효율적 구조 질의를 지원하는 바다-IV/XML 질의처리기의 설계 및 구현

이 명 철* · 김 상 균* · 손 덕 주** · 김 명 준** · 이 규 철*

Design and Implementation of BADA-IV/XML Query Processor Supporting Efficient Structure Querying

Myungcheol Lee* · Sang-Kyun Kim* · Duk-Joo Son** · Myung-Joon Kim** · Kyu-Chul Lee*

Abstract

As XML is emerging as the Internet electronic document language standard of the next generation, the number of XML documents which contain vast amount of information is increasing substantially through the transformation of existing documents to XML documents or the appearance of new XML documents. Consequently, XML document retrieval system becomes extremely essential for searching through a large quantity of XML documents that are stored in and managed by DBMS. In this paper we describe the design and implementation of BADA-IV/XML query processor that supports content-based, structure-based and attribute-based retrieval. We design XML query language based upon XQL (XML Query Language) of W3C and tightly-coupled with OQL (a query language for object-oriented database). XML document is stored and maintained in BADA-IV, which is an object-oriented database management system developed by ETRI (Electronics and Telecommunications Research Institute). The storage data model is based on DOM (Document Object Model), therefore the retrieval of XML documents is executed basically using DOM tree traversal. We improve the search performance using Node ID which represents node's hierarchy information in an XML document. Assuming that DOM tree is a complete k-ary tree, we show that Node ID technique is superior to DOM tree traversal from the viewpoint of node fetch counts.

※ 본 논문은 한국전자통신연구원의 "XML 멀티미디어 문서 모델링 도구 개발 (위탁계약번호 : 00-085)" 과제의 일부로 수행된 결과임.
* 충남대학교 컴퓨터공학과
** 한국전자통신연구원 컴퓨터 · 소프트웨어기술연구소

1. 서 론

XML[1]은 사용하는 플랫폼에 독립적이며, 문서 정보의 전송과 교환이 쉽고, 문서의 풍부한 의미를 그대로 나타낼 수 있다는 장점을 지니고 있다. XML이 1998년에 W3C(World Wide Web Consortium)에서 차세대 인터넷 표준으로 채택된 이후, XML 관련 응용이나 도구, 구문 분석기와 XML을 지원하기 위한 다른 표준들이 속속 나오고 있다. 또한, XML은 점차 기존의 웹 문서 포맷인 HTML[2]을 대체하고 있을 뿐만 아니라, 워드프로세서 문서, 전자상거래 문서, 전자교범 문서 등도 XML 문서로 만들어 지고 있다.

이러한 추세라면 향후 XML로 만들어지는 문서의 양은 기하급수적으로 많아질 것이다. 따라서, 이런 대량의 문서를 저장하고 관리하는 시스템은 필수적인 요소가 될 것이고, 대량의 저장된 문서에 대해서 원하는 문서를 찾는 검색 기능은 필수적이 될 것이다.

XML 문서는 기존의 문서와는 달리 문서의 의미적인 구조를 엘리먼트와 애트리뷰트로 나타내고 엘리먼트 안에 원하는 문서의 내용을 표현한다. 예를 들어, 책의 서론, 본론, 결론, 제목, 장, 절, 구 등을 엘리먼트로 나타내어 책의 구성이나 구조가 어떻게 되어 있는 지를 한 눈에 알 수 있으며, 서론에 대한 내용은 서론 엘리먼트에 기술되어 있으므로 서론 엘리먼트에서 찾을 수 있다. 이런 특성을 가진 XML 문서에 대한 검색을 위해서는 기존의 검색기에서 제공하던 문서의 내용에 대한 검색 뿐만 아니라 문서의 논리적인 구조 정보와 속성 정보에 대한 검색도 할 수 있어야 한다.

본 논문에서는 객체 지향 데이터베이스 관리 시스템(OODBMS)인 바다-IV[3]에 저장한 대량의 XML 문서를 효율적으로 검색하기 위한 연구로서 XML 문서에 대한 질의어의 설계와 질의처리기의 설계 및 구현에 대한 내용을 기술하였다.

본 논문에서 구현한 XML 질의처리기는 XML 문서의 구조적인 특성이 잘 유지될 수 있도록 OODB 내에 W3C의 표준인 DOM(Document Object Model)[4]을 기반으로 하는 저장 구조로 저장된 XML 문서에 대한 검색을 수행한다. XML 문서 검색을 위해서는 XML 문서가 지닌 내용 정보, 구조 정보, 속성 정보에 대해 처리가 가능하고, 사용자에게 쉽고 사용이 편한 질의 언어가 필요한데, 본 논문에서는 OODB의 OQL(Object Query Language)[5]과 XML 질의 언어인 XQL(XML Query Language)[6]을 결합하여 OODB 내에 저장된 XML 문서를 효율적으로 검색할 수 있는 질의어를 설계하였다. 또한, 데이터베이스에 대한 접근을 최소로 하며 XML 문서의 구조 정보 검색을 수행하기 위한 방법으로서 노드 ID 기법을 사용하였다.

논문의 구성은 다음과 같다. 먼저 2장에서 XML 질의 언어의 요구 사항과 XML 관련 질의 언어를 살펴 본다. 3장에서는 바다-IV/XML 질의 언어의 설계 내용을 기술한다. 4장에서는 바다-IV/XML 질의처리기의 설계 내용을 기술하고, 5장에서는 구현 내용을 기술하며, 6장에서는 성능 분석에 대해서 기술한다. 마지막으로 7장에서는 본 논문의 결론을 기술한다.

2. XML 질의 언어 분석

2.1 XML 질의언어 요구사항

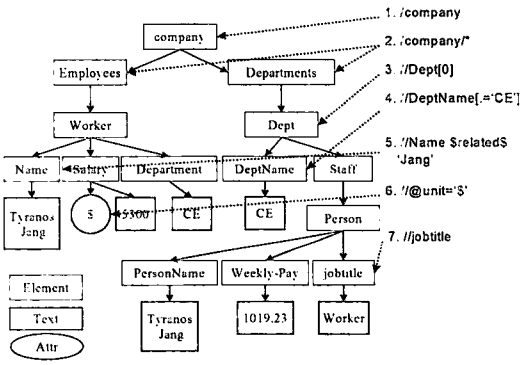
XML 질의 언어는 XML 문서의 모든 요소에 대한 질의가 가능해야 하며, DOM이나 다른 XML 응용에 대해서도 쉽게 사용이 가능해야 한다. XML 질의 언어가 지원해야 하는 기능에

대한 요구 사항[7]은 다음과 같다.

