Journal of Korea Multimedia Society Vol. 10, No. 3, March 2007(pp. 289-302)

XML 데이터베이스에서 CXQuery의 XQuery 변환 기법

이민영[†], 용환승[††], 이월영[†††]

요 약

XML 문서 구조를 모르고서도 질의할 수 있는 CXQuery라는 질의 언어에 대한 기존의 질의 처리 기법은 관계형 데이터베이스를 사용하기 때문에 XML 문서 구조를 관계형 테이블에 매핑하는 문제와, 질의 처리시나 결과를 반환하기 위하여 테이블간의 조인 때문에 운영 상에 어려움을 지니고 있다. 본 논문에서는, 표준화가 진행 중인 XQuery 질의 처리 기법을 이용하기 위하여 CXQuery를 XQuery로 변환하는 변환기를 개발하였다. 이 변환기의 변환 속도는 질의 처리하는 전체 시간에 비해 무시할 정도의 짧은 시간이 걸린다. 또한 기존의 관계형 데이터베이스와 관계없이 XML 문서에 대해 직접적으로 질의 처리가 가능하도록 하며, 사용자는 CXQuery를 이용하여 문서 구조를 모르고서도 질의할 수 있도록 하는 장점을 갖는다.

A Technique of Converting CXQuery to XQuery for XML Databases

Minyoung Lee[†], Hwan Seung Yong[††], Wol Young Lee[†††]

ABSTRACT

The existing query processing technique for CXQuery, which is able to query regardless of knowledge about XML document structures, is difficult to manage because of table join for query processing and results return, mapping XML documents into relational tables, and so on. In this paper, we have developed a converter capable of converting CXQuery to XQuery in order to make use of the query processing techniques for XQuery progressing standardization. The converting speed of the converter takes a trifling time as much as negligible quantities in comparison with the total query processing time. This is also able to query directly XML documents regardless of relational databases, and users can query without knowledge about XML document structures.

Key words: XML Database(데이터베이스), XQuery, Query Converter(질의 변환기)

1. 서 론

XML[1]은 다양한 데이터 종류를 임의 형태로 쉽게 표현할 수 있다는 장점 때문에 많은 분야에서 XML을 이용하여 데이터를 표현하고 있고, 이에 따라 사용자들이 원하는 것을 검색할 수 있도록 하기 위해 많은 질의 언어들도 제안되었다[2-5]. 그러나 지금까지 제안된 여러 가지 질의 언어들은 질의 표현에 있어, XML 문서 구조에 절대적으로 의존하고 있기 때문에 문서 구조가 달라질 때마다 다른 질의 표

※ 교신저자(Corresponding Author) : 용환승, 주소 : 서울시 서대문구 대현동 이화여자대학교 컴퓨터학과(120-750), 전화 : 02)3277-2592, FAX : 02)3277-2306, E-mail : hsyong@ewha.ac.kr

접수일 : 2006년 11월 27일, 완료일 : 2007년 1월 18일

[†] 삼성전자
(E-mail : seia2@hanmail.net)
[††] 종신회원, 이화여자대학교 컴퓨터학과
[†††] 서울기독대학교 국제경영정보학과
(E-mail :wylee@ewha.ac.kr)

※ 본 연구는 한국과학재단 목적기초연구(R01-2006-000-10609-0) 지원으로 수행되었음

현을 해야만 한다. 이 점은 사용자를 불편하게 할 뿐만 아니라 문서 구조를 모르는 경우 거의 질의가 불가능하게 된다. 따라서 사용자가 문서의 구조를 모르더라고 질의가 가능한 CXQuery가 제안되었다[6,18,19].

CXQuery는 단지 검색하고자 하는 데이터 이름과 그 값만을 명시하고 데이터들 사이의 경로를 명시하지 않는 내용 기반 애드 혹 질의를 지원하도록 한다. 하지만 현재까지의 CXQuery는 XML 문서를 파싱하여 관계형 데이터베이스에 저장을 한 뒤 질의를 처리하기 때문에 XML 문서에 대해 직접적인 질의 처리를 하지 못하고 관계형 데이터베이스로의 매핑을 위한 변환 과정과 질의 처리시뿐 아니라 결과를 반환할 때도 XML 형식의 문서 형태로 변환하기 위한 특별한 시스템을 구축해야 한다. 이런 방법은 사용자에게 편의를 제공하는 반면 시스템 구축에 시간 및 비용이 많이 들어가며 각 도메인 별로 새로운 엔진을 구축하여야 하는 트레이드 오프(trade-off)를 갖는다[6].

반면 XQuery와 같은 경로 기반 질의 언어들은, 사용자가 질의 하기 위해서 문서 구조를 모두 알아야 하는 단점이 있지만 기존의 XML 데이터베이스를 사용 가능하다는 장점을 가진다. 이에 사용자들이 문서 구조를 모르고도 편하게 질의할 수 있도록 CXQuery 문으로 질의 표현을 하고 이를 문서 구조에 기반한 XQuery문으로 바꾸어 줌으로써 XML 문서와 관계형 데이터베이스 사이의 매핑을 위한 여러 번의 변환 과정 없이 질의가 가능하다.

본 논문에서는 사용자들은 CXQuery를 사용하여 문서 구조에 독립적으로 질의하고 질의 처리는 기존의 DBMS에 독립적으로 할 수 있도록 XQuery로 변환하는 변환기를 개발하였다. 즉, 사용자들은 문서 구조를 모르고도 편하게 질의하지만 질의처리기는 CXQuery 질의문을 XML 문서 구조를 분석하여 여러 개의 XQuery문으로 변환하여 사용자에게 보여주고 사용자는 자신이 원하는 XQuery문을 선택하게 하도록 하였다. 이 점은 사용자가 문서 구조를 자세히 알지 못해도 문서 구조에 기반한 보다 정확한 질의 표현을 이용하여 질의할 수 있도록 한다. 또한 관계형 데이터베이스를 사용하지 않고 XML 문서에 대해 직접 질의 처리를 함으로써 XML 문서와 관계형 데이터베이스 사이의 구조상 차이로 인하여 발생하는 모든 문제점을 해결하였다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대해 소개하고, 어떻게 XML 문서에 대해 질의 하는지 알아본다. 3장에서는 본 논문에서 제안하는 CXQuery를 XQuery로 변환하여 질의하는 시스템의 구조와 방법에 설명하였다. 그리고 4장에서는 구현과 그 결과를 5장에서는 향후 연구 방향과 결론에 대해서 살펴본다.

2. 관련연구

XML 문서는 같은 데이터를 가지고서도 표현할 수 있는 방법이 다양하기 때문에 수없이 많은 문서 구조를 가질 수 있다. 이에 따라 기존에 대부분의 질의 언어들은 정확한 질의를 위해 질의 표현이 XML 문서 구조를 따라 항해하는 방식으로 표현하는 경로 기반으로 하게 되어 있고 이는 문서 구조를 모르는 사용자들은 질의를 할 수 없도록 하는 단점이 되었다. 이를 극복해 보고자 문서 구조에 독립적인 질의를 할 수 있도록 자연 언어에서와 비슷한 CXQuery라는 질의어가 제안되었다[6,18,19]. CXQuery에 대한 이해를 돕기 위하여 예로써 다음과 같은 질의를 가정해 보자.

