碩士學位論文

효율적 문서 검색 및 변경을 위한 엘리먼트 ID 상속 기반의 XML 저장 구조

濟州大學校 大學院

컴퓨터工學科

康 仁 碩

2004年 12月

효율적 문서 검색 및 변경을 위한 엘리먼트 ID 상속 기반의 XML 저장 구조

指導敎授 郭 鎬 榮

康 仁 碩

이 論文을 工學 碩士學位 論文으로 提出함

2004 年 12 月

康仁碩의 工學 碩士學位 論文은 認准함

審査 委員長 _____ 印

委 員 _____ 印

委 員 _____ 印

濟州大學校 大學院

2004 年 12 月

XML storage structure based on the Element ID inheritance for efficiency of document Searching and Updating

In-Suk Kang

(Supervised by professor Ho-Young Kawk)

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

GRADUATE SCHOOL

CHEJU NATIONAL UNIVERSITY

2003. 12.

목 차

그 림 목 차

표 목 차

Summary

XML storage structure based on the Element ID inheritance for efficiency of document Searching and Updating

In-Suk Kang
Dept. of Information Engineering
The Graduate School
Cheju National University

In this thesis, ID inheritance storage system model base on XML document and information of schema element structure has proposed. Proposed system parse XML document and read in DOM type then assign ID to element by using analysis module. At this time, ID assigned to child element is inherited from parent element ID and assigned ID will be saved to database as form of defined schema structure proposed for efficient search and updates of document.

As results, when specific element has inserted or updated, location information are not required to configured again.

Ⅰ. 서 론

최근 인터넷 사용과 정보의 양이 급증하면서 인터넷상의 정보를 보다 효과적으로 사용하고자 하는 연구가 활발히 진행되고 있다[1]. 한편 웹에서 보편화 되어진 HTML(HyperText Markup Language)은 문서의 구조보다 표현에 중점을 두고 있어 특정 응용분야의 구조를 표현하는 문서로는 기능이 부족하다는 단점이 가지고 있다[1,6,7]. 이에 웹 발전의 주도적인 역할을 하고 있는 W3C(World Wide Web Consortium)에서는 HTML의 편리성과 SGML(Standard Generalized Markup Language)의 확장성 등의 장점을 취합하여 웹 문서의 표준이라 말할 수 있는 XML(eXtensible Markup language)을 1996년 제안하였고[1,2,3], 이기종간의 시스템에 작성된 문서의 상호교환과 다양한 형식의 문서들을 일관성 있게 구조화하기 위하여 고안되었다[4].

또한 SGML을 간략화 시키고 HTML을 보다 사용자의 다양한 욕구를 충분히 수용할 수 있어 웹 문서뿐만 아니라 전자출판, 의학, 경영, 법률, 판매 자동화, EDI(전자 문서 교환), EC(전자 상거래), KMS(지식 관리 시스템), MathML(수학), e-business, VoiceML(음성인식 시스템), 디지털 전자도서관 등 다양한 분야에서 XML을 활용하고자 폭 넓은 연구에 대한 구체화가 빠르게 진행이 되어지고 있는 실정이다[2,5,6,7]. 이런 폭 넓은 연구에도 불구하고 대용량의 XML 문서를 기존 데이터베이스 저장과 검색을 지원하는 위한 효율적인 문서 관리 시스템의 필요성이 대두되었다[8,9]. 기존 데이터베이스 기반의 XML 문서 저장 구조 연구는 문서를 관계 또는 객체지향 데이터 모델로 변환하여 저장하는 모델링 연구와 효율적인 검색을 위한 인덱스 및 검색 구조 설계 연구로 나누어진다[10].

이런 연구들을 살펴보면 XML 문서 구조 갱신이 발생하는 경우, 이에 따르는 모든 구조 정보가 수정되어야 한다. 그리고 순차적 탐색 기반으로 리스트 형태의 정보를 데이터베이스에 저장하므로, 검색 시 효율이 떨어지거나 특정 위치에 대한 엘리먼트 탐색 시 문제점이 발생하게 되었다[10,11].

따라서, 본 연구는 기존 연구들의 장점을 최대한 수용하여 동적 환경에 적합

한 효율적 문서 검색 및 변경을 위한 엘리먼트 ID 상속 기반의 문서 저장 구조를 제안하고 모델링하였다.

엘리먼트 ID 상속 의미는 엘리먼트를 구분을 위해 사용된 위치정보(offset)를 말한다. 위치정보는 부모 엘리먼트의 ID 정보를 자식 엘리먼트 구성요소로 재사용하는 것을 말하며, 또한 이들 정보들을 저장하기 위한 별도의 저장 스키마는 데이터베이스에 새로운 스키마로 정의되어 저장되었다.

본 연구의 구성은 먼저, Ⅰ장 서론에서는 XML의 연구 동향에 대하여 서술하고, Ⅱ장에서는 XML 문서에 대한 이론과 구조정보를 모델링하는 방법과 저장하는 방법에 대한 기존 연구들을 살펴보겠다. Ⅲ장에서는 문서 검색 및 변경을 위한 엘리먼트 ID 상속 기반의 XML 문서 저장 구조 모델링과 엘리먼트 ID부여알고리즘에 대해서 설명을 하며, Ⅳ장에서는 본 연구에서 제안한 문서 저장 구조를 적용한 결과를 보였으며, 마지막으로 Ⅴ장에서는 본 연구에서 제안한 알고리즘의 실용성과 향후 연구 과제 및 방향에 대해 서술하였다.

