포넌트는 PATHTABLE에서 그 노드의 노드 경로를 통해 찾은 ParentTypeInfo과 노드 이름을 키(key) 값으로 이용하여 추출 할 수 있다.

그림 5는 노드 경로 '/purchaseOrder/shipTo/state'에 대응되는 state 노드의 스키마 정보를 PATHTABLE를 통해 XSLocalElement로부터 추출하는 과정을 보여준다.

노드 경로상의 노드의 타입에 따라 검색 대상이 되는 테이블이 다르다. 경로의 길이가 1이면 NS와 노드의 이름을 키 값으로 하여 XSGlobalElement 테이블을 검색해야 하며, 경로의 길이가 2이상이면 XSLocalElement 테블을, 경로 상의 노드가 에트리뷰트 노드라면 XS-

XSLocalElement

NSID	superType	Name	MinC	MaxC	order	RefInfo	Type	GroupList
...	...	...	...	...	...	...	...	...
3	3:purchaseOrderType	shipTo	1	1	1		3:USAddress	{s}
3	3:purchaseOrderType	billTo	1	1	2		3:KRAddress	{s}
3	3:purchaseOrderType	items	1	1	3		2:Items	{s}
3	3:USAddress	state	1	1	1		2:USState	{s}
3	3:USAddress	zip	1	1	2		1:positiveInteger	{s}
3	3:KRAddress	state	1	1	1		1:string	{s}

XSGlobalElement

NSID	Name	Type
...	...	...
3	purchaseOrder	3:purchaseOrderType

XSAttribute

NSID	superType	Name	Type	UseType
...	...	...	...	...
3	3:purchaseOrderType	orderDate	1:date	1

XSTypeHierarchy

Type	BaseTypes
...	...
3:USAddress	{2:Address}

XSComplexType

NSID	Type	IsGlobal	ContentType	BaseType
...	...	...	...	...
3	3:purchaseOrderType	1		
3	3:purchaseOrderType	1	1 (= complexContent)	2:Address

XSNamespaceTable

NSID	NSURI
1	http://www.w3.org/2001/XMLSchema
2	http://www.example.com/Address
3	http://www.example/IPO

그림 4 타입 별 저장 방법에 따라 그림 1의 order.xsd를 저장한 예

PATHTABLE

NS	PathID	Path	ParentTypeInfo
...	...	...	...
3	4	/purchaseOrder/shipTo/city	3:USAddress
3	5	/purchaseOrder/shipTo/state	3:USAddress
...	...	...	...

XSLocalElement

NSID	superType	...	Name	...	type	...
...	...	...	...	...	...	...
3	3:USAddress	...	city	...	1:String	...
3	3:USAddress	...	state	...	3:USState	...
3	3:USAddress	...	zip	...	1:positive Integer	...
...	...	...	...	...	...	...

그림 5 PATHTABLE을 이용한 스키마 추출 예

Attribute 테이블을 검색해야 한다. 이렇게 PATHTABLE을 이용하여 한번에 하나의 테이블 검색만으로 필요한 스키마 정보만을 간단하게 추출할 수 있으므로 추출 시간이나 메모리 이용 면에서 네임스페이스 별 저장 방법에 비해 효율적이다. 특히 XML Schema의 크기가 커지고, 노드 경로 상의 노드들이 선언된 XML Schema의 수가 많아질수록 성능 차는 더욱 커질 것이다.

7. 실험 및 성능 분석

5장에서 설명한 예상 갱신 부분 유효성 검증 방법의 효율성을 확인하고 6장에서 언급한 스키마 저장 방법에 따른 갱신에 대한 스키마 유효 검증 성능을 보이기 위

```
Class DOC_ROOT { DID Integer; DocName String; Root NODE; }
Class NODE { NID Integer; NS Integer; DID Integer; Name String; ParentID Integer; PathID Integer; AncestorIDs List(Integer); }
Class ELEMENT under NODE { Attributes Set(ATTRIBUTE); Children List(ELEMENT); Sorder Integer; }
Class ATTRIBUTE under NODE { Tvalue List(String); }
Class TEXT under NODE { Tvalue String; }
Table PATHTABLE { NS Integer; PathID Integer; Path String; ParentTypeInfo String; }
```

그림 6 클래스 스키마

한 실험을 수행하였다.