Q1: 1994년에 제작된 영화로서 genre가 action이고 'Jean Reno'이라는 actor가 주연한 영화의 제목은 무엇인가?

CXQuery는 사용자가 질의하고자 하는 데이터 이름과 값만을 안다면 구조에 대해서는 고민하지 않고 데이터의 형태 (Element or Attribute)나 데이터의 경로에 상관없이 이들 모든 문서에 대해 다음과 같은 똑같은 하나의 질의 표현으로 질의가 가능하도록 하였다.

```
for $c in doc()
where year="1994" and genre="action" and
actor="Jean Reno"                                (E1)
return title
```

이러한 표현법은 문서 구조를 모르는 사용자라 하더라도 자신이 검색하고자 하는 데이터 이름과 그 값 만을 알면 원하는 결과를 찾을 수 있도록 해준다. 그러나 CXQuery에서의 질의 처리 기법은 관계형 데이터베이스를 기반으로 하고 있기 때문에, 여러 가지 문제점을 안고 있다[6].

많은 애플리케이션에서 데이터 저장, 관리를 위해

관계형 데이터베이스가 훌륭한 역할을 수행하지만 XML 기반의 애플리케이션에 있어서는 관계형 데이터베이스가 적합하지 않을 수 있다. IBM DB2[7], Oracle[8], MS SQL 서버[9]과 같은 관계형 데이터베이스를 XML 데이터 저장소로 채택하는 경우, 설계 단계에서부터 XML을 관계형 테이블 구조로 매핑해야 하고 이에 따라 많은 수의 테이블이 만들어져야 한다. 더욱이 XML 데이터가 복잡하고 계층이 깊을수록 많은 수의 테이블을 필요로 한다. XML 문서를 저장, 질의 처리, 결과를 반환하는 과정에 이르기까지 XML과 관계형 테이블간의 구조적인 상이함으로 인하여 이를 맞추기 위한 노력은 성능의 저하를 가져올 수 있다. 또한 유지, 보수 단계에서도 다수의 테이블을 관리해야 하는데 따른 어려움이 있고, XML 데이터의 구조 변경에 유연하게 대처할 수가 없다. 대부분 XML 데이터는 데이터중심과 문서중심으로 명확하게 분리되기보다 두 가지 유형의 특성을 혼합하여 가지고 있는 경우가 많다. 데이터 중심 XML 저장 관리에 적합한 관계형 데이터베이스를 XML 데이터 저장소로 채택했다가 문서 중심 XML의 특성을 가진 데이터를 저장해야 할 경우 관계형 데이터베이스에서는 이를 수용할 수 없는 경우도 있을 것이다. XML 문서를 데이터베이스로부터 꺼냈을 때 원본 XML 문서를 손상 없이 그대로 복원해야 하는 경우에도 관계형 데이터베이스는 적합하지 않다[10-12].

이와 같이 관계형 데이터베이스가 수용하지 못하는 XML 데이터를 관리해야 하거나 XML 데이터를 다른 구조로 매핑, 변환하기 위해 생기는 문제점을 피하기 위해서는 네이티브 XML 데이터베이스를 검토해야만 한다. 대표적인 솔루션으로는, Software AG의 Tamino[13], X—Hive사의 X—Hive/DB[14], eXcelon 사의 eXtensible Information Server(XIS)[15], Ipedo사의 Ipedo[16]등이 있다.

3. CXQuery 변환기의 설계

본 논문에서 제안한 시스템은 크게 XML 문서의 저장, CXQuery[6,18,19]를 그에 해당하는 XQuery로 변환, 변환된 XQuery를 사용하여 저장된 XML 문서에 대해 질의를 실행하는 3단계로 거쳐 처리한다. 시스템의 전체 구조도는 그림 3-1과 같다. 문서 구조를 잘 아는 사용자나 모르는 사용자나 사용자

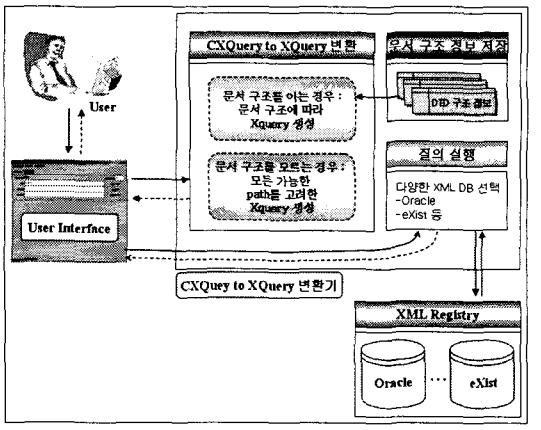


그림 3-1. 변환기를 사용한 XML DBMS 시스템 구조도

인터페이스를 통해 CXQuery를 이용하여 질의문을 입력하고 입력된 CXQuery문은 변환기를 통해 문서 구조에 따라 다양한 XQuery문으로 바꾸어 사용자에게 보여준다. 이 때 사용자는 자신이 원하는 보다 정확한 XQuery문을 고를 수 있다. 이것은 사용자에게 선택된 XQuery만을 실행함으로써 전체 XQuery를 실행하는 것보다 모호성을 줄이고 XQuery를 실행하는데 드는 시간과 비용을 절약할 수 있다. 구현한 CXQuery의 XQuery 변환기는 사용자가 CXQuery로 쉽게 질의가 가능하면서 기존의 XML 데이터베이스와 연동하여 처리할 수 있도록 해준다. 또한 XML 문서 구조를 아는 경우는 DTD에 따라 XQuery로 변환하도록 하였고, XML 문서 구조를 모른다 하더라도 질의가 가능하도록 하기 위해 사용자가 입력한 데이터 이름에 대한 모든 가능한 경로를 고려하여 질의를 생성하였다.

3.1 XML 문서 구조를 모르는 경우

본 시스템에서는 XML 문서에 DTD(Document Type Definition)가 없는 경우, CXQuery문에 주어지는 데이터 이름과 값을 가지고 이들이 가질 수 있는 모든 가능한 경로를 분석하여 XQuery문으로 변환하도록 하였다[19].

예를 들어, 다음과 같은 *E1* 질의문을 좀 더 간소화한 질의문 *E2*를 고려해 보자.

```
for $c in doc()
where genre="action" and actor="Jean Reno"  (E2)
return title
```

여기서 genre="action"이라는 조건에 대해 XQuery 질의문으로 바꾸어 주기 위해서는 먼저 XML 문서에서 genre이라는 데이터 이름과 데이터 값으로써 "action"이 가질 수 있는 모든 가능한 경로를 고려해 주어야 하는데 이러한 모든 경로를 분석하면 표 3-1과 같다.