Ⅱ. XML 이론 및 배경

1. XML 문서 구조

XML 문서 구조는 정형화된 구조이다. 일반적인 XML 문서 구조를 살펴보면 다음 Fig.1과 같다.

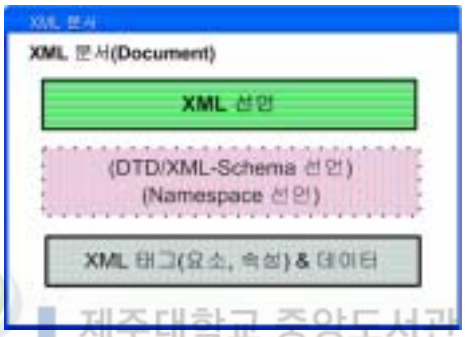


Fig. 1 Structure of XML document

XML 문서는 도입부에 XML 선언(Declaration)이 필요하며, XML 선언의 기본적인 형태는 처리지시(Processing Instruction) 요소인 〈?이름... ?〉과 동일하다. 최소화 형태의 XML 선언은 예약어와 XML 스펙 버전 번호로 구성된다. XML 문서 표현상의 인코딩에 관한 특별한 사항이 존재한다면, 인코딩에 따르는 정보를 넣을 수 있다. 다음으로 XML 문서의 정의 타입에 따르는 유효성을 검증하기 위한 DTD(Document Type Definition), XML 스키마(Schema) 선언, 네임스페이스(Namespace) 선언이 올 수 있다. 하지만 이 부분은 선택 사항이므로 필요한 경우에 선언할 수 있다. 마지막으로 XML 태그와 컨텐츠로 이루어진 몸체(body)가 있다. 몸체는 요소(Element)와 속성(Attribute)으로 구성된다. 요소는 시작태그와 종료태그 쌍과 그 사이에 들어 있는 컨텐츠인 데이터를 통칭하는 것이다. 속성은 XML 파서가 애플리케이션에게 보내는 값들을 뜻하며, 요소의 컨텐츠 값과는 서로 구별되는 값이다. 속성은 요소의 시작태그 안에서 선언되며, 해당 애플리케이

션에 보내는 값을 지정할 수 있다.

2. XML 저장 모델

XML 데이터는 계층적 구조로 표현이 되며, 이러한 구조를 저장하기 위해 먼저 선행되어야 하는 것이 데이터 모델링 부분이다. 데이터 모델링이란 데이터와 데이터들간의 관계를 기술하는 개념적 도구로서, 데이터베이스의 논리적 구조를 명시하는 과정이다. 이 과정은 DBMS(Database Management System)의 활용에 따라 관계형 모델과 객체지향 모델로 나누어진다[3, 7].

1) 관계형 모델

현재 가장 많이 사용되고 있는 관계 데이터베이스 관리 시스템(RDBMS)을 활용하는 경우이다. 이 모델은 안정된 DBMS의 기능을 활용할 수 있다는 장점과 관계 데이터베이스의 기능을 이용하여 데이터를 테이블 형태로 관리하기 때문에 사용자의 쉬운 접근이 가능하지만 여러 가지 문제점이 발생한다. 이러한 문제점들을 살펴보면 계층적 구조 형태의 XML 데이터를 단순한 테이블 구조로 변환해야 한다. 또한, XML 데이터에 자주 사용되는 다중값(collection) 속성이 별도의 테이블로 표현되어야 하며, 엘리먼트 간의 순서정보 유지를 위해 부가정보를 추가해야 한다. 그리고 XML 질의에 자주 사용되는 경로식에 대한 높은 조인 연산으로 변환되어 처리되므로 비효율적이다.

마지막으로, 질의 결과가 다중값을 포함하는 경우 다중값이 아닌 속성들에서 불필요한 중복이 발생하며, 이로 인해 질의 결과를 XML 데이터로 변환하는 알고리즘이 복잡하다. 이러한 여러 가지 이유로 관계 데이터베이스의 기능을 확장하지 않고서 XML 데이터 검색의 성능을 향상시키는 것은 어려운 일이다.

2) 객체 지향 모델

두 번째 방법은 객체 지향 데이터베이스 관리 시스템(ODBMS)를 활용하는 것

이다. 객체 지향 데이터베이스 관리 시스템은 객체 지향 개념을 이용할 수 있기 때문에 상속과 같은 객체 지향 특성을 이용할 수 있으며, 엘리먼트 간의 전후종속 관계를 클래스에 기반한 객체들간의 링크로 나타낼 수 있다. 관계 데이터베이스보다 데이터 모델이 XML과 유사하여 DTD로부터 객체 스키마를 생성하는 문제와 XML 질의를 OQL(Object Query Language)로 변환하는 과정이 쉽고 가변 필드의 사용에 제한이 없어서 대용량 데이터 형식(built-in data type)으로 지원되기 때문에 XML의 순서정보 표현이 간단하다. 또한, XML 질의에 자주 사용되는 경로식은 조인(Join)이 아니라 객체 참조 형태로 변환·처리되므로 질의 변환이 용이하고, 성능도 향상된다. 그리고 질의 결과도 중첩된 형태로 제공되므로 간단하게 XML 형태로 변환된다.

3) 그 밖의 모델

위의 두 가지 모델 외 다른 모델을 살펴보면, XML 데이터를 처리하기 위한 새로운 저장 모델인 Tamino, eXcelon 등이 있다. 이와 같은 시스템은 XML 데이터를 파일 단위로 저장하고, XML 데이터의 구조 정보를 인덱스로 구축하여 원하는 데이터를 검색할 때 사용된다. 데이터 모델 변환이나 데이터 저장에서 부담이 적지만, 데이터 검색을 위하여 여러 가지 인덱스를 참조해야 하므로 대규모 XML 데이터 처리에서 성능이 낮다고 알려져 있다.