- 질의 언어 문법 : 사람이 읽고 쓸 수 있는 문법을 가진 문자열로 표현되어야 하며, 절차적이기 보다는 선언적이어야 하고, 문서의 모든 요소를 지정 가능해야 하며, 사용이 쉽고 구현 가능해야 한다.
- 검색 범위 : 여러 DTD, 하나의 DTD, 여러 문서, 하나의 문서, 한 문서 안의 노드(엘리먼트, 애트리뷰트 등), 이전 질의의 결과 집합에 대한 검색을 할 수 있어야 한다.
- 검색 결과 : 검색의 결과는 XML 문서들의 리스트, 한 문서 안의 노드들의 리스트일 수 있고, 결과 내에서의 순서를 지원해야 한다.
- 구조적 관계 : XML 문서 내의 각 엘리먼트들의 부모/자식 관계, 조상/후손 관계, 형제 관계, 절대적, 상대적, 범위상의 위치를 지정할 수 있어야 한다.
- 내용에 대한 조건 : "같다, 같지 않다, 크다, 크거나 같다, 작다, 작거나 같다"와 같은 기본 비교 연산은 물론이고 패턴 매칭을 위한 LIKE 연산(%, _), 정보 검색을 위한 불리언 연산(AND, OR, NOT)을 지원해야 한다.

2.2 XQL(XML Query Language)

XQL[6]은 현재 W3C QL '98[8]에 제출된 XML 질의 언어들 중에서 사실상(de facto) 표준으로 받아 들여지고 있다. XQL은 XSL(Extensible Stylesheet Language)[9]의 패턴 언어를 확장한 형태이다. XSL은 XML 문서에 대한 스타일시트 언어인데, 유닉스에서의 디렉토리 표기와 유사한 구문을 사용하는 XSL 패턴 언어를 통해 특정 노드를 찾아서, 스타일시트를 적용한다. XQL은 XSL의 노드에 대한 처리에 불리언 논리, 필터, 노드의 집합에 대한 색인을 추가했고, 질의와 패턴을 사용하는 데에 단순한 구문을 사용하여 간결하다는 것이 특징이다. (그림 1)은 company라는 XML 문서에 대해 XQL로 질의를 하는 예이다.



(그림 1) XQL 질의 예

(그림 1)의 XQL 질의 예에 대해 간단히 설명하면 다음과 같다.

- /company : 루트 엘리먼트인 company 검색
- /company/* : company의 모든 자식 엘리먼트 검색
- //Dept[0] : 모든 Dept 엘리먼트 중 0번째 Dept 검색, XQL에서의 index는 0에서 시작하며 맨 마지막 요소는 1로 지정한다.
- //DeptName[.='CE'] : 모든 DeptName 엘리먼트 중에서 값이 CE인 DeptName 검색
- //Name $related$ 'Jang' : 모든 Name 엘리먼트 중 값이 Jang을 포함하는 Name 검색, $related$ 연산자는 본 논문에서 확장한 것이다.
- //@unit = '$' : 모든 unit 애트리뷰트 중 값이 $인 unit 검색
- //jobtitle : 모든 jobtitle 엘리먼트 검색

2.3 XQL 기능 분석

2.2절의 XQL에 대한 설명과 예에서 알 수 있는 것처럼, XQL은 간단하고 선언적이며, 사용자

에게 친숙한 장점이 있는 반면에, 조인이나 변환과 통합 등의 데이터베이스 기능이 부족하다는 단점이 있다. 2.1절의 XML 질의언어 요구사항을 기준으로 XQL의 기능을 분석해 보면 <표 1>과 같다.

〈표 1〉 XQL 기능 분석

특성 \ 언어	XQL
간단, 사용하기 쉬움	○
선언적	○
조인, 집단 연산	×
데이터 통합/변환	×
내용 검색	○
구조 검색	○
속성 검색	○
여러 문서 검색	×
전문 검색, LIKE	×

3. 바다-IV/XML 질의 언어의 설계

3.1 XQL의 확장

본 논문에서는 <표 1>에서 XQL의 부족한 기능 중 전문검색, LIKE를 지원하기 위해 XQL에 $like$와 $related$ 연산자를 확장했고 각각 다음과 같이 사용이 된다.

- //TITLE $like$ "%Hamlet%" : TITLE 엘리먼트에 Hamlet을 포함하는 TITLE 검색
- //TITLE $like$ "Caesar%" : TITLE 엘리먼트가 Caesar로 시작하는 TITLE 검색
- //TITLE $related$ "Romeo&Juliet" : TITLE 엘리먼트가 Romeo와 Juliet을 포함하는 TITLE 검색
- //TITLE $related$ "Hamlet|Romeo" : TITLE 엘리먼트가 Hamlet 또는 Romeo를 포함하는 TITLE 검색

본 논문에서의 $like$와 $related$ 연산자의 확장을 통해서 XML 문서에 대한 전문 검색 및 문자열 패턴 비교 연산이 가능하다.

또한, XQL이 하나의 문서 내에서만 검색이 가능한 단점을 개선하여, 여러 문서에서의 검색이 가능하도록 확장하였는데, 그 내용은 3.2절에서 자세히 설명하도록 한다.

3.2 OXQL의 설계

<표 1>에서 알 수 있듯이 XQL은 조인, 집단 연산, 데이터 통합 등 데이터베이스 기능을 제외하고는 많은 장점을 가진다. XQL에 데이터베이스 기능을 보완하기 위한 방법으로 객체 지향 데이터베이스의 OQL을 이용한다면 XQL에서 지원하기 힘든 데이터베이스 기능 연산들의 지원이 가능하다.

또한 XQL을 OQL과 결합하면 OQL을 사용하는 객체지향 데이터베이스 응용에서 XML을 사용할 수 있는 장점을 얻을 수 있다. 본 논문에서는 OQL과 XQL을 긴밀하게 결합시켜 질의 언어를 설계하였다. 즉, (그림 2)와 같이 OQL 구문의 WHERE 절에 <xql_predicate>를 추가하여 XQL 질의를 입력하는 것이다. 이렇게 OQL과 XQL이 긴밀하게 결합된 형태를 본 논문에서는 OXQL이라고 부른다.

```
OXQL : =
    'select' <select_list>
    'from' <class_name> [inherited_class_flag>]
    ['where' <search_condition>]
    ['order by' <sort_specification_list>]
search_condition : =
   <xql_predicate> | <comparison_predicate> |
   <in_predicate> | <like_predicate>
xql_predicate : =
   <value_expression>  'contains'  '('  <Bada-
IV/XQLQuery> ')'
```

(그림 2) OQL과 XQL을 결합한 OXQL의 구문 정의

OXQL에서 SELECT 절과 FROM 절은 OQL

의 문법과 같고 WHERE 절에 <xql_predicate>이 추가되었다. <xql_predicate>의 검색 대상은 객체나 객체의 애트리뷰트가 되고, 이 검색 대상에 'contains'를 적용하여 XQL 질의를 하게 된다.

(그림 2)에서 <Bada-IV/XQLQuery>는 XQL에 엘리먼트의 내용이나 애트리뷰트의 내용이 문자열인 경우, 와일드 문자를 포함한 패턴에 대한 비교 연산을 위한 LIKE 연산과 지정된 엘리먼트의 전문 내용에 대한 유사도 검색을 위한 RELATED 연산을 지원하도록 확장한 것이다. <Bada-IV/XQLQuery>의 자세한 BNF 표기는 XQL[6]에서의 BNF와 같으며, 단지 LIKE 연산과 RELATED 연산을 지원하기 위하여 $like$, $related$ 연산자에 관련한 부분만이 추가되었다.

