효율적인 질의 기반 XML 접근제어 수행 메커니즘

(An Efficient Query-based XML Access Control Enforcement Mechanism)

변 창 우 † 박 석 ††
(Changwoo Byun) (Seog Park)

요 약 다양한 사용자 및 응용 프로그램들이 XML을 기반으로 정보의 분산과 공유를 요구함에 따라 안전하고 효율적으로 XML 데이타를 접근하는 요구가 중요한 이슈로 부각되고 있다. 특히, 접근제어 규칙을 XPath로 표현함으로써 문서 단위 접근 범위의 한계를 극복하고 문서의 일부분 단위의 접근을 해결하면서 접근제어를 수행할 때 발생될 수 있는 충돌에 대한 해결책을 마련하는 안전성에 초점을 둔 연구들은 많았으나 접근제어를 수행할 때의 효율성에 초점을 둔 연구는 미비하다. 본 논문은 안정성뿐만 아니라 효율성을 고려한 XML 접근제어 시스템을 내용으로 하고 있다. 제안하는 방법은 사용자의 질의와 그 사용자의 접근제어 규칙들 중에 질의와 관련된 접근제어 규칙들만을 선택하게 하고 XPath 2.0에서 지원하고 있는 집합 연산을 적절히 연결시켜 접근 제어 정책을 준수하는 새로운 대체 질의로 변경하는 선처리 방법이다. 본 논문에서 제안하고 있는 방법은 어떠한 XML 데이타베이스 관리 시스템에도 적용가능하며 최소 단위의 접근제어 수행, 구현의 용이성, 낮은 실행시간, 그리고 안전하고 정확한 대체 질의 생성을 보장한다. 이와 같은 장점들을 실험을 통해 분석한다.

키워드 : XML 데이타, 보안, XML 접근제어, 필터링 기술

Abstract As XML is becoming a de facto standard for distribution and sharing of information, the need for an efficient yet secure access of XML data has become very important. To enforce the fine-level granularity requirement, authorization models for regulating access to XML documents use XPath which is a standard for specifying parts of XML data and a suitable language for both query processing. An access control environment for XML documents and some techniques to deal with authorization priorities and conflict resolution issues are proposed. Despite this, relatively little work has been done to enforce access controls particularly for XML databases in the case of query access. Developing an efficient mechanism for XML databases to control query-based access is therefore the central theme of this paper.This work is a proposal for an efficient yet secure XML access control system. The basic idea utilized is that a user query interaction with only necessary access control rules is modified to an alternative form which is guaranteed to have no access violations using tree-aware metadata of XML schemas and set operators supported by XPath 2.0. The scheme can be applied to any XML database management system and has several advantages over other suggested schemes. These include implementation easiness, small execution time overhead, fine-grained controls, and safe and correct query modification. The experimental results clearly demonstrate the efficiency of the approach.

Key words : XML Data, Security, XML Access Control, Filtering Technique

· 본 연구는 한국과학재단 특정기초연구(R01-2006-000-10609-0) 지원으로 수행되었음
† 학생회원 : 서강대학교 컴퓨터학과
chang@dblab.sogang.ac.kr
†† 종신회원 : 서강대학교 컴퓨터학과 교수
spark@dblab.sogang.ac.kr
논문접수 : 2006년 4월 24일
심사완료 : 2006년 10월 31일

1. 서 론

XML[1]이 인터넷 환경에서 데이타 저장과 전송을 위한 표준으로서 부각되면서 많은 사용자 및 응용 프로그램들이 XML 데이타 기반의 정보의 분산과 공유에 대한 요구가 날로 증가함에 따라, 효율적이고 안전하게

XML 데이타에 접근할 수 있는 방법이 매우 중요한 이슈로 부각하고 있다. 그럼에도 불구하고, 이와 관련된 연구들은 문서의 일부분 단위의 접근을 해결하면서 접근제어를 수행할 때 발생될 수 있는 충돌에 대한 해결책을 마련하는 안전성에만 주로 초점을 두어 왔으며 질의에 기반한 효율적인 접근제어를 수행하는 연구는 미비하다. 따라서, 본 논문의 주요 핵심은 질의 기반의 XML 데이타에 대한 안전하고 효율적인 접근제어 메커니즘을 제안하는 것이며, 사용자 질의에 맞는 접근제어 규칙들만 추출하는 방법 및 사용자 질의를 접근제어 정책을 준수하는 질의로 변경하는 문제점을 해결하는데 초점을 두고 있다.

사용자 질의에 맞는 접근제어 규칙들의 추출은 사용자의 접근제어 규칙들 중에서 그 사용자의 질의와 관련된 규칙들만 추출할 수 있는 방법을 개발하는 것을 의미한다. 한편, 사용자 질의 변경은 추출된 접근제어 규칙들을 이용하여 원래의 사용자 질의를 그 사용자의 접근제어 정책을 준수하는 안전하고 정확한 질의로 변경하는 효율적인 질의 재작성 방법을 개발하는 것이다.

논문의 의의. 다음의 특성들을 통해 효율적이고 안전한 XML 접근제어 시스템(SQ-Filter 시스템)을 개발하였고, 실험을 통해 접근 결정 시간의 빠른 성능과 정확한 질의 재작성 방법의 정확성을 입증하였다:

- **질의에 필요한 접근제어 정보만의 추출**: XML 데이타에 대한 전통적인 접근제어 메커니즘은 질의를 요구한 사용자의 접근제어 규칙들을 이용한다. 반대로 SQ-Filter 시스템은 접근제어 규칙의 PRE(order)와 POST(order) 값과 질의의 (PRE, POST) 값을 이차평면(이를 PRE/POST 평면 이라 함)에 사상시켜 그 질의에 관련된 조상(혹은 부모) 및 자손(혹은 자식) 관계에 있는 접근제어 규칙들만을 이용한다. 결국, 사용자의 접근제어 규칙들 중에 질의에 해당되는 최소의 접근제어 규칙들을 이용하기 때문에 처리량이 줄어드는 잇점과 빠른 질의 재작성에 영향을 준다.
- **효율적인 질의 재작성 알고리즘**: 비결정 유한 오토마타(NFA)방법 [16,17]은 "*" 노드 혹은 "//" 축을 갖고 있는 질의에 대한 재작성에 매우 느리고 부정확한 결과를 낳을 수 있는 단점을 갖고 있다. 본 논문에서 추구하는 개념은 간단한 수치 비교이다. 만약 DTD를 통해 "*" 노드의 전·후 노드를 알고 있다면 "*" 노드는 불필요하다. 또한 "//" 축 역시 "//" 축의 전·후 노드를 알고 있다면 그들 사이의 단일 경로를 알 수 있게 되어 "//" 축은 일련의 "/" 축들로 교체될 수 있다. 이것은 질의 재작성 처리기가 부정확한 질의를 만드는 것을 미연에 방지할 수 있는 중요한 요인이 된다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 관련 연구들과 그들의 단점을 간략히 설명한다. 3장에서는 본 논문의 배경 지식을 설명하고, Document Type Definition(DTD)의 메타데이타 및 SQ-Filter 시스템에서 사용하는 기본적인 용어를 설명한다. SQ-Filter 시스템에 대한 개요 및 SQ-Filter 시스템을 구성하는 여러 컴포넌트들 및 그 컴포넌트들을 생성하는데 필요한 알고리즘들을 4장에서 설명한다. 5장에서는 여러 실험들을 통해 비교 시스템인 Q-Filter보다 성능이 뛰어남을 보이고 최종적으로 6장에서 결론을 맺는다.

2. 관련 연구

최소 단위 XML 접근제어 모델. XML 데이타에 대한 권한부여로 XPath[2]를 이용하는 다양한 접근 단위의 객체를 보호하는 것과 관련되어 있다. 일반적으로, 현존하는 XML 접근제어 모델은 5-튜플(주체, 객체, 접근유형, 기호, 타입)로 구성된 접근제어 규칙 명세를 가정한다[4-9]. 주체는 권한을 갖고 있는 사용자 혹은 사용자 그룹을 말한다. 객체는 XML 데이타(혹은 일부분)와 관련되며 XPath[2]를 이용하여 표현하고 있다. 접근유형은 연산의 종류를 말하고, 기호는 접근의 허가(긍정적 접근제어 규칙: +) 혹은 불허(부정적 접근제어 규칙: -)를 말한다. 타입은 접근의 범위로서 전파허용(Recursive) 혹은 전파불허(Local)를 말한다. E. Damiani et al.[4,5], E. Bertino et al.[6,7], Gabilon과 Bruno[8], 그리고 Stoica와 Farkas[9]들의 연구에서는 XML의 계층적 성질에 의해 접근 범위를 다음과 같이 분류한다:

1) 노드 그 자체
2) 노드와 그 노드의 속성
3) 노드와 그 노드의 텍스트 자식 노드
4) 노드와 그 노드의 속성, 그리고 그 노드의 자손 및 자손들의 속성

만약 접근 범위가 (1)과 (2)라면 전파불허이고 (3)과 (4)라면 전파허용이 된다.

한편, 부정적(negative) 접근제어 규칙과 긍정적(positive) 접근제어 규칙을 혼합하여 사용하는 방법에는 두 가지 유형이 있다[3]. 첫 번째 유형은 긍정적 접근제어 규칙의 범위 내의 부정적 접근제어 규칙의 조합이고 두 번째 유형은 부정적 접근제어 규칙의 범위 내의 긍정적 접근제어 규칙의 조합이다. M. Murata et al.[11]은 첫 번째 유형을 '접근불허 하향 일관성(denial downward consistency)'이라 정의하고 있으며 특히, 두 번째 유형은 '불법적인 판독 접근성(invalid read accessibility)'의 문제를 안고 있다고 지적하고 있다.

효율적인 XML 접근제어 수행 메커니즘. 전통적인 XML 접근제어 수행 메커니즘[4-9]은 뷰-기반 수행 메

커니즘이다. 즉, 질의를 한 사용자와 관련된 접근제어 규칙들에 의해 생성된 문서의 특정 뷰를 기반으로 사용자의 접근을 제어한다. 트리 레이블링 기법으로 뷰를 계산할 수 있는 유용한 알고리즘을 제시하였지만, 뷰 생성의 높은 비용 및 뷰 유지 비용의 문제점을 가지고 있다. 또한, 사용자 증가에 따른 확장성 문제를 갖고 있다.

