# 갱신 연산을 고려한 XML 문서의 접근제어
## (Access Control of XML Documents Including Update Operators)

임 청 환 † 박 석 ††
(Chung-Hwan Lim) (Seog Park)

**요 약** 웹 상에 많은 정보들이 XML 형태로 표현되면서 XML 보안에 대한 요구가 커지고 있다. 현재까지 XML 보안에 대한 연구는 전자서명이나 암호화 기법을 이용한 통신상의 보안을 중심으로 진행되어 왔다. 하지만 XML 데이타가 방대해지고 복잡해짐에 따라 XML에 대한 통신상의 보안 뿐만 아니라 관리적인 보안까지 필요하게 되었다. 이러한 관리적인 보안은 접근제어를 통해 보장할 수 있는데, 기존의 XML 접근제어 모델에서는 검색 연산만 고려하고 있다. 이러한 모델은 XML 문서나 구조의 변경이 필요한 현실적인 환경에서 사용자의 갱신 질의의 경우에는 접근제어를 할 수 없는 한계점이 있다.

본 논문에서는 검색 연산 뿐만 아니라 갱신 연산까지 지원하는 XML 접근제어 모델과 기법을 제안하고, XML 갱신에 필요한 연산자를 정의한다. 또한 복잡한 접근권한 정보를 체계적으로 관리하고, 다양한 갱신 질의를 효율적으로 처리하기 위해 새로운 액션 타입(action type)을 정의한다. 이러한 액션 타입을 이용하면 DOM 기반의 DTD 검증 과정으로 인한 저장 공간과 검색 비용을 줄일 수 있고, 초기에 불필요한 질의를 여과함으로써, 전체적으로 접근제어의 단계를 줄일 수 있다. 제안한 접근제어 모델은 검색 질의의 경우 액션 타입을 결정하기 위한 약간의 오버헤드(overhead)가 발생하지만, 갱신 질의의 경우에는 기존의 접근제어 모델에 비해 좋은 성능을 보인다.

**키워드** : XML 보안, 접근제어, 액션 타입

***Abstract*** As XML becomes popular as the way of presenting information on the web, how to secure XML data becomes an important issue. So far study on XML security has focused on security of data communications by using digital sign or encryption technology. But, it now requires not just to communicate secure XML data on communication but also to manage query process to access XML data since XML data becomes more complicated and bigger. We can manage XML data queries by access control technique. Right now current XML data access control only deals with read operation. This approach has no option to process update XML queries.

In this paper, we present XML access control model and technique that can support both read and update operations. In this paper, we will propose the operation for XML document update. Also, We will define action type as a new concept to manage authorization information and process update queries. It results in both minimizing access control steps and reducing memory cost. In addition, we can filter queries that have no access rights at the XML data, which it can reduce unnecessary tasks for processing unauthorized query. As a result of the performance evaluation, we show our access control model is proved to be better than other access control model in update query. But it has a little overhead to decide action type in select query.

**Key words** : XML security, access control, action type

## 1. 서 론

XML은 웹 환경에서 데이타를 표현하기 위한 마크업 언어이다. 이러한 XML은 SGML의 복잡성을 제거하고, HTML의 고정된 태그의 한계에서 벗어나 사용자가 문서 구조를 정의할 수 있다. 이와 같은 장점 때문에 W3C에서는 XML을 웹 데이타의 표준으로 제정했다 [1,2]. XML의 표준화 이후 많은 데이타가 XML로 표현되기 시작했고, 현재는 웹 상에 다양한 XML 데이타가 존재하고 있다. 이러한 XML 데이타를 질의할 수 있는 질의어는 XQuery가 표준으로 제정되어 있다. 하지만

· 본 연구는 한국과학재단 목적기초연구(R01-2003-000-10395-0) 지원으로 수행되었음
† 비 회 원 : e 신한 정보기술팀 DBA로 근무
chlim@eshinhan.com
†† 종신회원 : 서강대학교 컴퓨터학부 교수
spark@dblab.sogang.ac.kr
논문접수 : 2003년 12월 22일
심사완료 : 2004년 8월 3일

아직까지는 검색 연산만을 포함하고 있어서, 사용자는 XML의 데이타에 대한 검색만 가능하다. 하지만 사용자는 XML 데이타의 검색 뿐만 아니라 삽입, 삭제, 수정과 같은 갱신 질의까지 요구하고 있다. 따라서 XML을 갱신할 수 있는 연산자를 XQuery에 포함하려는 연구가 활발히 진행되고 있기 때문에, 조만간 XQuery에 갱신 연산자가 포함될 전망이다[3]. 다른 데이타와 마찬가지로 XML 데이타 역시 부적절한 질의로 인해 데이타가 노출되거나 변경되어 훼손되지 않도록 보호되어야 한다[4]. 하지만 수많은 사용자가 존재하는 웹 환경에서 다양한 질의로부터 XML 데이타의 기밀성과 무결성을 보장하는 일은 그리 간단한 문제가 아니다.

본 논문은 갱신 연산을 고려한 접근제어 모델과 기법을 제안한다. 기존의 XML 접근제어 모델은 XML의 검색 연산만을 고려하고 있으므로, 데이타의 기밀성만을 보장하고 있지만 XQuery에 갱신 연산자가 포함된다면, 데이타의 기밀성 뿐만 아니라 무결성까지 보장해야 한다. 먼저, XML 갱신에 필요한 새로운 연산자를 정의한다. 새로운 갱신 연산자가 접근제어 모델에 추가되면 접근권한 정보의 관리가 복잡해지고, 접근제어 과정에서 레이블링 과정과 DTD 검증 과정이 반복적으로 필요하게 된다. 이러한 반복적인 레이블링 과정과 DTD 검증 과정은 XML 파싱 작업과 DOM 트리의 검색 작업 때문에 많은 메모리가 사용되므로, 시스템의 성능 저하를 초래할 수 있다. 이러한 문제점을 해결하기 위해 새로운 액션 타입을 정의했다. 액션 타입을 정의함으로써 사용자와 접근권한 정보의 관리가 용이해졌고, 불필요한 파싱 작업과 DOM 트리의 검색 작업을 제거함으로써 신속한 접근제어가 이루어질 수 있었다.

본 논문의 나머지는 다음과 같이 구성되어 있다. 2장에는 접근제어에 관련된 연구들을 살펴본다. 3장에서는 연구목표와 제안하는 접근제어 모델을 위한 가정과 환경을 설명한다. 그리고 XML 갱신 연산자와 액션 타입을 정의하고, 액션 타입을 이용한 새로운 DTD 검증 방법을 소개한다. 그런 후 제안하는 접근제어 기법의 알고리즘과 예제를 살펴보고, 전체 시스템 구조를 설명한다. 4장에서는 제안하는 접근제어 모델에 대한 평가와 성능 분석을 하고, 5장에서는 결론과 추후 연구 과제를 밝힌다.

## 2. 관련 연구

데이타 보호를 위한 접근제어에 대한 연구는 오래 전부터 데이타베이스 분야에서 연구되어 왔으며[5-7], 웹 환경이 빠르게 발달됨에 따라, XML 데이타의 접근제어에 대한 연구들이 진행되고 있다[8,9]. 이 장에서는 기존의 접근제어에 대한 연구와 XML의 특징을 고려한 접근제어에 관한 연구를 살펴본다.

### 2.1 일반적인 접근제어

컴퓨터의 시스템이 다수의 사용자에게 다수의 응용(application)을 제공하는 특성을 갖게 되면서 데이타 보안에 대한 관심이 높아지게 되었다. 접근제어란 데이타 보안을 위해 주체(subject: 사용자 혹은 프로세스 등)가 컴퓨터 내의 객체(object: 데이타 혹은 프로그램 등)에 대한 어떤 연산(판독, 기록, 실행 등)을 할 수 있는 능력을 가능하게 하거나 제한할 수 있는 수단이다[6,7]. 사용자는 컴퓨터 내의 정보나 자원에 접근 요청을 하면, 접근제어 규칙에 의해 접근을 거부할지 혹은 허용할지를 결정하고, 필요에 따라서는 요청에 대한 수정을 요구하기도 한다. 접근제어 규칙은 권한 정보, 보안 정책, 권한 부여 규칙에 의해 결정되고, 시스템 환경에 따라 다른 요인들이 포함된다. 이러한 접근제어를 통해서 비 인증된 사용자의 자원에 대한 접근을 제어하여, 컴퓨터내의 자원의 기밀성과 무결성을 보장할 수 있다.

### 2.2 XML 문서의 접근제어

일반적인 파일 구조는 파일 일부분의 접근이 복잡한 구조이기 때문에, 파일 일부분에 해당하는 권한을 부여하기가 어렵다. 그렇기 때문에 접근제어가 전체 파일 단위로 이루어진다. 하지만 XML은 트리 형태의 계층 구조로 이루어져 있다[4,9]. 하나의 XML 문서는 여러 엘리먼트들로 구성되어 있다. 각각의 엘리먼트는 트리의 노드에 해당하고, 노드들은 XML 데이타의 일부분을 표현하고 있기 때문에 XML에서는 엘리먼트에 권한을 부여함으로써, 데이타 일부분에 대한 접근제어가 가능하게 되었다.

• DOM 기반의 XML 접근제어

XML DOM 트리는 XML 문서의 엘리먼트에 접근할 수 있는 API(Application Program Interface)를 제공한다. [8,9]에서 제안한 접근제어 모델은 이러한 DOM 트리를 이용하여 XML 문서와 DTD의 엘리먼트에 접근권한을 설정하고, 설정된 접근권한 정보에 의해 사용자의 XML 데이타 접근을 제어한다. 이러한 접근제어 모델을 이용한 접근제어 과정은 그림 1과 같다.