다음으로 저장방식에 따른 분류에는 XML 문서 전체를 한꺼번에 저장하는 분할 저장 모델과 문서를 엘리먼트 단위로 나누어 저장하는 가상 분할 저장 모델로 나누어진다.

3. XML 저장 방법

구조적 XML 문서에 대한 효율적인 질의 처리 기법과 많은 양의 데이터 관리를 위한 저장 시스템 기법에 대한 연구들이 진행되었다. 특히 기존 데이터베이스 시스템(RDB : Relational DataBase, OODB : Object-Oriented DataBase)을 이용하

여 XML 문서를 저장하려고 하는 연구와 별도의 데이터베이스 시스템을 개발하는 방법으로 나누어지고, 문서 DTD(Document Type Definition)를 이용하여 관계형 데이터베이스의 스키마를 생성한 후 XML 문서를 저장하는 연구와 DTD 문서 없이 XML 데이터를 관계형 데이터베이스에 저장하려는 방법도 있다.

계층적 XML 문서 구조정보를 데이터베이스에 저장하는 방법들을 소개하겠다. XML 구조 정보를 저장하는 방법으로는 크게 분할 저장과 가상 분할(Virtual Fragmentation)로 나눌 수 있다.

1) 분할 저장 방법

문서의 트리구조의 각 객체가 실제 노드의 내용을 가지고 저장되는 분할 저장(Decomposition) 방법을 살펴보면, XML 문서를 엘리먼트 단위로 쪼개어 저장하고, 검색 시 구조 정보를 참조하여 해당 엘리먼트나 하위 엘리먼트의 조합을 생성하여 처리하는 방식이다.

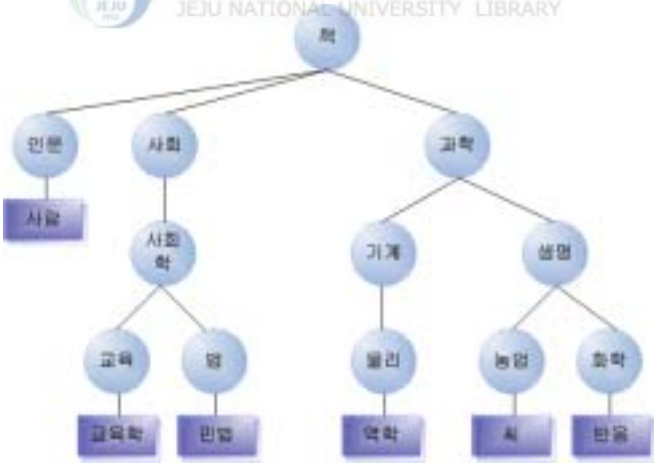


Fig. 2 Decomposition method

Fig. 2는 XML 문서를 분할 저장 방법을 이용하여 계층 구조로 표현한 것 이다. 분할 저장 방법은 엘리먼트 추가나 삭제 또는 수정 등의 연산의 발생하는 경우

관계되는 엘리먼트만을 수정하면 되므로 문서의 편집 및 관리가 쉽고 동일한 내용을 가지는 노드를 공유할 수 있다는 장점을 가지고 있지만, 문서를 검색하여 내용을 추출해야 하는 경우에는 각 엘러먼트의 내용을 조합하여 결과를 구성해야 하므로 검색 과정이 복잡하고 검색 시간이 오래 걸린다는 단점과 DTD에 의존적인 데이터 모델을 사용하므로 DTD 구조 변경이 발생했을 시 이와 관련된 모든 데이터들이 재구성 되어야 한다.

2) 가상 분할 저장 방법

트리상의 각각의 노드는 실제 문서의 논리적인 구조만을 기술하고 내용부분을 다른 영역에서 관리하여 각각의 노드가 이 영역에 대한 위치정보와 길이 등을 가지고 내용을 표시하는 가상 분할 방법이다.

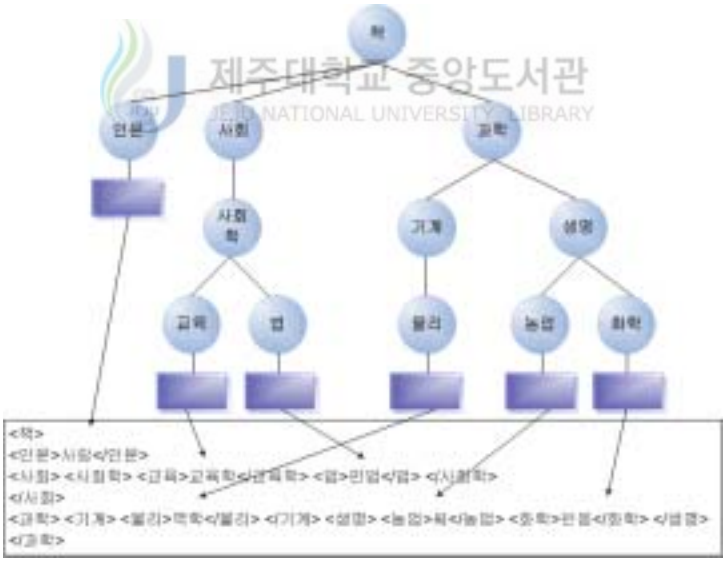


Fig. 3 Virtual fragmentation method

문서를 한꺼번에 저장하기 때문에 분할 저장 방법과 달리 통합 과정이 필요하지 않으므로, 단말 노드에 대한 검색의 경우 위치 정보를 이용한 구조적 검색이 가능하고, 검색 효율이 상대적으로 우수하다. 그러나 XML 문서의 일부부에 대하

여 엘리먼트가 추가, 삭제 되는 등의 갱신 연산이 발생하는 경우에는 문서 내의 모든 다른 엘리먼트 들의 위치 정보도 수정해야 하므로 이에 따른 오버헤드와 데이터베이스 일관성을 유지해야 하는 문제가 있다. 따라서 가상 분할 방식은 주로 검색을 위주로 한 시스템에 적합한 방식이라 할 수 있다.