OXQL은 여러 문서 검색 지원과 관련해서 다음과 같이 사용이 될 수 있다.

- 모든 DTD의 모든 문서 검색 : 데이터베이스에 저장된 모든 XML 문서 중에서 title 엘리먼트를 가진 XML 문서 검색

```
Select      *
From        PDocument*
Where       this contains ('//title')
```

- 특정 DTD의 모든 문서에 대한 검색 : 데이터베이스에 저장된 모든 Book DTD의 문서 중에서 title 엘리먼트를 가진 XML 문서 검색

```
Select      *
From        Book_Document
Where       this contains ('//title')
```

- 특정 DTD의 모든 엘리먼트에 대한 검색 : 데이터베이스에 저장된 모든 Book DTD의 엘리먼트 중에서 title 엘리먼트 검색

```
Select      *
From        Book_Element
Where       this contains ('//title')
```

OXQL을 사용한 질의의 예로는 다음과 같은 것이 있을 수 있다.

```
논문 DTD의 모든 문서에서 제목에 'XQL'이란 단어가 있는 모든 문서를 찾아라.
```

```
SELECT *
FROM 논문_Document
WHERE this CONTAINS ('제목 RELATED "XQL" ')
```

(그림 3)은 "제목이 'xql'인 장을 찾는" OXQL을 C++ 언어와 바인딩하여 수행하는 응용의 한 부분이다. Book DTD의 모든 엘리먼트 중에서, Title이 "xql"인 Chapter 엘리먼트를 찾아서, 노드 이름을 출력한다. 사용자는 검색 결과에 대해서 DOM의 인터페이스를 사용하여 접근할 수 있는데, (그림 3)에서는 Element의 getNode Name() 함수를 사용하였다.

```
  OM_LIST<Book_Element> result_eles;
  int num, i;
  if (query("SELECT * FROM Book_Element WHERE
this CONTAINS ('//Chapter[Title = "xql"]')",
result_eles) == OK_ERROR) {
     printf("Error in XQL Statement!\n");
     return OK_ERROR;
}
num = result_eles.cardinality();
for (i = 0; i < num; i++) {
 printf("%번째 결과 : %s\n", i,
   result_eles[i].getNodeName());
}
```

(그림 3) OXQL의 C++ 언어 바인딩 예

3.3 다른 시스템과의 비교

현재 질의언어로 XQL을 사용하는 대표적인 XML 문서 저장 검색 시스템으로는 eXcelon Corp.의 eXcelon과 Software AG의 Tamino가 있다.

2.1절의 XML 질의언어 요구사항을 기준으로 바다-IV/XML 질의언어와 위 두 시스템이 지

원하는 질의언어를 비교해 보면 다음과 같은 차이를 보인다.

3.3.1 eXcelon

eXcelon[10]은 eXcelon Corp.에서 개발한 XML 문서 저장 검색 시스템이다. OODBMS인 ObjectStore[11]를 하부 저장시스템으로 사용하며 질의언어로는 XQL을 사용한다. eXcelon은 XQL의 기본 사양은 물론 확장 기능까지 거의 대부분 구현한 시스템으로서 빠른 저장 및 검색을 지원하는 것이 장점이며 대표적인 XML 문서 관리 시스템으로서 주목을 받고 있다.

그러나 2.1절의 XML 질의언어 요구사항 중에서 eXcelon은 다음과 같은 기능을 제대로 제공하지 못한다.

- 여러문서 검색 : eXcelon은 하나의 문서에 대한 검색만을 지원하므로, 여러 문서 검색을 위해서는 하나의 문서에 대한 검색을 반복해야 한다.
- 전문검색, LIKE 검색 : eXcelon은 XQL의 기본 문자열 비교 연산(Exact Matching)만을 지원한다.

이에 비하여 본 논문에서 설계한 바다-IV/XML 질의언어는 XQL의 부족한 기능을 XQL 확장과 OQL과의 연동을 통하여 2.1절의 XML 질의언어 요구사항을 모두 만족시킬 수 있다는 점에서 비교 우위에 있다.

3.3.2 Tamino

Software AG[14] 사의 Tamino[15]는 임의의 형태의 데이터를 저장하고 검색하기 위하여 XML을 사용하는 인포메이션 서버이다. 즉, XML을 일종의 메타구조로 사용하여 각기 다른 저장 시스템의 데이터에서 추출한 정보들을 조합할 수 있도록 한다.

Tamino는 2.1절의 XML 질의언어 요구사항 중에서 여러문서를 지원할 수 없을 뿐만 아니라, XQL의 일부만을 구현하여 다음과 같은 XQL의 기능을 제공할 수 없는 것이 단점이라 할 수 있다.

- //로 시작하는 질의 : //title
- ieq, ine 등의 대소문자에 상관없이 비교하기 위한 연산자
- 질의 또는 서브질의 표현식을 부정하기 위한 불리언 연산자 not : //chap[not title]
- 서로 다른 지를 비교하는 연산자 ne : //chap[title ne 'Hamlet']
- XQL의 모든 함수 또는 메소드 : nodeType, nodeTypeString, nodeName, text, value, index, end, date, baseName, namespace, prefix, textNode, comment, pi, element, attribute, node, count, ancestor
- 중간 결과 또는 최종 결과들의 합집합, 교집합 : $union$, $intersect$
- 결과 셋에서 특정 위치의 결과들만을 추출하기 위한 Subscript 연산자 : //chap[0, 1, -1]

4. 바다-IV/XML 질의처리기의 설계

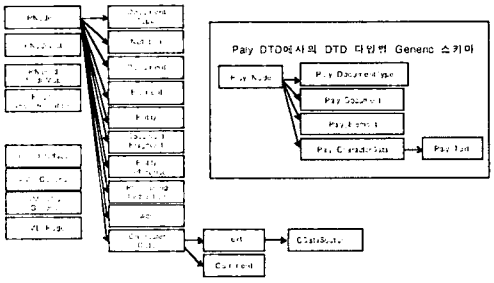
4.1 바다-IV/XML 질의처리기의 특징

본 논문에서 구현한 XML 질의처리기는 다음과 같은 세가지 특징을 가지도록 설계하였다

4.1.1 웹 표준인 DOM의 저장 구조를 이용한 검색

DOM(Document Object Model)은 프로그램 또는 스크립트에서 문서의 내용, 구조, 스타일을 동적으로 접근하고 변경할 수 있도록 W3C에서 표준으로 만든 플랫폼과 언어에 중립적인 인터페이스이다. 본 논문에서 구현한 XML 질의처리기는 DOM을 기반으로 하는 저장 구조를 따르는 시스템에 저장이 된 XML 문서를 대

상으로 질의를 처리한다. 바다-IV/XML에서는 XML 문서의 구조 정보와 내용 정보를 잘 유지하기 위해서 아래 (그림 4)와 같이 XML 문서를 DOM을 기반으로 하는 스키마를 사용해서 저장하고 관리한다. 그림에서 왼쪽의 클래스들은 DTD에 상관없이 공통된 정보들을 저장하기 위한 Generic 클래스들이고, 오른쪽의 사각형 안의 클래스들은 Play DTD의 객체들을 저장하기 위한 DTD 타입별 Generic 클래스들이다. 이렇게 각 DTD마다 다른 클래스에 XML 문서를 저장하게 되면, 전체 객체를 탐색할 필요가 없어서 검색 성능이 향상되고, DTD 단위의 질의가 가능하다는 것이 장점이 된다.