예를 들어 회사의 사원 정보가 XML 문서로 되어 있고, 접근권한 정보에 일반 사원은 다른 사원의 정보를 볼 수 없도록 권한이 설정되어 있다고 가정하자. 접근권한 정보는 XAS(XML Access Sheets) 문서에 명시되어 있다. 일반 사원이 사원 정보 문서를 보기 위해 접근 요청을 하면, 시스템은 그림 1의 하단부와 같은 작업을 수행한다. 먼저 DOM 트리를 얻기 위해 XML 문서를 파싱(parsing)한다. 그런 후 DTD와 XML문서의 XAS에 기초하여 DOM 트리의 노드에 접근 허가(+) 혹은 거부(-)를 의미하는 sign값을 설정하게 된다. 이렇게 DOM 트리의 노드에 접근권한을 설정하는 작업을 레이

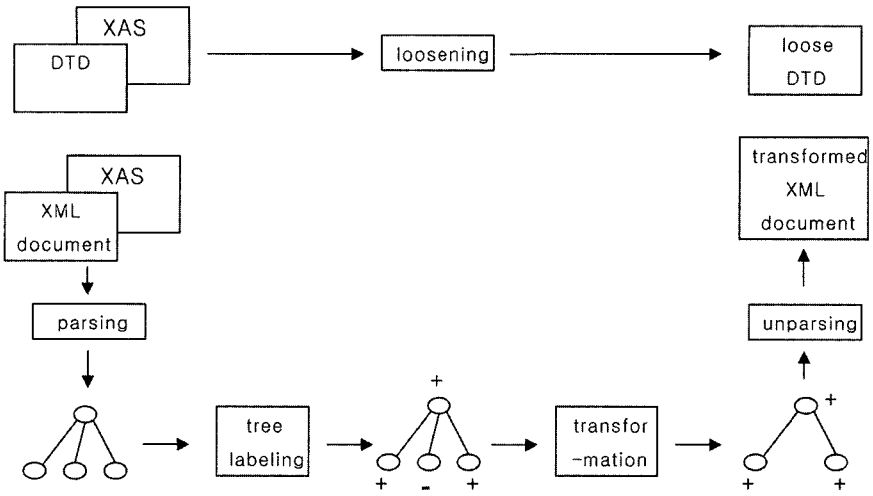



그림 1 접근제어 과정

블링이라 한다. 사원 정보 문서에서는 접속한 사원에 해당하는 엘리먼트는 + 로 설정되고, 다른 사원에 해당하는 엘리먼트는 - 로 설정된다. 최종적으로 레이블링 된 DOM 트리에서 sing 값이 - 로 설정된 노드는 제거되고, + 로 설정된 노드만을 XML 형식으로 사원에게 보여준다. 사원이 보는 문서는 전체 문서의 일부분이라는 점에서는 뷰(view)와 같지만, 미리 정의되어 있지 않고, 별도의 저장소에 저장되지 않는다는 점에서는 다르다. DOM 트리에서 노드가 제거된 XML 문서는 DTD에 유효(valid)하지 않을 수 있는 문제가 발생하는데, 이를 해결하기 위해 그림 1의 상단부와 같은 루즈닝(loosening) 과정이 필요하다. 이는 DTD에 설정된 모든 엘리먼트와 속성을 optional로 설정한다. 따라서 DOM 트리에서 노드가 제거되더라도 기존의 DTD를 유지할 수 있게 된다. 이러한 접근제어 과정을 통해 인증된 사용자는 XML 문서 중에서 권한이 있는 부분만 보게 되고, 비 인증된 사용자는 문서를 볼 수 없게 되므로 데이타의 기밀성이 보장될 수 있다. 하지만 이 모델은 read 연산자만 지원하고 있기 때문에, XML 갱신 연산으로 인해 발생하는 무결성 문제는 고려되지 않고 있다. 또한 전체의 DOM 트리가 메모리에 적재되어야 하고, DOM 트리의 모든 노드에 접근권한을 설정하기 위한 반복적인 트리의 검색으로 많은 메모리가 사용되므로, 시스템의 성능 저하를 초래할 수 있는 문제점이 있다.

## 3. 갱신 연산을 고려한 XML 문서의 접근제어 모델과 기법

### 3.1 연구의 목표

본 논문은 갱신 연산이 필요한 웹 환경에서 활용될 수 있는 접근제어 모델과 기법을 제안한다.

• XML 갱신 연산자의 정의

XML이 보편화됨에 따라 사용자들은 웹 상에 존재하는 XML 데이타에 검색 질의 뿐만 아니라 삽입, 삭제, 수정과 같은 갱신 질의까지 요구하고 있다. 또한 미러링(mirroring) 시스템과 같은 분산 환경에서는 데이타의 일관성을 유지하기 위해 갱신 질의가 필수적이다. 하지만 현재 XQuery에는 XML의 갱신에 대한 연산자가 포함되어 있지 않다. 따라서 본 논문은 갱신 질의를 지원하는 XML 접근제어 모델에 필요한 XML 갱신 연산자를 정의한다.

• 성능 개선

XML 접근제어 모델에 갱신 연산자를 추가하면, 반복적인 레이블링 과정이 필요하게 된다. 보안 관리자는 연산자 단위로 접근권한을 설정하기 때문에, 질의문에 포함된 연산자의 종류에 따라 레이블링 결과 값은 달라지게 된다. 따라서 동일한 사용자라 하더라도 XML 문서에 수행하고자 하는 연산이 바뀔 때마다 레이블링 과정이 반복적으로 필요하게 된다. 이러한 반복적인 레이블링 과정으로 인한 트리 검색 비용은 새로운 연산자가 접근제어 모델에 계속 추가될 수록 증가할 것이다. 또한 갱신 질의로 인해 XML 문서 구조의 변경 여부를 확인하기 위한 DTD 검증 과정이 필요하게 된다. 기존의 DTD 검증 방법에서는 갱신 질의로 인해 변경 될 XML 문서의 DOM 트리를 생성하기 위해 파싱 작업을 수행하고 새로 생성된 DOM 트리의 전 노드를 DTD와 비교하기 위해 방문한다. 만약 XML 구조 변경을 허용하지 않는 환경에서, 갱신 질의 수행 후 XML 문서가 DTD에 유효하지 않다면, 접근을 요청하는 갱신 질의는

거부되어야 하고, 갱신 질의 수행 전의 DOM 트리를 이용하여 나머지 접근제어 과정을 수행해야 한다. 하지만 갱신 질의 수행 후의 XML 문서가 DTD에 유효하다면, 새로 생성된 DOM 트리를 이용하여 나머지 접근제어 과정을 수행한다. DTD 검증 과정은 메모리에 두 개의 DOM 트리를 유지해야 하므로 많은 메모리의 공간이 필요하다. 또한 파싱 작업과 DOM 트리의 검색 작업으로 접근제어의 속도를 저하시킨다. 이러한 문제점들을 해결하기 위해 본 논문에서 새로운 타입의 데이타를 정의한다.

### 3.2 가정 및 환경

- 갱신 연산자가 포함된 XQuery의 표준화 작업이 진행되고 있다[10].
- XML의 엘리먼트 사이에 의미적 종속성은 없다.
- 사용자의 갱신 질의로 인해 XML 문서의 내용과 구조가 변경될 수 있다.
- 검색 질의보다 갱신 질의가 빈번하게 발생한다.

만약 엘리먼트 사이에 의미적 종속성 존재한다면, 갱신 질의로 인하여 엘리먼트 사이의 의미적 불일치가 발생할 수 있다. 예를 들어 은행의 계좌 정보가 XML 문서로 되어 있고, 잔액 엘리먼트는 입금 엘리먼트와 출금 엘리먼트를 자식 엘리먼트로 가지고 있다고 가정하자. 잔액은 입금과 출금의 가감연산으로 계산된다. 그러므로 입금이나 출금 엘리먼트의 값이 변하면 잔액 엘리먼트의 값도 변하게 된다. 또한 잔액 엘리먼트는 입금 엘리먼트나 출금 엘리먼트 중 하나가 삭제되거나 삽입되면 은행 업무에서의 잔액의 의미는 사라지게 된다. 그러므로 의미적인 종속성을 유지하기 위한 별도의 작업들이 필요하다. 이는 본 논문의 범위에 벗어나기 때문에 고려하지 않는다.

### 3.3 접근제어 모델

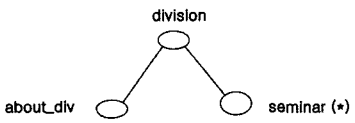
3.3.1. 갱신 연산자의 종류

XML 갱신 연산자의 종류는 [10]로부터 원용하고, 의미는 재정의한다. XML의 갱신 연산자들의 종류와 의미는 다음과 같다.

- Insert (content)
- InsertBefore | After (ref, content)
- Delete (child)
- Replace (child, content)
- Rename (child, name)

Insert는 삽입 연산자로, content에는 PCDATA, 엘리먼트, 속성이 올 수 있다. 만약 XML에 순서 정보가 있다면, InsertBefore 혹은 InsertAfter를 이용한다. InsertBefore는 ref에 지정된 엘리먼트 앞에, InsertAfter는 뒤에 삽입한다. Delete는 삭제 연산자로, child에는 PCDATA, 엘리먼트, 속성이 올 수 있다. Replace는 대체 연산자로, child에는 DTD에 정의된 속성이나 PCDATA가 포함된 단말 엘리먼트가 올 수 있다. Rename은 이름을 바꾸는 연산자로, child에는 엘리먼트, 속성이 올 수 있고, name에는 DTD에 |(or)에 의해 명시된 이름 값이나 새로운 이름 값이 올 수 있다. 이렇게 정의된 XML 갱신 연산자를 접근제어 모델에 적용하기 위해서는 연산자의 의미적 구별이 필요하다. 왜냐하면 동일한 갱신 연산자라 하더라도 권한의 검사범위와 연산의 수행을 위한 작업이 달라질 수 있기 때문이다. 그림 2는 그림 5의 SEC.XML의 DTD 그래프의 일부와 동일한 갱신 연산자가 서로 다른 의미를 가지고 있는 질의문의 예를 보여주고 있다.