표 3-1에서 보는 것처럼 genre와 "action"사이의 가능한 모든 경로를 고려하여 질의 조건을 표현한다면 다음과 같이 쓸 수 있다.

```
//genre="action" OR //@genre="action" OR
//genre//*="action" OR //genre//@*="action"
```

같은 근거로 CXQuery문의 actor="Jean Reno" 문도 다음과 같이 변환될 것이다.

```
//actor="Jean Reno" OR //@actor="Jean Reno"
OR //actor//*="Jean Reno" OR
//actor//@*="Jean Reno"
```

Return 절의 title의 경우에는 데이터 이름만 주어질 뿐, 값이 주어지지 않기 때문에 title이 엘리먼트인 경우와 애트리뷰트인 경우 2가지로만 나누어진다.

```
//title OR //@title
```

이러한 조건에 의해 문서구조를 모르는 경우, *E2*을 XQuery문으로 변환한다면 결과를 반환하는 title의 데이터 타입에 따라 다음과 같이 두 가지 질의문으로 변환된다.

```
for $c in doc()
where ($c//genre="action" or
       $c//genre//*="action" or
       $c//@genre="action" or
       $c//genre//@*="action") and
      ($c//actor="Jean Reno" or
       $c//actor//*="Jean Reno" or
       $c//@actor="Jean Reno" or
       $c//actor//@*=" Jean Reno")
return $c//title
```

또는

```
for $c in doc()
where ($c//genre="action" or
       $c//genre//*="action" or
       $c//@genre="action" or
       $c//genre//@*="action") and
      ($c//actor="Jean Reno" or
       $c//actor//*="Jean Reno" or
       $c//@actor="Jean Reno" or
       $c//actor//@*="Jean Reno")
return $c//@title
```

표 3-1. genre와 "action"사이의 모든 가능한 경로들

가능한 경로	XQuery 질의 표현
데이터 이름이 엘리먼트고 이 엘리먼트가 값을 가지는 경우	//genre="action"
데이터 이름이 애트리뷰트인 경우	//@genre="action"
데이터 이름이 엘리먼트고 이 엘리먼트의 자식 엘리먼트가 값을 가지는 경우	//genre//*="action"
데이터 이름이 엘리먼트고 이 엘리먼트의 자식 엘리먼트의 애트리뷰트가 값을 가지는 경우	//genre//@*="action"

3.2 XML 문서 구조를 아는 경우

XML 문서에 DTD가 있는 경우, 구조 정보를 파싱하여 저장해 두었다가 CXQuery 문으로 입력된 데이터 이름에 해당하는 노드를 찾아 완전한 경로를 적어줌으로써 문서 구조에 맞춰 XQuery로 변환하게 된다. 문서의 구조가 다르다면 데이터 경로도 달라지기 때문에 DTD마다 서로 다른 XQuery가 생성된다. 따라서 변환 과정은 DTD의 구조 정보를 저장한 뒤 CXQuery의 각 데이터 이름을 구조 정보 테이블에서 전체 경로를 찾아 CXQuery의 각 데이터에 치환한다.

3.2.1 구조 정보 테이블 저장

질의 Q1을 수행하는 문서의 DTD가 그림 3-1과 같다고 하자. 이들 중, PCDATA타입의 데이터를 부모를 따라 루트까지 추적하면 각 데이터의 전체 경로를 알 수 있다.

이러한 데이터 정보를 자신의 부모까지의 경로와 데이터 정보를 가진 테이블 형태를 표 3-2과 같이 저장한다.

```
<!ELEMENT movies (movie+)>
<!ELEMENT movie (country, genre, title,
          director, actor)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT genre (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT director (#PCDATA)>
<!ELEMENT actor (#PCDATA)>
<!ATTLIST movie year CDATA #REQUIRED>
```

그림 3-1. 예제 DTD 문서

표 3-2. 그림 3-1에 대한 구조 정보 테이블

Type	Data type	Path	Name
ELE	P	/	movies
ELE	P	/movies	movie
ATT	L	/movies/movie	year
ELE	L	/movies/movie	country
ELE	L	/movies/movie	genre
ELE	L	/movies/movie	title
ELE	L	/movies/movie	director
ELE	L	/movies/movie	actor

테이블에 저장된 각 열의 내용은 다음과 같다.

- **Type**: 데이터가 엘리먼트(Element)인지 애트리뷰트(Attribute)인지를 저장한다.
- **Data type**: 데이터의 위치를 알려준다. 즉, 데이터가 말단 노드인지 자식 노드를 가지고 있는지의 정보를 저장한다. 저장시 사용되는 약자는 다음을 뜻한다.
  - ⇨ L인 경우: 데이터가 말단 노드임을 뜻한다. 이 노드는 값을 가질 수는 있으나 (#PCDATA 형) 자식 엘리먼트는 가질 수 없다.
  - ⇨ P인 경우: 데이터가 자식 노드만을 가짐을 뜻한다. 이 노드는 자식 엘리먼트만을 가지며 자신은 값을 가질 수 없다.
  - ⇨ C인 경우: 데이터가 값을 가질 수 있으며 자식 엘리먼트도 가지고 있음을 뜻한다.
- **Path**: 데이터의 부모까지의 경로를 저장한다.
- **Name**: 데이터의 이름을 저장한다.

또한, XML 문서의 경우 같은 데이터 이름을 반복하여 가질 수 있다. 그림 3-2는 데이터 이름이 중복 사용된 DTD문서를 보여준다.

```
<!ELEMENT movie (year, country, genre, title,
          director, actor)+>
<!ELEMENT year (#PCDATA)>
<!ATTLIST year yyyy CDATA #IMPLIED>
<!ELEMENT country (#PCDATA)>
<!ATTLIST country name CDATA #IMPLIED>
<!ELEMENT genre (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ATTLIST title name CDATA #IMPLIED>
<!ELEMENT director (#PCDATA)>
<!ATTLIST director name CDATA #IMPLIED>
<!ELEMENT actor (#PCDATA)>
<!ATTLIST actor name CDATA #IMPLIED>
```

그림 3-2. 데이터 이름이 중복하여 나오는 경우

그림 3-2 문서 구조에서는 name이라는 데이터 이름이 4번이나 반복된다. 이에 대한 구조 정보 테이블은 표 3-3과 같다.

만약 CXQuery에서 name이라는 데이터 이름이 사용된다면 Name의 경로는 /movie/country/@name, /movie/title/@name, /movie/director/ @name, /movie/actor/@name 이렇게 4개의 경로를 리턴해 주게 된다.

3.2.2 XQuery 생성

문서 구조 정보 테이블을 이용하면 데이터 이름과 값만으로 질의를 하여도 그에 대응하는 데이터를 찾아 그 경로를 찾아 낼 수 있다. 따라서 문서 구조를 아는 경우의 CXQuery문 변환은 다음과 같은 순서를 거친다.

표 3-3. 그림 3-2를 위한 구조 정보 테이블

Type	Data Type	Path	Name
ELE	P	/	movie
ELE	P	/movie	year
ELE	L	/movie	country
ELE	L	/movie	genre
ELE	L	/movie	title
ELE	L	/movie	director
ELE	L	/movie	Actor
ATT	L	/movies/year	yyyy
ATT	L	/movie/country/@name /movie/title/@name /movie/director/@name /movie/actor/@name	name

1) CXQuery문에 입력된 데이터 이름에 대한 경로를 문서 구조 정보 테이블에서 찾는다.
2) 각 문서 경로로부터 동일한 경로를 구한다.
3) 각 데이터의 동일한 경로는 for문의 doc() 부분을 바꿔준다.
4) 각 데이터의 경로에서 동일한 경로 이후 부분으로 where과 return절의 데이터 경로로 수정한다. 수정 시 노드의 Data type을 고려하여 수정한다.