(그림 4) 바다-IV/XML 저장 스키마

4.1.2 XML의 구조 정보와 내용 정보에 대한 검색 가능

구조검색은 다음에 설명할 노드 ID를 기반으로 수행하고, 내용검색은 바다-IV의 저장시스템인 MIDAS[12, 13]의 정보검색엔진을 사용해서 수행한다.

바다-IV에서는 CLOB 형의 객체에 대해서만 내용 검색을 지원하므로, 바다-IV/XML에서는 다음과 같이 NodeValue와 같은 값을 가지는 CLOB 형의 content를 추가하고 내용 검색 인덱스를 생성하여 내용 검색을 수행한다.

```
class PNode : public bada4_defined_class {
    char * NodeValue;
    OM_CLOB content;
}
```

4.1.3 XML 검색기의 구조 검색 성능 향상을 위한 노드 ID 기법의 사용

자세한 내용은 4.2절에서 설명하기로 한다.

4.2 노드 ID를 이용한 구조검색

구조검색을 위한 노드(Node) ID는 타입(Type) ID를 이용해 생성한다.

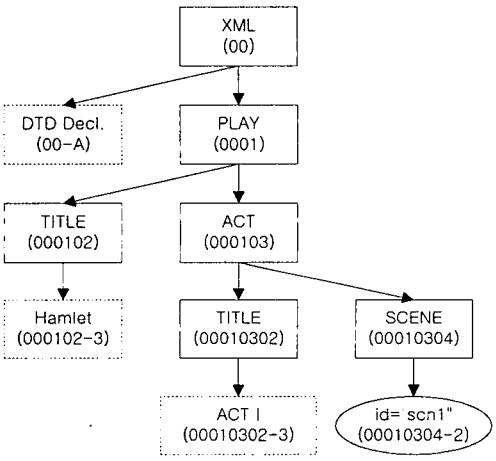
타입 ID는 XML 문서 내의 엘리먼트에 고유하게 부여하는 ID로서 DTD 내에 정의된 순서대로 부여하며 DTD 내의 각 엘리먼트를 유일하게 구분 가능하도록 해준다. 본 논문에서는 '0' → '9', 'A' → 'Z', 'a' → 'z'의 62 문자를 2자리 디지트로 조합하여 위 순서대로 부여한다. 따라서 하나의 DTD에서 최대 3844개의 엘리먼트에 타입 ID를 부여할 수 있어 거의 모든 DTD를 처리할 수 있다. 자식 노드를 가질 수 없는 엘리먼트 이외의 노드에 대해서는 일반 엘리먼트와 차별을 두기 위해 DOM의 양식을 따라 각각 -2에서 -A까지의 타입 ID를 부여한다. <표 2>는 PLAY라는 DTD에 타입 ID를 부여한 예이다.

<표 2> 타입 ID 부여 예

TypeName	TipeID
XML	00
PLAY	01
TITLE	02
ACT	03
ATTRIBUTE	-2
TEXT	-3
CDATA_SECTION	-4
ENTITY-REFERENCE	-5
PROCESSING_INSTRUCTION	-7
COMMENT	-8
DOCUMENT_TYPE	-A

노드 ID는 엘리먼트에 따라 정해진 타입의 식별자인 타입 ID를 엘리먼트에 부여하고, 부모

의 노드 ID에 자신의 타입 ID를 붙여서 자신의 노드 ID를 만든다. 부모 노드가 없는 루트 엘리먼트의 경우는 자신의 타입 ID를 노드 ID로 사용한다. 예를 들면 (그림 5)에서 루트 엘리먼트인 <PLAY>는 노드 ID가 "0001"이고, <PLAY>의 자식 노드인 <ACT>는 부모 노드의 ID인 "0001"에 자신의 타입 ID인 "03"을 붙여서 "000103"를 만들어 자신의 노드 ID가 된다. 여기서 노드 ID가 "00"인 XML은 문서 상에 존재하지는 않으나 XML 문서를 DOM 트리로 표현했을 때의 루트 노드를 나타낸다.



(그림 5) 노드 ID 부여 예

XQL 질의문에서의 구조 정보는 XQL 구문 분석기가 만들어 내는 XQL 파스 트리에 포함이 된다. 따라서, XQL 파스 트리가 가진 정보들을 통해 각 엘리먼트의 타입 ID를 알아 내고, 각 타입 ID를 연결하여 원하는 엘리먼트의 노드 ID를 조합해 낼 수 있다. 이때, XQL 표현식에서 노드 ID를 조합하는 일반적인 규칙은 다음 <표 3>과 같다.

예를 들어, "/PLAY//SCENE"의 경우 "0001%04"와 같이 노드 ID를 만들어 낼 수 있고, 데이터베이스에 저장된 PLAY DTD의 객체들에 대해서 노드 ID가 "0001%04"인 객체를 검색하면 원하는 결과를 바로 얻을 수 있다.

<표 3> XQL2NodeID 변환 규칙

XQL 표현	NodeID 문자열
/	""
//	"%"
*	"__"
.	""
Element's Name	Element's TypeID

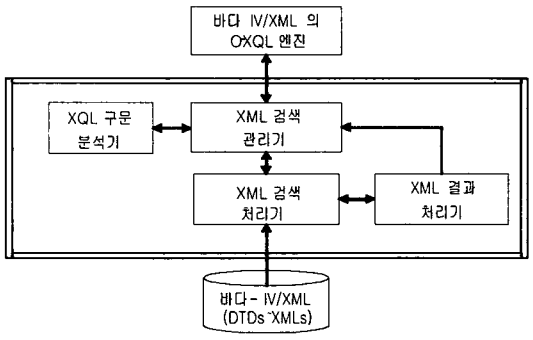
노드 ID를 이용한 검색을 하지 않을 경우, DOM 트리를 루트 노드에서부터 차례로 순회하면서 원하는 객체를 찾아 내야 한다.

따라서, DOM 트리의 자식 노드의 개수와 트리의 깊이에 따라 수배에서 수십배의 검색 속도 차이가 날 수 있다. 위 두 방식에 대한 성능 비교 분석은 본 논문 6.1절에서 자세히 기술한다.