뷰 기반의 문제점을 극복하기 위해 M. Murata et al.[11]은 접근 제어 정책을 만족하지 못하는 질의를 사전에 걸러낼 수 있는 필터링 방법을 제안하였다. J. M Jeon et al.[12]은 XML 문서에 대한 모든 접근제어 규칙에 대한 구조 요약 정보를 에지로 구성하고, 각 노드에 접근제어 규칙을 기록한 XML 접근제어 트리(XACT)를 두어 사용자의 질의를 XACT 접근제어 정보들을 비교하여 변형된 안전한 질의로 변형하는 방법을 제안하였다. XACT는 질의를 제출한 사용자뿐만 아니라 문서에 대한 모든 접근제어 규칙을 갖고 있기 때문에 불필요한 계산 시간을 갖고 있다. B. Luo et al.[13]은 [11]의 방법론을 한층 발전시켜 접근제어 정책의 일부분만을 만족하고 있는 질의에 대해 공유 비결정 유한 오토마타(shared NFA) 방법을 이용해 재작성하는 기법을 제안하였다. 그러나, 공유 비결정 유한 오토마타 방법은 사용자의 질의 관점에서 불필요한 접근제어 규칙들에 대한 오토마타까지 포함하고 있어 사용자 질의의 접근허가, 접근불허, 혹은 질의 재작성에 대한 결정을 내리는데 불필요한 시간 낭비를 초래한다. 추가로, 비록 제안된 NFA 기반의 알고리즘은 루트부터 시작하는 경로 질의의 재작성에는 유용하나 "//" 축을 갖고 있는 경로 질의에 대한 재작성에는 비효율적이고 NFA 상의 불필요한 탐색의 문제를 안고 있다. 또한, 부정확한 질의를 재작성하게 만들게 된다. 이와 같은 문제점의 원인을 4.5절에서 자세히 설명한다.

3. 배 경

이번 장에서는 본 논문에서 제안하는 XML 접근제어 방법의 배경 지식과 논문에서 제안하고있는 가정들 및 기본 용어들에 대해 설명한다.

3.1 XML

XML 문서는 엘리먼트, 속성, 그리고 텍스트 노드로 구성된다. 각 엘리먼트의 내용은 중첩된엘리먼트들의 일련의 순서 혹은 텍스트 노드이다. 하나의 엘리먼트는 속성 집합을 가질 수 있으며 이들은 이름(name)과 값(value)로 되어 있다. 속성은 엘리먼트에 대한 추가적인 정보를 제공한다.

XML 문서는 잘 정형화된(well-formed) 문서와 유효한(valid) 문서, 두 가지 유형이 있다. 다음의 세 가지 규칙들을 준수한 문서를 잘정형화된 문서라 한다.

- 문서는 적어도 하나의 엘리먼트를 포함해야 한다.
- 문서는 하나의 루트 엘리먼트를 포함해야 한다. 이것은 문서 전체를 둘러싼 유일한 시작 태그와 끝 태그이다.
- 다른 모든 엘리먼트는 엘리먼트들 간의 겹침(overlap)이 없이 중첩되어야 한다.

한편, 유효한 문서는 엘리먼트 혹은 속성 생성의 규칙들을 표현한 Document Type Definition(DTD) 혹은 XML Schema에 정의된 허용된 타입과 순서 규칙들을 준수한 잘 정형화된 문서를 말한다. 따라서, 유효한 문서 환경에서는 XPath 기반의 질의의 경로 구조(프레디키트는 제외)는 반드시 DTD(혹은 XML Schema)의 구조를 따르게 된다.

가정 1. 본 논문에서는 DTD 기반의 유효한 XML을 이용한다.

비록 설명되는 내용은 DTD 기반의 유효한 XML 환경이지만, XML Schema 기반의 유효한 문서 환경뿐만 아니라, 잘 정형화된 XML 환경에서도 같은 방법으로 적용될 수 있다. 잘 정형화된 XML 환경에서는 일시적인 유연한 DTD를 생성하여 할 수도 있고, 아니면 직접 XML 문서에 적용할 수 있다.

3.2 XPath와 접근제어

XPath는 XML 문서를 트리 형태로 표현하고 특정 부분을 나타내는 경로 표현 언어이다. 전형적인 경로 표현은 연속적인 스텝(location step)들로 이루어진다. 각 스텝은 문서 내의 노드 사이의 자식(child), 속성(attribute), 조상(ancestor), 혹은 자손(descendant) 등의 관계를 나타낸다. 경로 상의 어떤 스텝은 또한 선택되는 노드를 걸러주는 필터 역할을 하는 프레디키트(predicate)를 포함하기도 한다. 이와 같은 XPath의 특성을 이용하여 접근 단위의 최소화 요구사항을 만족시키기 위해 XML 문서에 대한 접근을 수행하는 권한부여 모델은 XPath를 이용한다[2].

가정 2. 본 논문에서는 '부모-자식' 관계를 나타내는 '/', '조상-자손' 관계를 나타내는 '//' 과 임의의 엘리먼트를 나타내는 '*' 스텝만을 가정한다.

일반적인 접근제어는 사용자의 질의가 들어오면, 그 사용자에 해당되는 접근제어 규칙들을 추출하여 사용자가 객체에 대해 접근이 가능한지를 따진다. 따라서, 객체에 관련이 없는 접근제어 규칙들 역시 추출되어 비교하는 추가적인 계산 시간이 생긴다. 그러나, XML 문서 기반의 접근제어 규칙이 XPath로 표현되고, 사용자는 XPath 기반의 질의를 하는 환경에서는 이와 같은 불필요한 접근제어 규칙들에 대한 추가 시간의 낭비를 줄일 수 있음을 간과하고 있다. 즉, XML 문서는 사용자의

질의를 기반으로 조상(Ancestor, 부모 포함), 후손(Descendant, 자식 포함), 그리고, 형제(Sibling, following-sibling과 preceding-sibling), 이렇게 세 부분으로 나뉠 수 있다. 결국, 사용자의 질의를 기반으로 필요한 접근제어 규칙들은 조상 혹은 후손 영역에 기술된 접근제어 규칙들만이다. 형제 영역에 기술된 접근제어 규칙들은 사용자 질의 관점에서는 불필요한 접근제어 규칙이 된다.

본 논문에서는 DTD에 대한 메타데이타인 PRE/POST 구조를 통해 사용자의 질의가 들어오면 그 사용자의 접근제어 규칙들 중 조상 혹은 후손 영역의 접근제어 규칙들만을 추출하는 효과적인 알고리즘을 제시한다.

3.3 접근제어 정책

계층적 데이타 모델(예를 들어, Object-Oriented Data Model, XML Data Model 등) [3]에서는 보안 관리자가 명시한 권한부여는 명시적(explicit)이라고 부르며, 명시적 권한부여를 기반으로 시스템이 파생한 권한부여를 묵시적(implicit)이라고 한다. 저장의 잇점을 얻고자 묵시적 권한부여 방법을 사용하면서 적절하게 '전파 정책(propagation policy)[1]'을 만든다. 여러 환경에 따라 최적의 전파 정책은 다르겠지만 일반적으로 '가장 특정화된 것을 우선으로 하는 정책(most specific precedence)'을 사용한다. 이와 같은 묵시적 권한부여에 의한 전파 정책에 의해 발생될 수 있는 '충돌(conflict)[2]' 문제를 해결하는 정책으로는 '거절 우선 정책(denial take precedence)'을 일반적으로 사용한다. 또한, 긍정적(positive) 권한부여와 부정적(negative) 권한부여를 혼합하여 사용하기 때문에 명시적인 권한부여가 없는 노드에 대한 '결정 정책(decision policy)'로 명시적 권한부여가 없는 노드는 접근을 허용하는 '개방(open) 정책'과 접근을 불허하는 '폐쇄(closed) 정책'을 사용한다.

일반적으로, 엄격한 데이타 보안을 위해 '거절 우선 정책'과 '폐쇄 정책'을 사용한다.

가정 3. 본 논문에서의 접근제어 정책은 '묵시적 권한부여' 기반의 '가장 특정화된 것을 우선으로 하는 정책'을사용하며 충돌 해결 정책으로 '거절 우선 정책' 및 긍정적/부정적 권한부여 혼용을 위해 '폐쇄 정책'을 사용한다.

XML 질의 언어와 XML 접근제어 방법에 XPath 표현식을 이용하고 가정 3을 접근제어 정책으로 하는 XML 접근제어 모델은 접근제어 규칙들간의 새로운 무결성 규칙이 존재하는데 본 논문에서는 다음과 같이 정의한다.

1) 한 노드에 대한 접근제어 결과가 그 노드의 후손에 전파되는지 여부를 결정하는 정책을 말한다.

2) 충돌이라 함은 어떤 노드가 '접근허용' 및 '접근불가' 규칙이 동시에 정의되어 있는 경우를 말한다.

정의 1. [접근제어 규칙의 무결성]:

① 한 사용자에 대해 임의의 노드에 동시에 긍정적 접근제어 규칙과 부정적 접근제어 규칙을 기술할 수 없다.

② 만약 긍정적 접근제어 규칙과 부정적 접근제어 규칙이 같이 존재한다면 부정적 접근제어 규칙이 우선한다.

③ '접근불허 하향 일관성(denial downward consistency)' 원칙을 따른다 [11].

④ 부정적 접근제어 규칙은 반드시 긍정적 접근제어 규칙의 범위 내에 기술되어야만 한다.

접근제어 규칙의 무결성 ①과 ②는 '거절 우선 정책'의 산물이다. 접근제어 규칙의 무결성 ③과 ④는 XML 데이타에 대한 접근제어를 수행하기 위해 추가된 규칙이다.

XML 데이타에 긍정적/부정적 권한부여 정책을 혼용하는 경우를 상기해 보면, 데이타를 접근하는데 있어 부정적 권한부여는 긍정적 권한부여를 보충하는데 사용한다. 즉, 계층적 구조를 가지고 있는 데이타 모델에서 긍정적 권한부여 범위 안에서의 부정적 권한부여는 데이타의 접근 단위를 더욱 최소화할 수 있는 방법론이다. [11]에서는 이를 '접근불허 하향 일관성' 원칙이라 정의하고 있다. 유사한 방법으로 부정적 권한부여 범위 안에서의 긍정적 권한부여 방법을 생각해 볼 수 있다. 그러나, 이 방법은 두 가지 문제점을 초래한다. 본 논문에서는 이를 '불법적인 판독 접근성'(invalid read accessibility) 문제와 '보이지 않는 조상'(invisible ancestor problem) 문제로 정의한다.

정의 2. [불법적인 판독 접근성]:

계층적 데이타 구조를 갖고 있는 XML 데이타 모델에서 긍정적 권한부여, 부정적 권한부여, 그리고 긍정적 권한부여의 경로에 대한 질의시 부정적 권한부여가 되어 있는 노드에 대한 판독이 이루어지게 되는데 이를 '불법적인 판독 접근성'이라 한다.

정의 3. [보이지 않는 조상 문제]:

계층적 데이타 구조를 갖고 있는 XML 데이타 모델에서 긍정적 권한부여, 부정적 권한부여, 그리고 긍정적 권한부여의 경로에 대한 질의에 대한 결과를 계층적 데이타 구조로 표현하였을 때 부정적 권한부여가 되어 있는 노드는 보이지 말아야 한다. 이때, 질의자가 결과를 보고 갱신 연산을 수행하였을 때 부정적 권한부여가 되어 있는 노드에 대한 계층적 구조 처리 문제가 발생하는데 이를 '보이지 않는 조상 문제'라 한다.

