XPath 표현식의 필터링을 통한 XML 접근 제어 기법

(An XML Access Control Method through Filtering XPath Expressions)

전 재 명 † 정 연 돈 †† 김 명 호 ††† 이 윤 준 †††
(Jae-myeong Jeon) (Yon Dohn Chung) (Myoung Ho Kim) (Yoon Joon Lee)

요 약 XML은 인터넷 상에서 데이타의 표현 및 전송 표준으로 인식되고 있다. XPath는 XML 문서의 특정 부분을 규정하는 표준으로, XML 질의 처리와 접근 제어에 적합한 언어이다. 본 논문에서는 XPath를 사용자 질의 및 접근 제어 정보를 표현하는 방법으로 사용하는 XML 접근 제어 방법을 제안한다. 제안하는 방법은 접근 제어 XPath 표현식을 통해 질의 XPath 표현식을 필터링하여 XML 문서에 대한 접근을 제어한다. 이를 위하여 XML 접근 제어 트리(XACT)를 정의하고, 이 트리를 이용하여 질의 XPath 표현식에서 접근 허용되는 부분만을 추출한다. XACT는 XML 엘리먼트들에 대한 구조적 요약으로 에지를 구성하고, 접근 제어 정보로 노드를 구성한 구조이다. 제안하는 방법의 정확성을 보이고, 기존 방법과의 성능을 비교한다.

키워드 : XPath, XML, 질의 처리, 접근 제어, 보안, 데이타베이스

Abstract XML (eXtensible Markup Language) is recognized as a standard of data representation and transmission on Internet. XPath is a standard for specifying parts of XML documents anda suitable language for both query processing and access control of XML. In this paper, we use the XPath expression for representing user queries and access control for XML. And we propose an access control method for XML, where we control accesses to XML documents by filtering query XPath expressions through access control XPath expressions. In the proposed method, we directly search *XACT* (*XML Access Control Tree*) for a query XPath expression and extract the access-granted parts. The *XACT* is our proposedstructure, where the edges are structural summary of XML elements and the nodes contain access-control information. We show the query XPath expressions are successfully filtered through the *XACT* by our proposed method, and also show the performance improvement by comparing the proposed method with the previous work.

Key words : XPath, XML, Query Processing, Access Control, Security, Databases

1. 서 론

XML[1]은 인터넷 상에서 데이타의 표현 및 전송을 위해 W3C에서 정의한 표준이다. XML의 특징인 응용과의 독립성, 확장성, 가독성 등으로 XML은 그 응용 분야를 급속히 넓히고 있으며, 최근에는 XML 보안에 대한 관심도 높아지고 있다. XML 문서의 보안에 관한 연구들로[1-7] 등이 있으며, XML 보안 이슈들 중에서 디지털 암호화(cryptography)와 서버에서 접근 제어에 대해 특히 많은 연구가 이루어지고 있다. 디지털 암호화란 XML 데이타를 부호화하여 WWW 상에서 전송하고, 접근이 허가된 사용자들이 복호화하는 과정들에 대한 연구이며[3-7], 서버에서 이루어지는 접근 제어란 사용자별로 주어진 접근 권한에 따라 접근 가능한 XML 데이타만을 선별하여 제공하는 방법에 대한 연구를 말한다[2,8-11]. 특히, 다수의 사용자 환경과 엘리먼트 수준의 접근 제어를 제공하는 서버측XML 접근 제어 방

· 본 연구는 대학 IT연구센터 육성 지원사업의 부분적인 지원을 받아 수행되었음
† 비 회 원 : 공군 E-X 사업단
jmjeon@dbserver.kaist.ac.kr
†† 종신회원 : 동국대학교 컴퓨터공학과 교수
(Corresponding author임)
ydchung@dgu.edu
††† 종신회원 : 한국과학기술원 전산학과 교수
mhkim@dbserver.kaist.ac.kr
yjlee@dbserver.kaist.ac.kr
논문접수 : 2004년 7월 2일
심사완료 : 2004년 11월 25일

법들은 많은 계산 시간 및 비용을 요구한다. 본 논문에서는 복잡한 XPath 표현식(XPath Expression: XPE)에도 적용 가능하고, 접근 제어에 필요한 처리 시간도 줄이는 효과적인 서버측 XML 접근 제어 기법을 제안한다.

XPath는 XML 문서의 부분을 기술하는 표준 언어로서, 특정 문맥에서 나타내는 일정 패턴의 데이타(노드)를 지정하거나 , 그 노드들에 대해 연산을 수행할 수 있다[12]. 이런 측면에서 XPath 표현식(XPE)은 XML 데이타에 대한 질의 처리 및 접근 제어 목적에 적합하다고 알려져 있다[2,9,11-14]. 본 논문에서는 사용자의 질의와 XML 문서의 접근 제어 정보를 기술하는 수단으로 XPath를 사용한다(사용자 질의를 나타내는 XPE를 '질의 XPE', 접근 제어 정보를 나타내는 XPE를 '접근 제어 XPE'라고 부르도록 한다.) 본 논문은 다음과 같은 환경을 가정한다.

- 사용자가 '질의 XPE'를 서버에게 제출한다.
- 서버에 존재하는 접근 제어 정보는 XPE로 표현 및 저장된다.
- 서버는 사용자가 제출한 '질의 XPE'와 '접근 제어 XPE'를 비교하고, XML 문서에서 사용자에게 접근이 허가된 부분만을 기술하는 XPE들을 생성한다. ('결과 XPE'이라고 부른다.)
- '결과 XPE'들은 서버의 질의 처리기에서 처리되고, 그 결과 데이타가 사용자에게 전달된다.

제안하는 방법은 'XML 접근 제어 트리(XACT: XML Access Control Tree)'를 정의한다. 이 트리는 XML 엘리먼트들에 대한 구조적 요약 정보를 에지(edge)로 구성하고, 각 노드에 접근 제어 XPE를 기록한 트리로서, 사용자가 제출한 '질의 XPE'에 대하여 XACT에 저장된 접근 제어 정보들을 비교하여 '결과 XPE'들을 생성하게 된다.

논문의 구성은 다음과 같다. 2장에서 본 논문의 배경 지식 설명을 위해 XML, XPath, XML 접근 제어 정책들과 XML 접근 제어에 대한 관련 연구들을 소개한다. 3장에서 '질의 XPE'와 '접근 제어 XPE'를 사용하여 XML 문서에서 접근이 허락되는 부분만을 기술하는 '결과 XPE'들을 찾아내는 문제를 정의한다. 4장에서 XACT 구조에 대한 정의와, 이 구조를 사용하는 접근 제어 방법을 제안한다. 5장에서 제안하는 방법의 성능 평가 결과를 보인 후, 6장에서 결론을 맺는다.

2. 배 경

2.1 XML

XML 문서는 중첩된 엘리먼트(element)들의 배열(sequence)로 구성되며, 각 엘리먼트는 내부 엘리먼트, 텍스트(text), 애트리뷰트(attribute)를 포함할 수 있다. 엘리먼트와 텍스트들을 시작 엘리먼트 '<tag-name>'와 끝 엘리먼트 '</tag-name>' 사이에 존재하며, 애트리뷰트는 시작 엘리먼트 '<tag-name>' 내부에 기술된다. XML의 태그(tag)들은 XML 문서의 포맷팅 규칙이나 데이타들 사이의 관계들을 나타내며, 이들을 통해 다양한 구조의 데이타를 XML 문서로 표현하게 된다. 그림 1은 예제 XML 문서와, 그 문서의 구조적인 성질을 보여주는 트리 형태의 표현 방식을 나타낸다.