위의 DTD 그래프는 세 개의 엘리먼트로 구성되어 있다. division 과 about_div 엘리먼트는 오직 한 개만 존재할 수 있지만, seminar 엘리먼트는 여러 개 존재할 수 있다. 오른편의 질의문은 about_div와 seminar 엘리먼트를 삭제하라는 내용이다[10]. About_div 엘리먼트의 삭제는 DTD에 위배되므로, 삭제 시 XML의 구조 변경을 초래한다. 하지만 seminar 엘리먼트의 삭제는 DTD에 유효하므로, 삭제되어도 XML 구조는 변경되지 않고 오직 DOM 트리의 변경만 이루어진다. 따라서, 첫 번째 delete 연산에 대한 접근제어 과정에는 구조변경에 대한 권한 검사가 포함되어야 한다. 만약 사용자가 XML 구조변경 권한이 있다면, DTD를 변경하는 작업까지 포함되어야 한다. 반면에 두 번째 delete 연산에 대한 접근제어 과정에는 XML 문서 변경에 대한 권한 검사만 포함되면 된다. 왜냐하면 이 연산은 DTD의 변경을 초래하지 않기 때문에 DTD 변경에 대한 권한 검사는 불필요하기 때문이다. 이와 같이 동일한 연산자라도 접근제어 과정이 달라질 수 있기 때문에 구별이 필요하다.




```
FOR $T1 IN document( "sec.xml")/division
    $AB IN $AB/about_div
    $SE IN $AB/seminar
UPDATE
    DELETE $AB
    DELETE $SE
```

그림 2 DTD 그래프와 갱신 질의문의 예

3.3.2. 액션 타입(Action Type)

• 액션 타입의 정의

접근제어 모델에 새로운 갱신 연산자가 추가되면 해당 연산자의 관련정보 또한 접근권한 정보에 포함되어야 하므로 접근권한 정보가 복잡해진다. 또한 레이블링 과정과 DTD 검증 과정에서 DOM 트리의 반복적인 검색과 파싱 작업 때문에 시스템의 성능을 저하시킨다. 이러한 관리적인 문제점과 성능상의 문제점들을 해결하고, 갱신 연산자의 의미적 구별을 위해 액션 타입을 정의한다. 액션 타입은 연산자들이 XML을 어느 수준까지 변경하는지를 기준으로 분류된 연산자들의 집합으로 다음과 같이 네 개의 타입으로 분류한다.

- R 타입 : XML에 어떠한 변경도 하지 않는 연산자들의 집합. 따라서 부가적인 작업이 필요 없다. Read 연산자가 속한다.
- U 타입 : XML 문서의 변경만 초래하는 연산자들의 집합. 새로운 DOM 트리를 얻기 위한 파싱 작업이 필요하다. Replace, rename, insert, delete 연산자가 속한다.
- D 타입 : XML 문서 뿐만 아니라 구조의 변경까지 초래하는 연산자들의 집합. DTD 변경 작업과 파싱 작업이 필요하다. Rename, insert, delete 연산자가 속한다.
- E 타입 : 예외 타입으로 연산자 단위의 접근제어를 보장하기 위해 필요하다. U 타입과 D 타입에 해당하는 연산자들이 속한다.

이러한 액션 타입은 그림 3과 같은 계층구조로 되어 있다. 상위 계층의 타입 권한은 하위 계층의 타입의 권한을 포함하고 있다. 가령, LIM이라는 사용자가 XPath로 지정된 XML 엘리먼트에 D 타입의 연산자를 수행할 권한이 있다면, LIM은 같은 엘리먼트에 U 타입과 R 타입에 해당하는 연산자를 수행할 권한이 있게 된다.

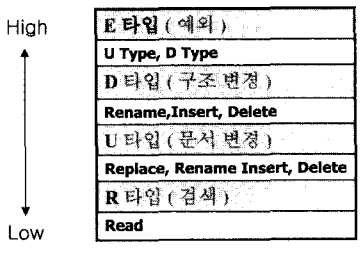



그림 3 ATH (Action Type Hierarchy)

Rename, insert, delete 연산자는 U 타입에도 속해 있고 D 타입에도 속해 있는데, 이들의 연산자는 DTD와 XML의 엘리먼트 개수에 따라 액션 타입이 변할 수 있기 때문이다. 위의 연산자들이 U 타입인지, D 타입인지에 따라서 접근제어 과정에서 권한 검사의 범위가 달라지고, 질의를 처리하기 위한 절차 역시 달라진다. 그러므로 위의 연산자들의 액션 타입이 명확하게 구별되어야 한다. 이러한 액션 타입 구별은 기존의 DOM 기반의 DTD 검증 과정을 통해서 해결할 수 있다. 만약 rename, insert, delete 연산자가 수행된다면 XML 문서는 변경된다. 이 때 변경된 XML 문서가 DTD에 유효하다면 U 타입이고, 유효하지 않다면 D 타입이 된다. 하지만 이러한 방법은 연구목표에서 언급한 것처럼 메모리의 사용량이 많다는 문제점이 있으므로 새로운 방법으로 액션 타입을 구별한다.

• 명시적인 액션 타입의 구별

Insert, delete, rename 연산자의 액션 타입은 DTD와 XML의 엘리먼트 개수에 따라 결정된다. 표 1은 DTD에서 엘리먼트가 가질 수 있는 기호와 각 기호의 경우에 해당하는 액션 타입을 나타내고 있다. DTD에서 엘리먼트에 아무런 기호가 없는 경우, 즉 DTD와 XML 문서의 노드가 1:1 관계인 경우는 모든 갱신 연산자 모두 D 타입으로 결정된다.

표 1 DTD 기호에 의한 연산자 액션 타입 설정

| DTD | None (1:1) | ? | + | * | \| |
|---|---|---|---|---|---|
| Insert | D | U or D | U | U | D |
| Delete | D | U | U or D | U | D |
| Rename | D | D | D | D | U or D |

? 기호의 insert와 + 기호의 delete의 경우는 XML 문서의 엘리먼트 개수에 따라 U 타입도 될 수 있고, D 타입도 될 수 있다. 예를 들어 DTD에 name 엘리먼트는 ? 기호로 명시 되어있고, XML 문서에 name 엘리먼트가 존재하지 않는다면 name 엘리먼트를 삽입하려는 질의는 U 타입으로 결정된다. 왜냐하면 삽입 질의를 하더라도 DTD의 ? 기호, 즉 0 또는 1의 조건을 만족하므로, DTD에 유효하다. 하지만 이미 name 엘리먼트가 존재하고 있다면 삽입 질의 후의 name 엘리먼트는 두 개가 존재하게 되므로, DTD를 위배하게 된다. 따라서 D 타입으로 결정된다. | 기호는 rename 연산자의 타입 결정에 영향을 미친다. 즉, 재명명 하려는 이름이 DTD에 명시된 이름이면 U 타입으로 결정되고, 새로운 이름이라면 D 타입으로 결정된다. 결과적으로 ? 기호의 insert 연산자, + 기호의 delete 연산자, | 기호의 rename 연산자를 제외한 경우는 DTD에 의해 명시적으로 타입이 결정된다. 위의 세 가지 경우의 액션 타입을 결정하기 위해서는 DTD 뿐만 아니라 XML의 문서까지 참조해야 한다. 만약 XML 문서에 대한 부가정보가 있다면, XML 문서를 참조하기 위해 매번 DOM 트리를

검색 할 필요가 없다. Insert, delete 연산자의 경우, XML 문서의 엘리먼트 중에서 ? 와 + 의 기호가 붙어 있는 엘리먼트의 수를 알고 있다면, 간단한 알고리즘에 의해 액션 타입을 구별할 수 있다. 또한 rename 연산자의 경우, | 연산자에 명시된 이름을 알고 있다면, 역시 간단한 알고리즘에 의해 액션 타입을 구별할 수 있다. XML 문서의 엘리먼트 수는 최초 파싱 과정에서 추출하여 유지할 수 있고, | 연산자에 명시된 이름은 DTD를 참조하여 < | (or) 엘리먼트의 XPath, NameSet > 과 같은 형식으로 접근권한 정보에 유지할 수 있다. 다음은 U 타입도 될 수 있고, D 타입도 될 수 있는 세 가지 경우, 액션 타입을 구별하는 방법이다.

- 기호에서의 insert 연산자의 타입 구별
  IF Opt + 1 > 1 THEN D 타입
  ELSE U 타입 (Opt는 ? (optional) 엘리먼트의 수)
- + 기호에서의 delete 연산자의 타입 구별
  IF Man - 1 < 1 THEN D 타입
  ELSE D 타입 (Man은 + (mandatory) 엘리먼트의 수)
- | 기호에서의 rename 연산자의 타입 구별
  IF Name ∈ NameSet THEN U 타입
  ELSE D 타입 (Name은 재명명 하려는 이름)

이와 같은 방법으로 XML 문서의 엘리먼트에 액션 타입이 결정되면 DOM 트리를 생성하지 않고, 갱신 질의의 연산자를 구별할 수 있다. 예를 들어 XML 문서에서 speaker 노드의 개수와, member 노드의 개수가 0이라면, 그림 5의 SEC.DTD의 경우 표 2같은 액션 타입 결정표를 얻을 수 있다.