7.1 XML 데이타 저장 스키마

갱신 방법은 데이타가 저장된 저장 시스템과 저장 스키마에 따라 달라진다. 본 논문에서는 본 연구실에서 객체 관계형 데이타베이스 시스템을 기반으로 개발한 XML 저장 관리 시스템에서 이용한 XML 저장 방법에 따른 갱신 모듈 구현하여 실험을 수행하였다. 그림 6은 XML 데이타 저장 스키마이다. 갱신이 발생하면 관련된 테이블을 갱신하게 된다. 엘리먼트 삽입은 ELEMENT 테이블에 새로운 튜플을 추가하고 부모 엘리먼트의 Children List를 갱신한다. 엘리먼트 삭제는 ELEMENT 테이블에서 삭제되는 엘리먼트에 대응하는 튜플과 그의 모든 자손들을 삭제한다. 이는 AncestorIDs에 삭제되는 엘리먼트의 노드 식별자(NID)를 포함한 튜플들을 삭제하므로써 매우 효율적으로 처리할 수 있다. 에트리뷰트의 삽입 및 삭제는 ATTRIBUTE 테이블에 튜플을 삽입 혹은 삭제하고, 부모 엘리먼트의 Attributes Set을 갱신한다.

7.2 실험 환경 및 데이타

실험에 사용된 시스템은 Pentium 4, 1.70GHz CPU, 512MB 메모리를 갖추었으며 운영체제는 윈도우 XP이다. 객체 관계형 데이타베이스 시스템은 Sun 5.8 Unix 워크스테이션에 설치된 UniSQL을 원격으로 접근하여 사용하였다. 프로그래밍 언어는 자바를 사용하였으며, 데이타베이스 시스템과의 통신을 위해 JDBC를 이용하였다. 실험에 사용된 XML Schema 데이타는 OASIS Committees에서 승인한 비즈니스 문서 스키마 UBL 2.0[12]이고 표 1은 스키마 UBL 2.0에 대한 특성을 보여준다. UBL 2.0은 31개의 비즈니스 종류별 스키마와 이들이 공통적으로 이용하는 타입들을 정의한 10개의 스키마들로 구성되어있다. XML Schema에서는 다른 스키마에 정의되어 있는 타입을 이용하여 새로운 타입을 정의하고 새로운 엘리먼트/에트리뷰트들을 선언 할 수 있다. 표 1에서는 참조해오는 타입이 선언되어 있는 외부 스키마 문서들의 수를 참조도로 표기하였다. 예를 들어, UBL 2.0에 포함된 UBL-Order-2.0.xsd와 같은 비즈니스 문서 스키마는 5개의 외부 스키마에 선언된 타입 혹은 엘리먼트/에트리뷰트들을 이용하여 새로운 타입을 정의하고 엘리먼트/에트리뷰트들을 선언한다.

갱신을 수행할 XML 데이타로는 UBL-Order-2.0.xsd 스키마를 만족하는 10MB 크기의 데이타를 만들어 사용하였다. 갱신 질의로 스키마에 유효한 엘리먼트 및 에트리뷰트 삽입, 삭제, 이동, 수정, 새 이름 할당 갱신 및 스키마에 유효하지 않은 갱신 질의들을 만들어 실험을 수행하였다. 본 실험에서는 타입 별 스키마 저장 방법의 효율성을 검증하기 위해 타입 별 저장 방법에 따라 스키마를 저장하고 갱신을 수행하여 유효 검증 시간과 실제 갱신에 걸리는 시간을 측정하였다. 또한, 네임스페이스 별 저장 방법에 따라 스키마를 저장하여 갱신을 하는 실험을 함께 수행하여 측정된 시간을 비교 분석 하였다.