2.2 XPath

XPath의 주요 목적은 XML 문서에서 특정 부분들을 나타내는 것이다. 따라서 XQuery[15], XSLT[16]와 같은 XML 질의 언어와 XML 접근 제어 방법들[9,11]에서 사용되고 있다. XPath 표현식(XPE)은 XML 문서

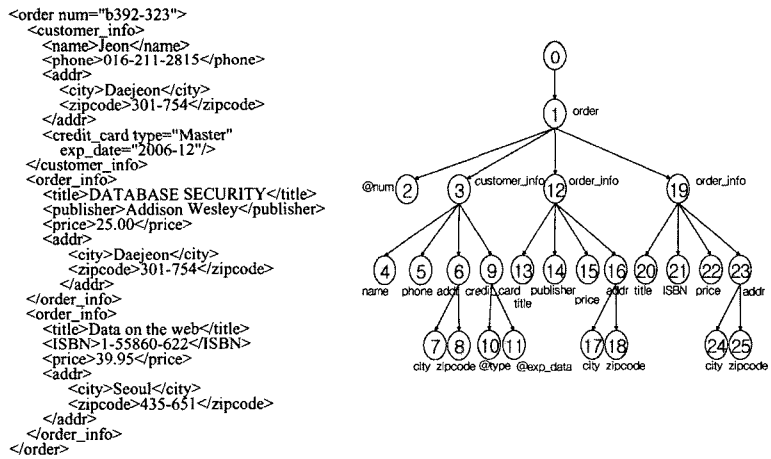


그림 1 XML 문서와 트리 형태의 표현

트리의 노드들에 부합되는 구조적인 패턴을 나타낸다. XPE는 엘리먼트와 엘리먼트들 사이의 관계를 기술하는 연산자들을 나타내는 로케이션 스텝(location step)들의 배열(sequence)이다. XPath의 로케이션 스텝에 사용되는 연산자들로는 '부모-자식' 관계를 나타내는 '/', '조상-후손' 관계를 나타내는 '//'과 임의의 엘리먼트를 나타내는 '*' 등이 있다. 또한 각 로케이션 스텝은 선택되는 노드 집합을 세밀하게 정의하는 프레디키트(predicate)를 포함할 수도 있다(프레디키트는 '['와 ']' 사이에 기술된다.) 프레디키트 사용으로 형제(sibling) 노드들 사이의 순서 및 위치를 기술할 수도 있다. 하지만, 실제 XML 문서의 사용에 있어 형제 노드들 사이의 순서를 사용하는 경우가 흔치 않기 때문에, 본 논문에서는 형제 노드들 사이의 순서를 고려하지 않는다.

정의 1. p_i와 p_j를 *XPE*라고 하고, $n\{p_i\}$를 XML 문서 트리에서 p_i에 의해 표현되는 노드들의 집합이라고 하자.

- $p_i \cup p_j$는 p_i 또는 p_j로 표현되는 노드들을 나타내는 *XPE (또는 XPE들)*이다. 즉, $n\{p_i \cup p_j\} = n\{p_i\} \cup n\{p_j\}$
- $p_i \cap p_j$는 p_i와 p_j 모두에 공통으로 표현되는 노드들을 나타내는 *XPE (또는 XPE들)*이다. 즉, $n\{p_i \cap p_j\} = n\{p_i\} \cap n\{p_j\}$
- $p_i - p_j$는 p_i로 표현되지만 p_j로는 표현되지 않는 노드들을 나타내는 *XPE (또는 XPE들)*이다. 즉, $n\{p_i - p_j\} = n\{p_i\} - n\{p_j\}$

예제 1. 그림 1에 나타나있는 XML 문서 트리에 대해, 다음의 두 XPE p_i = "order /customer_info//" 와 p_j = "order/customer_info/name"를 사용하여 정의 1에 기술된 세가지 집합 연산을 시행해 보자.

1. $p_i \cup p_j$ = "order/customer_info//"
2. $p_i \cap p_j$ = "order/customer_info/name"
3. $p_i - p_j$ = "order/customer_info/phone"; "order/customer_info/addr//"; "order/customer_info/credit_card//"

2.3 접근 제어 정책

XML 문서에 대한 접근 제어 정책에 대하여 [14]에서 정리한 결과를 살펴보자. '*전파 정책(propagation policy)*'은 한 노드에 대한 접근 제어 결과가 그 노드의 후손에 전파되는지 여부를 결정하는 것이다. '*충돌 해결 정책 (conflict resolution policy)*'는 어떤 노드가 '접근 허용' 및 '접근 불가'로 동시에 정의되어 있는 경우에 대한 해결 방법을 나타낸다. 그리고, '*결정 정책(decision policy)*'은 어떤 노드가 '접근 허용' 또는 '접근 불가' 중 어떤 것으로도 정의되어 있는 않은 경우에 대한 결정 방법을 이야기 한다. 그리고, 이들 접근 제어 정책들은 동시에 적용되는 것이 일반적이다.

본 논문에서는 '전파 정책'으로 XML 문서 트리에서 가장 세밀한(트리에서 아래 수준) 노드에 대한 접근 제어 정보를 우선하는 "*세밀 우선(most specific overrides)*" 정책을 사용하며, 엄격한 데이타 보안을 위해 '충돌 해결 정책'으로 "*불가 우선 정책(denial take precedence)*"을 사용한다. 그리고 '결정 정책'으로는 같은 이유에서 "*폐쇄(closed)*" 정책을 사용한다. 이는 접근 제어 정보가 정의되지 않는 노드에 대해서는 '접근 불가'로 결정하는 정책이다.

2.4 관련 연구

XML 문서 보안에 대하여 지금까지 많은 연구들이 이루어져 왔다. [8,9]는 XML 접근 제어에 대해 초기 모델을 제시한 연구로, 질의 XPE에 대한 DOM 트리를 생성한 후, 사용자의 접근 권한에 따라 트리의 각 노드를 '접근 가능' 및 '접근 불가'로 표시하는 방법이다. '접근 불가'로 표시된 노드들은 트리에서 삭제되고, 남아있는 노드들만 질의에 대한 결과로 사용자에게 전달된다. [17]은 위 방법을 질의 접근 시간 측면에서 성능 향상을 시킨 것으로, 각 사용자 별로 접근 제어 정보를 검색하기 위한 최적의 구조를 정의하고 있다. 하지만, 이 방법에서는 루트 노드에서 특정 노드까지의 완전한 경로를 요구하기 때문에 '//'나 '*' 등이 포함되는 XPE를 처리하지 못하고 있다. [11]은 사용자 질의를 XACL (XML Access Control Language)과 비교하는 세밀 접근 인가 모델을 제시하고 있다. 하지만, 이 연구에서는 질의 XPE와 XACL에 있는 XPE들을 비교는 문제의 계산 복잡도가 coNP-hard인 문제에 대해 해결책을 제시하지 못하고 있다. [18,19]의 연구는 XML 문서의 DTD 그래프가 주어질 때, 보안성을 제공할 수 있는 질의로의 안전하고 최적화된 변환(질의 재작성)에 대해 언급하고 있다.

