KIMF를 이용한 XML 인덱싱 알고리즘 설계 및 구현

Design and Implementation of XML-Based Indexing Algorithm Using KIMF

°김광남, *윤희병, **김화수
°*국방대학교 전산정보학과, **아주대학교 정보통신대학원

KwangNam Kim°, Heebyung Yoon*, Hwa-Soo Kim**
°*National Defence University, Dept. of Computer & Information Science
**Ajou Univ., Graduate School of Information & Communication Technology
E-mail : kma6471@hanmail.net

요 약

XML은 사용자 정의 태그를 이용한 정보 제공의 장점으로 인하여 웹 뿐만 아니라 EDI, CALS, RDF, HDML, WML등 많은 분야에서 사용하고 있다. 그러나 XML 문서는 다양한 사용자 정의를 포함하기 때문에 사용자 질의에 대하여 효율적으로 응답하기 위해서는 내용기반 질의로부터 구조, 내용과 구조가 포함된 질의까지 해결해 줄 수 있는 인덱싱 기법이 필요하다. 이를 해결하기 위해 DTD를 이용한 방법, K-ary 완전트리, 추상화기법, SCL 등이 제시되었으나 XML 노드의 거리관계를 이용한 방법은 제시되지 않았다.

본 논문에서는 국방대 인덱싱 모델 프레임워크인 KIMF를 이용하여 효율적으로 XML 문서를 인덱싱하고 설계 및 구현한다. 이를 위해서 KIMF 모듈에 대한 구성도를 제시하고, 깊이탐색과 최단거리, 깊이 차이를 이용하여 XML 문서를 인덱싱하는 알고리즘을 제안한다. 마지막으로 C# 을 이용하여 제안한 알고리즘을 설계 및 구현하고, 이를 기반으로 한 내용검색, 구조검색 및 혼합(내용+구조) 검색 결과를 또한 제시한다.

1. 서론

W3C에 의해 XML1.0이 표준으로 발표된 이래 e-business 트랜잭션과 같은 XML 어플리케이션의 빠른 증가로 여러분야에 확산되어 사용되고 있으며, XML 데이터의 효율적인 전달이 중요시되고 있다[1]. XML은 웹 문서뿐만 아니라 전자도서관, EDI, CALS, 채널기술의 CDF, 메타데이타를 기술하기 위한 RDF, 이동통신에서의 HDML, WML 등 여러 다양한 분야로 그 활용범위가 확대되고 있다.

이처럼 웹상에서 다양하고 무수히 많은 웹 문서로부터 사용자 질의에 맞게 검색하는 것은 중요한 과제 중의 하나이다. 사용자 질의에 대해 신속하고 정확하게 크롤링된 검색결과를 제공하기 위해서는 효율적으로 인덱싱 하는 것이 필요하다. 기존 HTML은 표현위주의 태그로 구성되어 있어 검색에 필요한 의미해석에는 많은 어려움을 가지고 있다. 이에 반해 XML은 사용자 태그를 이용하여 의미를 해석하고 도메인별로 웹 문서를 분류할 수 있는 방법을 제공한다. XML의 이러한 장점을 이용하여 인덱싱하기 위한 여러 가지 방법이 시도되었다. 대표적인 예로는 규칙을 기반으로 한 DTD,부모노드와 자식노드간의 관계를 tree구조로 표현한 추상화[2], 텍스트 간격을 이용하여 문서구조의 질의를 지원하는 SCL[3][4], 임의의 노드들 중 가장 많은 자식을 갖는 노드의 차수를 이용한 K-ary 등이 있으나 깊이탐색과 최단거리를 이용한 인덱싱 방법은 없었다.