표 1 실험에 사용된 스키마 정보

	이름	크기	타입 수	선언 수	참조도
공통 스키마	CCTS_CCT_SchemaModule-2.0.xsd	약 1.7MB	651 (개)	1980 (개)	0
	CodeList_CurrencyCode_ISO_7_04.xsd				0
	CodeList_LanguageCode_ISO_7_04.xsd				0
	CodeList_MIMEMediaTypeCode_IANA_7_04.xsd				0
	UBL-CommonAggregateComponents-2.0.xsd				3
	UBL-CommonBasicComponents-2.0.xsd				2
	UBL-CommonExtensionComponents-2.0.xsd				2
	UBL-ExtensionContentDatatype-2.0.xsd				0
	UBL-QualifiedDatatypes-2.0.xsd				1
	UnqualifiedDataTypeSchemaModule-2.0.xsd				4
문서 스키마	UBL-Order-2.0.xsd를 비롯한 31가지	약 1MB 각각 20~50KB	31(개)	938(개)	5

7.2 실험 결과

그림 7은 유효한 엘리먼트 삽입 및 삭제 갱신 수행 결과를 보여준다. 우선 엘리먼트 삽입의 경우, 삽입되는 서브 트리 상의 노드의 수를 증가시키며 실험을 수행하였다. 그래프 상에서는 1개의 엘리먼트 삽입 갱신과 22개의 엘리먼트와 에트리뷰트들로 구성된 서브 트리 삽입 갱신 결과만 보여준다. 괄호 안의 숫자는 서브 트리를 삽입하기 위해 검토해야 할 스키마 문서의 수를 의미한다. 예를 들어, 22개의 요소들로 구성된 서브 트리의 삽입(즉, 그림 7에서 Insertion 22Nodes(7)) 에 대한 유효 검증을 위해서는 총 7개의 스키마 문서를 살펴보아야 한다. 갱신에 총 소요된 시간은 유효 검증을 위해 갱신과 관련된 스키마 정보를 추출하는 시간, 갱신과 관련된 데이타베이스 정보 추출 시간, 유효 검증 시간, 마지막으로 데이타베이스 갱신 시간을 포함한다. 엘리먼트 삽입의 경우 삽입되는 엘리먼트 서브 트리를 탐색하면서 노드 별로 삽입에 대한 유효 검증을 수행하고 삽입을 수행하였기 때문에 전체 유효 검증 시간과 갱신 시간을 분리하지 못하였다. 하지만 알고리즘 상 전체 소요 시간 중 스키마 저장 방법에 따라 차이가 나는 부분은 스키마 정보 추출 시간이며 실험 결과 그림 7에서도 볼 수 있다시피 타입 별 저장 방법(Type)에 따라 XML Schema를 저장하였을 때, 네임스페이스 별 저장 방법(Namespace)에 따라 저장하였을 때보다 전체 소요 시간이 더 짧았으며 이는 타입 별 저장 방법의 경우 스키마 추출 시간이 더 적게 소요되었음을 의미한다. 이 결과는 타입 별 저장 방법의 경우 갱신과 관련된 스키마 정보 추출을 위해 데이타베이스에서 직접 필요한 스키마 정보 만을 가져올 수 있는 반면 네임스페이스 별 저장 방법의 경우에는 삽입되는 엘리먼트의 선언 혹은 타입이 정의된 스키마들을 메모리로 읽어드려 파싱하고 원하는 정보를 추출하기 위해 파스 트리를 탐색해야 하기 때문이다. 본 실험에서는 네임스페이스 별 저장을 위해 로컬 파일 시스템에 각 스키마 문서들을 저장하였다. 따라서 원격지의 데이타베이스 시스템을 접근 해야 하는 타입 별 저장 방법에 비해 스키마 추출이 유리하다. 하지만 특정 엘리먼트가 삽입될 때 엘리먼트의 타입이 선언된 스키마가 메모리에 존재하지 않으면 해당 스키마를 읽어드려 파싱해야 한다. 그러므로 그림 7에서 확인 할 수 있듯이 하나의 엘리먼트를 삽입하는 경우에도 검토해야 하는 스키마 문서의 수에 따라 유효 검증 시간이 차이가 난다. 그림 7에서 엘리먼트 삭제 갱신 결과는 유효 검증 시간(Validation for Deletion)과 실제 삭제 시간(Deletion)을 분리하였다. 실험에서 수행된 삭제 갱신은 엘리먼트 하나를 삭제하는 갱신이며, 갱신과 관련된 스키마 추출 시간은 유효 검증 시간에 포함된다. 실험에 이용된 삭제 갱신과 관련된 스키마 정보를 추출하기 위해서 살펴봐야 할 스키마 수는 단 하나이므로 타입 별 저장 방법과 네임스페이스 별 저장 방법간의 스키마 추출 시간에는 큰 차이가 없었다.