[20]는 질의를 표현하고 비교하는 방법으로 오토마타(automata)를 사용하고 있다. 질의 및 접근 제어 정보, 스키마(schema)에 대해 오토마타를 생성한 후, 이들 오토마타를 교차시켜(intersection) 결과를 생성해 내는 방법이다. 하지만, 이 방법은 프레디키트 및 재귀적(recursive) 질의 처리에 제약을 갖는 단점이 있다.

XML 질의 처리 분야에 있어 XPE들을 비교하는 부분에 대한 연구들이 있다. [21]에서는 XPE들의 클로져(closure) 특성에 대하여 설명하고 있다. 합집합(union)과 교집합(intersection)은 닫혀있는 반면 차집합(complementation)은 그렇지 않다. [22]에서는 분기(branching), 와일드 카드('*') 및 후손 관계('//')를 포함하는 XPE들 사이의 포함(containment) 및 동등성 (equivalence) 관계에 대해 연구하였다. 이 연구의 결과로

XPE들 사이의 포함 문제는 일반적으로 coNP-hard 이며 특별한 경우에 한하여 PTIME에 해결된다고 알려져 있다[22].

3. 문제 정의

주어진 XML 문서에 대하여 접근 허가된 XPE집합과 접근 불가능한 XPE 집합을 각각 와 로 나타내고, 를 질의 XPE라고 하자. 그러면 다음의 연산이 계산 가능할 것인가 ?

Result $XPE = (\Theta \cap \Pi) - \triangle$

이 식에서 각 집합들이 부분적으로 공유되어 있는 경우, 결과 XPE 집합을 추출해 내는 것은 어려운 문제이다. 게다가 결과 XPE는 대상 문서의 구조와 밀접하게 관련이 되기 때문에 대상 문서의 구조에 대해 사전에 정보를 가지고 있어야 한다. [22]에서는 '[]', '*' 및 '//'를 포함하고 있는 XPE들의 포함(containment) 관계 문제가 coNP-hard임이 증명된 바 있다. 따라서, XPE 집합들에 대한 교집합 및 차집합 문제 역시 coNP-hard 문제가 된다. 이 문제를 풀기 위해 [22]에서는 '후손' 축(axis)을 XML 문서를 참조하여 다른 노드(z-labeled node)로 변환하는 모델을 제시하고 있다. 하지만, 본 논문에서 다루고자 하는 XPE 집합의 연산에 있어 이러한 노드 변환 방법은 사용할 수 없다. 변환 노드들을 사용한 XPE 집합들의 연산이 차집합 연산에 대하여 닫혀(closed)있지 않기 때문이다. [9]에서는 접근 허가된 XPE들로 DOM 트리를 만들어 접근 허가된 데이타들을 추출한 후 질의 XPE를 처리하는 방식을 이용하지만, 이 역시 질의들이 여러 XML 문서를 접근하는 경우에는 매우 비효율적이다.

본 논문에서는 "$(\Theta \cap \Pi)-\triangle$" 연산을 통해 결과 XPE 집합을 생성함에 있어, XML 문서나 DOM 트리의 접근을 필요로 하지 않는 방법을 제안한다. 제안하는 방법에서는 XPE 집합들 사이의 효과적인 집합 연산을 위해 XML 접근 제어 트리(*XACT: XML Access Control Tree*)를 정의한다. 이 트리는 XML 엘리먼트들에 대한 구조적 요약 정보를 에지(edge)로 구성하고, 각 노드에 접근 제어 정보를 XPE 형태로 기록한 트리로서, 사용자가 제출한 '질의 XPE'에 대하여 *XACT*에 저장된 접근 제어 정보들을 비교하여 '결과 XPE'들을 생성하게 된다.

4. 제안하는 접근 제어 방법

이 장에서는 질의 XPE와 접근 제어 XPE들을 비교하여 질의 XPE로부터 접근 허가된 XPE들 만을 추출하는, 본 논문이 제안하는 접근 제어 방법에 대하여 설명한다. 그림 2는 본 논문이 제안하는 접근 제어 방법에 대한 전체 구조를 나타낸다. 먼저 *XACT 생성자 (XACT Constructor)*는 XML 문서의 구조와 접근 제어 정보들을 *XACT* 구조로 생성 및 갱신하는 모듈이다 (4.1절에서 *XACT* 생성 과정에 대해 설명한다). *XACT*의 각 노드는 접근 제어 정보를 XPE 형태로 표현하고 있는데, 그림에서 '10'번 노드에 포함되어 있는 접근 제어 정보를 별도로 표시하고 있다. 둘째, 질의 필터 (*Query Filter*) 모듈은 사용자가 입력한 '*Input Query*'

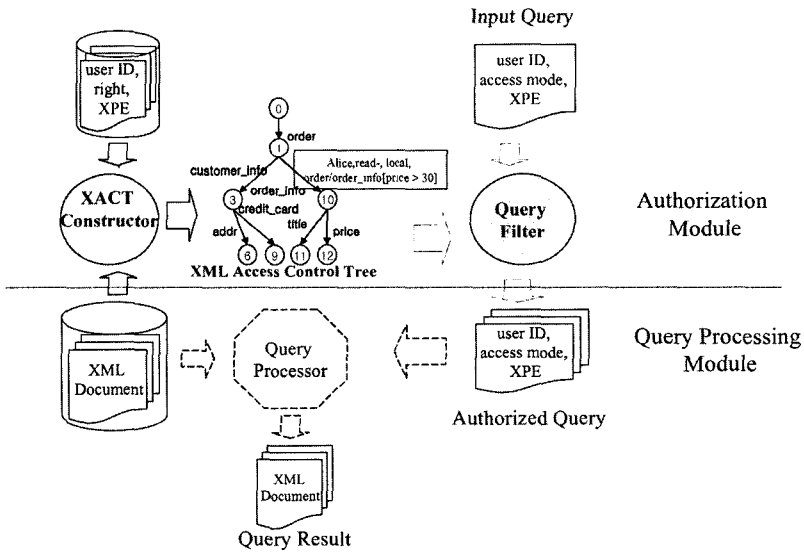


그림 2 제안하는 접근 제어 방법의 구조도

를 *XACT*와 비교하여 접근 허가가 확인된 부분들로만 구성된 '인가 질의(*Authorized Query*)'들을 생성한다 (4.2절에서 질의 필터 모듈에서 사용되는 알고리즘에 대해 설명한다.) 인가 질의는 사용자가 처음에 제시한 '*Input Query*' 대신 '질의 처리기(*Query Processor*)'에 전달되고, 질의 처리기의 처리에 따라 결과 데이타가 사용자에게 전달되게 된다.

4.1 XACT 구축

먼저, "(Θ-Π)"에 대한 접근 제어 정보를 *XACT*의 해당 노드에 저장한다. 이 정보는 질의 XPE가 주어질 때, 질의 XPE에 부합하는 *XACT* 노드를 검색하여 접근 가능 여부를 판단할 때 사용된다. 만일 해당 노드가 접근 가능으로 판별될 경우, 결과 XPE 집합에 해당 노드를 나타내는 XPE가 추가된다.

XACT 생성자는 XML 문서와 *ACL(Access Control List)* 형태로 주어지는 접근 제어 정보를 입력으로 받는다. 여기서 *ACL*은 다음과 같이 정의된다.

정의 2. 어떤 XML 문서에 대한 *ACL*은 3가지 원소 <*o, s, a*>로 구성되는 튜플들의 집합이다.