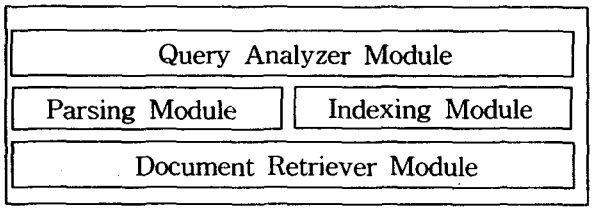
본 논문에서는 국방대 인덱스 모델 프레임워크인 KIMF를 이용하여 인덱싱 알고리즘을 설계

및 구현한다. 이를 위해 먼저 KIMF의 모듈을 설계하고, 깊이 탐색과 최단거리를 이용하여 노드를 식별하고 파싱하는 인덱싱 알고리즘을 제안한다. 그런 다음 제안한 알고리즘에 대해서 C#을 이용하여 설계 및 구현하고, 마지막으로 결론과 향후 연구방향을 제시한다.

2. KIMF 설계

2.1 모듈 구성도 설계

KIMF는 사용자의 질의에 효율적으로 검색결과를 제공하기 위하여 XML문서의 파싱, 인덱싱, 질의 분석을 수행한다. 그리고 수행결과에 대해서 내용, 구조, 혼합 검색을 가능하도록 해준다. 이와같은 KIMF의 4가지 모듈 구성도가 [그림 1]에 도시되어 있다.



[그림 1] KIMF 모듈 구성도

Query Analyzer Module은 사용자 질의를 내용, 구조, 혼합 검색으로 분석하여 인덱싱된 테이블로부터 질의결과를 보여준다. Parsing Module은 XML문서를 인덱싱하기 전에 노드별로 ID를 식별하여 부모노드, 자식노드, 형제노드로 구분한다. Indexing Module은 파싱된 각 노드를 Index DB에 저장한다. 사용자가 질의시 질의내용과 Index DB를 매핑한다. 마지막으로 Document Retrieval Module은 사용자 질의 내용을 포함하는 문서를 탐색하여 보여준다.

2.2 노드 식별

XML 문서 위주의 구조정보 표현시 노드의 ID를 다르게 부여할 수 있다[5]. 노드의 ID는 해당 노드를 유일하게 식별할 수 있게 한다. XML 문서에는 문서 내 또는 문서 사이에 동일 이름의 노드가 존재하고, 형제노드 사이뿐만 아니라 노드와 attribute간에도 동일한 이름이 존재한다. 따라서, 동일 이름을 갖는 노드들간의 관계는 해당 노드의 고유한 ID를 이용하여 추출한다.

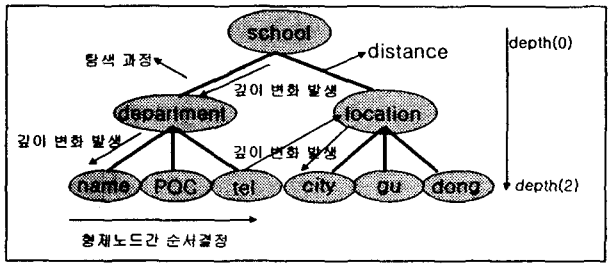
2.2.1·부모노드

XML문서는 tree구조로 표현이 가능하며 깊이 탐색과 노드간 최단거리를 통하여 부모노드를 식별할 수 있다. [그림 2]는 XML 문서의 한 예이다.

```
<school>
  <department>
    <name>
      Computer and Information Science
 </name>
      <POC>Kim ChulSoo</POC>
      <tel>02-123-1234</tel>
  </department>
  <location>
     <city>Seoul</city>
    <gu>EunPyung</gu>
    <dong>Susaek</dong>
  </location>
</school>
```

[그림 2] XML 문서

XML 문서는 계층적으로 표현이 가능하므로 tree 구조로 바꾸어 표현할 수 있다. 이와같은 tree구조의 계층적 표현을 이용하여 [그림 2]의 XML문서를 [그림 3]과 같이 표현할 수 있으며, 각 노드간 깊이(depth)변화와 거리(distance), 형제노드간 순서 또한 표시되어 있다.