표 2 액션 타입 결정표

| DTD | Insert | Delete | Rename |
|---|---|---|---|
| division | D | D | D |
| about_div | D | D | D |
| seminar (*) | U | U | D |
| address | D | D | D |
| member (+) | U | D | D |
| contact | D | D | D |
| title | D | D | D |
| speaker (?) | U | U | D |

표 2를 통해 member 엘리먼트에 새로운 엘리먼트를 삽입하려는 질의는 U 타입이고, member 엘리먼트를 삭제하려는 질의는 D 타입임을 알 수 있다. 만약 XML의 구조 변경을 허용하는 환경이라면, member를 삭제하려는 질의에 대해서는 사용자가 구조 변경을 할 수 있는지에 대한 권한 검사와 DTD 변경 작업이 접근제어 과정에 포함되어야 한다. 이러한 액션 타입 결정표가 없다면, 접근제어 권한과 과정을 결정하기 위해 매 질의마다 DTD와 XML 문서 참조를 위한 파싱 작업과 트리 검색 작업을 반복해야 한다

3.3.3 접근권한 정보

갱신 연산을 고려한 XML 문서의 접근제어 모델은 아래와 같은 여섯 개의 구성 요소로 이루어져 있다.

- Subject : 사용자 이름 또는 IP 주소 또는 컴퓨터 이름
- Object : XPath 1.0
- Action : Read, Insert, Delete, Replace, Rename
- Action Type : R, D, U, E
- Sign : + / -
- Type : LDH, RDH, L, R, LD, RD, LS, RS

Subject는 XML 문서에 접근하는 주체이다. 이들의 주체는 사용자의 그룹과 패턴을 지원한다. Object는 XML 문서의 엘리먼트이며, 이는 XPath로 표현된다. Action은 주체가 수행할 수 있는 연산이고, Action Type은 연산자들의 그룹이다. Sign은 권한의 허가 혹은 거부에 대한 표현이며, Type은 권한의 속성 값을 의미한다. 이 모델에서 접근권한 정보는 다음과 같은 형식으로 기술된다.

<Subject, Object, Action, Action Mode, Propagation, Option>

Action mode는 U+와 같이 액션 타입과 Sign의 조합으로 이루어져 있다. 그리고, Propagation은 권한의 전파 여부를 결정하는 정보로 R(recursive) 혹은 L(local)이 설정될 수 있다. 또한 Option은 접근권한의 우선순위 나타낸다. 위의 형식에서 Object, Type, Option에 명시된 값들을 보고, 접근제어 모델의 Type을 알 수 있다. 예를 들어 SEC.XML Object의 Propagation 은 R, Option은 Soft로 설정되어 있다면 Object의 Type 값은 RS가 된다. 접근권한 정보는 오직 보안 관리자에 의해서만 작성된다. 보안 관리자는 최초 접근권한 설정 시각 연산자의 액션 타입을 접근권한 정보에 명시해야 한다. 이때 액션 타입 결정표가 참조된다.

3.3.4 전파 규칙

XML은 계층적인 트리 구조이기 때문에 상위 노드의 권한이 하위 노드의 권한에 영향을 미칠 수 있고, 동일한 사용자라 하더라도, 속해있는 그룹, IP 주소, 컴퓨터 이름에 따라서 권한이 달라질 수 있다. 또한 권한이 XML 문서와 DTD 동시에 설정된다. 이러한 이유로 인해 보안 관리자가 정의해 놓은 접근권한 정보에서 동일한 노드에 서로 다른 권한이 충돌할 경우가 발생한다 [11]. 최종적으로 하나의 노드에는 하나의 권한만이 설정될 수 있기 때문에 이러한 충돌 문제를 해결해야 한다. 전파 규칙은 이와 같이 권한 충돌이 발생할 때 권한의 우선 순위를 결정한다. 권한의 충돌이 발생할 수 있는 모든 경우는 subject의 정보, 액션 타입의 부호,

type, ATH의 위치, 권한정보의 대상에 따라 아래와 같은 다섯 가지 경우로 분류될 수 있다. 다음은 권한 충돌이 발생하는 다섯 가지의 경우, 권한의 우선순위를 결정하는 전파규칙이다.

- 규칙 1 : 서로 다른 subject의 권한정보 충돌 시, 상세한 subject 정보가 명시되어 있는 권한정보에 우선순위가 높다.
  <LIM,*,*>, U- <==> <LIM,163,239,*> U+ : U+
- 규칙 2 : XML 권한정보와 DTD 권한정보의 충돌 시, XML 문서의 권한정보가 DTD 권 한정보 보다 우선순위가 높다. 단 DTD 권한정보에서 type이 hard로 설정되어 있는 경우, DTD의 권한 정보가 우선순위가 높다.
  R-, LD <==> R+, L : R+ R-, LDH <==> R+, L : R-
- 규칙 3 : 서로 다른 sign 충돌 시, 액션 타입이 같으면 - 가 우선하고, 다르면 ATH 낮 은 액션 타입이 우선순위가 높다.
  U+ <==> U- : U- R+ <==> U- : R+
- 규칙 4 : + 의 서로 다른 액션 타입의 sign 충돌 시, ATH가 높은 액션 타입이 우선 순위가 높다. 만약 이때 액션 타입이 사용자의 권한 등급보다 높아 질 경우에 는 ATH가 낮은 액션 타입이 우선순위가 높다. - 의 충돌의 경우는 ATH가 낮 은 액션 타입이 우선순위가 높다.
  R+ <==> U+ : U+ R- <==> U- : R-
- 규칙 5 : E 타입과 다른 액션 타입 충돌 시, sign이 같을 경우 E 타입은 우선순위가 낮고, sign이 다를 경우 우선순위가 높다.
  UE-(replace) <==> U- : U- UE(+replace) <==> U- : UE+(replace)

3.3.5 레이블링 알고리즘

레이블링은 보안 관리자가 정의한 접근권한 정보를 이용하여, 질의에서 접근을 요청하는 DOM 트리의 노드에 권한을 설정하는 과정이다[8,9]. 이렇게 레이블링 된 권한정보는 사용자의 질의를 거부할지 혹은 승인할지 결정하는데 사용된다. 기존의 레이블링 과정에서는 연산자 단위로 레이블링이 이루어지기 때문에, 질의문에 포함된 연산자의 종류 수 만큼 레이블링 과정을 반복해야 했다. 이러한 반복적인 레이블링을 제거하기 위해 액션 타입 단위로 레이블링하는 알고리즘을 제안한다. 액션 타입 단위로 레이블링을 하게 되면 연산자의 종류에 관계없이 결과값이 동일하므로, 오직 한번의 레이블링 과정만 필요하게 된다. 보안 관리자는 액션 타입의 E 타입을 이용해서, 기존의 연산자 단위의 레이블링 수준까지 보장할 수 있다. 전파규칙에서 XML 문서의 접근권한 정보는 DTD의 접근권한 정보보다 우선순위가 높기 때문에 레이블링 과정에서 이들의 구별이 필요하다. 이를 위해 전체 접근제어 정보에서 XML 문서에 해당하는 접근권한을 A_xml에, DTD에 해당하는 접근권한을 A_dtd로 분리하여 유지한다. 보안 관리자가 특정 엘리먼트에 명시적으로 권한을 설정하지 않았다면, 미리 정해진 기본 접근권한이 할당된다. 그러므로 모든 엘리먼트에는 액션 모드가 존재하게 된다. 그림 4의 상단부는 알고리즘에서 사용되는 변수와 함수에 관한 설명이고, 하단부는 액션 타입을 이용한 레이블링 알고리즘이다.

3.3.6 접근제어 기법

- A_xml := { a ∈ Auth | rq ≤ subject(a), uri(object(a)) = URI}
- A_dtd := { a ∈ Auth | rq ≤ subject(a), uri(object(a)) = dtd(URI)}
- r 은 URI 문서의 루트 노드, n은 r을 제외한 노드, p는 n의 부모노드
- AM( ) : Authorization rule에 명시된 노드의 action mode 반환, default( ) : 노드의 기본 액션모드 반환 , type( ) 은 Authorzation rule에 명시된 type 값 반환, propagation_rule( ) : 전파규칙에 의해 결정된 액션모드 반환

```
1.label (r)
2.For each c ∈ children (r) do label (n, p)
Procedure label (r)
  L1r = AM (r) in A_dtd
  L2r = AM (r) in A_xml
  if L1r U L2r = { }, Lr = default (r)
  else Lr = propagation_rule ([L1r , L2r])
```

```
Procedure lable (n,p)
  L1n = AM (n) in A_dtd
  L2n = AM (n) in A_xml
  if type (p) in {R, RD, RDH, RS}
     if L1n & L2n = { }, Ln = Lp
     else Ln = propagation_rule([Lp, L1n, L2n])
  else
     if L1n & L2n = { }, Ln = default (n)
     else Ln = propagation_rule([L1n, L2n])
```

그림 4 레이블링 알고리즘

지금까지의 XML 접근제어 기법은 DOM 트리의 레이블링 과정을 마친 후, 질의에 대한 접근을 허가할지 혹은 거부할지 결정했다. 그렇기 때문에 권한 검사에 필요한 모든 정보를 마지막까지 유지해야 하고, 이로 인해 불필요한 권한 검사가 수행되었다[8,9]. 이러한 불필요한 작업은 접근제어 속도를 느리게 한다. 이러한 문제를 해결하고자 제안하는 접근제어 기법에서는 접근제어 과정을 2 단계로 나누었다. 접근제어 과정을 2 단계로 나누면 권한에 위배되는 질의문은 초기에 제거되기 때문에, 거부될 질의문에 대한 추가적인 작업이 필요 없게 된다.

• 1 단계 접근제어

1 단계 접근제어 단계에서는 사용자의 권한 등급에 위배되는 질의의 연산자를 우선적으로 제거한다. 사용자의 권한 등급은 사용자가 XML 문서에 접근 요청을 하면 접근권한 정보에 의해 할당한다. 만약 할당받은 사용자의 권한 등급이 U 타입이라면, 이 사용자는 U 타입보다 ATH가 높은 D 타입의 질의를 할 수 없다. 즉, XML의 구조 변경 권한이 없는 사용자가 XML의 구조 변경을 초래하는 질의를 요청하면 이 질의는 사전에 제거된다. 이로서 불필요한 권한 검사 및 작업을 제거할 수 있다.