- o $\in$ *O*; *O*는 XPE의 집합이다.
- s $\in$ *S*; *S* = {ID_1, ID_2, ID_3, $\cdots$}, ID_i는 사용자 혹은 사용자 그룹에 대한 식별자이다.
- a $\in$ *A*; *A*는 XPE에 대한 접근 권한이다,. 즉 *A* = {read+, read-, write+, write-} □

*ACL*은 해당 XPE가 나타내는 *XACT*상의 노드에 저장된다. 하나의 XPE는 노드에 대한 경로와 조건을 기술하는 프레디키트로 구분될 수 있다. *XACT*에서 XPE 정보는 XACT의 노드 자체에서 그 정보를 포함하기 때문에 프레디키트 정보만 노드에 따로 저장하면 된다. '//'나 '*'로 끝나는 XPE는 접근 제어의 범위(scope)가 각각 '*후손 전체*(*recursive*)' 또는 '*자식*(*child*)'임을 나타낸다. (예: order/order_info// , order/order_info/*)

정의 3. XML 접근 제어 트리 XACT는 XML 문서의 엘리먼트을 나타내는 에지(E)와 네 개의 원소 <*s, a, c, f*>로 이루어진 튜플을 포함하는 노드(N)로 구성된다. 여기에서 $s \in S$, $a \in A$, $c \in C$, $f \in F$ 이다.

- 루트 노드를 제외한 각 노드 n은 그 노드를 가리키는 에지 e를 갖는다. ($n \in N$, $e \in E$)
- *path*(*n*)는 루트에서 노드 n에 이르는 경로를 나타내며, XACT에서는 중복 경로가 존재하지 않는다. 즉, $n_i \neq n_j$ 이면 $path(n_i) \neq path(n_j)$이다.
- *S* 는 사용자 또는 사용자 그룹에 대한 식별자들의 집합이다.
- *A* 는 XPE에 대한 접근 모드들의 집합이다. 즉, *A* = {read+, read-, write+, write-}
- *C* 는 노드 n에 대한 s의 접근 권한 범위(scope)를 나타낸다. 즉, *C* = { local, child, recursive}, *local*은 해당 노드에, *child*는 해당 노드의 자식 노드들까지, *recursive*는 해당 노드의 모든 후손 노드들까지를 범위에 포함시킨다.
- *F*는 노드 *n*에 대한 XPE의 프레디키트들의 집합이다. 프레디키트들은 사용자들이 노드 n을 접근할 수 있는 조건을 규정한다. 예를 들면. *F* = {price> 30, name = "Bob"}와 같이 프레디키트를 통해 접근 조건을 명시할 수 있다. □

*XACT*는 요약된 XML 트리와 접근 제어 정보로 구성된다고 볼 수 있다. 요약되었다는 의미는 중복 경로를 제거하였다는 점을 의미한다. 따라서, *XACT*의 각 노드는 유일한 경로를 지니게 된다. 접근 제어 정보는 정의 2에서 설명한 *ACL*로 기술된 정보로부터 추출된다. *ACL* 정보를 변환하여 *XACT* 노드에 저장하기 위해 *ACL*에 나타나는 XPE에 대하여 *XACT*에서 부합되는 에지 배열을 찾은 다음, 정의 3에 기술된 형태로 마지막 에지가 가리키는 노드에 저장하게 된다. 정의 3의 S와 A는 정의 2에서의 S 및 A와 동일하며, 정의 3의 C는 *ACL*에 있는 XPE의 범위에 따라 결정된다. 만일 XPE가 '//'나 '*'로 끝나는 경우, C는 각각 '*recursive*' 또는 '*child*'로 결정되며, 그 밖의 경우엔 '*local*'이다. 정의 3의 F는 XPE의 프레디키트로 노드 n의 조건이 된다. XACT의 경로는 엘리먼드의 배열을 의미하기 때문에 애트리뷰트나 텍스트 등의 비-엘리먼트들은 XACT의 노드로 매핑되지 못한다. 따라서. 비-엘리먼트들은 *XACT*에서 그들을 포함하는 엘리먼트에 대한 프레디키트들로 변환하여 저장된다.

그림 3은 *XACT* 생성 과정을 나타낸다. *XACT*의 에지들은 XML 엘리먼트들로 이루어진다는 점을 주목하자. XACT의 에지들이 만들어진 다음, *ACL*에 있는 각 XPE에 대해 매치되는 에지들을 찾아 해당 ACL 정보를 그 에지가 가리키는 노드에 저장한다. XPE에 프레디키트가 포함되어 있으면, 그 프레디키트가 '//'나 '*'를 포함하고 있는지 여부를 조사한 후, XACT의 에지들을 참고하여 '//'와 '*'가 없는 프레디키트로 변환시킨 후 노드에 저장한다. '//'와 '*'는 노드에 저장되는 접근 제어 범위 정보 역시 결정하게 된다. '//'는 '*recursive*'로, '*'는 '*child*'로 설정되며, 나머지 경우는 '*local*'로 결정된다. ACL에 존재하는 모든 XPE들에 대하여 위 과정을 반복하면 *XACT*가 완성된다.

예제 2. 다음과 같은 ACL이 있다고 가정하자.

(1) (Bob, read+, "order//")
(2) (Alice, read+, "order/order_info//")
(3) (Bob, read-, "//credit_card/")
(4) (Alice, read-, "//order_info[price > 30]/addr")

```
Algorithm XACT_Constructor
Input : ACL(ID, right, XPE), XML document
Output : XACT
(1)     Build XACT edges of XML document
(2)     for (each XACT node which is matched by the XPE)
(3)         if (the XPE contains predicates)
(4)             Current_node.predicate := Predicate_Search(node, XPE)
(5)         Current_node.ID := ID
(6)         Current_node.right := right
(7)         if (the XPE terminated with //) Current_node.scope := 'recur'
(8)         else if (the XPE terminated with *) Current_node.scope := 'child'
(9)         else Current_node.scope := 'local'
Algorithm Predicate_Search
Input : node, XPE
Output : predicate
(1)     for (each predicate)
(2)         if (the predicate contains '//' or '*' )    // search exact set of predicates
(3)             for (each path of the predicate)   predicate_path := predicate_path ∪ path
(4)         else predicate_path := predicate
(5)         predicates := predicates ∪ predicate_path
(6)     return(predicate)
```