그림 1(a)는 '불법적인 판독 접근성'에 대한 예를 설명하고 있다. 질의자가 //a//와 같은 질의를 던졌을 때, 그 질의자에게 접근불허된 노드 b를 어떻게 처리하는가에 대한 문제가 발생한다. 그림 1(b)는 '보이지 않는 조

상 문제'에 대한 예를 설명하고 있다. 원천 XML 문서의 경로는 //a/b/c/d 이고 임의의 사용자에게 노드 b는 접근불허인 접근제어 규칙이 있어 이 사용자에게 보여지는 문서의 구조는 //a/c/d인 경우, 이 사용자가 새로운 인스턴스로 //a/c/d 경로를 추가하려고 한다면 원천 XML 문서에 어떻게 반영을 해야하는가 하는 문제가 발생한다.

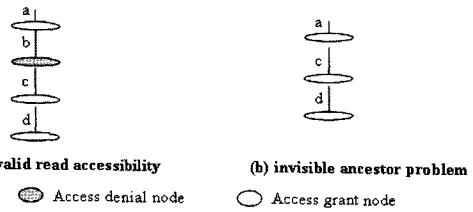


그림 1 긍정적/부정적 권한부여 혼용에 따른 문제

따라서, 본 논문에서는 접근제어 규칙을 생성하는데 있어서 접근불허 노드의 자손(자식 포함) 노드에는 접근허용 규칙이 존재하지 않음을 가정한다. 이것이 '접근불허 하향 일관성' 무결성 규칙이다. 또한, '폐쇄정책'을 가정하고 있기 때문에 '전파정책'에 의해서도 접근권한이 존재하지 않는 객체(object)에 대해 주체는 접근할 수 없다. 이런 이유로 부정적 접근제어 규칙은 긍정적 접근제어 규칙 범위 안에서 접근불허 부분을 정하는 것이 바람직한 방법이다. 이것이 '접근제어 규칙의 무결성' 네 번째 규칙이다.

3.4 문제 정의

본 논문에서 제안하고 있는 '안전한 XML 접근제어 필터 시스템'은 질의자의 임의의 XML 문서에 대해 질의를 하였을 때, 그 질의자의 접근제어 규칙들을 살펴 사용자의 질의를 접근제어 정책에 어긋나지 않는 질의로 변형하여 질의 처리기에서 변형된 질의를 넘겨주는 작업을 수행한다. 따라서, 긍정적 접근제어 규칙(ACR^+)과 부정적 접근제어 규칙(ACR^-)이 존재했을 때, 사용자 질의(Q)에 따른 변형된 질의(Q')는 다음과 같은 수식으로 정리할 수 있다.

$$Q' = Q \text{ INTERSECT } (ACR+ \text{ EXCEPT } ACR-)$$
$$= (Q \text{ INTERSECT } ACR+) \text{ EXCEPT } (Q \text{ INTERSECT } ACR-)$$

XPath 2.0[2]에서는 집합 연산(즉, UNION, INTERSECT, 그리고 EXCEPT)을 지원한다. 비록 이들 연산이 기술적으로 경로 표현식은 아니지만, 경로 표현식들 사이에 일정한 방식으로 사용된다. 본 논문에서는 이들 집합 연산을 이용하여 불안전한 사용자 질의를 안전한 질의로 변형하는데 유용하게 이용하고자 한다. XML spy 2006 버전에 이들 집합 연산이 지원되고 있음을 확인하였다 (http://www.altova.com).

이런 환경에서 본 논문에서 제기하는 문제점은 다음 두 가지이다:

- 첫째, 사용자에 대한 많은 접근제어 규칙들 중 질의와 관련된 접근제어 규칙만을 추출하는 방법
- 둘째, '*' 및 '//'을 포함하고 있는 사용자 질의 및 접근제어 규칙을 비교하여 효율적으로 정확한 XPath 경로 표현식(변형된 질의)을 생성하는 방법

본 논문은 유효한(valid) XML 문서에 대한 접근제어 기법을 가정하고 있다. 그리고, 보다 빠른 접근제어 수행을 위해 XML 스키마(본 논문에서는 DTD를 가정)에 대한 메타데이타를 이용하고 있다. 다음 절에서는 XML 스키마에 대한 메타데이타에 대한 설명이다.

3.5 PRE/POST 구조 (PRE/POST Structure)

XML 데이타 상에서 주어진 질의에 대한 결과는 접근제어 정책에 위반되는 데이타를 포함해서는 안 된다. [13]에서는 이를 '안전한 결과(safe answer)'라 정의하고 있다.

정의 4. [안전한 질의(safe query)]

안전한 결과를 나타내는 질의를 안전한 질의라 정의한다.

XPath[2] 기반의 질의는 XPath 경로 표현식에서 가장 마지막 노드의 하부-트리(sub-trees)를 그 결과로 나타낸다[13,14]. 특히, [13]에서는 이를 '하부-트리 결과(answer-as-subtrees)'라 정의하고 있다.

정의 5. [목적 노드(target node)]

XML 질의 언어 혹은 XML 접근제어 규칙에 사용된 XPath 경로 표현식의 목적 노드는 프레디키트(predicate)를 제외한 경로 표현식의 가장 마지막 노드이다.

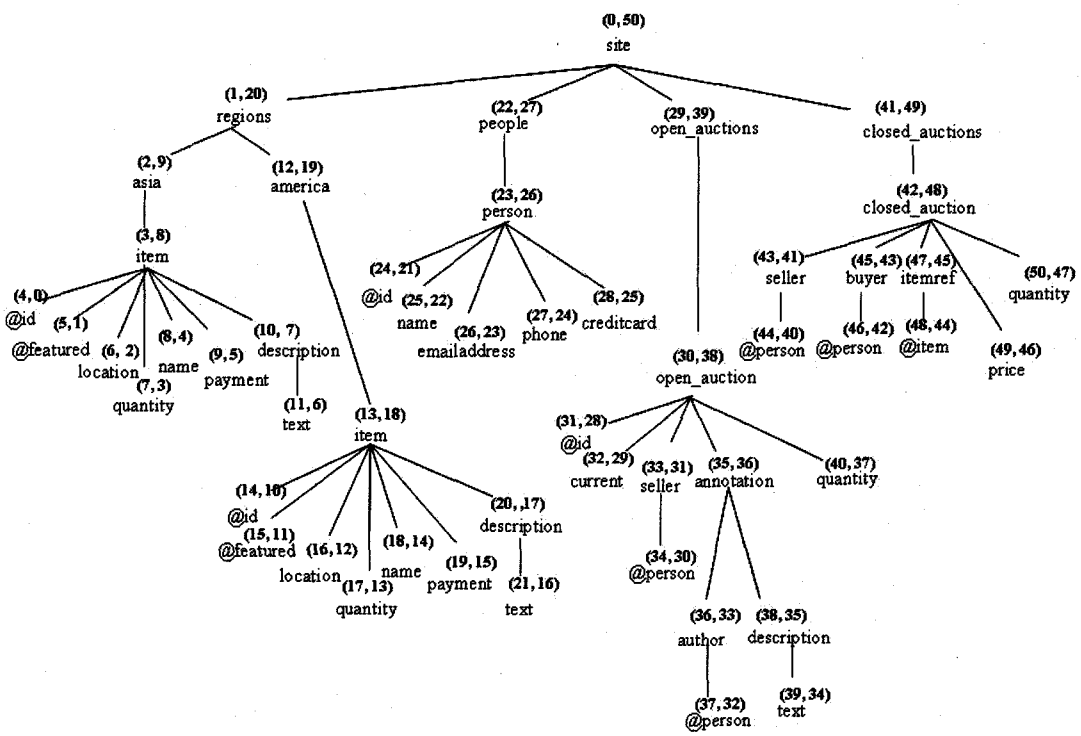
예를 들어, '/site/people/person'의 목적 노드는 person 노드이다. 또 다른 예로, '/site/regions/aisa/item [@id = "xxx"]'의 목적 노드는 item 노드이다.

그림 2(a)는 XMark[15]로부터 인용된 앞으로 논문에서 예제로 사용하는 auction.dtd를 보여주고 있다. 일반적으로 인터넷 옥션 사이트는 지역(region), 사람(people), 경매중인 내용(open-auctions) 그리고 경매종료 내용(closed_auctions)을 가지고 있다. 그림 2(b)는 DTD를 순차적으로 접근하여 얻은 PRE(order) 그리고 POST(order) 값이 부여된 DTD 트리를 나타낸다. 트리 구조에서 preorder와 postorder 값을 기술하는 방법은 XML 문서를 트리 구조로 만들어 XML 질의 언어를 위한 질의 처리기[10]에서 유용하게 사용하는 방법으로 본 논문에서는 이 방법을 DTD 트리에 인용하고 있으며 이를 **PRE/POST 구조**(PRE/POST Structure: PPS)라 한다. 그림 2(c)는 본 논문에서 제안하고 있는 SQ-Filter 시스템에서 구현한 DTD 트리에 대한 관계

```
<!ELEMENT site                (regions, people, open_auctions, closed_auctions)>
<!ELEMENT regions             (asia, america)>
<!ELEMENT asia                (item*)>
<!ELEMENT america             (item*)>
<!ELEMENT item                (location, quantity, name, payment, description)>
<!ATTLIST item                id ID #REQUIRED
                              featured CDATA #IMPLIED>
<!ELEMENT location            (#PCDATA)>
<!ELEMENT quantity            (#PCDATA)>
<!ELEMENT name                (#PCDATA)>
<!ELEMENT payment             (#PCDATA)>
<!ELEMENT description         (text)>
<!ELEMENT text                (#PCDATA)>
<!ELEMENT people              (person*)>
<!ELEMENT person              (name, emailaddress, phone?, creditcard?)>
<!ATTLIST person              id ID #REQUIRED>
<!ELEMENT emailaddress        (#PCDATA)>
<!ELEMENT phone               (#PCDATA)>
<!ELEMENT creditcard          (#PCDATA)>
<!ELEMENT open_auctions       (open_auction*)>
<!ELEMENT open_auction        (current, seller, annotation, quantity)>
<!ATTLIST open_auction        id ID #REQUIRED>
<!ELEMENT current             (#PCDATA)>
<!ELEMENT seller              EMPTY>
<!ATTLIST seller              person IDREF #REQUIRED>
<!ELEMENT annotation          (author, description?)>
<!ELEMENT author              EMPTY>
<!ATTLIST author              person IDREF #REQUIRED>

<!ELEMENT closed_auctions     (closed_auction*)>
<!ELEMENT closed_auction      (seller, buyer, itemref, price, quantity)>
<!ELEMENT buyer               EMPTY>
<!ATTLIST buyer               person IDREF #REQUIRED>
<!ELEMENT itemref             EMPTY>
<!ATTLIST itemref             item IDREF #REQUIRED>
<!ELEMENT price               (#PCDATA)>
```

(a)



(b)