Fig. 3은 가상 분할 모델을 이용하여 XML 문서를 DOM 파서를 통해 계층 구조로 표현하였고, 분할 저장과 가상 분할 저장 특징은 Table. 1에 나타내었다.

Table 1 Method comparison between decomposition and virtual fragmentation

	분할 저장 방법	가상 분할 저장 방법
저장 방식	문서를 나누어 저장	전문(Full-Text) 전체저장
검색 방법	해당 노드를 하나씩 직접 가져온다.	오프셋, 길이를 이용해 문서를 읽어 온다.
검색 시 취합여부	모든 리프 노드를 순회하여 취합	필요 없음

3) 페이징 저장 방법

위에서 언급한 두 가지 구조저장 방법의 경우에는 각각의 장·단점을 가지며 XML 문서의 내용 저장 모델로 사용하기에는 다른 특별한 메커니즘이 필요하게 된다. 페이지 기법은 가상 분할에서 문서의 Full-Text를 특정 영역에 저장하는 것이 아니라 일부분씩 나누어 페이지 단위로 저장하는 기법을 말한다. 여기서 만일 페이지의 단위가 하나의 노드만을 저장할 수 있는 단위라면 페이징 기법은 분할 모델이 된다. 즉, 페이징 기법에서 페이지의 크기가 중요한 성능 변수가 된다. 페이지는 BLOB(Binary Large OBject)나 CLOB(Character Large OBject)으로 처리해 줄 수 있지만 노드의 구조변경(노드 추가/삭제)이 아니라 노드내의 내용 변경(노드내의 문자 열 변경)이 많이 발생한다면 LOB 대신 문자열에 대한 리스트 구조로 페이지를 구성할 수 있다. 이 경우 페이지내의 특정 문자열 내용 변경 연

산은 페이지의 리스트 구조가 바뀌는 것이 아니기 때문에, 다른 노드의 변경을 최소화 할 수 있는 장점이 있다.

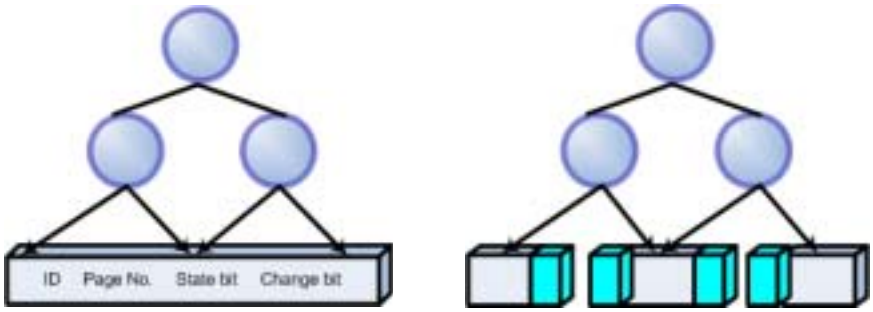


Fig. 4 Paging method

4. 구조 정보를 표현하기 위한 모델

XML로 표현된 구조화된 문서를 검색하기 위해서는 키워드에 의한 문서 단위의 내용 검색뿐만 아니라 엘리먼트를 기본 단위로 하는 구조 검색 및 애트리뷰트 검색이 지원되어야 한다. 이를 위해 구조 정보를 효율적으로 표현하기 위한 연구가 선행되어야 한다. 현재 구조 정보를 표현하기 위한 모델로는 SCL 모델, K-ary 완전 트리 모델과 ETID 모델 등이 있다.

1) SCL(Simple Concordance List) 모델

Fig. 5와 같이 구조 문서의 계층적 관계보다는 포함 관계를 이용한 표현 방법으로서 SC-List라는 데이터 타입을 통해 중첩된 정보를 허용하므로 리스트와 같은 순환 구조를 다룰 수 있다는 장점이 있다. 이 모델은 텍스트와 마크업에 대해 색인 넘버를 부여한 후, 불용어를 제외한 텍스트 어휘들을 텍스트 인덱스에 색인 넘버로 저장하고, 마크업은 시작 태그와 종료 태그의 쌍으로 마크업 인덱스에 저

장한다. 그러나 SCL 구조는 트리의 깊이를 표현 할 수 없다는 단점이 있다.

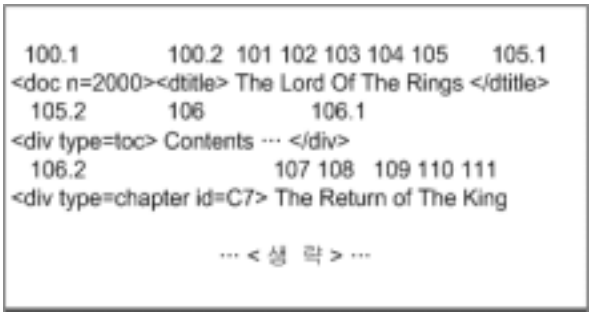
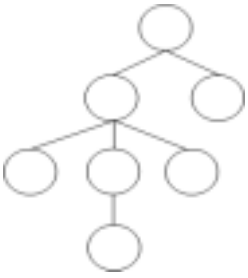