그림 3 XACT 생성 알고리즘

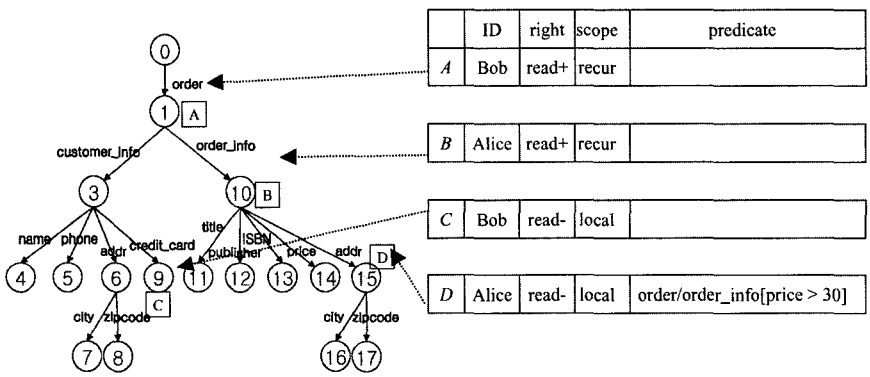


그림 4 XACT 노드 내부의 접근 제어 자료

이 ACL이 변환되어 저장된 *XACT*는 그림 4와 같다. (1), (2)의 내용은A, B처럼 변환되어 노드 '1'번과 '10'번에 각각 저장된다. 이때 XPE가 '//'로 끝났기 때문에, 접근 제어 범위는 '*recursive*'로 결정된다. (3)은 C와 같이 변화되어 '9'번 노드에 저장되며, (4)는 D와 같이 변환되어 '15'번 노드에 저장된다. 프레디키트는 해당 노드의 경로에 따라 수정이 이루어지는데, 예를 들어 (4)에서 '//order_info[price>30]' 프레디키트가 저장될 '15'번 노드에 맞게 'order/order_info [price>30]'로 수정됨을 알 수 있다. □

XML트리에서 루트로부터 엘리먼트로 이르는 하나의 경로가 *XACT*에서 에지들의 배열로 변환되는데, 중복 경로가 모두 제거된다. 따라서 *XACT*는 다음과같은 성질들을 갖는다.

특성 1. 프레디키트를 포함하지 않는 XPE p가 *XACT*에서 에지들의 배열을 나타낼 경우, p는 *XACT*의 에지에해당하는 XML 문서의 엘리먼트 집합을 나타내고, 그 역도 성립한다. □

프레디키트를 포함하는 XPE의 경우, XML 문서에서 p에 대한 엘리먼트들의 집합이 XACT에서 p에 해당하는 에지들이 나타내는 XML 엘리먼트 집합과 다를 수 있다.

특성 2. p를 프레디키트가 포함되어 있는 XPE라고 하자. 그리고, $n_{XML}(p)$를 XML 문서 트리에서 p에 대한 노드들의 집합이라고 하고, $n_{XACT}(p)$를 *XACT*에서 p에 대한 에지들이 가리키는 노드들의 집합이라고 하자. 그러면, $n_{XML}(p) \subseteq n_{XACT}(p)$이다. □

특성 2는 XML 문서 트리에서 조상 노드들의 배열은 같지만, 자식이 다른 두 노드들이 존재하는 경우 나타난다. 정의 3에 따라, *XACT*에서는 중복되는 경로들이 제거되고 하나의 경로만 표현된다. 따라서 두 자식 노드 집합이 *XACT*에서는 서로 형제 노드들이 되기 때문이다. 예를 살펴보자. 그림 5에서 보면 'order/order_info[ISBN]/publisher'라는 XPE는 XML 문서상의 어떤 노드에도 해당되지않는다. 하지만 *XACT*에서는 '12'번 노드에 해당된다. 이 예는 $n_{XML}(p) \subset n_{XACT}(p)$인경우

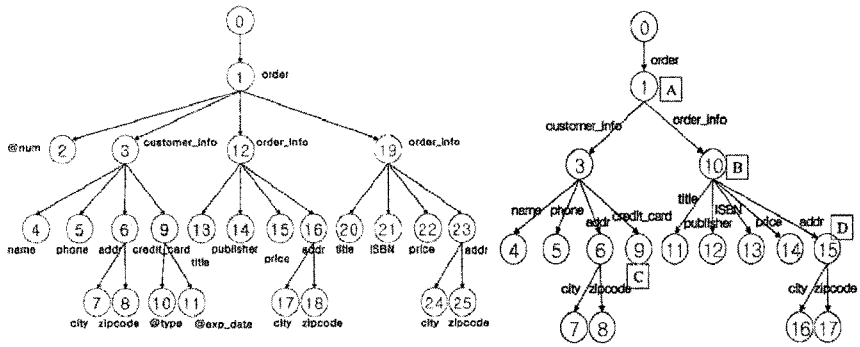


그림 5 XML 문서와 이 문서의 XACT

이다. 두 집합이 일치하는 경우도 가능하다.

어떤 XPE가 '*부정(negation)*'을 포함하는 특별한 경우에 대하여 생각해 보자(예: 'order/order_info[not (ISBN)]/publisher')이 XPE는 XML 문서 트리에서 '14' 번 노드에 해당이 되지만 XACT에서는 어떤 노드에도 해당되지 않는다. 이러한 오류를 방지하기 위해, '부정'을 포함하는 프레디키트에 대하여 특별한 처리를 하여야 한다. 그림 6의 Query Filter 알고리즘에서, 'not'이 포함된 프레디키트를 지니는 XPE에 대해서는 그에 대응하는 XACT 노드들을 검색하지 않고, 프레디키트를 애트리뷰트처럼 취급하여 해당 노드의 부모 노드에 그 정보를 저장하도록 한다. 이런 방법을 통해 XML 문서상에서 어떤 XPE(프레디키트 유무에 관계없이)에 대한 엘리먼트들의 집합은 XACT를 통해서도 동일한 엘리먼트들로 표현된다(특성 1, 2에 대한 자세한 증명은 [22]를 참조하라).

정의 4. 두 XPE p_j와 p_k가 있을 때, XML 문서 트리 D_i에서 그들의 대상 노드 집합 $n(p_j)$과 $n(p_k)$가 동일할 때, 두 XPE가 *동등(equivalent)*하다고 말하며, $p_j \equiv_{Di} p_k$로 나타낸다.

정리 1: XML 문서 Di에 대해 XPE 집합 P와 XACT를 통해 변환된 XPE 집합 P'은 동등하다(즉, $P \equiv_{Di} P'$).

증명: XML 문서 D_i에서, $n(P)$를 XPE 집합 P에 대한 노드들의 집합으로, g를 P에서 $n(P)$로의 함수라고 하자. 그리고, f를 $XACT$를 통한 P에서 P'으로의 함수고 하자(즉, $f(P) = P'$). P의 모든 원소 p와 그 변환 p'이 존재할 때($p' \in P'$), $g(f(p)) = g(p)$이면, p이고 $p' \equiv_{Di} p$이다. 우선, 프레디키트가 없는 p와 그 변환 p'은 특성 1에 따라 $g(p) = g(p')$이 레디키트가 있는 XPE p와 p'의 경우는 다음의 4 황이 가능하다. (1) $g(f(p)) = n(p)$ 이고 $g(p) =$ 정우 ($f(p) = p'$ 이고 $g(p') = g(p) = n(p)$인 p 가 존재한다.) (2) $f(p) = \emptyset$ 이고 $g(p) = \emptyset$ 인 경우 (XML 문서와 $XACT$ 모두에서 p에 대응되는 노드가 존재하지 않기 때문에, $p \equiv_{Di} p'$ 이다.) (3) $g(f(p)) = \emptyset$ 이고 $g(p) = \emptyset$인 경우 ($XACT$에서는 $f(p) = p'$*이지만, XML 문서에서는* $g(p') = \emptyset$ 인 p가 있을 수 있다. 하지만 g 함수의 결과가 공집합이기 때문에 $p \equiv_{Di} p'$이다.) (4) $f(p) = \emptyset$ 이고 $g(p) = n(p)$인 경우 ($XACT$에서는 p에 대응되는 노드가 없지만 XML 문서 트리에는 대응되는 노드가 존재하는 경우를 나타내지만, 특성 1, 2에 따라 $g(p) \subseteq f(p)$이기 때문에 이러한 상황은 발생하지 않는다. 따라서 임의의 p($\in P$)와 p'($\in P'$)에 대해 $p \equiv_{Di} p'$이며, 따라서 $P \equiv_{Di} P'$이다.