그림 8은 유효한 에트리뷰트 삽입 및 삭제 실험 결과를 보여준다. 그림 8에서는 유효 검증 시간과 실제 갱신 시간을 분리하였다. 이 실험에서도 스키마 저장 방법에 따라 영향을 받은 부분은 스키마 추출 시간이며 이는 유효 검증 시간에 포함된다. 실험에 사용된 에트리뷰트 삽입에 사용된 질의는 삽입되는 에트리뷰트의 수는 단 하나지만 삽입을 위해 참조해야 할 스키마 수는 총 4개인 질의로 타입 별 저장 방법에 따라 스키마를 저장하였을 때 스키마 추출 성능이 더 좋았음을 확인 할 수 있다. 여기서, 참조하는 스키마 수가 4개인 이유는 유효 검증을 위해 에트리뷰트가 삽입되는 위치의 엘리먼트의 타입 및 삽입되는 에트리뷰트의 선언을 추출 해야 하며 이들 타입 및 선언이 이미 정의된 타입을 상속 받은 경우 참조된 엘리먼트 타입 및 에트리뷰트의 선언 역시 검토해야 하는데, 이때 이들이 모두 서로 다른 스키마에 정의된 경우이다. 에트리뷰트의 삭제의 경우에는 유효 검증을 위해 필요한 스키마 정보는 에트리뷰트 선언이며 하나의 스키마에서 추출이 가능한 질의를 수행하였