[그림 3] XML tree 구조

부모노드의 추출은 노드의 깊이변화를 이용하여 다음의 두 가지 단계에 의해 수행된다. 첫 번째는 현재 노드에 대한 자신의 상위노드를 깊이 탐색을 이용하여 식별하는 단계이며, 두 번째는 상위노드에 대하여 가장 짧은 연결선의 거리를 구하는 단계이다. [그림 3]에서 POC의 경우 상위노드는 department와 location이 되지만 가장 짧은 거리에 있는 department가 부모노드임을 알 수 있다. 따라서, 상위노드를 찾기 위한 식은 다음과 같이 부모노드 p로 정의된다.

$$p=(depth(n)-1)\wedge distance_{oth}(oth,n) \text{-------(1)}$$

여기서 depth는 노드 깊이를, n은 현재 노드를, distance는 노드사이 거리를 oth는 상위노드들을 나타낸다. 식 (1)에서 부모노드 p는 현재 노드의 깊이보다 한 단계 위에 있어야 하며, 상위노드 사이에서 가장 짧은 거리에 위치하는 노드이다.

2.2.2 형제노드

형제노드는 깊이탐색중 깊이차이가 발생하는 경우에 대하여 별도의 Queue를 정의하고, Queue의 길이와 깊이간의 차이를 비교하여 형제노드를 식별하고, 순서를 계산 할 수 있다. 이에 대한 알고리즘을 [그림 4]에 나타난다.

```
Let Q(i) be Length of Queue Sibling nodes locate
Let Dep(j) be the depth of j node
Let n be a name of node
procedure
begin
    if(j==0)                          ------------ ①
      add 1 into Q(0)
    else if(i==j)                     ------------②
    {
     extract depth(j)
     set 1 into Q(i)
     }
    else if(i>j)                      ------------③
    { extract Q(depth(j))
      Update Q(depth(j)) into Q(depth(j)+1)
    }
end
```

[그림 4] 형제노드 추출 알고리즘

[그림 4]의 알고리즘을 [그림 3]을 이용하여 설명하면 school은 root 노드이므로 ①에 해당하며 Queue의 0위치에 저장된다. 노드의 깊이변화가 발생하는 school → department, department → name, tel → location, location → city의 경우에는 ②에 해당하며, Queue에 저장한 길이와 노드 깊이 값이 같은 경우에는 해당되는 노드 깊이의 순서값이 저장된 Queue의 위치에 1로 설정한다. 형제노드간의 순서를 결정하는 경우에는 ③에서 제시하는 것처럼 대상 노드의 깊이에서 순서값이 저장된 Queue의 현재값에 1을 더하며, name, POC, tel이 해당된다. 즉, name, POC, tel의 경우에는 깊이가 2이므로 Queue(2)의 위치에서 name은 1의 값을, POC는 name의 다음순서이므로 1의 값을 더한 2, tel은 POC의 다음 순서값이므로 name의 순서값에 1을 더한 3이 된다.

2.3 테이블 설계

추출된 부모노드와 형제노드의 ID를 통하여 노드 사이의 관계를 식별한 다음에는, 구조정보를 테이블에 저장한다. 그런 다음, 사용자 질의에 매핑되는 관련 노드들의 정보를 이용하여 확인할 수 있는 문서 및 노드 테이블을 설계하며, 이들 테이블은 [그림 5]와 [그림 6]에 나타나 있다.