추가 정리 1: XPE들은 *합집합*과 *교집합* 연산에 대하여 닫혀있으며, $XACT$를 사용하는 경우 *차집합*연산에 대해서도 닫혀있다. **증명:** [22] 참조

4.2 XACT를 이용한 접근 제어 알고리즘

이 장에서는 *사용자 질의(Input Query)*와 $XACT$의 접근 제어 정보를 비교하여 *접근 허가 질의 (Authorized Query)*를 생성하는 '*Query Filter*'과정을 설명한다. 와 를 각각 접근 허가 및 접근 불가 XPE들의 집합이라고 하고, 를 질의 XPE 집합이라고 하자. 우리의 목적은 '$(\oplus \cap \Pi) - \triangle$'를 계산하여 *Authorized Query*를 생성해 내는 것이다. 4.1절에서 설명한 바와 같이 질의 XPE와 비교할 접근 제어 정보들은 $XACT$에 저장되어 있다.

그림 6에 기술된 *Query Filter*알고리즘에 대하여 알아보자. XACT를 *순회(traverse)*하면서 *Input Query*의 대상 노드들이 해당 사용자에게 허가되어 있는지 그렇지 않은지를 판단한다. 질의 XPE와 대응되는 에지를 발견하게 되면(Algorithm *Query_Filter* (1)행), 노드의 접근 권한을 확인한다. 만일 프레디키트를 포함하는 노드가 '접근 불가'인 경우 (Algorithm *Query_Filter* (2) 행), 조상 노드들의 접근 권한이 본 노드로 상속되었는

```
Algorithm Query_Filter
Input : Input Query(ID, access mode, XPE), XACT
Output : Authorized Query(ID, access mode, XPE)
(1)  for (each node of XACT ∈ XPE)
(2)      if (current_node is access-denied) ∧ (current_node has a predicate)
(3)        ancestor_result := Ancestor_Check(current_node, ID, access_right)
(4)        if (ancestor_result = 'grant')
(5)          not_predicate := Predicate_Comparison(current_node, Input Query)
(6)          if (predicate_XPE != NULL)
(7)                predicate_XPE   := concat('not(', not_predicate, ')')
(8)      else if (current_node is access-denied)
(9)      else if (current_node is access-granted) ∨ (current_node is access-undefined)
(10)         if (current_node is access-undefined)
(11)           ancestor_result := Ancestor_Check(current_node, ID, access_right)
(12)         if (ancestor_result = 'grant') ∨ (current_node is access-granted)
(13)           if (XPE is terminated with '//' or '*')
(14)             descendant_XPE := Descendant_Check(current_node, XPE)
(15)           if (XPE contains predicate)
(15)             for (each descendant_XPE)
(17)               predicate_XPE := Predicate_Comparison(current_node,
(18)                                Query(ID, access mode, descendant_XPE))
(19)     if (the XPE contains predicate) ∧ (predicate_XPE != NULL)
(20)               Auth_XPE := Auth_XPE ∪ predicate_XPE
(21)     else if (the XPE does not contain predicate) ∧ (descendant_XPE != NULL)
(22)               Auth_XPE := Auth_XPE ∪ descendant_XPE

Algorithm Descendant_Check
Input: current_node, XPE
Output  access-granted XPE
(1)  if (XPE terminated with '//')
(2)      if (all descendants are granted) grant_XPE := XPE
(3)      else
(4)        for (each descendant of XPE)
(5)          if (descendant is granted) grant_XPE := grant_XPE ∪ descendant
(6)  else if (XPE terminated with '*')
(7)        if (all child nodes are granted) grant_XPE := XPE
(8)        else
(9)          for (each child of XPE)
(10)           if (the child is granted) grant_XPE := grant_XPE ∪ child

Algorithm Ancestor_Check
Input: current_node, ID, access_right
Output: access right of current_node
(1)  ancestor_node := parent_of_current_node
(2)  if (ancestor_node is access-granted and scope is 'child') return 'grant'
(3)  for (each ancestor_node)
(4)      if (ancestor_node is access-granted and scope is 'recur') return 'grant'
(5)      else if (ancestor_node is access-denied or scope is 'local') return 'denied'
(6)      else ancestor_node := parent_of _ancestor_node     // when ancestor_node is access-undefined

Algorithm Predicate_Comparison
Input : current_node, Input Query(ID, access mode, XPE)
Output : XPE with access-granted predicate
(1)  grant_XPE := path of current_node
(2)  for (each predicate of XPE)
(3)      if ( predicate contains '//' or '*' )
(4)          for (each path of predicate)
(5)              if(path is access-granted) predicate_path := predicate_path ∪ path
(6)      else predicate_path := current_predicate
(7)      for (each access-granted predicate_path)
(8)          node_predicate := node_predicate ∪ predicate_path
(9)      if (node_predicte = NULL) return null
(10)     grant_XPE := node_predicate ∪ grant_XPE
(11) return grant_XPE
```

그림 6 *Query Filter* 알고리즘

지를 확인한다 (Algorithm *Ancestor_Check*, 그림 4에서 '10'번 노드의 접근 권한이 '15'번 노드로 전파된다.) 만일 접근 허가된 조상 노드가 본 노드로 상속되었다면, 프레디키트가 지정하는 부분들을 제외한 나머지 부분에 대하여 질의 XPE는 접근 허가된다. 가령, 그림 4에서 'order/order_info[price=30]/addr'라는 XPE는 *'Alice'*에게 허가된다. 프레디키트 (예: 'order/ order_info[price]')의 접근 권한은 *Predicate_Comparison* 알고리즘에서 처리된다. 만일 프레디키트 없이 어떤 노드가 '접근 불가'라면 그 노드는 접근이 거부된다. 접근 권한이 *'access-undefined'*라면, 그 노드의 접근 권한은 조상 노드들로부터 상속 받을 수 있다. 따라서, *"most*

specific overrides" 정책에 따라, 부모 노드에서부터 경우에 따라 루트 노드까지 확인되어야 한다 (Algorithm *Ancestor_Check*). 물론, 노드에 '*access-granted*' 값이 저장되어 있다면, 그 노드는 접근 허가된 노드로 간주한다. 따라서, XPE가 '//' 또는 '*'로 끝나는지를 확인하여 접근 권한의 범위가 '*recursive*', '*child*' 또는 '*local*'인지의 여부를 확인하여야 한다(Algorithm *Query_Filter* (13)행). 이 경우에 후손 노드들에 대한 접근 권한 확인은 *Descendant_Check* 알고리즘에서 이루어진다. XPE가 프레디키트를 포함하고 있는 경우는 *Predicate_Comparison* 알고리즘에서 처리된다. '//' 및 '*'가 포함된 프레디키트는 프레디키트가 나타내는 모든 경로를 프레디키트 집합으로 만든 후 각 프레디키트들을 확인하게 된다. *XACT*의 모든 에지를 처리하면 *Query Filter* 알고리즘은 종료된다.

예제 3. 4가지 사용자 질의와 이들에 대한 접근 허가 질의를살펴보자. XACT는 그림 4에 나타난 내용을 사용한다.