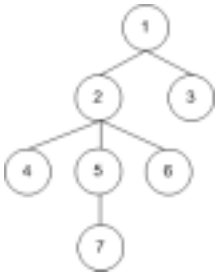
Fig. 5 Simple Concordance List model

2) K-ary 완전 트리 모델

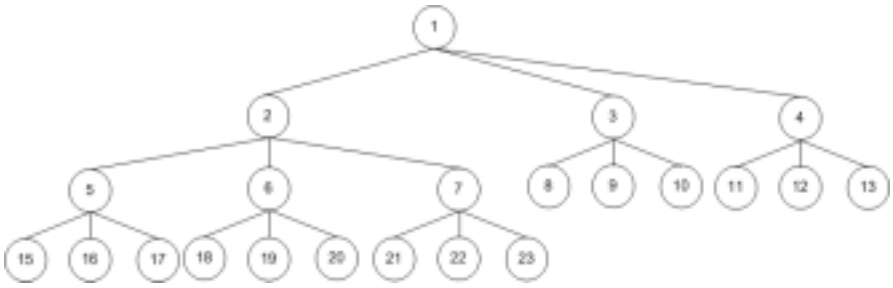
문서에 대한 트리로부터 이들 노드 중 가장 큰 차수 K를 구하여 K-ary 완전 트리로 재구성한 후, 여기에 문서 트리를 매핑하여 각 노드에 모든 번호를 부여한다. 이 모델은 문서 구조 사이의 계층 관계를 간단한 공식을 통해 쉽게 구할 수 있다는 장점이 있는 반면, 매핑 과정이 Null 노드가 많아질 수 있고 노드의 깊이가 깊어질수록 노드 변화가 커진다는 단점이 있다. Fig. 6은 K-ary 완전 트리 모델의 예이다.



(a) 파싱 트리



(c) 노드값을 부여한 파싱 트리



(b) K-ary Tree (K=3)

Fig. 6 K-ary tree model

3) ETID(Element Type ID) 모델

Fig. 7은 같은 엘리먼트들 간의 계층 정보와 동일 부모 엘리먼트를 갖는 자식 엘리먼트들의 순서 정보, 그리고 동일한 부모 엘리먼트를 갖는 자식들 중 동일한 타입의 엘리먼트들에 대한 순서 정보를 통해 구조 문서를 표현한다. 이 방법은 기준 엘리먼트로부터 특정 엘리먼트에 대한 계층 정보와 순서 정보를 간단한 문자열 조작만으로 쉽게 구할 수 있다는 장점이 있는데 반해 트리의 깊이가 깊어질수록 각 노드를 표현하기 위한 공간이 무한대로 늘어난다는 단점이 있다.

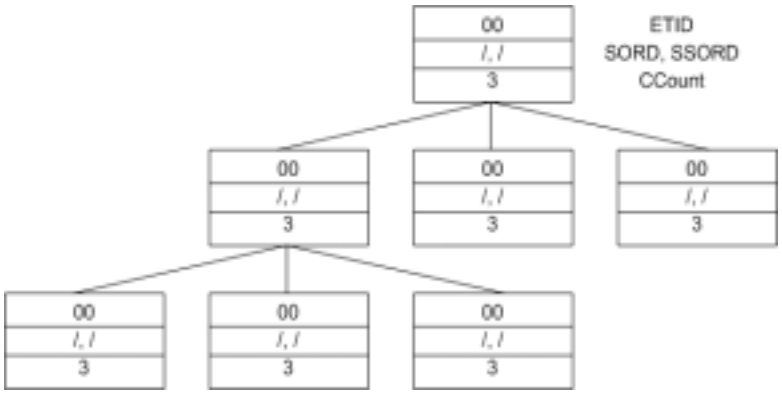


Fig. 7 Element Type ID model

5 XML 문서의 접근

일반적인 파일 구조는 파일 일부분의 접근이 복잡한 구조이기 때문에, 파일 일부분에 해당하는 권한을 부여하기가 어렵다[12]. 그렇기 때문에 접근제어가 전체 파일 단위로 이루어진다. 하지만 XML은 트리 형태의 계층 구조로 이루어져 있다. 하나의 XML 문서는 여러 엘리먼트들로 구성되어 있고, 각각의 엘리먼트는 트리의 노드에 해당하며 노드들은 XML 데이터의 일부분을 표현하고 있기 때문에 XML에서는 엘리먼트에 권한을 부여함으로써 데이터 일부분에 대한 접근이 가능하게 되었다. XML 문서를 접근하기 위해 DOM 트리를 이용하여 XML 문서와 DTD의 엘리먼트에 접근권한을 설정하고, 설정된 접근권한 정보에 의해 사용자의 XML 데이터 접근을 제어한다. 먼저 DOM트리를 얻기 위해 XML 문서를 파싱한다. 그런 후 DTD와 XML 문서 기반의 접근권한 정보를 기초하여 DOM 트리의 노드에 접근 허가(+) 혹은 거부(-)를 의미하는 sign값을 설정하게 된다. 이렇게 DOM 트리의 노드에 접근권한을 설정하는 작업을 레이블링(Labeling)이라 한다. 최종적으로 레이블링된 DOM 트리에서 sign 값이 (-)로 설정된 노드는 제거되고, (+)로 설정된 노드만을 XML 형식으로 사용자에게 보여준다. DOM 트리에서 노드가 제거된 XML 문서는 DTD에 유효하지 않을 수 있는 문제를 해결하기 위해 DOM 트리에서 노드가 제거되더라도 기존의 DTD를 유지할 수 있게 하는 루이닝(Loosening) 기법을 제안하고 있다. 그러나 이 방법은 구조 정보를 상실하게 되어 의미적 충돌(Semantic Conflict) 문제를 초래한다.