Tag-Name	PRE	SIZE	LEVEL	POST	PARENT
site	0	50	0	50	
regions	1	20	1	20	site
asia	2	9	2	9	regions
item	3	8	3	8	asia
@id	4	0	4	0	item
@featured	5	0	4	1	item
location	6	0	4	2	item
quantity	7	0	4	3	item
name	8	0	4	4	item
payment	9	0	4	5	iem
description	10	1	4	7	item
text	11	0	5	6	description
...	...	...	...	...	...
people	22	6	1	27	site
person	23	5	2	26	people
@id	24	0	3	21	person
name	25	0	3	22	person
emailaddress	26	0	3	23	person
phone	27	0	3	24	person
creditcard	28	0	3	25	person

Tag-Name	PRE	SIZE	LEVEL	POST	PARENT
open_auctions	29	11	1	39	site
open_auction	30	10	2	38	open_auctions
@id	31	0	3	28	open_auction
current	32	0	3	29	open_auction
seller	33	1	3	31	open_auction
@person	34	0	4	30	seller
annotation	35	4	3	36	open_auction
author	36	1	4	33	annotation
@person	37	0	5	32	author
description	38	1	4	35	annotation
text	39	0	5	34	description
quantity	40	0	3	37	open_auction
closed_auctions	41	9	1	49	site
closed_auction	42	8	2	48	closed_auctions
seller	43	1	3	41	closed_auction
@person	44	0	4	40	seller
buyer	45	1	3	43	closed_auction
...	...	...	...	...	...
quantity	50	0	3	47	closed_auction

(c)

그림 2 (a) auction.dtd, (b) auction.dtd의 PRE/POST 구조, (c) PRE/POST 구조의 관계형 저장 형태

형 저장 구조(실제는 해쉬 테이블로 구현함)를 나타내고 있다. LEVEL은 DTD 트리에서의 레벨을 의미하며, SIZE는 임의의 트리 노드에서의 하부-트리의 노드 수를 나타낸다. POST 값은 다음과 같은 수식을 통해 얻는다:

$$POST = PRE + SIZE - LEVEL \quad [10] \qquad (1)$$

식 (1)에 의해 PRE(order)와 POST(order)는 DTD 트리에 대해 한번의 스캔 작업으로 얻게 된다. PARENT는 임의의 노드에서의 부모 노드를 나타낸다. PPS에 대한 활용은 4장에서 자세히 설명한다.

4. SQ-Filter 시스템

SQ-Filter 시스템의 목적은 사용자 질의를 처리하기 위해 필요한 접근제어 규칙들만을 선택하여 불안전한 사용자 질의를 자신의 접근제어 정책에 맞는 안전한 질의로 재작성을 하는 것이다. 이번 장에서는 SQ-Filter 시스템의 구조를 설명하고 각각의 핵심 컴포넌트들을 설명한다.

4.1 SQ-Filter 시스템 구조

사용자의 질의를 입력으로 받는 SQ-Filter 시스템은 XML 데이타베이스 시스템에 사용자의 접근제어 정책에 어긋나는 질의(denial), 접근제어 정책을 완전히 준수하는 질의(acceptatnce), 일부분만이 접근제어 정책을 준수하는 질의(rewriting) 인가를 판단하고, 만약 일부분만이 접근제어 정책을 준수하는 질의인 경우 그 불안전한 부분을 제거하여 안전한 질의로 변경하여 최종적으로 XML 질의 처리기에 질의를 넘겨주는 작업을 한다. 이와 같은 처리 개념은 QFilter[13]와 유사하다. 그림 3은 SQ-Filter 시스템의 구조를 보여주고 있다.

SQ-Filter 시스템은 크게 두 부분으로 나뉜다. 컴파일 시간에 PPS 정보를 구축하는 부분과 접근제어 규칙에 대한 데이타베이스를 구축하는 부분(3.5 절), 그리고 런타임 시간에 사용자 질의를 분석(QUERY ANALYZER)하고 필요한 접근제어 규칙을 찾고(ACR-FINDER), 최종적으로 사용자 질의를 안전한 질의로 변경하는(QUERY EXECUTOR) 컴포넌트들이다. 이들에 대해서는 다음 절부터 자세히 설명하고, 이번 절에서는 접근제어 규칙에 대한 데이타베이스 부분을 설명한다.

일단 보안 관리자(혹은 정책 관리자)에 의해 그림 4와 같은 사용자(혹은 역할)별 접근제어 규칙들이 만들어지면, ACRs Converter 컴포넌트는 접근제어 규칙 명세의 객체 부분에 대한 XPath 경로 표현식에서 경로에 대한 정보와 프레디키트에 대한 부분을 그림 5처럼 데이타베이스화 한다. 예를 들어, 접근제어 규칙 R1의 목적 노드는 item이다. DTD 트리 해쉬 테이블에서 item에 대한 (PRE, POST) 값을 찾는다((3, 8)과 (13, 18)). 이들을 접근제어 규칙(ACRs) 테이블의 PRE와 POST 컬럼의 값으로 저장한다. 추가로 R1은 [location='xxxx'] 프레디키트를 갖고 있기 때문에 *PREDICATES* 테이블의 Parent-PRE 컬럼에 (3, 13) 값,

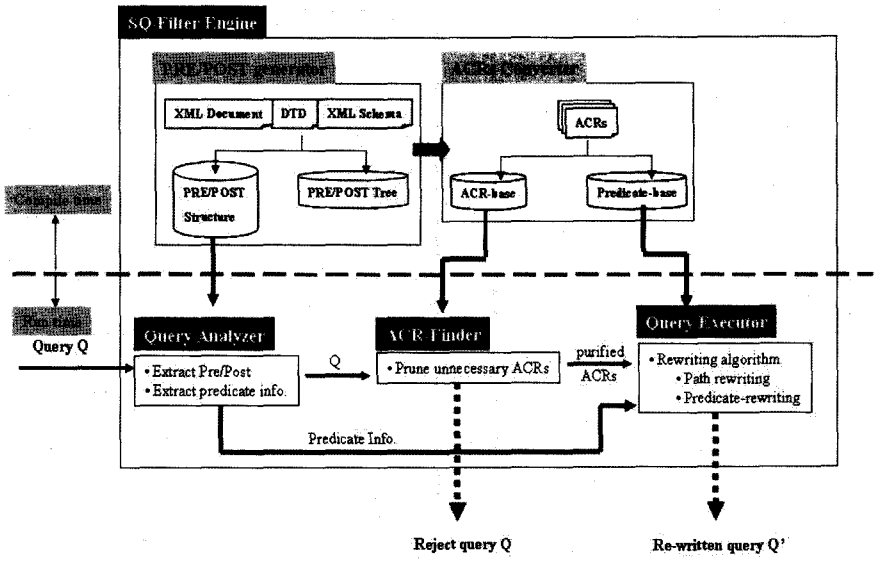


그림 3 SQ-Filter 시스템 구조

Positive ACRs

(R1): /site/regions/*/item[location="LA"]

(R2): /site/people/person[name = "chang"]

(R3): /site/open_auctions/open_auction

(R4): //open_auction[quantity]/seller

Negative ACRs

(R5): /site/regions/*/item/payment

(R6): /site/people/person/creditcard

(R7): /site/*/open_auction[@id>50]/seller[@person="chang"]

그림 4 긍정적/부정적 접근제어 규칙 예제

ACR' base

rule	path	Pre	Post	P_link
R1	/site/regions/*/item	3, 13	8, 18	P1
R2	/site/people/person	23	26	P2
R3	/site/open_auctions/open_auction	30	38	
R4	//open_auction[quantity]/seller	33	31	P3

ACR' base

rule	path	Pre	Post	P_link
R5	/site/regions/*/item/payment	9, 19	5, 15	
R6	/site/people/person/creditcard	28	25	
R7	/site/*/open_auction[@id>50]/seller[@person="chang"]	33	31	P4

PREDICATES base

P-id	Parent-PRE	property	operator	value
P1	3, 13	location	=	LA
P2	23	name	=	chang
P3	33	quantity		
p4	33	@person	=	chang

그림 5 ACRs 테이블과 PREDICATES 테이블 예제

property, operator, 그리고 value 컬럼 각각에 'location', '=', 그리고 'xxxx' 값을 저장한다. 마지막으로 프레디키트 아이디 P1을 *ACRs* 데이타베이스의 P_link 컬럼 값으로 저장한다. 다른 접근제어 규칙들도 유사하게 *ACRs* 데이타베이스와 *PREDICATES* 데이타베이스에 저장된다.

4.2 QUERY ANALYZER 컴포넌트

QUERY ANALYZER(QA)의 목적은 사용자 질의로부터 정보(목적 노드의 (PRE, POST) 값, 그리고 프레디키트 정보)를 얻는 것이다.

Q1: /site/people/person[name="chang"]/phone/

사용자 질의 Q1에 대해 일단 목적 노드 phone을 확인한다. 그런 후, QA는 PRE/POST 구조(그림 2(c) 참조)로부터 목적 노드 phone에 대한 (PRE, POST) 쌍인 (27, 24)를 얻는다. 또한, Q1의 프레디키트 정보를 얻는다. 질의에 따라 (PRE, POST) 쌍은 집합이 될 수도 있다.

사용자 질의가 다음과 같다고 가정하자.

/site/regions/america/item

목적 노드는 item이고, PPS를 통해 (3, 8)과 (13, 18)을 얻게 된다. 만약 america 노드 대신 "*" 노드였다면 둘 다 질의에 대한 목적 노드의 (PRE, POST) 쌍이 되겠지만, (13, 18)만이 america 노드의 자식 노드의 (PRE, POST) 쌍이 된다. 즉, 목적 노드의 (PRE, POST) 쌍이 집합인 경우, 불필요한 (PRE, POST) 쌍을 제거하는 핵심 개념은 질의의 각 노드의 preorder (postorder) 값은 목적 노드의 preorder(postorder) 값보다 작아야(커야) 한다. 그림 6은 이에 대한 알고리즘을 보여준다.

```
Input: a user's query
Output: suitable (PRE, POST) values of target node of the query
BEGIN
1. for each (Pr_tn, Po_tn) value of the target node of the query
2. {    for (Pr_step, Po_step) value of each node of the query
3.          if (!(Pr_step < Pr_tn and Po_step > Po_tn))
4.                  break;
5.      suitable (PRE, POST) set ← (Pr_tn, Po_tn)
6. }
END
```

그림 6 Prune-TNs 알고리즘

4.3 ACR-FINDER 컴포넌트

ACR-Finder(ACR-F) 컴포넌트의 목적은 사용자의 접근제어 규칙들 중 QA의 질의 정보를 기반으로 질의와 관련된 접근제어 규칙들만을 추출하는 것이다. 4.2절에서 제시한 질의 Q1을 다시 예를 들면, QA에 의해 목적 노드 phone에 대한 (27, 24) 값을 얻는다. 그림 7(a)에서 보듯이, 접근제어 규칙들에 대한 PRE/POST 이차평면에 목적 노드 (27, 24) 점을 기준으로 네 개의 사분면이 형성된다.