Q1: (Bob, read, order/customer_info//)
➡ (Bob, read, order/customer_info; order/customer_info/name; order/customer_info/phone; order/customer_info/addr//)
Q2: (Bob, read, //price)➡(Bob, read, order/order_info/price)
Q3: (Alice, read, order/customer_info/name) ➡ Ø
Q4: (Alice, read, //order_info[ISBN]/addr)
➡ (Alice, read, order/order_info[ISBN and not(price > 30)]/addr)

Q1에 대한 대상 노드들은 그림 7에서 '3'번 노드와 그 후손 노드들이다. 이 중에서 '9'번 노드를 제외한 모든 대상 노드들이 이 사용자(Bob)에게 접근 허가되어 있다 Q2는 '14'번 노드에 해당되며, 이 노드는 질의 제출자(Bob)에게 접근 허용된다. Q3는 '4'번 노드에서 접근이 거부된다. Q4는 '15'번 노드에서 프레디키트를 통해 접근이 조절된다. '10'번 노드는 "*most specific overrides*" 정책에 따라 부모 노드들을 검사하여 *재귀적 접근 허용(recursively access-granted)*으로 판별되고, '15'번 노드로 그 결과가 전파된다. '15'번 노드의 경우 프레디키트를 통해 전파되는 권한 중에서 일부를 제한하게 된다. 따라서, '[not(price > 30)]'와 같은 조건을 Q4의 XPE에 부가하여 접근 허가된 질의 XPE를 만들게 된다. □

아래에 기술되는 소정리 및 정리에 사용되는 기호들에 대한 설명은 표 1에 있다.

소정리 1. XML 문서 D_i에 대하여 $(\Theta - \Delta) \equiv_{Di} (\Theta' - \Delta')$이다. **증명:** [22] 참조 □

두 XPE사이의 차집합 연산은 접근 허가 XPE에서 접근 불가 XPE를 뺄 때 사용되며, 두 XPE의 교집합 연산은 질의 XPE와 접근 허가 XPE에 공통되는 부분을 선택하고자 할 때 사용된다.

소정리 2: XML 문서 D_i에 대하여 $(P \cap \Pi) \equiv_{Di} (P' \cap \Pi)$이다. **증명:** [22] 참조 □

정리 2: XML 문서 D_i에 대하여 $((\Theta - \Delta) \cap \Pi) =_{Di} \Pi'$이다.

증명: 본 논문에서는 *DOM 트리 레이블링* 방법[9]을

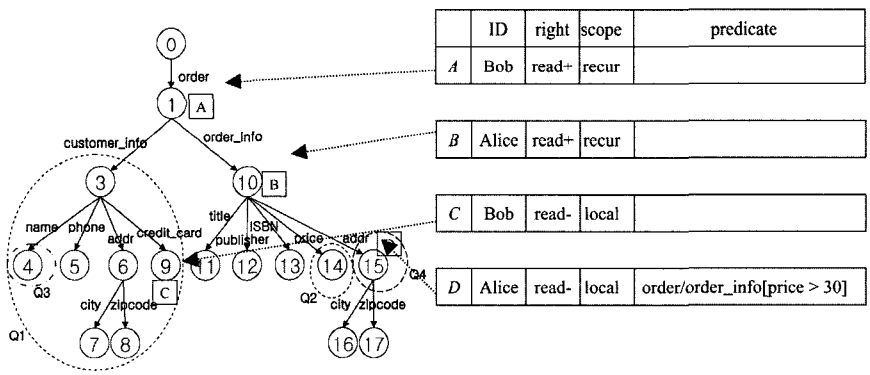


그림 7 예제 3의 *Input Query*에 따른 *XACT* 노드들의 구성

표 1 기호 설명

기호	설명	기호	설명
P	접근제어 XPE들의 집합	P'	XACT Constructor에 의해 P로부터 변환된 XPE들의 집합
Θ	접근 허용 XPE들의 집합	Θ'	XACT Constructor에 의해 Θ 로부터 변환된 XPE들의 집합
Δ	접근 불가 XPE들의 집합	Δ'	XACT Constructor에 의해 Δ 로부터 변환된 XPE들의 집합
Π	질의 XPE들의 집합	Π'	Query Filter를 통해 Π 로부터 필터된 XPE들의 집합

정확한 XML 접근 제어 방법으로 간주한다. 따라서, *DOM 트리 레이블링* 방법의 결과($(\Theta-\Delta)\cap\Pi$)와 제안하는 방법의 결과(Π')가 동일함을 증명하여 제안하는 방법의 정확성(correctness)를 보인다. 주어진 XML 문서 D_i에 대하여 Query Filter 알고리즘에 따라 $\Pi' = (\Theta'-\Delta')\cap\Pi$ 이다. 그리고 정리 1에 의해 $\Theta \equiv_{Di} \Theta'$과 $\Delta \equiv_{Di} \Delta'$임을 알 수 있다. 따라서, 소정리 1에 의해 $\Theta-\Delta \equiv_{Di} \Theta'-\Delta'$ 이고, 소정리 2에 의해 $(\Theta-\Delta)\cap\Pi \equiv_{Di} (\Theta'-\Delta')\cap\Pi$이기 때문에 $(\Theta-\Delta)\cap\Pi \equiv_{Di} \Pi'$임이 증명된다. □

정리 1과 정리 2를 통해서, 본 논문에서 제안하는 *ACL*의 *XACT*로의 변환과 *XACT*를 이용한 접근 제어 방법이 옳은 결과를 생성한다는 사실을 보였다.

5. 성능 평가

제안하는 방법의 성능을 평가하기 위하여 *DOM 트리 레이블링* 방법[9]과 *CAM*[17] 방법 그리고 제안하는 방법을 비교 실험하였다. 측정 기준은 각 방법을 사용하는 데 필요한 노드들의 접근 회수이다. 실험은 질의 XPE가 '*단순(simple) XPE*'와 '*복잡한(complex) XPE*'인 경우로 구분하여 시행하였다. '단순 XPE'란 '//'나 '*'가 포함되지 않은 XPE를 말한다.

실험은 XML 문서의 엘리먼트 개수를 증가시키면서 실험하였으며, 접근 허가 노드를 80%로, 복잡한 XPE의 경우 검색되는 노드의 비율을 30%로 설정하였다. 각 사용자별 *ACL*에 저장되는 XPE들의 수는 평균 10개로 하였으며, 복잡한 XPE에 대응되는 *XACT* 경로의 수는 평균 10으로 설정하였다.

그림 8은 실험 결과를 나타낸다. *CAM* 방법은 단순 XPE를 사용하는 경우에만 적용 가능한 방법이기 때문에, 복잡한 XPE에 대해서는 실험하지 않았다. 단순한 XPE에 대한 실험에서 *CAM*과 제안하는 방법은 동일한 성능 결과를 보인다. 반면 *DOM 트리 레이블링* 방법은 단순 XPE와 복잡 XPE 모두에서 제안하는 방법보다 매우 많은 노드 접근 회수를 보이고 있다.