DocId	DocName	DocPath

[그림 5] 문서 테이블

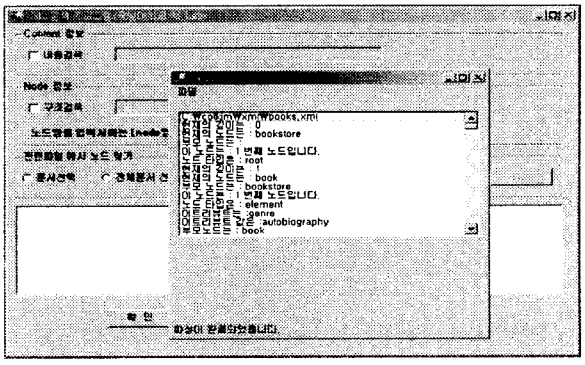
Eid	DocId	Name	Pid	depth	Sorder	type	content

[그림 6] 노드 테이블

문서 테이블에서 DocId는 문서의 ID이며, DocName은 문서의 이름이고, DocPath는 문서가 저장되어 있는 위치를 나타낸다. 노드 테이블은 노드의 정보를 나타낸 매핑 테이블로서, Eid는 노드에 대한 ID이고, DocId는 문서 ID를 나타내며, 이것은 문서 테이블의 DocId와 동일한 값을 갖는다. Name은 노드명, Pid는 부모노드를 식별하기 위한 ID로 상위노드 ID를 나타낸다. depth는 노드의 깊이를 나타내는 값이며, Sorder는 형제노드간의 순서를 나타낸다. type은 element, attribute, root 노드 등을 식별하기 위한 값이며, content는 노드가 가지고 있는 텍스트를 저장한다. 만약, type이 attribute를 나타내면 content는 attribute가 갖는 텍스트 값을 의미하고, type이 element를 나타내면 노드가 갖는 텍스트의 값을 나타낸다.

3. 구 현

지금까지 제시하고 설계한 내용을 기반으로 하여 C#을 이용하여 구현한다. 파싱은 전체 검색된 모든 XML문서를 대상으로 할 수도 있고, 특정 XML 문서를 선택하여 파싱을 할 수도 있다. 파싱결과가 [그림 7]에 나타나 있다.



[그림 7] XML 문서의 파싱결과

[그림 7]에서 각 문서의 파싱결과로 나타나는 노드별 Type과 깊이, 형제노드 순서, 텍스트 내용등을 화면에서 보여준다. 현재 저장되어있는 문서가 없을 경우에는 "해당되는 문서가 없음"을 표시한다. 파싱과 인덱싱이 완료되면 생성된 테이블에 매핑된다. 그 결과가 [그림 8]과 [그림 9]에 나타나 있다.

DocId	DocName	DocPath	Inputdate
1	books.xml	C:₩cptkim₩xm	2004-02-27 오후
2	EID.xml	C:₩cptkim₩xm	2004-02-27 오후
3	KNOWLEDGE.xr	C:₩cptkim₩xm	2004-02-27 오후
4	mail.xml	C:₩cptkim₩xm	2004-02-27 오후
5	Noname1.xml	C:₩cptkim₩xm	2004-02-27 오후

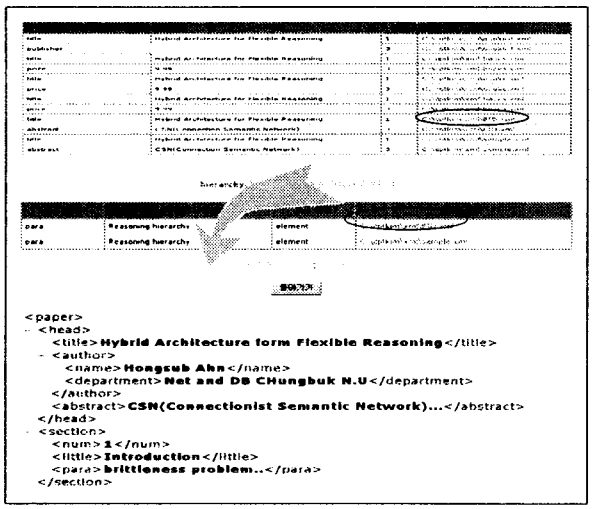
[그림 8] 생성된 문서 테이블

Eid	DocId	EName	Pid	depth	Sorder	EType	Content
1	1	bookstore	0	0	1	root	
2	1	book	1	1	1	element	
3	1	genre	2	1	1	attribute	autobiography
4	1	publicationdate	2	1	2	attribute	1981
5	1	ISBN	2	1	3	attribute	1-861003-11-0
6	1	title	2	2	1	element	Hybrid Architecti
7	1	author	2	2	2	element	
8	1	first-name	7	3	1	element	Herman
9	1	last-name	7	3	2	element	Melville
10	1	price	2	2	3	element	9.99
11	1	book	1	1	2	element	