문서의 구조 정보를 유지하기 위해 트리 레이블링 기법을 사용하는데 이 방법의 문제는 거부(-) 레이블링이 된 구조 정보와 데이터의 존재를 보여주게 되어 기밀성에 위배되는 문제점을 앉고 있다. 거부(-) 레이블링이 된 노드에 대한 접근을 금지하는 강력한 레이블링 기법을 적용하고 있다. 이 방법은 금지된 노드의 하부 노드 역시 접근을 금지하게 되는데 데이터의 이용성에 한계가 있다[8][9].

위에서 언급한 연구들은 전체의 DOM 트리가 메모리에 적재되어야 하고, DOM 트리의 모든 노드에 접근권한을 설정하기 위한 반복적인 트리의 검색으로

많은 메모리가 사용되므로, 시스템의 성능 저하를 초래할 수 있는 문제점이 있다. 또한, 판독 연산만 지원하고 있기 때문에, 갱신 연산으로 인해 발생하는 무결성 문제는 고려되지 않고 있다. 많은 응용은 잘 정형화된 XML보다는 스키마 기반의 유효한 XML 데이터를 이용하여 판독뿐만 아니라 갱신 연산을 요구하고 있다. 특히, XML 데이터를 이용한 응용은 그림 8과 같은 환경이다. 하나의 스키마에 하나 이상의 인스턴스가 있으면서 그 인스턴스를 이용한 여러 응용이 얽혀 있는 환경이다. Fig. 8과 같은 환경에서 이용되는 XML 데이터에서 갱신 연산을 지원하기 위해서는 다음과 같은 XML 문서만이 갖고 있는 고유한 특성이 있다.

사용자의 갱신 질의에 의해 XML 문서의 내용과 구조의 변경이 가능하게 되고 이것은 곧 스키마의 변경이 가능함을 의미한다[14]. 결국 갱신 연산에 의한 인스턴스의 변경은 스키마와의 불일치를 야기하고 기존 스카마와 관련된 다른 인스턴스와의 불일치 혹은 그 인스턴스를 사용하고 있는 다른 응용과의 불일치를 초래하는 문제점을 갖게 된다.

Fig. 8 Use of XML Data Application environment

6. XML 데이터에 대한 갱신 연산

XML 문서에 대한 갱신 질의에 대한 연구는 그리 활발하지 못하다. eXcelon에서는 엘리먼트, 애트리뷰트, 텍스트, CDATA, 주석(Comment)을 추가 또는 삭제

할 수 있으며, 속성의 이름을 바꿀 수 있다. 그렇지만, 엘리먼트의 재명명은 되지 않는다. 하지만, 엘리먼트 순서를 변경할 수 있으며, 순서화된 XML 문서인 경우, 엘리먼트 추가 시 위치 선정-after, before-과 하부 엘리먼트(Sub-Element) 추가 시 위치 선정-Firstchild, lastchild-을 지원한다[8]. PCDATA, 속성, IDREF, 엘리먼트에 대한 삽입 연산, 속성이나 PCDATA 값이 포함된 엘리먼트에 대한 대체 연산, PCDATA 값이 포함된 엘리먼트에 대한 대체 연산, non-PCDATA나 IDREF에 재명명(ReName) 연산을 지원한다.

위에서 언급한 갱신 연산은 잘 정형화된(Well-Formed) XML에 한정되어 있다. 스키마(DTD or XML Schema)를 수반하고 있는 유효한(Valid) XML 문서에 대해서는 고려하고 있지 않다, 결국, 유효한 XML 문서에서의 갱신은 갱신된 XML 문서와 그 문서가 갖고 있는 스키마의 검증이 필요하게 된다. 특히, 여러 가지 요인들을 인지하고 있지 않은 경우에는 XML 문서와 스키마의 불일치를 초래하게 된다.

7. DOM(Document Object Model)

XML 문서는 트리 형태의 데이터 구조를 가지고 있다. 그림 9에서와 같이 문서의 하나의 요소가 자식 요소를 가지는 트리 구조를 보여주고 있다. 이와 같은 형태의 자료 구조를 접근하기 위해서는 특별한 방법이 필요하다. XML 문서의 각 요소들을 하나의 객체로 표현하고 그 요소들 사이의 관계를 통해 각 요소들을 접근할 수 있는 방법을 제공해준다. DOM(Document Object Model) 인터페이스를 인터넷 문서의 표준을 제정하는 W3C에서는 XML 문서를 위한 메모리 모델의 표준으로 채택하고 있다. XML 문서를 파싱해서 문서의 각 요소들을 객체로 만들어 놓고 메모리상에 객체의 트리 구조를 만들어 놓게 된다. 이런 트리 구조의 객체를 접근할 수 있도록 API를 제공해 주는데 이것이 DOM API이다.

DOM은 저장소에 저장되어 있는 XML 문서들을 파싱해서 그 결과로 생성되는 객체들을 메모리상에 상주시킨다. 다양한 응용 프로그램들은 메모리상에 있는 객

체들을 DOM 표준 인터페이스를 통해서 참조하는데 이번 방법을 통해서 응용 프로그램들은 XML 문서를 접근하고 조작할 수 있다.