• 2 단계 접근제어

1 단계 접근제어 과정에서 승인된 질의는 2 단계 접근제어 과정에서 최종 권한 검사를 통해 질의의 수행 여부가 결정된다. 1 단계 접근제어 과정에서는 XML 전체 문서 단위로 권한 검사가 이루어지는 반면에, 2 단계 접근제어 과정에서는 XML 문서의 엘리먼트 단위로 권한 검사가 이루어진다. 질의문에서 요청하는 엘리먼트에 접근권한이 검사하기 위해 레이블링 된 DOM 트리를 검색해야 한다. 이때 1 단계 접근제어 과정에서 권한이 없는 질의들은 사전에 제거되었기 때문에 이들을 위한 트리 검색은 할 필요가 없다.

• 접근제어 기법의 예

그림 5와 같은 XML 문서와 DTD, 그리고 접근권한 정보가 있다고 가정하자.

```
<division name = "Dblab">
  <about_div>
     <address> SEOUL </address>
     <member> SONG </member>
     <member>LIM</member>
     <contact> tuner1004@dblab.sogang.ac.kr </contact>
  </about_div>
  <seminar category = "public">
     <title> update를 고려한 XML 접근제어 기법 <title>
     <speaker> LIM </speaker>
  </seminar>
  <seminar category = "private">
     <title> 반복적인 색채 노출에 대한 효과 </title>
     <speaker>  SONG </speaker>
  </seminar>
</division>
```

< SEC.XML >

```
<! ELEMENT division (about_div, seminar*>
<! ELEMENT about_div ( address, member +,  contact)>
<! ELEMENT address (#PCDATA)>
<! ELEMENT member (#PCDATA)>
<! ELEMENT contact (#PCDATA)>
<! ELEMENT seminar (title, speaker ?)>
<! ELEMENT title <#PCDATA>
<! ELEMENT speaker <#PCDATA>
<!ATTLIST division name CDATA #REQUIRE>
<!ATTLIST seminar category (public | private) #REQUIRE>
```

< SEC.DTD >

```
<<ADMIN, *, *>, sec.xml:/divisoin, insert, D+, R, -- >
<<PUBLIC, *,*>, sec.dtd:/division/about_div, read, R+ , R -- >
<<PUBLIC, *,*>, sec. xml:/division/seminar[@cat=private], read, R-, R, -- >
<<LIM, *.*>, sec. xml:/division/seminar[@cat=private]/speaker, read, R+, L , -- >
<<LIM.163.239.*.*>, sec.dtd:/division/seminar, delete, U+, R, -- >
<<LIM, *.*>, sec. xml:/division/seminar[@cat=public]/speaker, insert, U+, L, -- >
<<LIM, 163.239.*.*>, sec.dtd:/division/about_div/member, read, R-, L, Hard >
<< LIM, 163.239.*.*>, sec.xml:/division/about_div/member, read, R+, L, -- >
<<DBLAB, *,*>, sec.dtd:/division/seminar[@cat=public]/title, rename, DE+, L , --
<<SOGANG, *,*>, sec.dtd:/division/about_div/contact, insert, U+, L , -- >
<<PUBLIC, 163.239.*>, sec.xml:/division/about_div/address, delete, U-, L, -- >
<<LIM, 163.*>, sec.xml:/division/about_div/address, replace, UE+, L, -- >
<<*.163.239.131.116,*>, sec.xml:/division/about_div/contact, insert, U-, L, -- >
<<KANG, *,*>, sec://xml:/division/seminar, read, R-, R, -- >
```

< 접근권한 정보 >

그림 5 XML 문서, DTD, 접근권한 정보의 예

접근권한 정보로부터 ADMIN은 division 노드에 insert할 수 있는 권한이 있고, 이 insert는 D 타입이기 때문에 구조를 변경시킨다는 것을 알 수 있다. 또한 KANG은 seminar 엘리먼트를 포함해서 하위 엘리먼트를 볼 수 없음을 알 수 있다. 그림 6의 질의는 about_div의 하위 노드에 내용이 database인 엘리먼트를 삽입하고, address의 내용을 PUSAN으로 갱신하며, contact 엘리먼트를 삭제하려는 내용이다.

LIM이라는 사용자가 이와 같은 질의를 한다면, 1 단계 접근제어 과정에서 권한이 없는 연산자의 접근 요청은 거부된다. 사용자의 권한 등급은 접근권한 정보를 통해 알 수 있고, 질의에 포함된 연산자의 타입은 액션 타입 결정표를 참조하여 알 수 있다. 사용자 LIM의 권한 등급은 U 타입이므로, XML 문서의 구조변경 권한은 없다. 하지만 질의문에서 첫 번째 delete 연산자와 insert 연산자는 구조 변경을 초래하는 D 타입이다. 따라서 이들 연산자는 거부되고 권한이 있는 U 타입 연산자에 대해서 1차 승인을 한다. 그런 후 2 단계 접근제어 과정에서 DOM 트리에 설정된 권한과 비교하여 접근권한이 있는 연산자들만 수행된다. 1 단계 접근제어 단계에서 승인된 연산자가 접근을 요청하는 address 엘리먼트와 contact 엘리먼트에 설정된 권한을 그림 7에서 보면, address는 UE+ (replace), contact는 R- 로 설정되어 있음을 알 수 있다. 따라서 중첩 질의에서 replace가 포함된 절은 접근이 허가되어 연산을 수행할 수 있지만, delete가 포함된 절은 거부된다. 사용자는 contact 엘리먼트에 대해 read 권한이 없기 때문에 delete 권한 역시 없게 된다.

이렇게 2 단계로 접근제어 과정을 나눈 결과 불필요한 접근제어 과정을 줄일 수 있었다. 표 3은 XML 구조 변경을 허용하는 환경에서 그림 8의 질의문을 처리하기 위한 접근제어 과정을 기존의 접근제어 기법과 제안한 접근제어 기법을 비교한 것이다.

사용자의 권한 등급이 U 타입일 경우 1차 접근제어

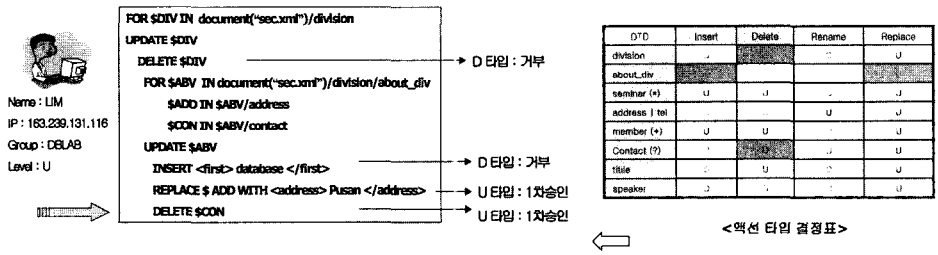



그림 6 1 단계 접근제어 과정

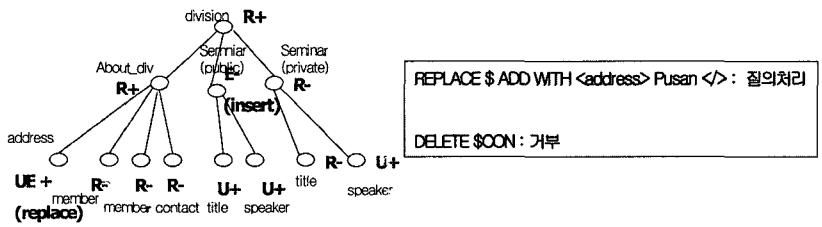



그림 7 2 단계 접근제어 과정

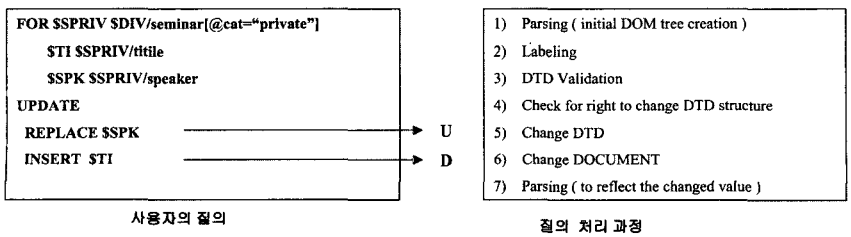



그림 8 갱신 질의를 처리하기 위한 일반적인 접근제어 과정

표 3 접근제어 과정 비교

| 사용자정보 | Securing XML Documents [4] | 제안하는 기법 |
|---|---|---|
| Read 허용 (R) | 1 - ② - (2) | 1 - 2 |
| 구조변경 금지 (U) | 1 - ②-③-⑥-⑦ - (2)-(3)-(4) | 1 - 2 - ⑥ - 7 |
| 구조변경 허용 (D) | 1 - ②-③-⑥-⑦ - (2)-(3)-(4)-(5)-(6)-(7) | 1 - 2 - ⑥ - (5) - (6) - 7 |

단계에서 insert 연산의 접근이 거부되므로, 이후 insert 연산에 대한 추가적인 작업은 필요 없다.

표 3의 O 안의 숫자는 replace 연산을 처리하기 위한 그림 8에서의 절차 번호를 의미하며, ( ) 안의 숫자는 insert 연산을 처리하기 위한 절차 번호이다. 위의 표에서 사용자의 권한 등급이 U 타입의 경우 [9]에서 사용된 접근제어 기법을 사용한다면 다음과 같은 절차가 필요하다. Replace 연산자의 권한 검사를 위해 XML 문서를 파싱하고, 접근권한 정보를 이용하여 DOM 트리에 권한을 레이블링 한다. 그런 후 DTD를 검증단계에서 구조변경을 하는지 검사한다. DTD 검증 결과 replace 연산이 구조 변경을 초래하지 않는다는 것이 확인되면 replace 연산을 처리한다. 연산 처리 후 XML의 내용이 변경되었으므로, 새로운 DOM 트리를 얻기 위해 파싱 작업을 한다. 그런 후 insert 연산의 권한 검사를 한다. 이를 위해 DOM 트리에 권한을 레이블링하고 DTD를 검증한다. 검증 결과 insert 연산자는 DTD의 변경을 초래하므로 insert 연산자는 거부된다. 하지만 제안한 접근제어 기법에서는 insert 연산자는 1차 접근제어 단계에서 거부되므로, replace 연산에 해당하는 접근제어 작업만 수행하면 된다. 따라서 제안한 접근제어 기법에서는 replace 연산자가 요청하는 엘리먼트에 대한 권한 검사를 위해 XML 문서를 파싱하고 권한을 레이블링 하는 작업만 필요하다.