[그림 9] 생성된 노드 테이블

[그림 8]은 파싱결과 생성된 문서 테이블로서 문서 ID 1은 books.xml 파일을 나타내며, 현재 저장되어 있는 절대 경로와 인덱싱된 일자가 표시된다.[그림 9]에서 노드 ID가 10인 price의 경우에 문서 ID는 1이며, [그림 8]의 문서 ID와 일치하므로 books.xml에 저장이 되어 있다는 것을 알 수 있으며, Pid가 2를 나타내므로 부모노드는 book이라는 것을 알 수 있다. 또한 형제노드의 순서는 3이고 첫째 형제노드는 first-name, 두 번째 형제노드는 last-name임을 알 수 있다. 또한 type은 엘리먼트 즉, 노드를 나타내고, price 노드가 가지고 있는 텍스트는 9.99라는 것을 알 수 있다.

KIMF를 이용한 XML 인덱싱 알고리즘의 효율성 여부를 분석하기 위하여 C#과 SQL Server2000을 이용하여 구현한 결과가 [그림 10]에 나타나 있다.



[그림 10] 내용 및 구조 검색 결과

[그림 10]의 상위 테이블은 내용검색에 대한 결과로서 노드명, 노드가 가지고 있는 텍스트, 형제노드 순서, 그리고 현재 노드를 가지고 있는 문서의 절대주소 항목이 포함되어 있다. 그리고 구조검색에 대한 결과 또한 하위 테이블에 나타나 있다. 테이블의 절대주소를 클릭하면 [그림 10]과 같이 해당문서에 대한 XML 문서가 상호 연결되어 있음을 보여준다. 이 결과를 통하여 구현된 KIMF는 DTD나 Schema등의 구조적인 규칙 기반이 아닌 노드의 깊이와 최단거리를 이용하였으므로, 규칙에 제한을 받지 않으며, 단지 파싱 알고리즘 내부에 유효성 검사를 포함하고 있다. 또한 DocumentType에 무관하게 구현이 가능하다는 장점이 있다.

4. 결론 및 향후 연구방향

본 논문에서는 KIMF를 이용한 XML 인덱싱 알고리즘에 대한 설계 및 구현을 하였다. 이를 위해 KIMF를 구성하는 모듈과 인덱싱 알고리즘, 그리고 제안된 알고리즘을 매핑하기 위한 테이블을 설계하였고, 내용 및 구조, 그리고 혼합검색에 대한 구현 결과를 제시하였다.

향후 연구과제로서 KIMF를 확장하여 순차적 패턴을 이용한 클러스터링과 이와 연계한 모델을 연구할 예정이다.

6. 참고문헌

[1] L.H. Yang et al., "Mining Frequent Query Patterns from XML Queries," 8th International Conference on Database Systems for Advanced Applications (DASFAA'03), 2003.

[2] J.H. Chow et al., "Indexing Design for Structured Documents Based on Abstraction," 6th International Conference on Database Systems for Advanced Applications, pp.89-96, 1999.

[3] Tuong Dao, "An Indexing Model for Structured Documents to Support Queries on Content, Structure and Attributes," Proceedings of ADL, pp.88-97, 1998.

[4] Tuong Dao et al., "An Indexing Scheme for structured Documents and its Implementation," 5th International Conference on Database Systems for Advanced Applications(DASFAA), pp.125-134, 1997.

[5] Y.K. Lee et al., "Index Structures for Structured Documents," Proc. Digital Library 96, pp.91-99, 1996.