각 사분면은 질의와 접근제어 규칙 간의 특별한 관계를 의미한다. 이들 관계를 XPath의 목적 노드에 대한 (PRE, POST) 쌍의 값 비교로 다음과 같이 정의한다. 정의에 앞서, 수식에 표현되는 용어를 다음과 같이 가정한다. 질의 XPath(Q), 접근제어 규칙 XPath(ACR)와 변형된 질의 XPath(Q')의 목적 노드에 대한 (PRE, POST) 쌍을 각각 (Pr_Q, Po_Q), (Pr_{ACR}, Po_{ACR}), 그리고 ($Pr_{Q'}$, $Po_{Q'}$)라 하자.

정의 6. [1사분면: FOLLOWING 규칙] 임의의 XML 문서에 대해 접근제어 규칙에 표현된 XPath에 의한 영역은 질의 XPath의 영역에 대해 FOLLOWING 영역으로써, 이와 같은 접근제어 규칙을 질의에 대한 FOLLOWING 규칙이라 정의한다.

$$Pr_Q < Pr_{ACR},\ Po_Q < Po_{ACR} \quad (2)$$

정의 7. [2사분면: ANCESTOR 규칙] 임의의 XML 문서에 대해 접근제어 규칙에 표현된 XPath에 의한 영역은 질의 XPath의 영역에 대해 ANCESTOR 영역(PARENT도 포함)으로써, 이와 같은 접근제어 규칙을 질의에 대한 ANCESTOR 규칙이라 정의한다.

$$Pr_Q > Pr_{ACR},\ Po_Q < Po_{ACR},\ Pr_{Q'} = Pr_Q,\ Po_{Q'} = Po_Q \quad (3)$$

정의 8. [3사분면: PRECEDING 규칙] 임의의 XML 문서에 대해 접근제어 규칙에 표현된 XPath에 의한 영역이 질의 XPath의 영역에 대해 PRECEDING 영역으로써, 이와 같은 접근제어 규칙을 질의에 대한 PRECEDING 규칙이라 정의한다.

$$Pr_Q > Pr_{ACR},\ Po_Q > Po_{ACR} \quad (4)$$

정의 9. [4사분면: DESCENDANT 규칙] 임의의 XML 문서에 대해 접근제어 규칙에 표현된 XPath에 의한 영역이 질의 XPath의 영역에 대해 DESCENDANT 영역(CHILD도 포함)으로써, 이와 같은 접근제어 규칙을 질의에 대한 DESCEDANT 규칙이라 정의한다.

$$Pr_Q < Pr_{ACR},\ Po_Q > Po_{ACR},\ Pr_{Q'} = Pr_{ACR},\ Po_{Q'} = Pr_{ACR} \quad (5)$$

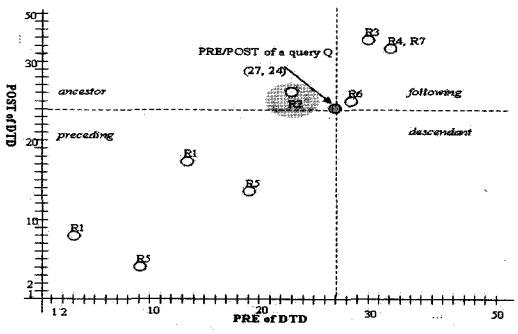
정의 10. [SELF 규칙] 임의의 XML 문서에 대해 접근제어 규칙에 표현된 XPath에 의한 영역이 질의 XPath의 영역에 대해 SELF 영역으로써, 이와 같은 접근제어 규칙을 질의에 대한 SELF 규칙이라 정의한다.

$$Pr_{Q'} = Pr_Q = Pr_{ACR},\ Po_{Q'} = Po_Q = Po_{ACR} \quad (6)$$

여기서 질의와 관련된 사분면은 2사분면(ANCESTOR 규칙)과 4사분면(DESCENDANT 규칙) 만이고 추가로 SELF 규칙도 질의와 관련된 접근제어 규칙이다. 즉, 예제에서는 R1에서 R7 중에 R2 만이 질의와 관련된 접근제어 규칙이 된다. 그림 7(b)는 이와 같이 사용자 질의에 대한 ANCESTOR 규칙, DESCENDANT 규칙 혹은 SELF 규칙에 해당되는 접근제어 규칙을 찾는 알고리즘이다.

정리 1. ACR-FINDER 알고리즘에 의해 추출된 접근제어 규칙들은 사용자 질의와 관련된 그 사용자의 모든 접근제어 규칙들이다.

ACR-FINDER 알고리즘은 ANCESTOR 규칙(정의



(a)

Input: (Pr_Q, Po_Q) := QA(query), ACRs
Output: suitable ACRs'
BEGIN
for each rule R1 in ACRs
  if (($Pr_Q \geq Pr_{R1}$ and $Po_Q \leq Po_{R1}$) *or* // *ACENSTOR*
    ($Pr_Q \leq Pr_{R1}$ and $Po_Q \geq Po_{R1}$)) // *DESCENDANT*
    ACRs' ← R1;
END

(b)

그림 7 (a) 접근제어 규칙에 대한 PRE/POST 이차평면의 의미, (b) ACR-FINDER 알고리즘

7), DESCENDANT 규칙(정의 9), 그리고 SELF 규칙(정의 10)을 추출하는 알고리즘이다. 따라서, 나머지 FOLLOWING 규칙과 PRECEDING 규칙이 질의와 관련이 없음을 증명하면 된다.

증명. 사용자 질의 Q에 대한 목적 노드의 (PRE, POST) 쌍을 (Pr_{tn}, Po_{tn})라 하자. 따라서, 이 질의에 대한 영역의 임의의 노드 sn에 대한 (PRE POST) 쌍은 다음과 같은 값을 갖게 된다.

$$Pr_{tn} < Pr_{sn} < Pr_{tn} + SIZE\ (tn),\ Po_{last\_sibling\_node\_of\_tn} < Po_{sn} < Po_{tn} \quad (7)$$

그리고, 질의 Q에 대한 목적 노드에 대한 임의의 following 노드(FN)들은 다음과 같은 수식을 만족한다.

$$Pr_{tn} < Pr_{FN},\ Po_{tn} < Po_{FN} \quad (8)$$

특히, 첫 번째 following 노드의 PRE(order) 값은 다음과 같다.

$$Pr_{first\_FN} = Pr_{tn} + SIZE(tn) + 1 \quad (9)$$

식 (7)과 (8)에 의해, POST(order) 값은 $Po_{sn} < Po_{tn} < Po_{FN}$ 인 관계를 나타내기 때문에 질의의 영역의 일부의 노드가 following 접근 규칙 영역에 존재하는가만 살펴보면 된다. 식 (7)과 (9)에 의해 다음의 수식이 유추된다:

$$Pr_{sn} < Pr_{tn} + SIZE\ (tn) < Pr_{first\_FN} = Pr_{tn} + SIZE(tn) + 1 \quad (10)$$

식 (10)의 의미는 질의 영역의 임의의 노드(sn)는 질의의 목적 노드(tn)에 대한 첫 번째 following 노드(first_FN)의 PRE(order) 값보다 작다는 것이다. 결국, 첫 번째 following 노드보다 작다는 것은 질의의 목적 노드에 대한 임의의 following 노드보다 작다는 것을 의미한다. 따라서, 질의 영역의 임의의 노드는 following 접근 규칙 영역에 포함되지 않음을 알 수 있다.

PRECEDING 접근제어 규칙 역시 유사한 방법으로 증명된다.

4.4 QUERY EXECUTOR 컴포넌트

QUERY-EXECUTOR(QE)의 목적은 사용자 질의와 그 질의와 관련된 접근제어 규칙들과 XPath 2.0의 집합 연산을 통해 안전한 질의로 변경하는 것이다.

ACR-F를 통과한 사용자 질의는 긍정적 접근제어 규칙과 부정적 접근제어 규칙 이렇게 두 가지 그룹으로 나뉘고 각 그룹은 SELF 접근제어 규칙, ANCESTOR 접근제어 규칙, 그리고 DESCENDANT 접근제어 규칙 이렇게 세 가지의 규칙으로 나뉜다. QE의 처리 과정은 다음과 같다:

1. 사용자 질의와 각각의 부정적 접근제어 규칙 간의 교집합 질의를 만든다.
2. UNION 연산자를 통해 이들 교집합 질의를 합친다.
3. 사용자 질의와 각각의 긍정적 접근제어 규칙 간의 교집합 질의를 만든다.
4. UNION 연산자를 통해 이들 교집합 질의를 합친다.
5. EXCEPT 연산을 통해 4번의 결과에 2번의 결과를 합친다.
6. 5번의 결과 질의를 질의 처리기에 보낸다.

자세한 설명에 앞서, 다음과 같은 두 개의 질의를 가정한다:

(Q2): /site/people/person/creditcard/

(Q3): //open_auction[@id<100]

QE의 수행과정을 설명하기 전에 처리과정 1과 3의 교집합 질의를 만들면서 '*' 를 원천 노드로 만들고, '//' 축을 일련의 '/' 축으로 만드는데 이용되는 *REFINE* 함수와 질의 XPath 및 접근제어 규칙 XPath의 프레디키트를 처리하는 *PREDICATE* 함수를 설명한다.

4.4.1 REFINE 함수와 PREDICATE 함수

REFINE 함수는 PPS(그림 2(c)) 정보를 통해 변경되는 안전한 질의의 '*' 노드를 실제 노드로 바꾸고 "//" 축을 일련의 "/" 축으로 이루어진 단일 경로로 바꾸는 함수이다.

예를 들어, 변경되는 질의가 다음과 같다고 하자:

/site/regions/*/item

목적 노드 item의 (PRE, POST) 쌍은 (3, 8)과 (13, 18) 이므로, *REFINE* 함수는 첫 번째로 (3, 8)에 대해 수행된다. 역 순서로, "*" 노드를 만나면, PPS를 통해 그 전의 노드 (item)의 (3, 8)의 부모 노드 aisa를 얻는다. 더 이상의 "*" 노드나 "//" 축이 없으면 /site/regions/asia/item를 리턴한다(그림 8 알고리즘의 (5) - (9) 줄). 그 다음 (13, 8)에 대해 같은 방법을 수행하고 /site/regions/america/item를 리턴한다.

이번에는 "//" 축을 갖고 있는 경우이다: /site/people//name

비록 목적 노드 name의 (PRE, POST) 쌍은 (8, 4), (18, 14) 그리고 (25, 22)이지만, 그림 6의 Prune_TNs 알고리즘에 의해 (25, 22)만이 선택된다. *REFINE* 함수가 "//" 축을 만나면, "//" 축의 전 노드 (name)과 다음 노드 (people)를 얻는다. PPS 정보를 통해 각 노드의 LEVEL 값을 비교한다. $LEVEL_{name}$ - $LEVEL_{people}$ = 2이기 때문에 "//" 축은 people/*/name으로 교체된다. 다시 "*" 노드를 처리하면 최종적으로 /site/people/person/name 경로 표현식을 리턴한다. 두 노드의 LEVEL 차가 1이면 두 노드는 parent/child 관계가 되어 "//" 축은 바로 "/" 축으로 교체된다. 만약 LEVEL 차가 3 이상이면, 전 노드의 부모 노드를 PPS에서 찾고 다시 부모 노드와 "//" 다음 노드에 대해 *REFINE* 함수를 재호출한다 (그림 8 알고리즘의 (18) - (25) 줄).