기존의 XML 저장 시스템에서 제공하는 갱신 및 유효 검증에 대해서 간단히 살펴보고자 한다. 우선 파일 시스템에 XML 문서를 저장한 경우 갱신을 위해서는 문서 전체를 메모리에서 파싱하여, 갱신 후 재 저장해야 하므로 갱신 성능 자체가 좋지 않으며 스키마를 동반한 경우 스키마에 대한 유효 검증을 위해서는 갱신 후 XML 문서 전체에 대한 유효 검증이 불가피하다. 객체지향 데이터베이스 시스템을 기반으로 한 대표적인 XML 저장 시스템인 eXcelon[3]에서는 스키마로 DTD만을 지원하고 있으며 새로운 XML 문서가 저장 될 때, 이 문서에 대한 유효 검증은 사용자 옵션으로 제공하고 있으나 갱신에 대한 유효 검증 메커니즘은 제공하고 있지 않다.

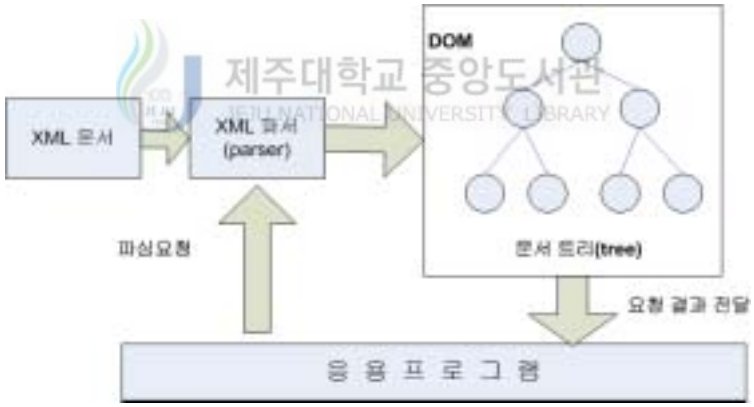


Fig. 9 Access XML document of use DOM

관계형 및 객체 관계형 데이터베이스 시스템을 기반으로 한 대표적인 XML 저장 시스템인 Oracle XML 데이터베이스[4]에서는 XML Schema로부터 관계형 스키마를 생성하여 데이터를 저장하기 때문에 새로운 분서를 저장할 때나 삽입 갱신이 발생 했을 때에는 XML Schema에 정의되지 않은 엘리먼트들이나 속성들을

검출해내는 등의 간단한 유효 검증은 제공하지만 XML Schema에 정의된 모든 제약 조건들을 체크하기 위해서는 갱신 후 문서 전체에 대하여 유효 검증을 수행해야 하며 이는 굉장한 오버헤드를 초래한다.

Ⅲ. 저장 구조 모델링

1. 제안된 저장구조 모델

1) 기존 저장구조 모델

Fig. 10은 기존 저장구조 모델 구성도이다. 데이터베이스 스키마 생성기는 XML 문서로부터 객체 데이터베이스 스키마를 생성하는 모듈이다. 그리고 XML 저장관리자는 XML 데이터를 분할하여 객체 데이터베이스 클래스에 저장 또는 구조정보 추출기에 의해 인덱싱 되어진 XML 문서에 대한 위치정보를 XML 인스턴스 관리기에 전달한다. 인스턴스 저장기는 위치정보에 대한 값을 저장하는 곳이다. 마지막으로 본 연구에서 중요하게 다룰 내용이 구조정보 추출기이다.

기존 구조정보 추출기는 XML 저장 관리자가 생성한 스키마를 가지고 문서를 엘리먼트, 속성, 컨텐츠로 나누어 위치정보의 값을 생성한다.

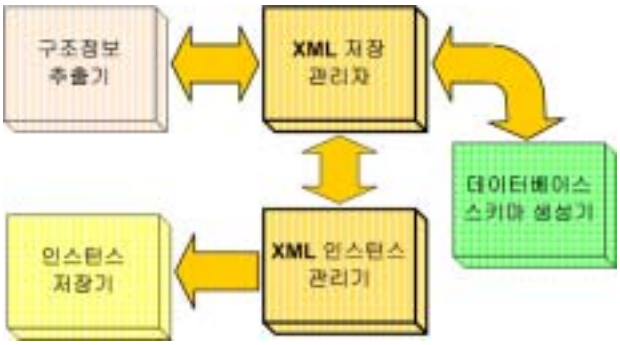


Fig. 10 Existing storage structure model

2) 제안 저장구조 모델

본 연구는 가상 분할 모델이 가지는 XML 문서에 대한 효율적 검색에 대한 장점과 분할 모델이 가지는 문서 변경 시 변경된 부분만 동적으로 반영할 수 있다는 장점을 이용하여 시스템을 설계한다.

Fig. 11는 본 연구에서 제안하는 전체적인 모델링 구조도이다. XML 문서가 들어오면 구조정보 추출기를 호출하여 XML 문서가 정형화 되었는가를 분석을 한 후, 문서를 순차적으로 읽어 내려가면서 요소들의 연관 관계를 추출한다. 구조정보 추출기를 통해 분류되어진 요소들은 매핑 테이블로 옮겨지고 데이터베이스 스키마를 생성한다. 생성된 스키마는 인덱스 관리자로 이동하고 각각의 엘리먼트, 속성, 컨텐츠에 인덱스를 부여한다. 요소들과 인덱스들은 데이터베이스에 저장이 된다. 구조 정보를 검색하기 위해서는 사용자 인터페이스 통해 질의를 하고, 질의 처리기는 사용자가 질의한 키워드를 기반으로 구조적인 정보를 수집하여 원하는 구조 정보를 검색하여 사용자에게 다시 전달을 한다.