5. XML 질의처리기의 구현

5.1 바다-IV/XML 질의처리기의 전체 구조

바다-IV/XML 질의처리기는 크게 XML 검색 관리기, XML 검색 처리기, XML 결과 처리기, XQL 구문 분석기의 네 부분으로 구성되며, 구조는 (그림 6)과 같다.



(그림 6) 바다-IV/XML 질의처리기의 구조

4개의 모듈에 대해 간략히 설명을 하면 다음과 같다.

5.1.1 XQL 구문 분석기

바다-IV/XML의 OXQL 엔진 모듈에서 전달된 질의 문을 구문 분석하여 XQL 파스 트리를 만들어 XML 검색 관리기를 통하여 XML 검색 처리기에 전달한다. XQL 파스 트리를 구성하는 기본 클래스는 XQLNode 클래스이며 각 노드는 가지고 있는 정보에 따라 Unit, Rval, Path, Invocation, Filter, Bang, Param 클래스 등으로 구분된다.

5.1.2 XML 검색 관리기

XML 검색 관리기는 OXQL 엔진 모듈로부터 XQL 질의 문을 입력 받고, 결과를 OODB의 OXQL 엔진에 전달해 주는 역할을 한다. 그리고 입력 받은 XQL 질의에 대해 XQL 파스 트리를 만들기 위해 XQL 구문 분석기를 이용하고 질의를 처리하기 위해 XML 검색 처리기를 이용하며 필요한 결과 형태를 만들기 위해 XML 결과 처리기를 이용한다.

5.1.3 XML 검색 처리기

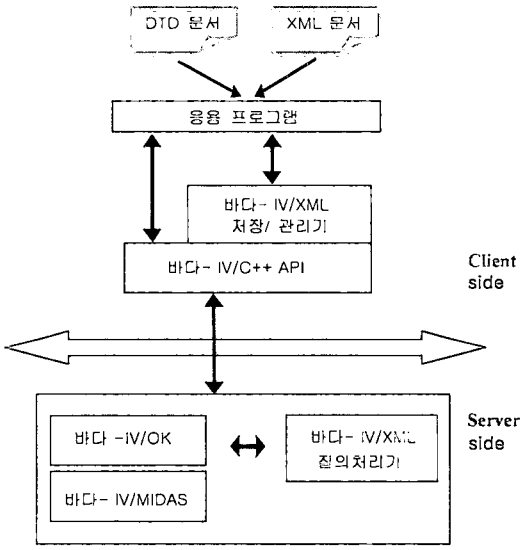
XQL 구문 분석을 통하여 나온 XQL 파스 트리를 XML 검색 관리기로부터 전달 받고, 전달 받은 XQL 파스 트리를 필요에 따라 질의 처리 계획을 이용하거나 트리를 순회하며 기본 술어 처리나 정보 검색 및 DOM API에서 제공하는 기본적인 조건 처리 함수들을 차례로 수행하여 결과를 구한다. 이때, 각 노드에서는 Unit, Path, Rval, Invocation, Filter, Bang, Param 등을 구분하여 각각 적절한 방식으로 처리한다. 처리 결과는 XML 결과 처리기를 통해 XML 검색 관리기로 전달 된다.

5.1.4 XML 결과 처리기

XML 검색 처리기에 의해 수행된 결과를 전달 받는다. 이때 전달 받은 내용은 단일형이거나 정보 검색의 결과로 나오는 가중치 값 등을 포함하는 복합형이 될 수가 있다. 그리고, 결과가 하나의 객체가 될 수도 있고 여러 객체의 집합이 될 수도 있는데 이런 여러 가지 결과 유형에 대한 처리를 한다.

5.2 바다-IV/XML 전체 구조

(그림 7)은 바다-IV가 XML 저장관리기/XML 질의처리기와 결합된 전체 구조이다. XML 저장관리기는 클라이언트 부분에 바다-IV/C++ API를 이용해서 구현이 되어 있고, XML 질의처리기는 서버 부분에 바다-IV/OK 엔진과 통합되어 구현이 되어 있다.



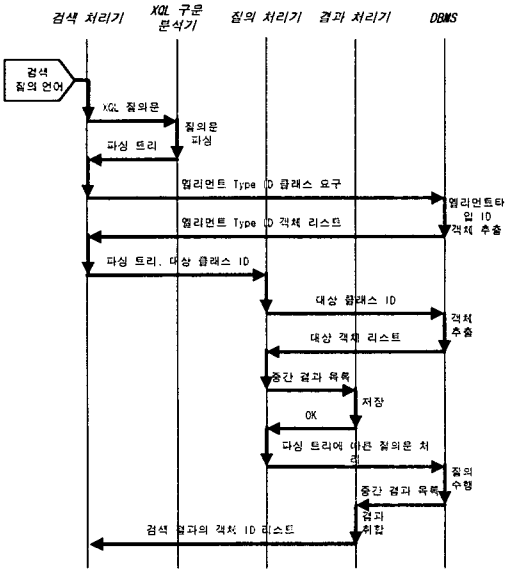
(그림 7) 바다-IV/XML의 전체 구조

5.3 OXQL의 처리 과정

OXQL로 작성된 질의의 처리 과정은 (그림 8)과 같다.

먼저, XQL 구문 분석기에서는 바다-IV/XML의 OXQL 엔진 모듈에서 전달된 질의 문을 구

문 분석하여 XQL 파스 트리를 만든다. XQL 파스 트리의 각 노드는 구문 분석 결과에 따라 Unit, Rval, Path, Invocation, Filter, Bang, Param 등이 된다. 구문 분석기에서는 다음과 같은 작업을 수행한다.



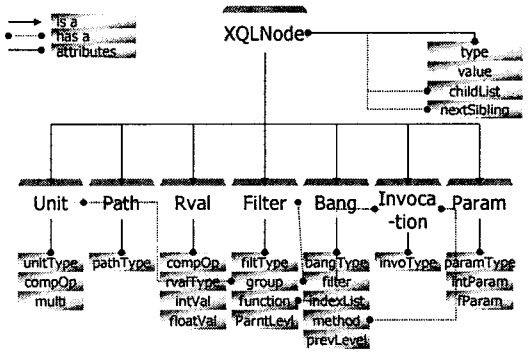
(그림 8) OXQL 질의 언어의 처리 과정

주어진 입력 질의 문을 구문 분석하여 XQL 문법에 올바른 질의문인지 검사한다. 올바른 질의문인 경우, 질의의 조건을 XQL 파스 트리로 구성한다.

사용자의 질의가 들어 오면 검색 관리기에서는 XQL 구문 분석기를 통하여 질의 언어를 파싱하고, 노드 ID를 이용한 검색을 위해서 미리 엘리먼트 이름과 타입 ID의 매핑 정보를 가지고 있는 객체를 얻어 온다. 그리고, 검색 관리기는 파스 트리와 검색 대상 클래스의 ID를 질의 처리기에 전달한다. 질의 처리기는 클래스 ID와 노드 ID를 이용해서 해당 클래스의 객체들 중에서 노드 ID가 일치하는 객체들을 추출하고 추출된 객체들에 대해서 파스 트리의 나머지 노드들을 순회하면서 질의 처리를 수행하게 된다. 질의 처리의 중간에 생성되는 결과값들은 결과 처리기에서 임시로 저장을 하고, 질의가 최종적으로 완료되면 결과 처리기는 결과들을 취합해서 검색 관리기에 결과를 넘겨 준다.