즉, 그림3-1과 같은 DTD를 갖는 문서에 대해 *E2*을 XQuery로 변환한다면, 우선 CXQuery 문에 사용된 데이터 이름 genre, actor, title에 대해 표 3-2의 구조 정보 테이블로부터 그들의 경로를 찾는다. genre의 완전 경로는 /movies/movie/genre, actor의 완전 경로는 /movies/movie/actor, title의 완전 경로는 /movies/movie/title이 된다. 여기서 genre, actor, title의 동일 경로는 **/movies/movie** 임을 알 수 있다. 따라서 CXQuery의 for $c in doc() 는 for $c in /movies/movie로 바꾸어 준다. 또한 where절과 return절에 쓰인 각 데이터들을 총 데이터 경로에서 동일한 경로를 제외한 나머지 경로로 바꾸어 준다. 이러한 과정에 의해 바뀐 XQuery문은 다음과 같다.

```
for $c in /movies/movie
where $c/genre="action" and $c/actor="Jean Reno"
return $c/title
```

위의 구조 정보는 각 데이터들의 데이터 타입이 L인 말단 노드인 경우에 변환된 XQuery이다. 말단 노드의 경우 바로 자신이 스트링 타입의 값을 가질 수 있기 때문에 데이터 이름에 바로 값을 대입하는 형태로 변환이 가능하다.

만일 구조 정보 테이블에서 찾은 데이터의 데이터 타입이 P나 C라면, 데이터 이름과 그 값 사이에 또 다른 데이터 이름이 삽입되어 있는 경우이기 때문에 이 점을 고려하여 변환하여야 한다. 즉, 데이터 타입에 따라 각각 다음과 같은 해석을 가지며 경로 변환도 이에 따라 해주어야 한다.

- L 인 경우: 데이터 이름이 스트링 값을 가질 수 있다.
  - /movies/movie/genre="action"
- P 인 경우: 자식 노드를 갖는 경우이므로, 애트리뷰트, 자식 노드, 자식 노드의 애트리뷰트가 값을 갖는다.
  - /movies/movie/genre/@*="action"
  - /movies/movie/genre//*="action"
  - /movies/movie/genre//*/@*="action"
- C 인 경우: 위의 두 가지 가능성을 모두 가진다.
  - /movies/movie/genre="action"
  - /movies/movie/genre/@*="action"
  - /movies/movie/genre//*="action"
  - /movies/movie/genre//*/@*="action"

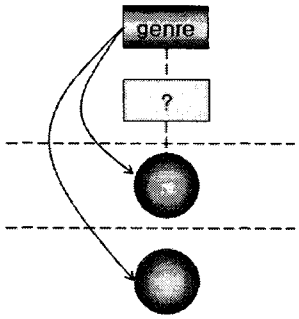
따라서 구조 정보 테이블에서 찾은 데이터의 타입이 C였다면 생성되는 XQuery 문은 다음과 같다.

```
for $c in /movies/movie
where ($c/genre="action" or
       $c/genre//*="action" or
       $c/genre//*/@*="action" or
       $c/genre/@*="action") and
       $c/actor="Jean Reno"
return $c/title
```

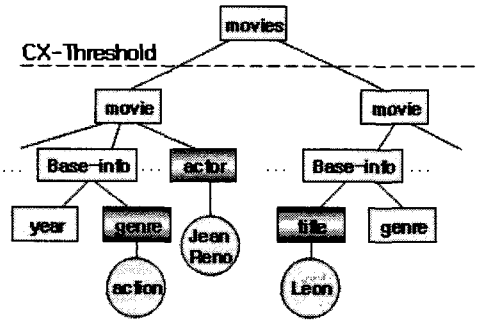
3.3 신뢰도 조절

CXQuery는 조건절에 주어지는 데이터 이름과 데이터 값 사이에 모든 경로를 생략하여 표현한다[19]. 이것은 사용자가 질의할 때 자신이 알고 있는 데이터 이름과 그 값 사이에 어떤 데이터가 삽입되는지 또는 데이터 이름 사이의 경로는 어떻게 되는지에 대해 전혀 걱정할 필요가 없다는 것을 의미한다. 이에 반해 질의 처리기는 이렇게 질의문에 생략된 경로를 모두 분석하여 문서상에서 그러한 데이터 이름과 값을 포함하고 있는 모든 문서 조각들을 검색해 내야 한다는 것을 의미한다. 따라서 질의 결과에는 사용자가 원하는 정확한 데이터 외에 부수적인 데이터가 포함될 수 있다. 본 논문에서는 이러한 단점을 극복하고자 간단한 방법으로 생략된 경로의 공통 경로를 제외한 상이한 경로의 레벨 차에 따라 한계 값(threshold)를 둠으로써 신뢰도를 조절하였다.

예를 들어, 그림 3-3의 (a)처럼 데이터 이름과 값 사이에 임의의 데이터가 삽입되어 있는 경우, 이것은 3.2.2절에서 설명한 것처럼 구조 테이블에서 찾은 데이터 타입에 따라 질의문을 변환한다. 또한 데이터 이름과 값 사이의 거리의 한계 값을 지정함으로써 신뢰도를 조절하였다.



(a) 데이터 이름과 값 사이 거리



(b)데이터간 거리

그림 3-3. 데이터간의 거리

그림 3-3의 (b)에서 genre와 actor는 같은 movie에 속해 있지만 title은 다른 movie에 속해 있다. (b)를 가지고 질의를 처리할 경우 사용자가 원하지 않는 결과를 가져다 줄 가능성이 크다. 따라서 각 데이터 거리에 따른 한계 값을 둠으로써 genre, actor, title이 같은 영역 안에서 결과를 반환할 수 있도록 하여야 한다. 이에 간단히 사용자가 최소 동일 패스를 조정할 수 있도록 하였고 이를 CX-Threshold이라고 하겠다. 즉 사용자가 받고 싶은 결과인 return 절의 데이터를 기준으로 return절의 데이터 경로와 상이한 경로의 레벨을 CX-Threshold으로 지정하였다.

- CX-Threshold 이 0인 경우: where절의 데이터의 경로는 return 절의 데이터의 경로를 포함한다.
- CX-Threshold이 1인 경우: where절의 데이터의 경로는 return 절의 데이터의 부모까지의 경로를 포함한다.

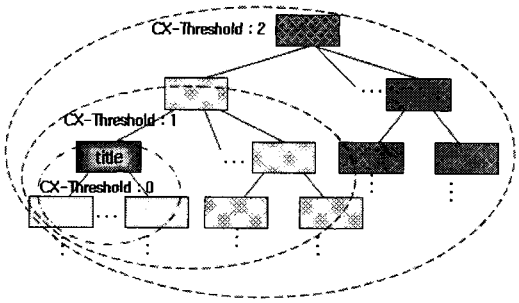
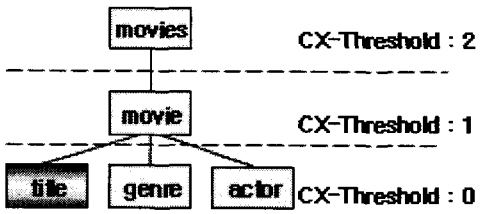


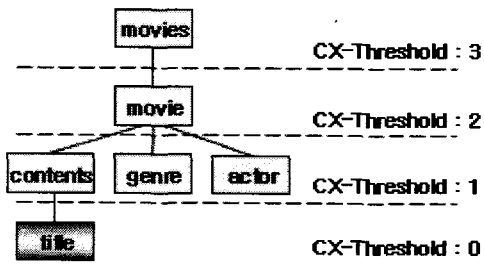
그림 3-4. CX-Threshold에 따른 거리 계산

위의 한계 값으로 신뢰도를 조정할 경우 그림 3-4의 경우에 대해 생성된 XQuery는 달라지게 된다. 즉 /movies/movie/title, /movies/movie/genre, /movies/movie/actor의 경우 return절의 title의 경로를 기준으로 CX-Threshold이 0일 경우 공통 경로로 /movies/movie/title이 모든 데이터 경로에 포함 되어야 하며, CX-Threshold이 1인 경우에는 공통 경로로 한 레벨 위인 /movies/movie가 공동 패스로, 2인 경우는 두 레벨 위인 /movies가 공동 패스로 포함되어야 한다.