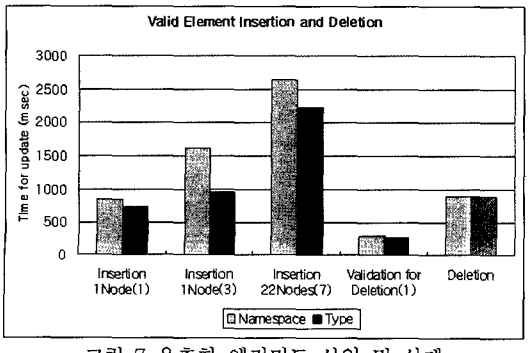

그림 7 유효한 엘리먼트 삽입 및 삭제

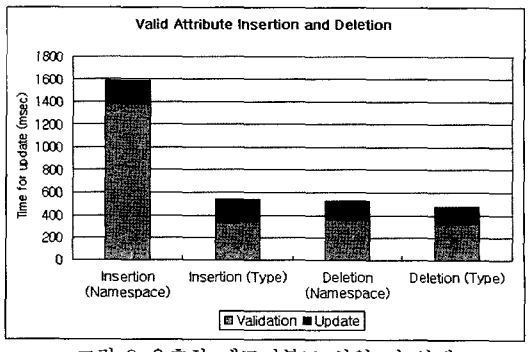

그림 8 유효한 에트리뷰트 삽입 및 삭제

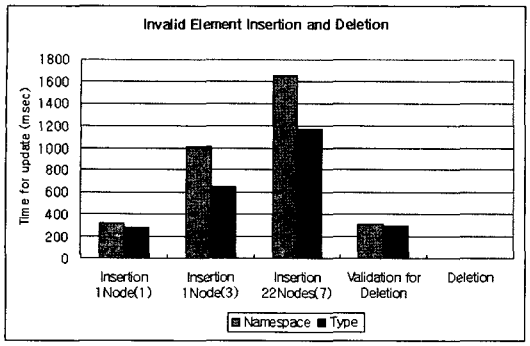

그림 9 유효하지 않은 엘리먼트 삽입 및 삭제

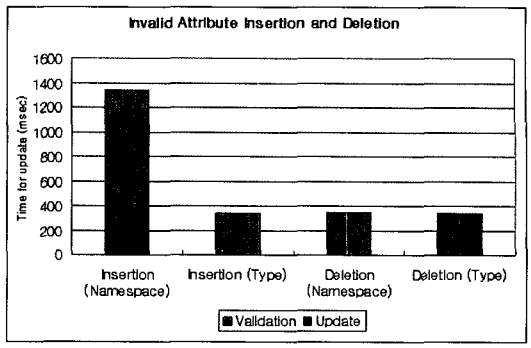

그림 10 유효하지 않은 에트리뷰트 삽입 및 삭제

다. 따라서 스키마 저장 방법 간의 유효 검증 시 요구되는 시간 차이가 거의 없다.

그림 9와 10은 유효하지 않은 엘리먼트의 삽입 및 삭제 갱신과 유효하지 않은 에트리뷰트의 삽입 및 삭제 갱신 질의 수행 결과를 보여준다. 이들 질의의 경우 유효 검증 결과 유효하지 않은 질의로 판명됨으로써 갱신을 수행하지 않는다. 또한, 유효한 갱신이 되기 위한 여러 조건 중 단 하나라도 만족하지 않는 경우 유효하지 않은 갱신으로 판단하고 더 이상의 유효 검증은 진행하지 않고 처리를 중단하게 되므로 전체 수행 시간은 유효한 갱신 처리 시간에 비하여 짧다. 즉, 유효하지 않은 갱신의 경우 갱신을 수행하기 이전에 유효하지 않은 갱신이라 판단하여 갱신을 차단함으로써, 갱신 후 전체 데이타에 대한 유효성을 체크하여 갱신의 유효 여부를 판단하는 방법에 비해 시간을 크게 절약하게 된다.

이외에도 엘리먼트 및 에트리뷰트 이동, 엘리먼트 및 에트리뷰트의 새 이름 할당, 엘리먼트, 에트리뷰트 및 텍스트 값 수정과 같은 여러 갱신들에 대해서도 실험을 수행하였으며 결과 역시 앞서 보인 결과와 거의 유사하였으므로 실험 결과 그래프는 본 논문에서 생략한다.