5.4 구문 분석기의 클래스 정의

XQL 구문 분석기는 입력 질의를 노드들에 대한 트리로 분리를 한다. 각 노드의 타입으로는 QUERY, UNIT, RVAL, PATH, BANG, SUBSCRIPT, FILTER, INVOCATION, PARAM 등이 있다. 이들 타입에 대해 각각 클래스로 정의하여 기능을 분리했다. XQL 구문 분석기의 결과인 XQL 파스 트리의 각 노드는 (그림 9)와 같이 XQLNode를 상속 받는 여러 클래스들로 이루어 진다.



(그림 9) XQL 구문 분석기의 클래스 구조

XQL 파스 트리의 루트 노드는 Query 노드이며 전체 질의를 나타낸다. 질의는 Unit 노드의 집합으로 이루어 지고, Unit 노드 자체로도 의미있는 질의 를 구성할 수 있다. 각 Unit 노드는 다음 타입 중에 하나이다.

- PATH_UNIT : 계층 정보만으로 이루어진 질의를 나타낸다. 예를 들면 /book/title은 하나의 PATH_UNIT으로 표현 가능하다.
- COMP_UNIT : 비교 연산자를 포함하는 질

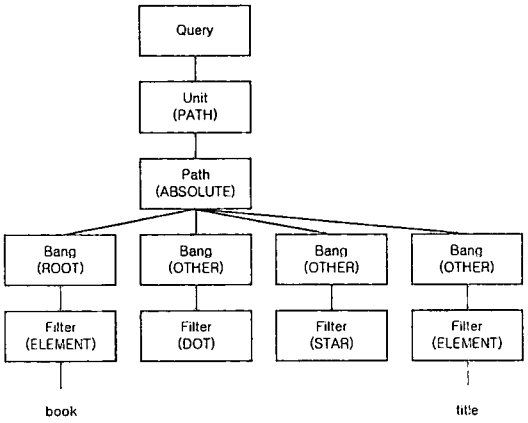
의를 나타낸다. 예를 들면 eq ('='), ne ('!=')를 포함하는 질의어는 COMP_UNIT으로 표현 가능하다.

- AND_UNIT : and 연산자를 포함하는 질의를 나타낸다.
- OR_UNIT : or 연산자를 포함하는 질의를 나타낸다.
- UNION_UNIT : $union$ ('|') 연산자를 포함하는 질의를 나타낸다.
- INTERSECT_UNIT : $intersect$ 연산자를 포함하는 질의를 나타낸다.
- NOT_UNIT : not 연산자를 포함하는 질의를 나타낸다.

Path 노드는 절대 경로와 상대 경로 둘 중에 하나를 표현한다. Path 노드는 질의의 부분 질의 내용을 갖고 있는 Bang 노드로 구성된다. 예를 들면, 하나의 Path 노드로 나타낼 수 있는 /book/title의 경우 "/"를 기준으로 book과 title 두개의 Bang 노드를 가지고 있다. Bang 노드는 Filter 노드에 대한 포인터를 가지고 있다. 각 Filter 노드는 다음 타입 중에 하나이다.

- DOT_FILTER : 현재까지의 검색 결과(current search context)을 나타낸다.
- INVOCATION_FILTER : ancestor()와 같은 함수나 메소드를 나타낸다.
- ELEMENT_FILTER : book이나 title 같은 엘리먼트를 나타낸다.
- STAR_FILTER : 계층 구조에서 한 레벨을 건너 뛴다. 예를 들어, /book/*/title은 book의 자식의 자식인 title 엘리먼트를 나타낸다.
- ATTRIBUTE_FILTER : @price와 같이 애트리뷰트를 나타낸다.
- GROUP_FILTER : 괄호와 같은 그룹 연산자를 나타낸다.

예를 들어 하나의 Path 노드로 나타낼 수 있는 /book/.*/title의 경우, (그림 10)에서와 같이 4개의 Bang 노드와 2개의 ELEMENT_FILTER, 1개의 DOT_FILTER, 1개의 STAR_FILTER로 구성된 트리로 표현할 수 있다.



(그림 10) XQL 파스 트리 예

Comp 노드는 비교 연산자를 포함하는 질의를 나타내며 Rval 노드를 가리키고 있다. Rval 노드는 비교 연산자를 중심으로 오른쪽에 있는 값을 나타낸다. 각 Rval 노드는 다음 타입 중에 하나이다.

- PATH_RVAL : 오른쪽에 있는 값이 Path를 나타내는 표현식이다.
- FLOAT_RVAL : 오른쪽에 있는 값이 부동소수이다.
- INT_RVAL : 오른쪽에 있는 값이 정수이다.
- TEXT_RVAL : 오른쪽에 있는 값이 문자열이다.

Invocation 노드는 메소드 또는 함수 호출을 포함하는 질의를 나타내며, Param 노드를 가리키고 있다. Param 노드는 메소드 또는 함수 호출의 인자를 나타낸다. 각 Param 노드는 다음 타입 중에 하나이다.

- UNIT_PARAM : 인자가 하나의 완벽한 질

의 형태를 갖는다.

- INT_PARAM : 인자가 정수 형태이다.
- FLOAT_PARAM : 인자가 부동소수 형태이다.
- TEXT_PARAM : 인자가 문자열 형태이다.

위에 정의된 클래스들을 사용하여 모든 가능한 XQL 질의를 트리 형태로 구성할 수 있고, XML 질의 처리기에서는 (그림 10)과 같은 XQL 파스 트리를 이용하여 검색을 수행한다.

6. 구조 검색 성능 분석

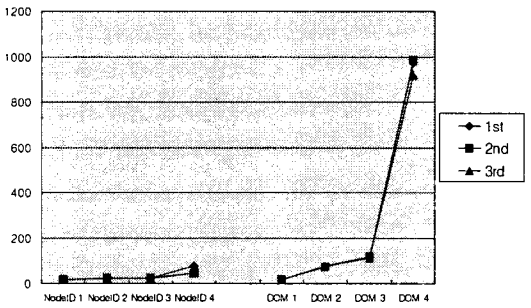
본 연구의 XML 질의처리기는 저장 구조가 DOM을 따르고 있다고 가정하므로 DOM 트리를 순회하면서 XML 문서의 구조 정보를 찾는 것을 기본으로 한다. DOM 트리를 순회하며 각 노드를 일일이 접근하다 보면 데이터베이스에 대한 접근이 많아 지고 처리가 지연되게 된다. 그래서, 각 노드의 정보에 노드 ID 정보를 추가하여 DOM 트리를 관리하고 이 노드 ID를 색인 정보에 추가하여 검색 응답 시간을 단축시켰다.