PREDICATE 함수는 질의 재작성 과정에서 원래 질의의 프레디키트 정보와 접근제어 규칙의 프레디키트 정보를 변경되는 질의에 유지하는 함수이다. 접근제어 규칙에 포함된 프레디키트의 위치 정보는 그림 5의 *PREDICATES* 데이타베이스에 저장되어 있고, 사용자 질의의 프레디키트는 4.2절의 QA에서 추출한다. QE에 의해 *REFINE* 함수가 호출되고 *REFINE* 함수는 최종적으로 변형된 질의를 만들 때 *PREDICATE* 함수를 호출하여 접근제어 규칙에 있던 프레디키트 정보를 추가한다. 그림 9는 변형된 질의에 접근제어 규칙에 있는 프레디키트를 추가하는 알고리즘을 나타내고 있다. *PREDICATE* 함수의 핵심은 내부에서 호출하는 *MERGE* 함수이다. *MERGE* 함수는 최종 변형된 질의에 접근제어 규칙의 프레디키트를 추가할 때, 같은 경로 위치에 사용자의 프레디키트와 접근제어 프레디키트가

Input: PPS, an original XPath, and (Pr_p, Po_p) of the target node of the XPath
Output: Refinedpath //Refined XPath expression
BEGIN
1. Refinedpath := null string;
2. for reverse order of path
3. { if node is not * and $ // $ means "//" axis
4. Refinedpath := concatenate("/node", Refinedpath);
5. else if node is *
6. { find ($Pr_{before\text{-}node}$, $Po_{before\text{-}node}$) value set before "*" node;
7. for each ($Pr_{before\text{-}node}$, $Po_{before\text{-}node}$) value
8. if (($Pr_{before\text{-}node} \le Pr_p$) and ($Po_{before\text{-}node} \ge Po_p$))
9. Refinedpath := concatenate(Get_parentnode($Pr_{before\text{-}node}$, $Po_{before\text{-}node}$), Refinedpath));
10. }else if node is $ // $ means "//" axis
11. { while(1)
12. { find ($Pr_{before\text{-}node}$, $Po_{before\text{-}node}$) value set before "//" node;
13. for each ($Pr_{before\text{-}node}$, $Po_{before\text{-}node}$) value
14. if (($Pr_{before\text{-}node} \le Pr_p$) and ($Po_{before\text{-}node} \ge Po_p$))
15. { next-node := get the next node of "//" node
16. find ($Pr_{next\text{-}node}$, $Po_{next\text{-}node}$) value set of next-node;
17. for each ($Pr_{next\text{-}node}$, $Po_{next\text{-}node}$) value
18. if (($Pr_{next\text{-}node} \le Pr_p$) and ($Po_{next\text{-}node} \ge Po_p$))
19. if($Level_{before\text{-}node} - Level_{next\text{-}node} = 1$)
20. pass;
21. else if($Level_{before\text{-}node} - Level_{next\text{-}node} = 2$)
22. Refinedpath := concatenate(Get_parentnode($Pr_{before\text{-}node}$, $Po_{before\text{-}node}$), Refinedpath));
23. else {
24. Refinedpath := concatenate(Get_parentnode($Pr_{before\text{-}node}$, $Po_{before\text{-}node}$), Refinedpath));
25. break while;
26. }
27. } } } }
28. return RefinedQuery;
END

그림 8 *REFINE* 함수 알고리즘

```
Input  : parent-node, PREp, POSTp, ACR_P_link, Query_P_link
Output : RefinePredicate
BEGIN
1. ACR_P_link := ACR-predicate index set
2. Query_P_link := Qeury-predicate index set
3. find (PRE, POST) of parent-node by (Prp, Pop)
4. for each predicate-data in ACR-predicate index set
5. { get predicate-data from PREDICATES database
6.     if(PRE-parent = Pr predicate-data)
7.         if(type of predicate-data is simplepredicate)
8.             ACR_simple_predicate = predicate-data;   // array
9.         else if(type of predicate data is pathpredicate)
10.            ACR_path_predicate = pathpredicate; // array
11. }
12. for each predicate-data in query-predicate index set
13. {   get predicate-data from PREDICATES database
14.     if(PRE-parent = Pr predicate-data)
15.         if(type of predicate data is simplepredicate)
16.             query_simple_predicate = predicate-data;   // array
17.         else if(type of predicate data is pathpredicate)
18.             query_path_predicate = pathpredicate; // array
19. }
20. if(query_path_predicate != null && ACR_path_predicate != null)
21.         Refinedpredicates := concatenate path_predicate by merge function;
22. else if(query_path_predicate != null)
23.         Refinedpredicates := concatenate query_path_predicate;
24. else if(ACR_path_predicate != null)
25.         Refinedpredicates := concatenate ACR_path_predicate;
26. if(query_ simple _predicate != null && ACR_ simple_predicate != null)
27.          Refinedpredicates := concatenate simple _predicate by merge function;
28. else if(query_ simple _predicate != null)
29.         Refinedpredicates := concatenate query_ simple _predicate;
30. else if(ACR_ simple _predicate != null)
31.         Refinedpredicates := concatenate ACR_ simple _predicate;
END
```

그림 9 PREDICATE 함수 알고리즘

질의의 프레디키트	부정적 접근제어의 프레디키트	긍정적 접근제어의 프레디키트	최적화
[@id = "1"]	[@id = "1"]		[@id = "1"]
[@id = "1"]	[@id = "3"]		reject
[@id > "1"]	[@id > "3"]		[@id > "3"]
[@id > "1"]	[@id < "3"]		[@id > "1" and @id < "3"]
[@id = "1"]		[@id = "1"]	[@id = "1"]
[@id = "1"]		[@id = "3"]	[@id = "3"]
[@id > "1"]		[@id > "3"]	[@id > "3"]
[@id > "1"]		[@id < "3"]	[@id < "3"]

그림 10 완성된 프레디키트 최적화 예제

존재하는 경우 최적화를 고려하는 함수이다. 특히, 부정적 접근제어 규칙의 프레디키트와 긍정적 접근제어 규칙의 프레디키트를 다르게 처리해야 하는데 초점을 두고 있다.

질의와 긍정적 접근제어 규칙의 프레디키트를 서로 비교하여 그림 10처럼 결과가 나오게 하고, 질의와 부정적 접근제어 규칙은 부정적 접근제어 규칙만을 그대로 유지하게 하고 있다. 그 이유는 부정적 접근제어 규칙은 긍정적 접근제어 규칙의 범위 내에 존재하고, 그것은 바로 질의와 긍정적 접근제어의 프레디키트 최적화에 포함되어 있기 때문에 부정적 접근제어 규칙에 다시 적용할 필요가 없다.

4.4.2 Query Executor 수행 과정

QE의 수행과정을 살펴본다. 부적절한 질의를 사전에 거절시키는 방법이 전체적으로 빠른 성능을 내기 때문에 본 논문에서는 질의와 부정적 접근제어를 먼저 처리한다.

단계 1. (부정적 접근제어 처리)

- **경우 1.1 (SELF 접근제어 규칙)**. SELF 규칙의 의미는 질의의 목적 노드 (PRE, POST) 쌍과 접근제어 규칙의 목적 노드의 (PRE, POST) 쌍이 같은 경우이다. 만약 사용자 질의에 대해 부정적 접근제어 규칙이 SELF 규칙이라면, QE의 결과는 질의 거절이다. 예를 들어, Q2의 (PRE, POST) 쌍은 (28, 25)이고 부정적 접근제어 규칙 R6 역시 (28, 25)이므로 질의 Q2는 R6에 의해 거절된다. 그러나, 만약 부정적 접근제어 규칙에 프리티키트가 존재한다면, 같은 경로들 중 프리티키트에 해당되는 경로만 접근 금지이기 때문에 질의는 거절이 되지 않고 *REFINE* 함수를 호출하여 질의를 재구성한다.
- **경우 1.2 (ANCESTOR 규칙)**. ANCESTOR 규칙의 의미는 사용자 질의의 preorder (postorder) 값이 접근제어 규칙의 그것보다 작은 (큰) 경우를 말한다. 만약 사용자 질의에 대한 부정적 접근제어 규칙이 ANCESTOR 규칙이라면, QE의 결과는 질의 거절이다. 왜냐하면 부정적 접근제어의 범위에 사용자 질의의 범위가 포함되기 때문이다. 그러나, 경우 1.1처럼 부정적 접근제어 규칙에 프레디키트가 존재하면 질의는 거절이 되지 않고 *REFINE* 함수를 호출하여 질의를 재구성한다.
- **경우 1.3 (DESCENDANT 규칙)**. DESCENDANT 규칙의 의미는 사용자 질의의 preorder (postorder) 값이 접근제어 규칙의 그것보다 큰 (작은) 경우를 말한다. 만약 사용자 질의에 대한 부정적 접근제어 규칙이 DESCENDANT 규칙이라면, QE는 *REFINE* 함수를 호출하여 질의를 재구성한다. 예를 들어, 질의 Q3 (30, 38)에 대한 R7 (33, 31)는 DESCENDANT 규칙이므로 QE는 *REFINE* 함수를 호출하여 다음과 같이 변경한다.

  Q3': Q3 EXCEPT (/site/open_auactions/open_auction[@id>50]/seller[@person="chang"]).

  그런 후, 긍정적 접근제어 규칙 R4를 고려하기 위해 단계 2로 간다.
- **경우 1.4 (Null)**. 질의와 관련된 부정적 접근제어 규칙이 없으면 QE는 단계 2로 간다.

단계 2. (긍정적 접근제어 처리)

- **경우 2.1 (SELF 규칙)**. 만약 질의에 대한 긍정적 접근제어 규칙이 SELF 규칙이라면, QE는 *REFINE* 함수를 호출한다.
- **경우 2.2 (ANCESTOR 규칙)**. 만약 질의에 대한 긍정적 접근제어 규칙이 ANCESTOR 규칙이라면, QE는 *REFINE* 함수를 호출한다.
- **경우 2.3 (DESCENDANT 규칙)**. 만약 질의에 대한 긍정적 접근제어 규칙이 DESCEDANT 규칙이라면, 사용자 질의는 접근이 허용되지 않는 범위를 포함하게 된다. 따라서, *REFINE* 함수를 호출하여 안전한 질의로 변경된다.