8. 결 론

본 논문에서는 유효한 XML 데이타 갱신에 대한 효율적인 XML Schema 유효 검증 및 갱신 방법에 대한 연구를 수행하였다. 우선 XML 데이타와 수반된 XML Schema에서 XML 요소와 그에 대한 선언간의 노드 경로를 통한 매핑 방법을 제안하였다. 그리고 갱신이 발생한 노드와 갱신으로 인해 영향을 받는 일부 영역에 대한 유효 제약 조건을 체크 함으로서 유효 검증 제공하는 '예상 갱신 부분 유효 검증 기법'과 갱신 별 유효 검증 알고리즘을 설계하였다. 추가적으로 갱신에 대한 유효 검증 시 갱신과 관련된 스키마를 추출하기 위한 XML Schema 저장 방법으로 '타입 별 저장 방법'을 제안하였고, '타입 별 저장 방법'으로 XML Schema를 저장하였을 때 PATHTABLE 테이블을 활용하여 특정 데이타 노드에 대한 스키마 정보를 효율적으로 추출하는 방법을 제안하였다. 그리고 마지막으로 실험을 통해 본 논문에서 제안한 XML Schema 저장 방법에 따른 유효 검증을 제공하는 갱신 성능을 측정함으로써 '타입 별 저장 방법'이 유효 검증을 위한 효율적인 스키마 추출을 제공하는 스키마 저장 방법임을 확인하였다. 일반적으로 XML Schema 스키마 크기는 데이타만큼 크지 않다. 따라서 스키마 전체를 메모리에 유지시키는데 별 문제가 없고, 스키마를 파싱하여 특정 부분을 검색하는 시간도 짧은 편이다. 따라서 스키마의 참조도가 낮은 스키마를 기반으로한 데이타 갱신, 혹은 동일한 스키마 정보를 요구하는 연속적인 갱신이 발생하는 경우에는 데이타베이스로부터 필요한 스키마 정보만을 추출해오는 이득이 그다지 크지 않을 것이다. 하지만 관리하는 스키마의 종류가 많고, 각기 다른 스키마에 기반한 데이타들에 대한 갱신이 다수 발생하는 경우에는 매 갱신마다 새로운 스키마를 읽어드리거나, 참조할 가능성이 있는 스키마들을 모두 메모리상에 유지하는 것은 갱신 성능 및 메모리 이용 효율을 저하 시키게 된다. 따라서, 본 논문에서 제안하는 유효 검증 기법 및 XML Schema 저장 방법은 수많은 종류의 XML 데이타, 즉, 다양한 스키마를 기반으로 한 데이타들을 동시에 유지 관리하는 XML 데이타관리 시스템에 유용하게 이용될 수 있을 것이다.

참 고 문 헌

[1] Extensible Markup Language(XML) Specification, W3C Recommendation, 04 February 2004, http://www.w3.org/TR/REC-xml

[2] D. Suciu, "Semistructured Data and XML," Proceeding of The 5th International Conference of Foundations of Data Organization, pp.1-12, 1998.

[3] DTD (Data Type Declaration) in XML Specification W3C Recommendation, 04 February 2004,

http://www.w3.org/TR/REC-xml
[4] XML Schema, W3C Recommendation, 02 May 2001, http:// http://www.w3.org/TR/xmlschema-0/
[5] eXcelon Corporation. Updating XML data. eXcelon 3.0 User Guide, 2001.
[6] I. Tatarinov et al, "Updating XML," Proceeding of ACM SIGMOD Conference, 2001.
[7] XQuery: An XML Query Language W3C Working Draft, 23 July 2004, http://www.w3.org/TR/xquery
[8] Oracle XML DB, Oracle9i XML Database Developer's Guide-Oracle XML DB Release2(9.2), October, 2002.
[9] R. Murthy et al, "XML Schemas in Oracle XML DB," Proceeding of VLDB Conference, pp.1009-1018, 2003.
[10] D. Barbosa et al, Efficient Incremental Validation of XML Documents, Proceeding of ICDE, 2004.
[11] Namespace in XML, W3C, http://www.w3.org/TR/REC-xml-names, January 1999.
[12] Universal Business Language-Library, approved as an OASIS Committee Specification, http://docs.oasis-open.org/ubl/cs-UBL-2.0/

이 지 현

2002년 숭실대학교 컴퓨터공학부(학사) 2004년 한국과학기술원 전산학과(석사) 2004년~현재 한국과학기술원 박사과정 재학 중. 관심분야는 데이타베이스, 시멘틱 웹, XML, OWL

박 명 제

2001년 한국과학기술원 전산학과(학사) 2003년 한국과학기술원 전산학과(석사) 2003년~현재 한국과학기술원 박사과정 재학 중. 관심분야는 데이타베이스, 시멘틱 웹, XML, OWL

정 진 완

1973년 서울대학교 공과대학 전기공학과(학사). 1983년 University of Michigan 컴퓨터공학과(박사). 1983년~1993년 미국 GM 연구소 선임연구원 및 책임 연구원. 1993년~현재 한국과학기술원 전산학과 부교수 및 교수. 관심분야는 시멘틱 웹, XML, 멀티미디어 데이타베이스, 스트림 데이타 및 센서 네트워크 데이타베이스