그림 3-5에서 한계 값에 따라 생성된 XQuery의 결과는 표 3-4와 같다.



(a) 문서1



(b) 문서2

그림 3-5. CX-Threshold의 값에 따른 공통 경로

표 3-4. CX-Threshold값에 따른 XQuery 생성

CX-Threshold : 1	CX-Threshold : 2
(a) for $c in /movies/movie where $c/genre="action" and $c/actor="Jean Reno" return $c/title	(a) for $c in /movies where $c/movie /genre="action" and $c/movie /actor="Jean Reno" return $c/movie /title
(b) 생성되지 않음	(b) for $c in /movies/movie where $c/genre="action" and $c/actor="Jean Reno" return $c/contents/title

3.4 CXQuery의 다의성 해결

CXQuery의 경우 데이터 이름과 데이터 값만을 명시해 줌에 따라 질의 처리시 그러한 데이터 이름과 값을 포함하는 모든 문서의 조각들을 검사한다. 예를 들어, actor와 "Jean Reno"의 쌍을 생각해보자.

```
<actors>
        <theater>
                <actor> Jean Reno </actor>
        </theater>
        <movie>
                <actor> Jean Reno </actor>
        </movie>
</actors>
```

사용자는 movie의 actor만을 검색하길 원할지라도 CXQuery는 theater의 actor 또한 주어진 조건을 만족하기 때문에 질의 처리기는 위의 두 가지 경우를 모두 찾아 준다. 이 경우 사용자는 본인이 원하는 결과 이외에도 불필요한 결과들을 얻게 된다.

```
for $c in doc()
return name
```

사용자가 title의 name을 알고 싶어 그림 3-2와 같은 DTD를 가진 문서에 대해 위와 같은 CXQuery 문으로 질의하였을 때 질의 처리기는 name에 해당하는 4개의 경로를 찾아내게 된다.

/movie/country/@name
/movie/title/@name
/movie/director/@name
/movie/actor/@name

사용자가 title의 name을 찾고 싶었다 할지라도 질의 처리기는 name이 포함된 모든 경로를 찾아낸다. CXQuery가 정확한 경로 대신 데이터 이름만으로 질의를 하기 때문에 한 데이터가 두 가지 이상의 경로를 가질 수 있는 다의성을 가지고 있다.

다의성은 말 그대로 '의미가 많다.'라고 해석할 수 있다. 이러한 다의성은 실제 생활에 있어서 사람들의 대화 속에서도 흔히 찾아 볼 수 있다. 예를 들어, '철수와 영희가 집에 간다.'라는 문장의 경우 '철수랑 영희가 각각 따로따로 집에 간다.'라는 의미와 '철수가 영희가 같이 집에 간다.'라는 두 가지로 생각할 수 있다.

이러한 다의성의 경우 대화하는 상대방조차 의미를 파악하기 어려운 경우가 많으며, 그 의미는 그 말을 한 당사자가 가장 정확히 알고 있다 하겠다. 따라서 다양한 해석을 통하여 생성된 XQuery문들을 사용자에게 반환하여 자신이 원하는 XQuery를 선택하도록 함으로써 CXQuery문의 다의성을 줄이도록 하였다.

4. 시스템 구현

본 논문의 CXQuery의 XQuery 변환기는 Microsoft Windows XP professional 환경을 기반으로 관계형 데이터베이스 Oracle10g와 네이티브 XML 데이터베이스인 eXist1.0[17]을 사용하였다. CXQuery를 XQuery로 변환하기 위한 부분은 JDK1.4를 사용하여 Java로 구현되었으며 데이터베이스와의 연결을 위해 Oracle JDBC Driver가 사용되었다.