  예를 들어, 질의 Q3 (30, 38)에 대한 R4 (33, 31)는 DESCENDANT 접근제어 규칙이기에 *REFINE* 함수를 호출하여 /site/open_auctions/open_auction[quantity][@id<100]/seller의 결과를 얻는다. 둘째, QE는 경우 1.3에서 얻어진 결과를 EXCEPT 연산자를 이용하여 불필요한 부분을 제거한다. 따라서, 최종적으로 변경된 안전한 질의는 다음과 같다:

  (/site/open_auctions/open_auction[quantity][@id<100]/seller)

  EXCEPT

  (/site/open_auactions/open_auction[@id>50]/seller[@person="chang"])

4.5 QFilter와의 비교

이번 절에서는 질의 재작성 관점에서 4.4절의 Q3 질의 예제를 갖고 본 논문의 비교 대상인 QFilter(메타데이타가 없는 NFA-기반의 방법)의 잘못된 질의 재작성의 문제점을 지적한다. DTD의 메타데이타가 없는 상황에서 사용자의 질의에 "//child" 경로가 존재하고 공유 비결정 유한 오토마타에 "/-child" 혹은 "//-child" 상태가 존재하지 않는다면, 공유 비결정 유한 오타마타에 대한 항해는 최종적으로 최종상태 (final state)에 도달하게 되고, "노드 결과 (answer-as-nodes)" 모델인 경우 질의를 거절하게 되고 "하부-트리 결과 (answer-as-subtrees)" 모델인 경우, QFilter는 최종 상태에 "//-child" 추가하여 재작성된 질의로 결과를 내게 된다. 다음은 Q3의 결과이다:

(/site/regions/* item[@location="LA"]//open_auction[@id<100]) UNION
(/site/people/person[name="chang"]//open_auction[@id<100]) UNION
(/site/open_auctions/ open_auction[@id<100]) UNION
(//open_auction[quantity] [@id<100]/seller))
EXCEPT
((/site/regions/*/item/payment//open_auction[@id<100]) UNION
(/site/people/person/creditcard//open_auction[@id<100]) UNION

(/site/*/open_auction[@id<100 and @id>50]))

이것은 4.4절의 최종 재작성된 질의와 다름을 알 수 있고, 그림 2(a)의 DTD와 비교했을 때 잘못된 질의 재작성임을 알 수 있다.

4.6 정확성

4.5절에서 제안하고 있는 QE는 사용자의 질의를 사용자의 보안 정책을 준수하는 질의로 변형하는 것이다. 따라서 질의 재작성 알고리즘의 정확성(correctness)는 재작성된 질의의 목적 노드의 PRE(order) 값과 POST (order)값이 부정적 접근제어 규칙에 포함되지 않으며, 질의에 대한 긍정적 접근제어 규칙을 모두 포함하고 있음을 보이는 것으로 증명할 수 있다. 아래의 수식은 3.4절에서 제시한 수식이다.

$$Q' = Q \text{ INTERSECT } (ACR^+ \text{ EXCEPT } ACR^-) = (Q \text{ INTERSECT } ACR^+) \text{ EXCEPT } (Q \text{ INTERSECT } ACR^-) \quad (11)$$

(Q INTERSECT ACR^+)의 의미는 XPath 질의에서 긍정적 접근제어 규칙이 아닌 부분을 제거하는 것이고, (Q INTERSECT ACR^-)의 의미는 XPath 질의에서 부정적 접근제어 규칙에 해당되는 부분을 찾는 것이다. 최종적으로, 긍정적 접근제어 규칙을 준수하는 변형된 질의에서 부정적 접근제어 규칙을 제거한 변형된 질의를 생성하는 것이다.

(Q INTERSECT ACR^+) 연산과 (Q INTERSECT ACR^-)의 의미는 다르지만, 처리 관점에서는 동일하게 질의에서 접근제어 규칙에 해당되는 부분을 찾는 것이다. 따라서, 본 논문에서는 포괄적으로 변형된 질의가 접근제어 규칙에 해당하는 영역에 대한 질의임을 보인다.

임의의 두 개의 XPath 표현식 p_i와 p_j에 대해 교집합 '∩'은 다음과 같이 정의된다.

정의 11. **[$p_i \cap p_j$]** $p_i \cap p_j$은 p_i 와 p_j 모두에 공통으로 표현되는 노드들을 나타내는 XPath 표현식이다. 따라서, 같은 경로 상의 p_i 의 목적 노드가 p_j 의 목적 노드보다 상위의 노드이면 다음 수식을 만족한다:

$$Pr_{pi \cap pj} = Pr_{pj},\ Po_{pi \cap pj} = Po_{pj} \quad (12)$$

한편, 같은 경로 상의 p_i 의 목적 노드와 p_j 의 목적 노드가 같으면 다음 수식을 만족한다:

$$Pr_{pi \cap pj} = Pr_{pj} = Pr_{pi},\ Po_{pi \cap pj} = Po_{pj} = Po_{pi} \quad (13)$$

정리 2. 프레디키트가 없는 사용자 질의와 긍정적 접근제어 규칙만이 존재하는 경우 Query Executor 알고리즘은 정확하게 안전한 질의를 생성한다: 즉,

$$Pr_{Q'} = Pr_{Q \cap ACR},\ Po_{Q'} = Po_{Q \cap ACR} \quad (14)$$

증명. Q는 질의 XPath 표현식이고, ACR를 접근제어 XPath 표현식이라 하면, Q∩ACR는 정의 11에 의해 Q와 ACR 모두에 공통으로 표현되는 노드들을 나타내는 XPath 표현식이다. 따라서, 정확하게 안전한 질의를 생성한다는 의미는 변형된 질의 XPath 표현식의 목적 노드가 Q∩ACR에 의해 생성된 XPath 표현식의 목적 노드와 같다는 것을 보이면 된다. 그 이유는 XML 스키마(DTD 또는 XML Schema)에 대한 PPS 구조 정보(그림 2(b))에서 루트에서부터 목적 노드까지의 경로 구조 정보는 유일하기 때문이다.

표 1 QFilter에서 Q3의 처리 과정

QFilter Construction	• Constructs **four** NFA for each **positive** ACR • Combines all the NFAs in the way that identical states are merged.
	• Constructs **three** NFA for each **negative** ACR • Combines all the NFAs in the way that identical states are merged.
QFilter Execution	• Rewrites the given query against the positive shared NFA /site/regions/*/item[@location="LA"]//open_auction[@id<100] • /site/people/person[name="chang"]//open_auction[@id<100] • /site/open_auctions/ open_auction[@id<100] • //open_auction[quantity] [@id<100]/seller
	• Rewrites the given query against the negative shared NFA • /site/regions/*/item/payment//open_auction[@id<100] • /site/people/person/creditcard//open_auction[@id<100] • /site/*/open_auction[@id<100 and @id>50]
	• Outputs a rewritten query, finally. ((/site/regions/* /item[@location="LA"]//open_auction[@id<100]) UNION (/site/people/person[name="chang"]// open_auction[@id<100]) UNION (/site/open_auctions/ open_auction[@id<100]) UNION (//open_auction[quantity] [@id<100]/seller)) EXCEPT ((/site/regions/*/item/payment//open_auction[@id<100]) UNION (/site/people/person/creditcard//open_auction[@id<100]) UNION (/site/*/open_auction[@id<100 and @id>50]))

4.3절에서 설명한 ACR-F에 의해 질의(Q)에 대한 접근제어 규칙(ACR)들은 'ANCESTOR 규칙', 'DESCENDANT 규칙', 그리고 'SELF 규칙'들이다.

ACR이 'ANCESTOR 규칙'이면(정의 7), 같은 경로 상의 ACR의 목적 노드가 상위 노드가 된다. 따라서, 변형된 질의(Q')의 목적 노드는 질의의 목적 노드를 선택한다:

$$Pr_{Q'} = Pr_Q,\ Po_{Q'} = Po_Q \quad (15)$$

한편, 정의 11에 의해 $Pr_{ACR} < Pr_Q$, $Po_{ACR} > Po_Q$ 이기 때문에 다음 수식을 만족한다:

$$Pr_{Q\cap ACR} = Pr_Q,\ Po_{Q\cap ACR} = Po_Q \quad (16)$$

따라서, 식 (15)와 식 (16)은 식 (14)를 만족한다.

ACR이 'DESCENDANT 규칙'이면(정의 9), 같은 경로 상의 ACR의 목적 노드가 하위의 노드가 된다. 따라서, 변형된 질의(Q')의 목적 노드는 ACR의 목적 노드를 선택한다:

$$Pr_{Q'} = Pr_{ACR},\ Po_{Q'} = Po_{ACR} \quad (17)$$

한편, 정의 11에 의해 $Pr_Q < Pr_{ACR}$, $Po_Q > Po_{ACR}$ 이기 때문에 다음 수식을 만족한다:

$$Pr_{Q\cap ACR} = Pr_{ACR},\ Po_{Q\cap ACR} = Po_{ACR} \quad (18)$$

따라서, 식 (17)과 식 (18)은 식 (14)를 만족한다.

ACR이 'SELF 규칙'이면(정의 10), 같은 경로 상의 ACR의 목적 노드는 질의의 목적 노드와 같다. 따라서, 변형된 질의(Q')의 목적노드는 질의(혹은 ACR)의 목적 노드를 선택한다:

$$Pr_{Q'} = Pr_Q(=Pr_{ACR}),\ Po_{Q'} = Po_Q(=Po_{ACR}) \quad (19)$$

한편, 정의 11의 식 (12)에 의해 다음 수식이 만족된다:

$$Pr_{Q\cap ACR} = Pr_Q(\text{or } Pr_{ACR}),\ Po_{Q\cap ACR} = Po_Q \quad (20)$$

따라서, 식 (19)와 식 (20)은 식 (14)를 만족한다.

정리 3. 프레디키트를 포함한 사용자 질의와 긍정적 접근제어 규칙만이 존재하는 경우 Query Executor 알고리즘은 정확하게 안전한 질의를 생성한다.

증명. 같은 경로 구조 정보를 갖고 있는 프레디키트를 포함한 XPath 표현식(pp)과 프레디키트가 없는 XPath 표현식(p)의 관계는 다음과 같다:

$$Pr_{pp} = Pr_p\ ,\ Po_{pp} = Po_p$$

$$pp\cap p = pp \quad (20)$$

따라서, 프레디키트를 포함한 사용자 질의(Q)와 그 질의에서 프레디키트만을 제거한 질의(Q*), 프레디키트를 포함한 접근제어 규칙(ACR)와 그 접근제어 규칙에서 프레디키트를 제거한 접근제어 규칙(ACR*) 사이에는 다음과 같은 관계가 성립한다.

$$Pr_Q = Pr_{Q^*}\ ,\ Po_Q = Po_{Q^*}$$

$$Q\cap Q^* = Q \quad (21)$$

$$Pr_{ACR} = Pr_{ACR^*}\ ,\ Po_{ACR} = Po_{ACR^*}$$

$$ACR\cap ACR^* = ACR \quad (22)$$

Q'(= (Q∩ACR))을 프레디키트를 포함한 사용자 질의 Q와 접근제어 규칙 ACR에 의해 변형된 질의라 하고, Q*' (= (Q*∩ACR*))을 프레디키트를 제거한 사용자 질의 Q*와 접근제어 규칙 ACR*에 의해 변형된 질의라 하자.