### 3.4. 시스템 구조

제안하는 접근제어 시스템의 구조는 그림 9와 같다.

사용자는 XQuery를 이용하여 XML 문서의 엘리먼트에 대한 접근을 요청한다. 질의를 입력 받은 접근제어 시스템은 DOM Generator에 의해 XML 문서를 파싱하여 DOM 트리를 생성한다. 이 파싱 과정에서 액션 타입 결정표의 액션 타입을 결정하기 위해 DTD에서 ? 기호와 + 기호가 붙은 엘리먼트의 개수를 추출하여 메타 데이타로 유지한다. Query Analyzer는 파싱 과정에서 추출된 메타 데이타와 접근권한 정보를 참조하여 질의문의 연산자와 사용자에 액션 타입을 할당한다. 그런 후 Label Maker는 사용자의 이름, IP 주소, 컴퓨터 이름과 일치하는 권한을 DOM 트리에 레이블링 한다. 1 Phase Access Controller에서는 XML 문서 전체에 대한 권한 검사가 이루어지고, 2 Phase Access Controller에서는 요청한 XML 엘리먼트에 대한 권한 검사가 이루어진다. Process Classifier는 XML의 구조의 변경 여부를 확인한다. 이때 XML 문서의 구조가 변경되었다면 Schema Changer에서 DTD를 변경한 후 Query Processor에 의해 질의가 수행한다. 질의에 의해 XML 문서 혹은 구조가 변경되었다면, Refresher는 변경된 내용을 DOM 트리에 반영하기 위해 파싱을 수행한다. 만약 사용자의 질의가 접근제어 단계에서 모두 거부되었다면, 남은 단계를 거치지 않고 사용자에게 결과 메시지를 반환한다.

## 4. 제안 모델 평가 및 성능 분석

• 모델 평가

기존의 모델[8,9,12]는 오직 검색 연산만을 고려했기

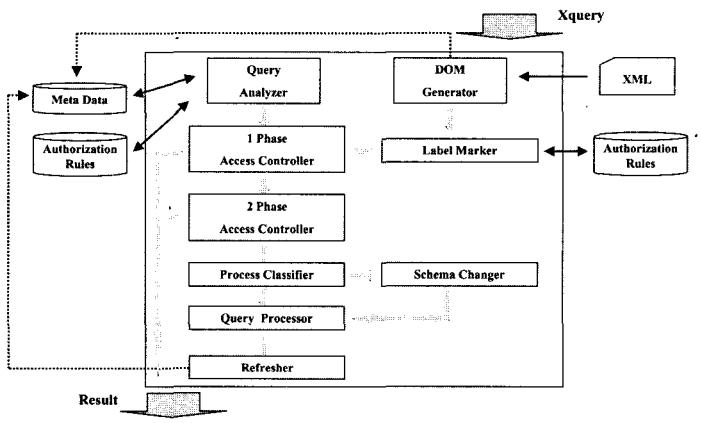



그림 9 접근제어 시스템 구조

때문에, 현실 세계에서 발생하는 갱신 연산에 대한 접근제어를 할 수 없다. 하지만 제안한 접근제어 모델은 검색 연산 뿐만 아니라 갱신 연산에 대해서도 접근제어를 할 수 있다. 또한 XML 문서의 변경과 구조의 변경 모두 허용하는 환경을 지원하므로, 유효한 XML(valid XML) 문서와 잘 정의된(well formed XML) XML 문서까지 접근제어의 대상이 될 수 있다. 그리고 액션 타입을 정의함으로써, 사용자들과 연산자를 그룹 단위로 관리할 수 있게 되었다. 그러므로 복잡하고 다양한 접근권한 정보를 체계적으로 관리할 수 있고, 보안 관리자가 사용자에게 XML 문서에 대한 권한을 부여하거나 회수할 때, 각각의 연산자에 대한 권한을 반복해서 부여하고 회수할 필요가 없다. 그리고 접근제어 과정을 2 단계로 분리함으로써 불필요한 작업을 제거할 수 있다는 장점이 있다.

• 성능 평가

성능 비교의 대상은 [8,9]에서 제안한 XML 접근제어 기법과 본 논문에서 제안한 XML 접근제어 기법이고, 평가 기준은 주어진 질의문에 대한 전체 접근제어 시간이다. [8,9]에서 제안한 모델은 갱신 연산자를 지원하지 않지만, 갱신 연산자도 검색 연산자와 같은 과정으로 접근제어 할 수 있다고 언급되어 있기 때문에, 갱신 연산자도 검색 연산자와 동일한 과정으로 접근제어가 이루어진다고 가정한다. XML 문서와 DTD는 [3]에서 XML 벤치마크를 위해 제공하고 있는 XML 문서와 DTD를 사용한다. 성능 평가를 위한 질의문 Q1, Q2, Q3는 다음과 같다.

• 검색 질의 Q1 :

```
FOR $O IN document(auction.xml)/site/people/person
RETURN
<Homepage> $O/homepage </Homepage>
```

• 갱신 질의 Q2 :

```
FOR $C IN document(auction.xml)/site/people/
person/creditcard
UPDATE
  DELETE $C
```

• 중첩 질의 Q3 :

```
FOR $P IN document(auction.xml)/site/people/person
  $N IN $P/name
  $CT IN $P/city
  $PH IN $P/phone
UPDATE
  DELETE $N
  REPLACE $CT WITH <City> California </City>
  INSERT <Person> AFTER $P
            <Name> SHAN </Name>
            <Email address> tuner1004@dblab.
            sogang.ac.kr </Email address>
          </Person>
```

또한 질의문에 대한 접근제어 시간을 측정하기 위한 매개 변수는 다음과 같다.

• FSD (Full Search DTD) : DTD의 전체 검색 시간
• ARP (Authorization Rules Parsing) : 접근권한 정보의 파싱 시간
• FSAR (Full Search Authorization Rules) : 접근권한 정보의 검색 시간
• XP (XML Parsing) : XML 문서의 파싱 시간
• FSX (Full Search XML) : XML 문서의 전체 검색 시간
• SN (Search Node) : 질의문에 포함된 객체의 검색 시간

이러한 매개 변수의 단위는 밀리세컨드(msec)이다. 표 4는 서로 다른 크기의 XML 문서인 auction3.xml (3M), auction5.xml(5M), auction10.xml(10M)과 DTD인 auction.dtd(5.16KB), 접근권한 정보인 authorization.xas(273KB)에 해당하는 매개 변수 값들을 CPU Pentium Ⅳ 1.7 GHz, 메모리 512MB인 컴퓨터에서 실험한 결과 값이다. 실험에 사용된 데이타와 소스 코드는 부록 1과 부록 2에 첨부한다.

문서의 크기가 커짐에 따라 파싱 시간(XP)과 트리의

표 4 매개 변수들의 값

| | XP | FSX | SN (homepage) | SN (credit) | SN (name) | SN (phone) | SN (city) |
|---|---|---|---|---|---|---|---|
| auction3.xml (3MB) | 1672 | 437 | 16 | 15 | 16 | 16 | 16 |
| auction5.xml (5MB) | 2406 | 703 | 20 | 16 | 17 | 16 | 16 |
| auction10.xml (10MB) | 4766 | 953 | 31 | 32 | 32 | 31 | 31 |
| authorization.xas (273KB) : ARP (534), FSAR (109)<br>auction.dtd : FSD (15) | | | | 단위 : msec | | | |

검색 시간(FSX, SN)은 증가한다. 그리고 매개 변수들 중에서 파싱 시간(XP)이 시스템의 성능에 가장 영향을 미치는 변수가 된다. Q1, Q2, Q3의 질의문에 대한 전체 접근제어 시간은 아래와 같은 공식에 의해 얻을 수 있다.

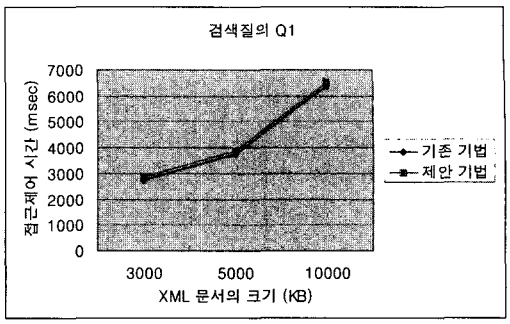
[ 공식 ]

기존 기법[6] : 파싱(XP)
+ 레이블링 (ARP + (연산자 종류 * (FSAR + FSX + SN)))
[ + DTD 검증 (갱신 연산자 종류 * (XP + FSX + FSD)) ]

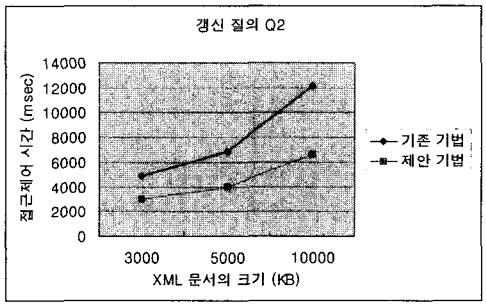
제안 기법 : 액션 타입 할당 (ARP + FSAR)
+ 파싱 (XP)
+ 레이블링 (FSAR + FSX + (연산자의 종류 * SN))
[ + DTD 검증 (FSAR)]

그림 10은 Q1, Q2, Q2 질의문에 대한 접근제어 시간을 기존의 접근제어 기법과 비교한 결과이다.