4.1 실험 데이터

예제 데이터는 그림 4-1과 같이 다양한 구조를 가지고 있는 movie라는 문서들을 대상으로 하였다. 각

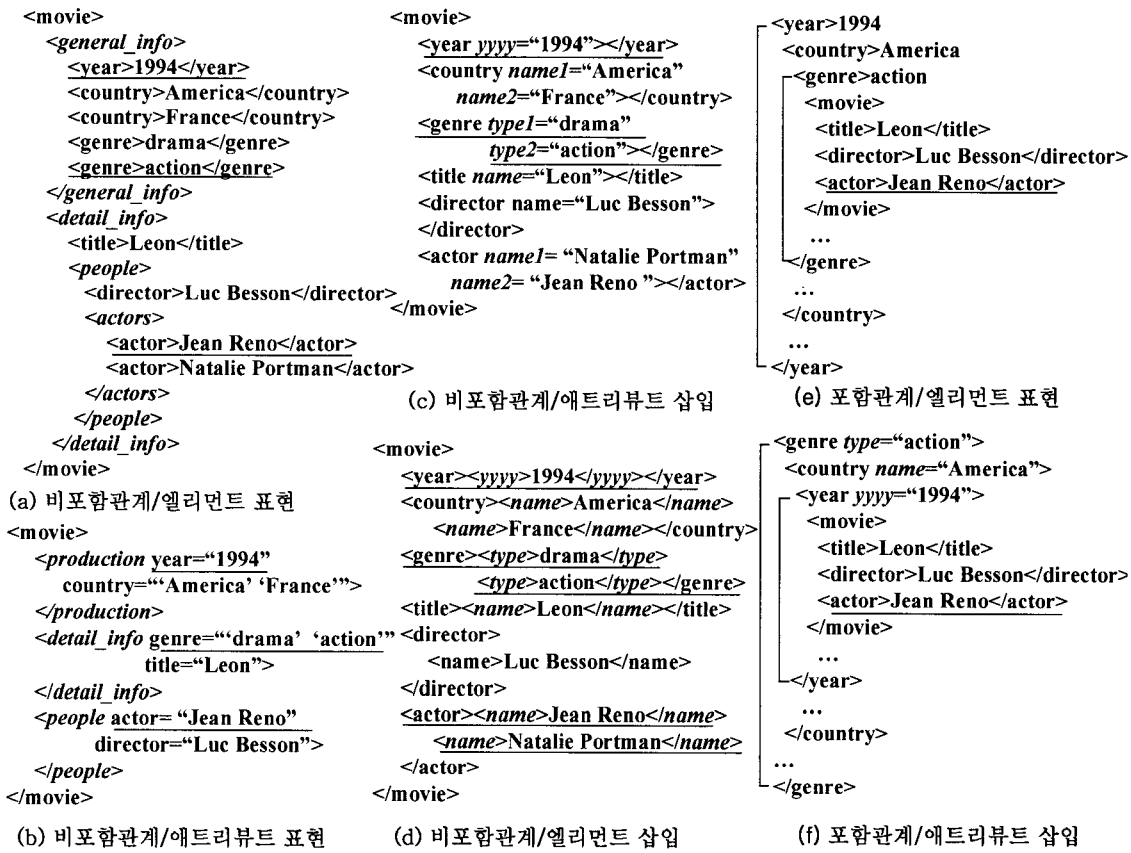


그림 4-1. movie 데이터의 다양한 문서 구조

DTD는 루트의 데이터 이름부터 서로 다른 다양한 문서 구조를 보이고 있음에도 그것이 표현하고 있는 정보는 {genre, year, title, actor, country, director}로 같은 정보임을 알 수 있다. 질의문 *E1*을 고려해 보자. 여기서 데이터 이름 year, actor, genre 사이의 관계가 포함 관계인지 아닌지, 또한 데이터가 엘리먼트인지 애트리뷰트인지에 따라 크게 그림 4-1과 같이 나눌 수가 있다. 우선 그림 4-1(e)에서 year, genre, actor는 엘리먼트로서 year안에 genre가, genre안에 actor가 포함된 모습을 갖는데 (a)는 데이터들 사이에 포함관계를 갖지 않는다. 또한 그림 4-1(f)에서 year, genre, actor는 포함관계로서 이들의 값은 애트리뷰트이고, (b)는 애트리뷰트 값을 갖지만 서로 포함관계는 아니다. 그림 4-1(c)에서 year, genre, actor는 서로 대등한 관계로서 애트리뷰트를 삽입하여 값을 할당하고 있지만 (d)는 엘리먼트를 삽입하여 값을 할당하고 있다. 이와 같이 문서의 구조가 포함이냐 비포함이냐 또는 엘리먼트냐 애트리뷰트냐에 따라 이러한 문서를 대상으로 하는 질의문이 달라진다. 즉, 문서 구조는 데이터 사이의 포함/비포함, 엘리먼트/애트리뷰트에 따라 나눌 수 있고, 질의문의 형태도 이에 따라 달라진다. 따라서 실험 데이터의 종류는 이러한 문서 구조를 기준으로 골고루 선택하였고 실험은 문서 구조를 모르는 경우와 문서 구조를 아는 경우로 나누어 처리하였다.

4.2 실험 처리 과정

사용자는 사용자 인터페이스를 통해 CXQuery INPUT의 for, where, return절을 차례로 입력한 후 한계 값을 직접 입력하여 distance를 조절할 수 있다. 입력이 끝나면 With DTD, No DTD, All XQuery 버튼 중 하나를 클릭하면 XQuery로 변환된 결과를 얻을 수 있다.

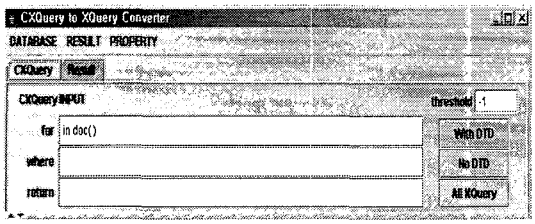


그림 4-2. CXQuery의 입력 형식

- With DTD: 문서 구조를 아는 경우의 XQuery를 생성한다. DTD파일이 저장된 디렉토리는 PROPERTY _> Set DTD Directory에서 저장할 수 있다.
- No DTD: 문서 구조를 모르는 경우의 XQuery를 생성한다.
- All XQuery: 위의 두 가지 경우의 XQuery를 모두 생성한다.

위의 변환기에 *E2*의 CXQuery 문을 입력한 후 문서 구조를 모르는 경우(No DTD 버튼)와 문서 구조를 아는 경우(With DTD)를 각각 실행시킨 결과는 그림 4-3와 그림 4-4와 같다. 즉, 그림 4-3의 경우는 문서에 DTD가 없기 때문에 그림처럼 질의문을 입력하면 문서 구조에 상관없이 검색할 수 있도록 3.1절과 같은 형태의 질의문으로 변환한다. 또한 그림 4-4와 같은 경우는 문서에 DTD가 있기 때문에 입력한 질의문은 3.2절에서 언급한 것처럼 각 문서들의 DTD를 분석하여 와일드카드(*)가 사용되지 않은 정확한 질의문으로 변환한다.

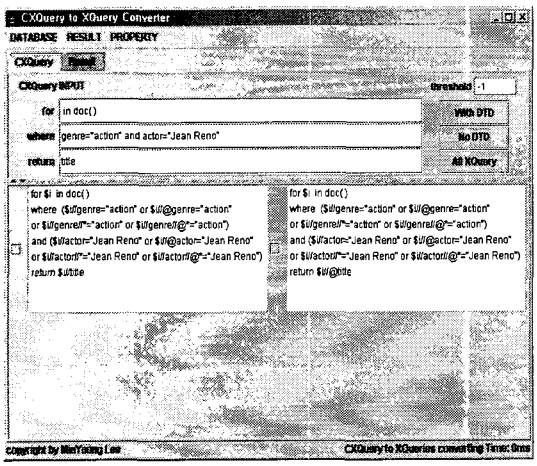


그림 4-3. 문서 구조를 모르는 경우

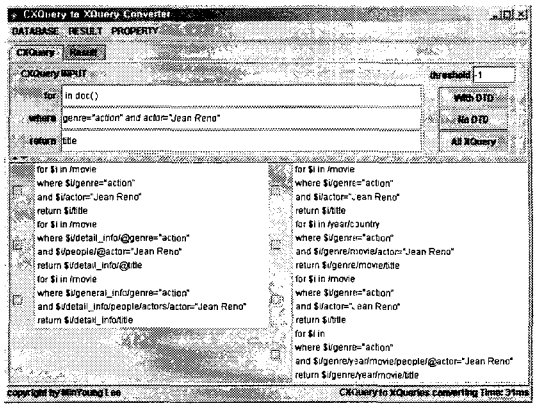


그림 4-4. 문서 구조를 아는 경우

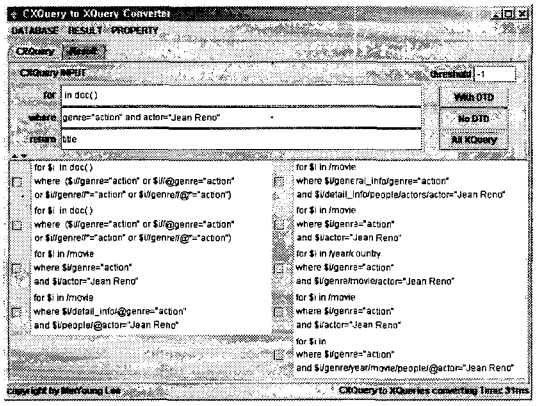


그림 4-5. 문서 구조 정보의 유무 모두 고려

또한 문서 구조 정보에 대한 유무를 모두 고려한 결과는 그림 4-5과 같은데 이는 그림 4-3에서 생성한 DTD가 없는 경우의 질의문과 그림 4-4에서 DTD가 있는 경우에 생성한 질의문을 모두 합쳐 놓은 것과 같다.

CX-Threshold을 지정하지 않은 경우 각 데이터는 최대한의 동일 패스를 찾아 XQuery를 생성한다. 문서 구조 (a)~(f)까지 7개에 대해 생성된 XQuery 질의문에 CX-Threshold을 지정하지 않았을 때의 결과는 표 4-1과 같다.