식 (20)에 의해 Q'=Q'∩Q*'이고 풀어서 쓰면 Q'=Q∩ACR∩Q*∩ACR*이다.

식 (21)과 식 (22)에 의해 Q'=Q1∩ACR 이 된다.

5. 실 험

SQ-Filter, Q-Filter[13], 그리고 SQ-Filter_NFA(메타데이타를 이용한 NFA 방법의 SQ-Filter)을 자바 프로그램 언어를 이용하여 구현하였다. XMark[15]에 의해 생성된 실험적 데이타 집합[3]을 이용하여 두 가지 관점에서 성능을 비교해 보았다.

5.1 실험 I: 거절 질의에 대한 정확한 탐지

거절 질의(Rejection Query)라 함은 항상 거절이 되어야 하는 사용자 질의를 말한다. 접근제어 정책에 의해 각각의 질의 유형에 맞는 20개의 의도된 거절 질의를 만들고 실제 시스템에서 거절 질의의 탐지 개수를 비교해 보았다. 그 결과는 그림 11과 같다. 여기서 "/" 축으로만 되어 있는 질의인 경우는 어느 시스템이나 100% 거절 질의를 탐지하지만, "//" 축으로 되어 있는 질의인 경우 SQ-Filter와 SQ-NFA 시스템은 완전히 거절 질의를 탐지하지만, QFilter는 그렇지 않음을 확인하였다. 그 원인은 "//" 축에 대한 경로 항해의 끝이 없기 때문에 질의를 거절하지 못하고 잘못된 질의로 재작성하기 때문이다.

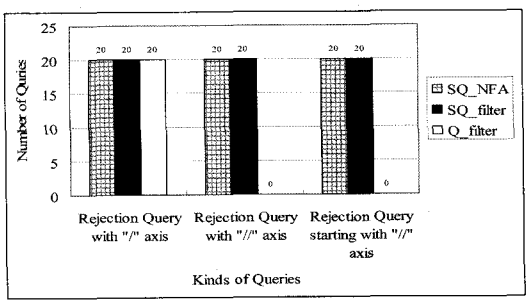


그림 11 거절 질의 탐지에 대한 결과

5.2 실험 II: 수행시간

무작위로 추출한 질의 개수(30, 50, 100, 200, 300, 그리고 500)에 따른 평균 수행시간을 측정하였다. 매번 수

3) · 25 개의 접근제어 규칙 (7 개의 긍정적 접근제어 규칙과 18개의 부정적 접근제어 규칙)
· 펜티엄 IV 2.66GHz, MS-Windows XP OS 그리고 메인 메모리 1 GB

행할 때마다 약간의 시간 차가 생기는 것은 시스템의 문제라기 보다는 자바 언어의 문제이므로 본 실험에서는 각각 100번을 수행하여 비슷하게 나오는 시점의 시간을 실험 데이타로 사용하였다. 일단 예비조건을 생성하는 시간을 비교하기 위해 SQ-Filter 시스템(혹은 SQ-Filter_NFA)의 PPS 정보 생성 시간과 Q-Filter의 접근제어 규칙들에 대한 공유 비결정 유한 오토마타를 생성하는 시간을 비교해 보았으며 그림 12에서 그 결과를 보여주고 있다. SQ-Filter 시스템에서의 PPS 정보를 이용하는 것이 시스템의 성능에 큰 영향을 미치지 못함을 확인할 수 있다.

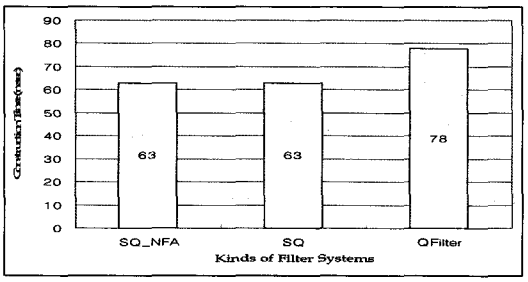


그림 12 각 필터 시스템의 예비조건 생성 시간

3.5절의 PPS 정보 및 4.3절의 ACR-F 컴포넌트에 의해 SQ-Filter 시스템과 SQ-NFA 시스템은 매우 적은 양의 접근제어 규칙을 이용하여 보다 정확하고 안전한 질의 재작성 과정을 수행한다. 그림 13은 그 결과를 보여주고 있으며 각 수행 시간은 예비조건 생성 시간을 포함하고 있다.

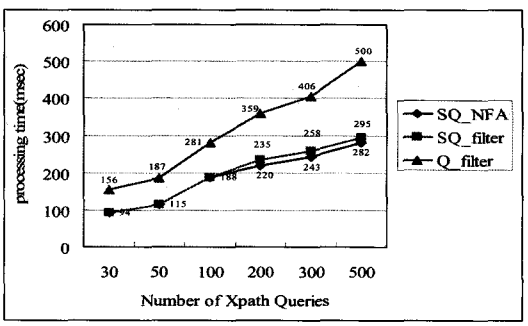


그림 13 질의에 대한 보안 처리 수행 시간

6. 결론 및 향후 연구

본 논문에서 기술한 효율적이고 안전한 XML 접근제어 수행을 위한 SQ-Filter 시스템은 사용자의 질의 처리에 필요한 접근제어 규칙들만을 추출하기 위해 PRE/POST 트리 특성을 이용하였다. 특히 PRE/POST 인코딩 기법은 질의 재작성을 하기 전에 사전에 접근 권한이 없는 질의를 거절시키는데 효율적으로 이용된다. 추가로 DTD에 대한 메타데이타를 이용하여 보다 쉽게 "*" 노드에 대한 실제 노드를 찾고, "//" 축을 일련의 단일 경로로 표현하고 있다. 최종적으로, XPath 2.0에서 지원하는 집합 연산을 이용하여 사용자의 질의를 그 사용자의 접근제어 정책을 준수하는 안전한 질의로 변경한다.

질의 기반의 필요한 접근제어 규칙만을 이용하여 효율적이고 안전한 XML 접근제어를 수행하는 방법론은 본 논문이 처음이며, 실험을 통해 뛰어난 성능을 보였다. 또한 XML 데이타 환경에서 접근제어 관점에서 정확하고 안전한 질의 재작성에 대한 개념 역시 본 논문이 처음이며, 실험을 통해 제안하는 방법의 우수성을 보였다.

추후 연구로는 접근 권한 모델(강제적 접근 권한 모델, 임의적 접근 권한 모델, 역할 기반 접근제어 모델)과 조합된 XML 접근제어 시스템 개발을 진행하고 객체-지향 데이타베이스 관리 시스템의 접근제어 시스템에 적용하고자 한다.

참 고 문 헌

[1] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau. eXtensible Markup Language (XML) 1.0, World Wide Web Consortium (W3C), 2004. (http://www.w3.org/TR/REC-xml).

[2] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon. XPath 2.0, World Wide Web Consortium (W3C), 2005. (http://www.w3.org/TR/xpath20/).

[3] F. Rabitti, E. Bertino, W. Kim and D. Woelk. A Model of Authorization for Next-Generation Database Systems. *ACM Transaction on Database Systems*, Vol. 126, No. 1, PP. 88-131, March 1991.

[4] E. Damiani, S. Vimercati, S. Paraboachk and P.Samarati. Design and Implementation of Access Control Processor for XML Documents. *Computer Network*, pp. 59-75, 2000.

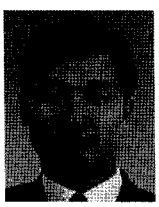
[5] E. Damiani, S. Vimercati, S. Paraboachk and P.Samarati. A Fine-grained Access Control System for XML Documents. *ACM Trans. Information and System Sec.*, Vol. 5, No. 2, pp. 169-202, May 2002.

[6] E. Bertino, S. Castano, E. Ferrari, and M. Mesiti. Specifying and Enforcing Access Control Policies for XML Document Sources. *WWW Journal*, Baltzer Science Publishers, Vol. 3, No. 3, pp. 139-151, 2000.

[7] E. Bertino, S. Castano, and E. Ferrai. Securing XML documents with Author-x. *IEEE Internet Computing*, May.June, pp. 21-31, 2001.

[8] A. Gabillon and E. Bruno, "Regulating Access to XML Documents," In Proc. IFIP WG11.3 Working Conference on Database Security, pp. 299-314, 2001.
[9] A. Stoica and C. Farkas, "Secure XML Views," In Proc. IFIP WG11.3 Working Conference on Database and Application Security, pp. 133-146, 2002.
[10] T. Grust, "Accelerating XPath Location Steps," In Proc. of the 21st Int'l ACM SIGMOD Conf. on Management of DataMadison, Wisconsin, USA, pp. 109-120, June 2002.
[11] M. Murata, A. Tozawa, and M. Kudo, "XML Access Control using Static Analysis," In ACM CCS, Washington D.C., pp. 73-84, 2003.
[12] Jae-Myeong Jeon, Yon Dohn Chung, Myoung Ho Kim, and Yoon Joon Lee, Filtering XPath expressions for XML access control. *Computers & Security*, 23, pp. 591-605, 2004.
[13] B. Luo, D. W. Lee, W. C. Lee, and P. Liu, "Qfilter: Fine-grained Run-Time XML Access Control via NFA-based Query Rewriting," In Proc. of the Thirteenth ACM Conference on Information and Knowledge Management 2004(CIKM'04), pp. 543-552, 2004.
[14] S. Mohan, A. Sengupta, Y. Wu, and J. Klinginsmith, "Access Control for XML- A Dynamic Query Rewriting Approach," In Proc. of the Thirteenth ACM Conference on Information and Knowledge Management 2005(CIKM'05), pp. 251-252, 2005.
[15] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse. The XML Benchmark Project. Technical Report INS-R0103, CWI, April 2001.

변 창 우
1999년 서강대학교 컴퓨터학과 학사 2001년 서강대학교 컴퓨터학과 공학석사 2001년~현재 서강대학교 컴퓨터학과 박사과정. 관심분야는 동적인 데이타베이스에서의 트랜잭션 관리, 역할기반 접근제어 모델, XML 접근제어, 프라이버시



박 석
1978년 서울대학교 계산통계학과(이학사). 1980년 한국과학기술원 전산학과(공학석사). 1983년 한국과학기술원 전산학과(공학박사). 1983년 9월~현재 서강대학교 컴퓨터학과 교수. 1989년~1991년/2002년~2003년 University of Virginia 방문교수. 1997년 2월~현재 한국정보보호학회 이사. 2005년. 한국정보과학회 부회장, 2004년 1월~2005년 12월 한국정보과학회지 편집위원장 1999년~현재 DASFAA Steering Committee 멤버. 관심분야는 데이타베이스 보안, 실시간 시스템, 트랜잭션 관리, 데이타웨어하우스, 웹과 데이타베이스, XML, XML 데이타베이스 시스템에서의 필터링 기법, 역할기반 접근제어 임