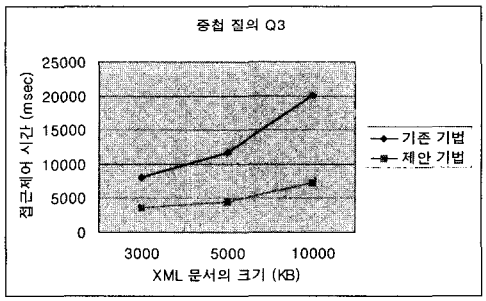
첫 번째 검색 질의 (a)의 경우, 기존 기법에서는 XP + ARP + FSAR + FSX + SN의 접근제어 시간이 소요되고, 제안 기법에서는 ARP + FSAR + XP + FSAR + FSX+ SN의 접근제어 시간이 소요된다. 제안 기법에서는 연산자의 액션 타입을 할당 하기 위해 접근권한 정보 authorization.xas를 검색해야 한다. Authorization.xas은 XML 문서로 작성되어 있으므로, 접근제어 정보를 참조하기 위해서는 authorization.xas를 파싱(ARP)하고, 트리를 검색(FSAR)해야 한다. Authorization.xas의 파싱(ARP)작업은 레이블링 과정에서 접근권한 정보를 참조하기 위해 필요한 작업이므로, 검색질의 (a)에서는 FSAR(109msec) 만큼의 오버헤드가 발생하게 되는 것이다. 두 번째 갱신 질의 (b)의 경우, 기존 기법에서는 XP + ARP + FSAR + FSX + SN + XP + FSX + FSD의 접근제어 시간이 소요되고, 제안 기법에서는 ARP + FSAR + XP + FSAR + FSX + SN + FSAR 시간이 소요된다. 갱신 연산자가 포함된 질의문에서는 갱신 연산자로 인해 구조의 변경이 초래되는지를 검사하기 위해 DTD 검증 과정이 필요하다. 기존 기법에서는 이를 위해, 갱신 연산을 수행하여 변경된 XML 문서를 파싱하고(XP), 새로운 DOM 트리의 모든 노드를 DTD와 비교하면서 방문하게 된다(FSX + FSD). 하지만 제안 기법에서는 액션 타입을 통해 구조 변경 여부를 알 수 있다. 따라서 DTD 검증 과정에서는 접근권한 정보를 검색하는 시간(FSAR)만 필요하게 된다. 세 번째 중첩질의 (c)는 delete, replace, insert의 세 종류의 연산자가 포함되어 있다. 이러한 경우, 기존 기법에서는 XP + ARP + 3 * ( FSX + FSAR + SN ) + 2 * (XP + FSX + FSD)의 시간이 소요되고, 제안 기법에서는 ARP + FSAR + XP + FSX + FSAR + (3 * SN) + (2 * FSAR)의 시간이 소요된다. 기존 기법에서는 연산자 단위로 레이블링 하기 때문에 연산자의 종류 수 만큼 레이블링(3 * (FSX + FSAR + SN)을 해야 하지만, 제안 기법에서는 액션 타입 단위로 레이블링을 하기 때문에 오직 한번의 레이블링 과정(FSX + FSAR)과 각 연산자가 요청한 노드의 레이블링 결과값을 확인하기 위한 검색(3 * SN)만이 필요하다. Replace 연산자는 엘리먼트나 속성 값을 변경하는 연산자이기 때문에 구조의 변경을 초래하지 않는다. 따라서 replace 연산자의 경우는 DTD 검증 과정이 필요 없다. 하지만 insert, delete는 구조 변경을 초래할 수 있으므로, DTD 검증 과정이 필요하다. 그러므로 기존 기법에서는 두 번




(a) 검색 질의에 대한 접근제어 시간




(b) 갱신 질의에 대한 접근제어 시간




(c) 중첩 질의에 대한 접근제어 시간

그림 10 접근제어 시간의 비교

의 파싱과 트리 검색(2 * (XP + FSX + FSD) 시간이 필요하지만, 제안 기법에서는 접근제어 정보에 두 번의 검색(2 * FSAR)시간이 필요하게 된다.

제안한 접근제어 기법은 검색 질의의 경우, 연산자의 액션 타입을 할당하기 위한 오버헤드가 존재하지만, 갱신 질의와 갱신 연산자가 포함된 중첩 질의에 대해서는 접근제어를 위한 많은 시간을 줄일 수 있다. 따라서 제안한 접근제어 기법은 갱신 질의가 빈번하게 발생하는 환경에서 좋은 성능을 나타낼 수 있다.

## 5. 결론 및 추후 과제

웹 환경이 발달하고, XML이 웹 데이타의 표준으로 제정되면서, 사용자들은 웹 상에 존재하는 XML 데이타에 검색 질의뿐만 아니라 삽입, 삭제와 같은 갱신 질의를 포함한 다양한 질의를 요구하고 있다. 또한 XML이 보편화됨에 따라 XML 데이타에 대한 보안의 요구사항도 생겨나기 시작했다. 기존의 XML 접근제어 모델은 갱신 연산을 고려하지 않기 때문에 갱신 질의에 대한 효율적인 접근제어를 지원하지 못하고 있다.

본 논문은 갱신 질의가 발생하는 현실적인 환경에 적합한 XML 접근제어 모델과 기법을 제안하였다. 갱신 연산을 지원하는 XML 모델을 제안하기 위해 XML 갱신 연산자를 정의했고, 이를 접근제어 모델에 포함시켰다. 접근제어 모델에 갱신 연산자를 추가하면서 발생하는 성능상의 문제점들을 해결하기 위해 새로운 액션 타입을 정의했다. 액션 타입을 정의함으로써 복잡한 접근권한 정보를 체계적 관리할 수 있었다. 또한 기존의 DOM 기반의 DTD 검증 과정 개선하고, 반복적인 트리의 검색을 제거하여 성능상의 문제점들을 해결하였다. 제안한 접근제어 모델에서는 접근제어의 과정을 2 단계로 나누었다. 이와 같이 접근제어 과정을 2 단계로 나눔으로써, 1 단계에서 거부되는 연산자들로 인한 불필요한 작업이 제거되어, 결과적으로 접근제어의 단계를 줄일 수 있다. 제안한 접근제어 모델은 검색 질의에 대해서는 약간의 오버헤드가 존재하지만, 갱신 질의에 대해서는 좋은 성능을 보인다. 따라서 제안한 모델은 사용자의 권한이 명확하게 구분되어 있고, 질의의 대부분이 갱신 연산인 물량 주문 시스템이나 데이타의 일치성을 보장하기 위해 갱신 질의가 반드시 필요한 미러링 시스템과 같은 환경에 적합하다. 하지만 제안하는 접근제어 모델 역시 DOM을 기반으로 하고 있기 때문에 트리에 대한 검색은 불가피하므로 성능상의 한계점을 내포하고 있다. 그러므로 성능 개선을 위해 데이타베이스의 빠른 엔진을 활용하는 연구가 추후에 필요하다.

## 참 고 문 헌


[1] David Hunter (Editor), et al., "Beginning XML," wrox, 2001.

[2] T.Bray et.al. (ed.), "Extensible Markup Language (XML) 1.0," World Wide Web Consortium (W3C), February 1998. http://www.w3.org/TR/REC-xml.

[3] S.Boag, D.Chamberlin, M.F. Fernandez, D.Florescu, J.Robie, J.simeon, and M.Stefanescu, "XQuery 1.0: An XML query language," http://www.w3.org/TR/xquery/, 30 April 2002. W3C working draft.

[4] Rutgers Security Team, "WWW Security, A survey," 1999. http://www-ns.rutgers.edu/www-security/.

[5] Oracle, "Database Security in Oracle 8i," February 1999.

[6] S.Castano, M.Fugini, G.. Martella and P.Samarati, "Database Security," Addision-Wesley, 1995.

[7] T.F. Lunt, "Access Control Policies for Database Systems," In C.E. Landwehr, editor, Database Security, II:status and Prospects, North-Holland, Amsterdam, 1989.

[8] E.Damiani, S.Vimercati, S.Paraboachk and P.Samarati, "Design and implementation of an access processor for xml documents," In Proceedings of the 9th international WWW conference, Amsterdam, May 2000.

[9] E.Damiani, S.Vimercati, S.Paraboschi, and P.Samarati, "Securing xml document," In Proceedings of the 2000 International Conference on Extending Database Technology(EDBT2000), pp 121-135, Konstan, Germany, March, 2000.

[10] Igor Tatarinov, Zachary G. Ives, Alon Y.Halevy, Daniel S.Weld, "Updating XML," ACM SIGMOD, pp 413-424, Santa Barbara, California, USA, May, 2001.

[11] S.Jajodia, P.samarati, and V.S Subrahmanian, "A Logical Language for Expressing Authorization," In Proceeding of the IEEE Symposium on Security and Privacy, pages 31-42, Oakland, CA, May 1997.

[12] Yue Wang, Kian-Lee Tan, "A Scalable XML Access Control System," In Proceedings of the 10th International WWW Conference(Poster), 2001.

[13] Albrecht Schmidt, Florian Waas, Martin Kersten, Michael J.Carey, Ioana Manolescu and Ralph Busse, "Xmark: A Benchmark for XML Data Management," Proc. VLDB, Hong Kong, China, 2002.

[14] Kevin Williams (Editor), et al., "Professional XML Databases," wrox, 2001.

[15] P.Samarati, E.Bertino, and S.Jajodia, "An Authorization Model for a Distritbuted Hypertext System," IEEE TKDE, 8(4):555-562, August 1996.