CX-Threshold을 다양하게 조절하여 XQuery를 생성한 결과는 표 4-2와 같은데 문서 구조는 그림 4-1 (a)~(f)를 대상으로 하였다. 즉, CX-Threshold이 1인 경우에는 공통 경로로 한 레벨 위인 /movies/movie가 공동 패스로, 2인 경우는 두 레벨 위인 /movies가 공동 패스로 포함되어야 한다. 또한 두 경우 모두 (e)~(f)처럼 데이터들 사이의 관계가

표 4-1. CX-Threshold 를 지정하지 않았을때 생성된 XQuery

생성된 XQuery	문서구조
for $i in /movie where $i/genre="action" and $i/actor="Jean Reno" return $i/title	(a)비포함관계/ 엘리먼트 표현
for $i in /movie where $i/detail_info/@genre="action" and $i/people/@actor="Jean Reno" return $i/detail_info/@title	(b)비포함관계/ 애트리뷰트 표현
for $i in /movie where $i/general_info/genre/@type="action" and $i/detail_info/people/actors/actor/@name="Jean Reno" return $i/detail_info/title	(c)비포함관계/ 애트리뷰트 삽입
for $i in /movie where $i/general_info/genre="action" and $i/detail_info/people/actors/actor="Jean Reno" return $i/detail_info/title	(d)비포함관계/ 엘리먼트 삽입
for $i in /year/country where $i/genre="action" and $i/genre/movie/actor="Jean Reno" return $i/genre/movie/title	(e)포함관계/ 엘리먼트 표현
for $i in doc() where $i/genre="action" and $i/genre/year/movie/people/@actor="Jean Reno" return $i/genre/year/movie/title	(f)포함관계/ 애트리뷰트 표현

표 4-2. CX-Threshold 값의 변화에 따른 XQuery

CX--Threshold	생성된 XQuery	문서구조
1	for $i in /movie/movies where $i/genre="action" and $i/actor="Jean Reno" return $i/title	(a)
	for $i in /movie/movies where $i/detail_info/@genre="action" and $i/people/@actor="Jean Reno" return $i/detail_info/@title	(b)
	for $i in /movie/movies where $i/general_info/genre/@type="action" and $i/detail_info/people/actors/actor/@name="Jean Reno" return $i/detail_info/title	(c)
	for $i in /movie/movies where $i/general_info/genre="action" and $i/detail_info/p eople/actors/actor="Jean Reno" return $i/detail_info/title	(d)
2	for $i in /movie where $i/genre="action" and $i/actor="Jean Reno" return $i/title	(a)
	for $i in /movie where $i/detail_info/@genre="action" and $i/people/@actor="Jean Reno" return $i/detail_info/@title	(b)
	for $i in /movie where $i/general_info/genre/@type="action" and $i/detail_info/people/actors/actor/@name="Jean Reno" return $i/detail_info/title	(c)
	for $i in /movie where $i/general_info/genre="action" and $i/detail_info/p eople/actors/actor="Jean Reno" return $i/detail_info/title	(d)

포함인 경우는 movie나 movies 엘리먼트가 상위 레벨의 엘리먼트가 아니고 이들도 genre나 actor의 엘리먼트 사이에 포함되는 구조를 갖기 때문에 이들이 공통패스로 사용되지 않는다.

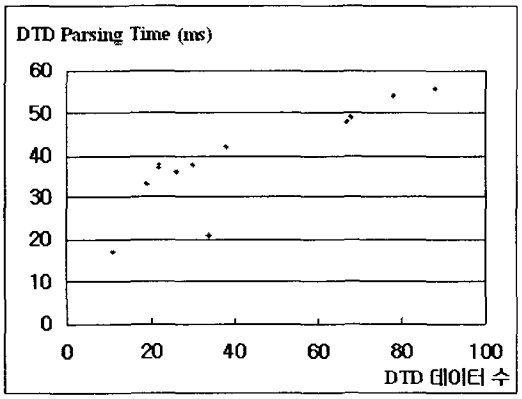
4.3 실험 결과

DTD를 파싱하여 문서 구조 정보 테이블을 만드는데 걸린 시간은 그림 4-6과 같다. DTD 파일의 구조상 같은 데이터 이름이 중복하여 나오는 경우 그 경로 또한 여러 가지가 생성될 수 있다. (a)는 DTD 파일의 데이터 수를 기준으로 한 실행 시간 이며 (b)는 만들어진 문서 구조 정보 테이블의 가능한 총 경로 수를 기준으로 한 실행 시간이다. DTD 파싱이 파일로부터 한 데이터씩 읽어 이미 만들어진 문서 구조로부터 부모 데이터의 경로에 자신을 추가하는 구조로 이루어져 있기 때문에 문서의 총 데이터 수에 실행 시간이 의존함을 알 수 있다.

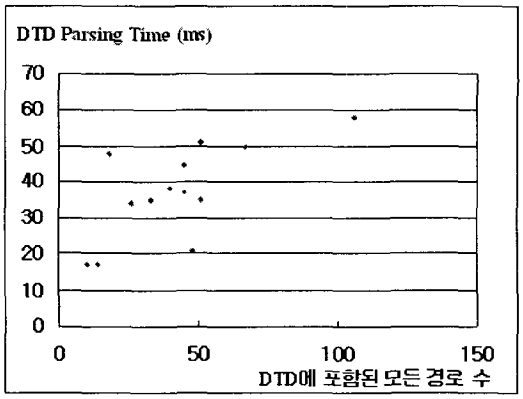
DTD 문서의 길이가 100줄일 때 실행 시간이 60ms 보다 작으므로 파싱 시간은 매우 짧음을 알 수 있다. 또한 한번 파싱한 문서 구조는 계속 사용 가능하다는 점에서 DTD 를 파싱하는 시간은 무시할 만큼 작다고 볼 수 있다.

또한 그림 4-7에서 보는 것처럼 CXQuery를 XQuery로 변환하는데 걸리는 시간도 무시할 정도로 적었으며 문서 구조에 따라 만들어질 수 있는 모든 경로를 따져 XQuery가 변환되어 사용자가 원하는 질의를 선택하도록 함으로써, 사용자가 원하는 질의 결과를 모두 구할 수 있었다.