6. 결 론

본 논문에서는 XML 데이타에 대한 접근 제어 방법을 제안하였다. 제안하는 방법은 *ACL* 형태의 입력으로 주어지는 사용자 접근 제어 정보를 *XACT*라고 부르는 트리 형태의 구조로 변환하여 *Query Filter* 알고리즘에 따라 질의 XPE와 비교하여 접근이 가능한 부분만을 추출하여 '접근 인가 질의 XPE'를 생성한다. XML 문서를 접근하지 않고도 접근이 가능한 부분을 찾아낼 수 있는 방법이다. 제안하는 방법의 정확성을 증명하였으며, 기존 방법들과의 성능 비교 실험을 통해, 제안하는 방법의 우수함을 보였다. 특히 프레디키트, '//' 또는 '*' 등을 포함하는 복잡한 형태의 XPE에 대해서도 접근 제어가 가능하며, XML 문서의 *실체화 뷰(materialized view)*에 대한 접근 제어에도 쉽게 적용할 수 있는 특성을 지닌다.

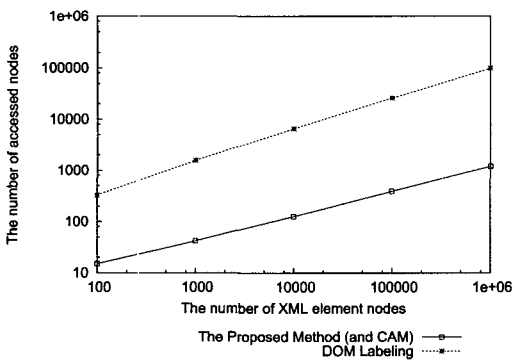
참 고 문 헌

[1] http://www.nue.et-inf.uni-siegen.de/geuerpoll-mann/xml_security.html

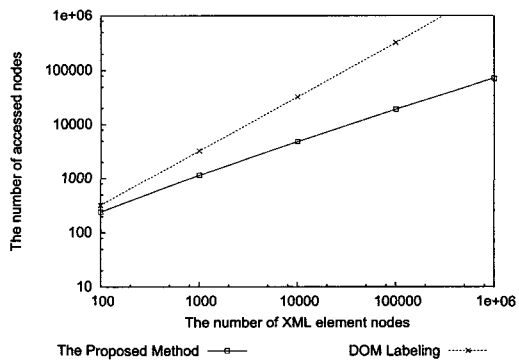
[2] OASIS, eXtensible Access control Markup Language(XACML) Version 1.0, OASIS Standard, February 2003. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

[3] OASIS, Security Assertion Markup Language (SAML) Version 1.1, OASIS Standard, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

[4] W3C, Canonical XML Version 1.0, W3C Recom-



(a) 단순한 XPE에 대한 실험



(b) 복잡한 XPE에 대한 실험

그림 8 성능 비교 실험 결과

mendation, March 2001. http://www.w3.org/TR/xml-c14n.
[5] W3C, SOAP Security Extensions: Digital Signature, W3C Note, February 2001. http://www.w3.org/TR/SOAP-dsig/
[6] W3C, XML Key Management Specification (XKMS), W3C Note, March 2001. http://www.w3.org/TR/xkms/
[7] W3C, XML-Signature Syntax and Processing, W3C Recommendation, February 2002. http://www.w3.org/TR/xmldsig-core/
[8] E. Bertino, S. Castano, E. Ferrari "Securing XML documents with Author-X," IEEE Internet Computing, 5(3):21-31, 2001.
[9] E. Damiani, S. De Capitani di Vimercanti, S. Paraboschi, P. Samarati "Securing XML Documents," In Proc. of the 2000 Int'l Conference on Extending Database Technology (EDBT2000), pp. 121-135, Germany, March 2000.
[10] E. Damiani, S. De Capitani di Vimercanti, S. Paraboschi, P. Samarati "Securing SOAP e-services," IJIS 1:100-115, 2002.
[11] M. Kudo, S. Hada "XML Document Security based on Provisional Authorization," CCS 2000, pp. 87-96, Athens, Greece
[12] G. Miklau, D. Suciu "Containment and Equivalence for an XPath Fragment," ACM PODS, pp. 65-76, Wisconsin, June, 2002.
[13] W3C, XML Path Language (XPath) Version 2.0, W3C Working Draft, 2003, http://www.w3.org/TR/2003/WD-XPath20-20030502/
[14] S. Jajodia, P. Samarati, M.L. Sapino, V. S. Subrahmanian "Flexible Support for Multiple Access Control Policies," ACM Trans. On Database Systems, 26(2):214-260, June 2001.
[15] W3C, XQuery: A Query Language for XML. W3C Working Draft, May 2003. http://www.w3.org/TR/2003/WD-xquery-20030502/
[16] W3C, XSL Transformations (XSLT) Version 1.0, W3C Recommendation, November 1999. http://www.w3.org/TR/xslt
[17] T. Yu, D. Srivastava, L. Lakshmann, H. Jagadish "Compressed Accessibility Map: Efficient Access Control for XML," In Proc. of the 28th VLDB Conference, pp. 478-489, Hong Kong, China, 2002.
[18] S. Cho, S. Amer-Yahia, L. Lakshmanan, D. Srivastava "Optimizing the Secure Evaluation of Twig Queries," In Proc. of the 28thVLDB Conference, pp. 490-501, Hong Kong, China, 2002.
[19] S. Cho, S. Amer-Yahia, L. Lakshmanan, D. Srivastava "LockX: A System for Efficiently Querying Secure XML," In Proc. of the SIGMOD 2003 Conference, pp. 669, San Diago, CA, 2003.
[20] M. Murata, A. Tozawa, M. Kudo "XML Access Control Using Static Analysis," CCS 2003, pp. 73-84, Washington, DC, USA, 2003.
[21] M. Benedikt, W. Fan, G. Kuper "Structural Properties of XPath Fragments," ICDT 2003, pp. 79-95, Italy, January 2003.
[22] J. M. Jeon, Y, D, Chung, Y. J. Lee and M. H. Kim "Filtering of XPath Expressions for XML Access Control," Technical Report (CS-TR-2004-199), Division of Computer Science, KAIST, 2004.

전 재 명
1988년 2월 공군사관학교 전산학과 학사 1996년 2월 오클라호마 주립대 전산학과 석사. 2005년 2월 한국과학기술원 전자전산학과 전산학전공 박사. 현재 공군 E-X 사업단. 관심분야는 XML, Database Security, Database Systems 등

정 연 돈
1994년 2월 고려대학교 전산과학과 학사 1996년 2월 한국과학기술원 전산학과 석사. 2000년 8월 한국과학기술원 전자전산학과 전산학전공 박사. 2000년 9월~2001년 8월 한국과학기술원 정보전자연구소 Post-Doc. 연구원. 2001년 9월~2003년 2월 한국과학기술원 전자전산학과 전산학전공 연구교수. 2003년 3월~현재 동국대학교 컴퓨터공학과 교수. 관심분야는 XML Database, Stream Data Processing, Mobile Databases, Sensor Networks, Database Systems 등

김 명 호
1982년 서울대학교 컴퓨터공학과 학사. 1984년 서울대학교 컴퓨터공학과 석사. 1989년 미국 Michigan State University 전산학과 박사. 현재, 한국과학기술원 전산학전공 교수. 관심분야는 데이타베이스 시스템, 분산시스템, XML, Sensor Networks 등

이 윤 준
1977년 서울대학교 계산통계학과 학사. 1979년 한국과학원 전산학과 석사. 1983년 프랑스 INPG-ENIMAG 전산학과 박사. 현재, 한국과학기술원 전산학전공 교수. 관심분야는 데이타베이스 시스템, 멀티미디어 정보 검색, WWW 등