## 부 록

[부록 1] auction3.xml, auction5.xml, auction10.xml의 DTD auction.dtd

```
<!-- DTD for auction database -->
<!-- $Id: auction.dtd,v 1.15 2001/01/29 21:42:35 albrecht Exp $ -->

<!ELEMENT site      (regions, categories, catgraph, people, open_auctions, closed_auctions)>
<!ELEMENT categories      (category+)>
<!ELEMENT category        (name, description)>
<!ATTLIST  category       id ID #REQUIRED>
<!ELEMENT name            (#PCDATA)>
<!ELEMENT description     (text | parlist)>
<!ELEMENT text            (#PCDATA | bold | keyword | emph)*>
<!ELEMENT bold            (#PCDATA | bold | keyword | emph)*>
<!ELEMENT keyword         (#PCDATA | bold | keyword | emph)*>
<!ELEMENT emph            (#PCDATA | bold | keyword | emph)*>
<!ELEMENT parlist         (listitem)*>
<!ELEMENT listitem        (text | parlist)*>
<!ELEMENT catgraph        (edge*)>
<!ELEMENT edge            EMPTY>
<!ATTLIST  edge           from IDREF #REQUIRED to IDREF #REQUIRED>
<!ELEMENT regions         (africa, asia, australia, europe, namerica, samerica)>
<!ELEMENT africa          (item*)>
<!ELEMENT asia            (item*)>
<!ELEMENT australia       (item*)>
<!ELEMENT namerica        (item*)>
<!ELEMENT samerica        (item*)>
<!ELEMENT europe          (item*)>
<!ELEMENT item            (location, quantity, name, payment, description, shipping, incategory+,
                          mailbox)>
<!ATTLIST item            id ID #REQUIRED, featured CDATA #IMPLIED>
<!ELEMENT location        (#PCDATA)>
<!ELEMENT quantity        (#PCDATA)>
<!ELEMENT payment         (#PCDATA)>
<!ELEMENT shipping        (#PCDATA)>
<!ELEMENT reserve         (#PCDATA)>
<!ELEMENT incategory      EMPTY>
<!ATTLIST  incategory       category IDREF #REQUIRED>
<!ELEMENT mailbox         (mail*)>
<!ELEMENT mail            (from, to, date, text)>
<!ELEMENT from            (#PCDATA)>
<!ELEMENT to              (#PCDATA)>
<!ELEMENT date            (#PCDATA)>
<!ELEMENT itemref         EMPTY>
<!ATTLIST   itemref       item IDREF #REQUIRED>
<!ELEMENT personref       EMPTY>
<!ATTLIST   personref     person IDREF #REQUIRED>
<!ELEMENT people          (person*)>
```

```
<!ELEMENT person (name, emailaddress, phone?, address?, homepage?, creditcard?,
                  profile?, watches?)>
<!ATTLIST person id ID #REQUIRED>
<!ELEMENT emailaddress (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT address (street, city, country, province?, zipcode)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT province (#PCDATA)>
<!ELEMENT zipcode (#PCDATA)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT homepage (#PCDATA)>
<!ELEMENT creditcard (#PCDATA)>
<!ELEMENT profile (interest*, education?, gender?, business, age?)>
<!ATTLIST profile income CDATA #IMPLIED>
<!ELEMENT interest EMPTY>
<!ATTLIST interest category IDREF #REQUIRED>
<!ELEMENT education (#PCDATA)>
<!ELEMENT income (#PCDATA)>
<!ELEMENT gender (#PCDATA)>
<!ELEMENT business (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT watches (watch*)>
<!ELEMENT watch EMPTY>
<!ATTLIST watch open_auction IDREF #REQUIRED>
<!ELEMENT open_auctions (open_auction*)>
<!ELEMENT open_auction (initial, reserve?, bidder*, current, privacy?, itemref, seller, annotation,
                        quantity, type, interval)>
<!ATTLIST open_auction id ID #REQUIRED>
<!ELEMENT privacy (#PCDATA)>
<!ELEMENT initial (#PCDATA)>
<!ELEMENT bidder (date, time, personref, increase)>
<!ELEMENT seller EMPTY>
<!ATTLIST seller person IDREF #REQUIRED>
<!ELEMENT current (#PCDATA)>
<!ELEMENT increase (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT interval (start, end)>
<!ELEMENT start (#PCDATA)>
<!ELEMENT end (#PCDATA)>
<!ELEMENT time (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT amount (#PCDATA)>
<!ELEMENT closed_auctions (closed_auction*)>
<!ELEMENT closed_auction (seller, buyer, itemref, price, date, quantity, type, annotation?)>
<!ELEMENT buyer EMPTY>
<!ATTLIST buyer person IDREF #REQUIRED>
<!ELEMENT price (#PCDATA)>
<!ELEMENT annotation (author, description?, happiness)>
```

```
<!ELEMENT author          EMPTY>
<!ATTLIST   author        person IDREF #REQUIRED>
<!ELEMENT happiness       (#PCDATA)>
```

[부록 2] 접근제어 시간 측정 프로그램 소스 ( estimate.java )

```
package estimatedom;
import org.jdom.*;
import org.jdom.input.SAXBuilder;  // JDOM 사용
import java.io.*;
import java.util.*;
public class EstimateDom {
public static void main(String[] args) {
        SAXBuilder builder = new SAXBuilder(false);
        Document doc=null;
  try {
                long s = new Date().getTime();
                doc = builder.build(new File("auction5.xml")); // aution5.xml 문서 로딩
                long e = new Date().getTime();
                System.err.println("XML Document is loaded.");
                System.err.println("XML pasing time : " + (e-s) + "ms"); // XP, ARP 변수 값
                }
                catch (JDOMException ex) {
                        System.err.println("Can't open file.");
                        System.exit(1);
                }
                        // 전체 DOM 트리 검색시간 측정
                System.err.println("전체 Dom트리 순회");
                long start = new Date().getTime();
                int elmsize =traverseDom(doc);
                long end = new Date().getTime();
                System.err.println("Elements Count: "+ elmsize);
                System.err.println("Time : "+ (end - start)+ "ms");
                        // site/people/person/homepage 노드 검색시간 측정
                System.err.println("\n모든 Homepage 찾기");
                start = new Date().getTime();
                elmsize = findNode1(doc);
                end = new Date().getTime();
                System.err.println("Elements(homepage) count: "+ elmsize);
                System.err.println("Time : "+ (end - start)+ "ms");
                        // site/people/person/creditcard 노드 검색시간 측정
                System.err.println("\n모든 Creaditcard 찾기");
                start = new Date().getTime();
                elmsize = findNode2(doc);
                end = new Date().getTime();
                System.err.println("Elements(creditcard) count: "+ elmsize);
                System.err.println("Time : "+ (end - start)+ "ms");
                        // site/people/person/name,phone,address/city 노드 검색시간 측정
```

```
        System.err.println("\n모든 name, phone, city 찾기");
        start = new Date().getTime();
        elmsize = findNode3(doc);
        end = new Date().getTime();
        System.err.println("Elements(name,phone, city) count: "+ elmsize);
        System.err.println("Time : "+ (end - start)+ "ms");
}
public static int traverseDom(Document doc) { // FSX, FSD, FARS 변수 값
        int i=0;
        Element root = doc.getRootElement();
        LinkedList queue = new LinkedList();
        queue.addLast(root);
        i++;
        while(!queue.isEmpty()){
                Element elm = (Element) queue.removeFirst();
                List list = elm.getChildren();
                i += list.size();
                queue.addAll(list);
        }
        return i;
}
public static int findNode1(Document doc) { // SN 변수 값
        Element root = doc.getRootElement();
        LinkedList nodeList = new LinkedList();
        Element people= root.getChild("people");
        List persons = people.getChildren("person");
        Iterator itr = persons.iterator();
        while(itr.hasNext()){
                Element person = (Element) itr.next();
                nodeList.add(person.getChildren("homepage"));
        }
        return nodeList.size();
}
public static int findNode2(Document doc) { // SN 변수 값
Element root = doc.getRootElement();
LinkedList nodeList = new LinkedList();
Element people= root.getChild("people");
List persons = people.getChildren("person");
Iterator itr = persons.iterator();
while(itr.hasNext()){
        Element person = (Element) itr.next();
        nodeList.add(person.getChildren("creditcard"));
}
return nodeList.size();
}
public static int findNode3(Document doc) { // SN 변수 값
Element root = doc.getRootElement();
LinkedList nodeList = new LinkedList();
```

```
        Element people= root.getChild("people");
        List persons = people.getChildren("person");
        Iterator itr = persons.iterator();
        while(itr.hasNext()){
                Element person = (Element) itr.next();
                Element phone = person.getChild("phone");
                Element name = person.getChild("name");
                Element address = person.getChild("address");
                if(phone != null) nodeList.add(phone);
                if(name != null) nodeList.add(name);
                if(address != null)    nodeList.add(address.getChild("city"));
        }
        return nodeList.size();
        }
}
```

임 청 환

2001년 2월 숭실대학교 컴퓨터공학과 졸업. 2003년 2월 서강대학교 데이타베이스연구실 졸업(공학석사). 2003년 3월~2004년 8월 뉴욕 NYBS 시스템 엔지니어로 근무. 2004년 9월~현재 e 신한 정보기술팀 DBA로 근무

박 석

1978년 서울대학교 계산통계학과(이학사). 1980년 한국과학기술원 전산학과(공학석사). 1983년 한국과학기술원 전산학과(공학박사). 1983년 9월~현재 서강대학교 컴퓨터학과 교수. 1989년~1991년 University of Virginia 방문교수. 1996년~1998년 한국정보과학회 데이타베이스 연구회 운영위원장. 1997년 2월~현재 한국통신정보보호학회 이사. 2001년 12월 한국정보과학회 이사 2000년 4월~현재 DASFAA Steering Committee 멤버. 관심분야는 데이타베이스 보안 멀티미디어 데이타베이스, 트랜잭션 관리, 데이타웨어하우스, 웹과 데이타베이스, 워크플로우