생성된 XQuery는 사용자가 DTD 문서 구조에 대한 지식을 가지고 만드는 것보다 더 많은 질의를 생성함으로써 사용자가 원하는 질의를 선택만 하면 되

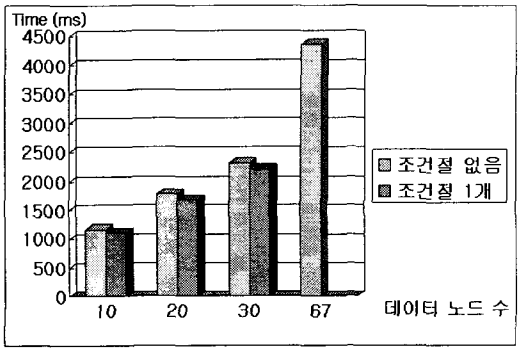


(a) 데이터 수에 따른 DTD Parsing 실행 시간

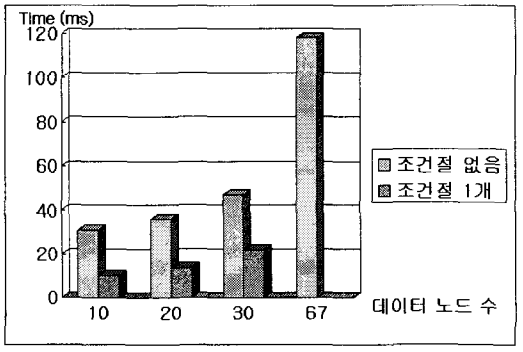


(b) DTD의 가능한 모든 경로 수와 DTD Parsing실행 시간

그림 4-6. 문서 구조 테이블 생성 시간



조건절 개수에 따른 실행시간



조건절 개수에 따른 생성된 쿼리 수

그림 4-7. 변환 시간과 생성된 XQuery

도록 하였다. 이는 사용자가 모든 문서 구조를 알지 않아도 그 데이터 이름과 관련된 모든 질의를 반환하여 주기 때문에 사용자가 정보를 얻고자 하는 DTD로부터 모든 가능한 질의를 빠짐없이 반환해 준다.

5. 결 론

XML은 웹 상에서 구현된 문서를 전송 가능하도록 설계된 표준화된 텍스트 형식의 마크업 언어로써 인터넷에서 바로 사용 가능한 문서를 표현하는 표준이다. 인터넷을 통해 주고 받는 방대한 XML 데이터를 효율적으로 저장, 질의, 검색, 조작하는 등의 "XML 데이터 관리"의 문제를 해결하기 위해 XML 문서를 RDB로 변환하는 시간과, XML을 RDB로 매핑하면서 발생하는 불일치 문제로 인해, XML 문서 자체를 저장하는 XML DB 개념이 탄생하게 되었다.

본 논문에서는 XML DB의 확산에 따른 XML 문서에 효과적으로 질의, 검색이 중요한 이슈로 떠오름에 따라 사용자가 문서 구조에 대한 지식 없이도 질의가 가능한 CXQuery가 가능하도록 시스템을 구현하였다. CXQuery의 XQuery 변환기를 통한 질의는 사용자가 가지고 있는 모든 문서 구조를 자동으로 파싱하여 그 정보를 가지고 사용자가 입력한 데이터 이름과 값에 대한 XQuery 질의를 생성한다. 또한 실험을 통해, CXQuery가 XQuery로 변환되는 시간이 매우 작으므로 사용자가 각 문서 구조를 분석하여 질의를 하는 것보다 편리성을 제공한다는 점에서 의의가 있다 하겠다. 우리는 CXQuery가 비슷한 정보를 다루는 분산 데이터 베이스 환경에 적합하게 설계되었기 때문에 CXQuery의 XQuery 변환을 하나의 노드로 놓고 분산 환경하에서의 CXQuery 질의를 통해 분산된 데이터에 CXQuery를 다른 노드에게 보내 다른 노드로부터 만들어진 XQuery를 취하는 시스템을 향후 과제로 제안한다.

참 고 문 헌

[1] W3C Consortium, "XML1.0 (Second Edition), W3C Recommendation 06 Oct. 2000," W3C Recommendation 16 August 2006, available at http://www.w3.org/TR/REC-xml.

[2] S. Adler, A. Berglund, J. Caruso, S. Deach, T. Graham, P. Grosso, E. Gutentag, A. Milowski, S. Parnell, J. Richman, and S. Zilles, "Extensible Stylesheet Language (XSL) Version 1.0," W3C Proposed Recommendation Aug. 2001, available at http://www.w3.org/TR/xsl/.

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener, "The Lorel Query Language for Semistructured Data," *International Journal on Digital Libraries*, Apr. 1997.

[4] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, and J. Siméon, "XQuery 1.0: An XML Query Language," W3C Working Draft 16 Aug. 2002, available at http://www.w3.org/TR/xquery/.

[5] W3C Consortium, "XML Path Language (XPath) Version 1.0," W3C Recommendation 16 Nov. 1999, available at http://www.w3.org/TR/xpath.html.

[6] 이월영, "XML 데이터베이스에서 문서 구조 독립적인 질의 처리 기법," 이화여대 과학기술대학원 박사학위논문, 2005. 2.

[7] IBM Corporation, "DB2 XML Extender," IBM Corporation, 2000, available at http://www.4.ibm.com.

[8] S. Banerjee, "Oracle XML DB," Oracle Corporation Technical White Paper Release 9.2, Jan. 2002.

[9] S. Howlett and D. Jennings, "SQL Server 2000 and XML: Developing XML-Enabled Data Solutions for the Web," MSDN magazine, Jan. 2002, available at http://msdn.microsoft.com/library/ default.asp?url= /msdnmag/issues/0800/sql2000/toc.asp.

[10] George Feinberg, "Native XML Database Storage and Retrieval," *Journal of Linux*, 2005.

[11] Matthias Nicola, and Jasmi John, "XML Parsing: a Threat to Database Performance," *Proc. Of the12th Information and Knowledge Management archive*, 2003.

[12] T. Fiebig, S. Helmer, C.-C. Kanne, G. Moerkotte, J. Neumann, R. Schiele, and T.

Westmann, "Anatomy of a Native XML base Management System," *Journal of VLDB*, 2002

[13] Tamino, http://www.tamino.co.kr.

[14] X-Hive Corporation, "X-Hive/DB 6.0," X-Hive Corporation, 2001, available at http://www.x-hive.com/.

[15] eXcelon Corporation, "eXtensible Information Server(XIS)," eXcelon Corporation, 2007, available at http://www.xml.com/pub/p/381.

[16] Ipedo Corporation, "Product Overview: Ipedo XIP - The Only EII Platform That Uses Both Relational And XML Data Models," Ipedo Corporation, 2007, available at http://www.ipedo.com/.

[17] eXist Corporation, "Open Source Native XML Database," eXist Corporation, 2007, http://www.exist-db.org.

[18] W. Kim, W. Lee, and H. Yong, "On Supporting Structure-Agnostic Queries for XML," *Journal of object technology*, Vol. 3, No. 7, pp. 27-35, July-August 2004.

[19] W. Kim, W. Lee, and H. Yong, "On Query-Processing Issues for Non-Navigational Queries for XML," *Journal of object technology*, Vol. 3, No. 10, pp. 19-26, November-December 2004.

이 민 영

2004년 2월 이화여자대학교 컴퓨터학과 학사

2006년 2월이화여자대학교 컴퓨터학과 공학석사

현재 삼성전자

이 월 영

1988년 2월 이화여자대학교 전자계산학과 학사

2000년 2월 이화여자대학교 과학기술대학원 컴퓨터학과 공학석사

2005년 2월 이화여자대학교 과학기술대학원 컴퓨터학과 공학박사

2005년 9월~현재 서울기독대학교 국제경영정보학과 교수

관심분야: : XML 데이타베이스, 정보검색, 데이타 마이닝

용 환 승

1983년 2월 서울대학교 컴퓨터공학과 학사

1985년 2월 서울대학교 컴퓨터공학과 공학석사

1994년 2월 서울대학교 컴퓨터공학과 공학박사

1995년~현재 이화여자대학교 컴퓨터학과 교수

관심분야 : 객체관계 데이타베이스, 데이타 마이닝, 유비쿼터스 데이타베이스