6.1 성능 분석

아래 (그림 11)은 노드 ID를 사용한 검색과 DOM 트리 순회 방법을 사용한 검색의 응답 시간을 나타낸다. 여기서, XQL 질의어의 깊이(depth)는 각각 1에서 4까지 (/PLAY, /PLAY/ACT, /PLAY/ACT/SCENE, /PLAY/ACT/SCENE/TITLE) 설정했고, 각각 3번의 검색을 수행했다.

그림에서 알 수 있듯이, DOM 트리를 순회하는 검색의 경우, 트리의 깊이가 증가할수록, 검색에 소요되는 시간은 지수적으로 증가한다. 그러나 노드 ID를 사용한 검색의 경우, 타입 ID를 한 번 얻어 오는 정도의 시간 증가만 발생하는 것을 볼 수 있다.

DOM 트리의 자식 노드의 개수가 증가할 수록, DOM 트리의 깊이가 증가할 수록 노드 ID를 사용한 검색이 DOM 트리 순회를 사용한 검색보다 월등한 성능 차이를 보일 것이다.



(그림 11) 노드 ID vs DOM 트리 순회 비교

일반적으로 어떤 특정 노드를 검색하기 위해 필요한 노드 탐색 횟수를 기준으로 DOM 트리를 순회하는 검색과 노드 ID를 사용하는 검색과의 성능을 다음과 같은 가정 하에서 비교해 보았다.

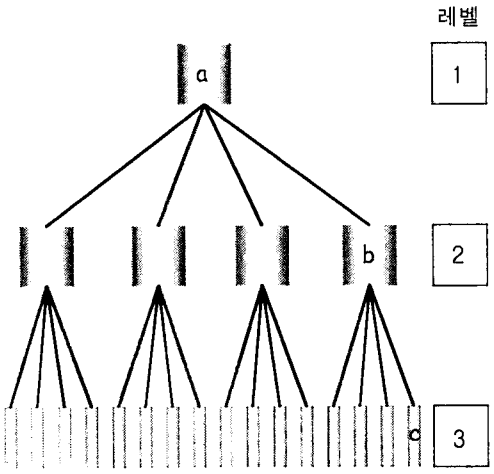
- DOM 트리는 완전 트리이다.
- 한 노드에서 자식의 수는 k개로 완전 k-ary 트리이다. (k >= 1)
- 자식 노드 중에서 원하는 자식 노드가 있을 확률은 p로 일정하다. (0 < p <= 1)

완전 k-ary 트리의 변수 k, p, i, d, f(i) 는 다음과 같이 정의를 한다.

- k 자식(children) 노드의 개수
- d 트리의 깊이(depth)
- p 자식 노드 중에서 원하는 엘리먼트가 있을 확률
- i 노드의 레벨(level)
- f(i) 루트 노드로부터 i 레벨까지 검색할 때에 방문하는 노드의 수

아래 (그림 12)는 4-ary 완전 DOM 트리의

예이다.



(그림 12) 4-ary 완전 DOM 트리의 예

6.2 DOM 트리 순회 방법

예를 들어서 (그림 12)와 같이 k = 4이고, d = 3, p = 0.5인 때에, DOM 트리 순회 방법을 사용하는 경우,

a의 노드 접근 회수는
f(1) = 1이고,
a/b의 노드 접근 회수는
f(2) = f(1) + k = 5 이고,
a/b/c의 노드 접근 회수는

$f(3) = f(2) + (kp)^{d-2} + k = 5 + (2)^{3-2} \times 4 = 13$

이다.

위의 예로부터, DOM 트리 순회 방법에서 d 레벨까지 방문하는 노드의 수인 $f(d_{dom})$을 구해 보면 다음과 같다.

$f(1) = 1$

$f(2) = f(1) + (kp)^0 \times k = f(1) + k$

$f(3) = f(2) + (kp)^1 \times k = f(2) + k^2p$

$f(d) = f(d-1) + (kp)^{d-2} \times k$

$$\begin{cases} f(d_{dom}) = \dfrac{k(kp)^{d-1}-1}{kp-1} - 1 & (kp>1 \text{ and } k=!1) \\ f(d_{dom}) = d & (kp=1 \text{ or } k=1) \end{cases}$$

6.3 노드 ID 적용 방법

예를 들어서, (그림 12)와 같이 k = 4이고, d = 3, p = 0.5일 때에, 노드 ID를 적용한 방법을 사용하는 경우,

a의 노드 접근 횟수는
f(1) = 1이고,
a/b의 노드 접근 회수는
f(2) = $(kp) \times f(1)$ = 2이고,
a/b/c의 노드 접근 회수는
f(3) = $(kp) \times f(2)$ = 4

이다.

위의 예로부터, 노드 ID를 사용하는 방법에서 d 레벨까지 방문하는 노드의 수인 $f(d_{nid})$를 구해 보면 다음과 같다.

$f(1) = 1$

$f(d) = (kp) \times f(d-1)$

$f(d_{nid}) = (kp)^{d-1}$이다.

6.4 DOM 트리 순회 방법과 노드 ID 적용 방법의 비교

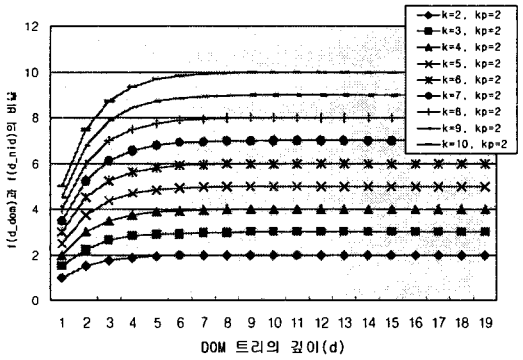
$f(d_{dom})$과 $f(d_{nid})$를 비교해 보면 다음과 같이 DOM 트리 순회 방법이 약 $\frac{k}{kp-1}$배 만큼 많은 노드를 거치게 된다.

$$f(d_{dom}) : f(d_{nid}) = \frac{k((kp)^{d-1}-1)}{kp-1} + 1 : (kp)^{d-1}$$

$$= \frac{k}{kp-1}\left(1 - \frac{1}{(kp)^{d-1}}\right) + \frac{1}{(kp)^{d-1}} : 1$$

$$\approx \frac{k}{kp-1} : 1$$

(그림 13)은 kp가 2일 때의 성능 비교이며, 자식 노드의 개수(k)가 증가할수록 DOM 트리를 순회하는 것보다 노드 ID를 사용하는 것이 월등히 뛰어난 성능을 보임을 알 수 있다.



(그림 13) k가 2일 때의 성능 향상 비율

7. 결 론

기존의 HTML의 부족한 점과 SGML의 복잡함 때문에 전자 문서 표준을 선정하지 못하던 많은 연구 기관과 업계에서는 XML에 대한 큰 관심을 갖고 연구를 계속하고 있으며, 많은 XML 구문 분석기, XML 문서 편집기 등 툴들이 개발되고, XML에 대한 많은 응용들이 나오면서 대량의 XML 문서가 생성되고 있다.

기존의 대량의 HTML 문서에 대한 웹 검색 엔진이나 메타 검색 엔진이 등장한 것처럼 대량의 XML 문서에 대한 검색 시스템이 필요하다. 그러나, XML 문서는 HTML처럼 단순하지 않고 내용 정보 뿐만 아니라 구조 정보와 속성 정보들이 다양한 형태로 구성이 되므로 구조 정보와 속성 정보에 대한 검색을 지원할 수 있는 질의처리기가 필요하다.

본 논문에서는 XML 문서의 내용 정보와 다양한 구조 정보를 검색할 수 있는 질의처리기를 설계 및 구현하였다. 이 XML 질의처리기에서 무엇보다 필요한 것은 질의 언어인데, 본 논문에서는 쉽고 사용하기 편한 XQL을 객체 지향 질의 언어인 OQL과 결합해서 XML 질의 언어로 제안했다. 이것은 기존의 관계형 SQL이나 객체형 OQL에서 XML을 지원하지 못하는 문제점을 보완한 것이다. 구조 정보인 부모와 자식간의 관계, 조상과 후손간의 관계, 형제들간의 관계에 대한 검색과 속성 검색은 XQL을 통해서 하고, 내용 검색과 데이터베이스 기능인 문서간의 통합과 조인은 OQL을 통해 지원한다.

본 논문에서 설계한 XML 질의처리기는 저장 구조가 DOM 기반으로 되어 있기 때문에 기본 연산은 부모와 자식 노드를 찾는 연산과 이전 형제 노드와 다음 형제 노드를 찾는 연산으로 검색을 한다. 그러므로, DOM 기반의 저장 구조에서 XML 문서를 검색할 수 있는 시스템으로 구현이 되었다. 그리고 DOM 기반의 저장 구조에서 각 노드의 정보에 노드의 ID를 두어 구조 정보를 검색할 때에 효율을 높였고, 앞 장에서 설명한 바와 같이 노드의 ID를 사용했을 때 DOM 트리를 순회하면서 탐색할 때보다 노드의 자식의 수에 비례하여 효율이 좋아지는 것을 보았다.

본 논문에서는 객체 지향 데이터베이스 시스템인 바다-IV에서 바다-IV의 OQL 안에 XQL을 추가하여 검색 질의 언어를 설계하고 구현했으며, 바다 IV/XML 질의처리기는 DOM 트리의 각 노드를 하나의 객체로 다루어 검색을 하고 결과 또한 각 노드를 하나의 결과 객체로서 처리를 하도록 구현이 되었다.

향후 연구 과제로는 XML 문서의 링크 정보에 대한 검색과 결과를 추출할 때 결과를 사용자가 원하는 형태의 XML 문서로 재구성하는 기능과 XML 문서의 그림 등 외부 개체의 검색에 대한 연구가 필요하다.

참 고 문 헌

[1] Tim Bray, Jean Paoli, C.M. Sperberg-Mc Queen," Extensible Markup Language (XML) 1.0," http://www.w3.org/TR/REC-xml-19980210/, 1998.

[2] HTML, http://www.w3.org/MarkUp/, W3C

[3] 이미영, 허대영, 김명준, "바다-III 멀티미디어 DBMS 설계", 한국전자통신연구원

[4] Apparao V., S. Byrne, et al., "Document Object Model (DOM) Level 1 Specification," http://www.w3.org/TR/REC-DOM-Level-1/, 1998.

[5] Cattell R.G.G., "The Object Database Standard : ODMG-93," Morgan Kaufmann, San Francisco, California, 1994.

[6] Jonathan Robie, Joe Lapp, and David Schach, "XML Query Language (XQL)," http://www.w3.org/TandS/QL/QL98/pp/xql.html, 1998.

[7] 김용훈, "다양한 구조 검색을 지원하는 XML 검색기의 설계 및 구현", 충남대학교 석사 학위 논문, 1999.

[8] W3C, "QL'98 - The Query Languages Workshop," http://www.w3.org/TandS/QL/QL98/

[9] Stephen Deach, "Extensible Stylesheet Language (XSL) 1.0," http://www.w3.org/TR/2000/WD-xsl-20000301/, 2000.

[10] eXcelon, http://www.exceloncorp.com/

[11] ObjectStore, http://www.exceloncorp.com/products/objectstore.html

[12] 박순영, 정광철, "MIDAS-III 블록 설계서", 한국전자통신연구원.

[13] 박순영, 정광철, "MIDAS-III 인터페이스 규격서", 한국전자통신연구원.

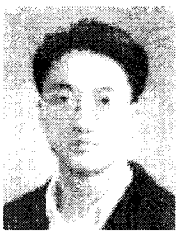
[14] Software AG, http://www.softwareag.com/

[15] Tamino 1.2.1 Documentation, http://softwareag.penta.co.kr/sagpublicman/Ino1211/overview.htm

저자소개

이 명 철

충남대학교 컴퓨터공학과를 졸업하고, 충남대학교에서 석사과정 중이다. 주요관심분야는 데이터베이스, XML, 정보통합, WWW분야이다.



김 상 균

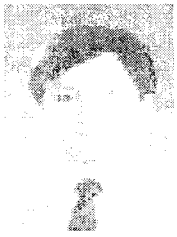
충남대학교 정보통신공학과를 졸업하고, 충남대학교에서 석사과정 중이다. 주요관심분야는 데이터베이스, XML, 정보통합, WWW 분야이다.

손 덕 주

서울대학교 수학교육과를 졸업하고, KAIST 전산학과에서 석사학위를 취득하였으며 한국전자통신연구원에서 분산처리 기술개발 및 멀티미디어 문서 데이터베이스기술 개발업무를 담당하였다.현재 인터넷서비스 연구부 책임연구원으로 재직하고 있으며 주요관심분야는 이동 컴퓨팅, 클러스터 기반의 데이터베이스 시스템 분야이다.

김 명 준

서울대학교 계산통계학과를 졸업하고, KAIST에서 석사, 그리고 프랑스 Nancy 제1대학교에서 박사학위를 취득하였으며 한국전자통신연구원에서 책임연구원으로 근무 중이다. 현재 컴퓨터소프트웨어기술연구소 소장으로 재직하고 있으며 주요 관심분야는 데이터베이스, 분산 시스템, 소프트웨어 공학 분야이다.



이 규 철

서울대학교 컴퓨터공학과를 졸업하고, 서울대학교에서 석사, 그리고 서울대학교에서 박사학위를 취득하였으며, 미국 Almaden Research Center에서 객원 연구원으로, 미국 Syracuse 대학 CASE Center에서 객원 교수로 근무하였다. 현재 충남대학교 컴퓨터공학과 정교수로 재직하고 있으며 주요관심분야는데이터베이스, XML, 정보통합, 멀티미디어 시스템 분야이다.