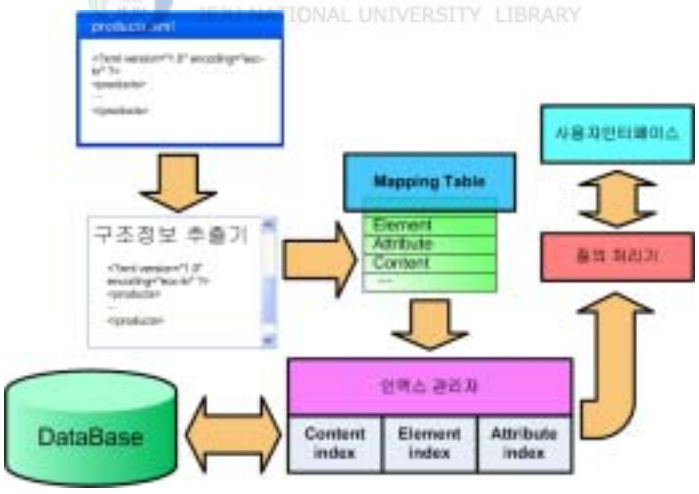


Fig. 11 Proposal storage structure model

2. XML 문서 구조의 그룹화

정형화된 XML 문서는 내용정보와 구조정보를 동시에 포함하고 있다. XML 문서의 구성은 세 부분으로 나누어진다. 첫 부분은 XML 선언부이다. 두 번째 부분은 DTD/XML 스키마, 네임스페이스 선언이며, 이 부분은 생략이 가능하다. 그리고 마지막 부분은 몸체를 이루는 XML 태그와 컨텐츠로 구성된다.

Fig. 12은 구조정보를 표현하기 위한 예제 XML 문서이며, 이 문서를 크게 두 부분으로 나누었다. XML 선언부와 몸체이다. ①번은 XML 선언부이고 ②번은 몸체부분이다. ③번부터 ⑧까지는 각각의 엘리먼트 단위로 나누었다.

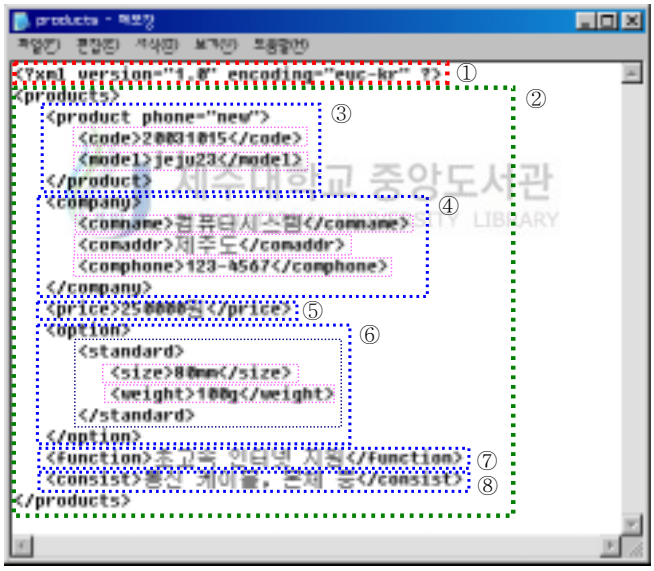


Fig. 12 XML document of Example

이렇게 나누어진 정보들은 제안 모델 구성도 Fig. 11에서 보았던 구조정보 추출기를 통해서 각 엘리먼트로 분리되어 매핑 테이블로 이동을 한다. Fig. 13는 구조정보 추출기를 통해 얻어진 정보들을 이용하여 DOM으로 파싱하고 얻어진 결과물을 계층 구조의 형태로 표현한 것이다. 이러한 분석 후 데이터베이스에 저장을

하기 위한 스키마 사전 정보 테이블을 생성하고, 생성된 인덱스 정보들을 저장을 한다.

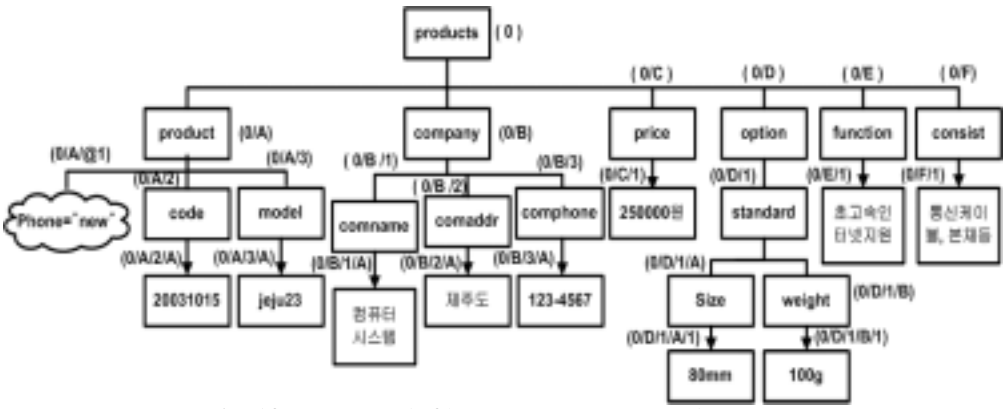


Fig. 13 Expressed Class structure XML document

3. ID 부여 방식

기존의 구조정보를 표현하기 위한 모델들로 II 장에서 살펴보았던 SCL 모델과 K-ary 모델이 있다. SCL 모델은 계층적 관계보다는 포함관계를 이용한 방법으로 이 모델이 단점은 XML 문서의 단계(Depth)를 표현할 수 없다는 것이고, K-ary 모델은 노드 중 가장 큰 차수를 구하여 트리의 형태로 재구성하는 방법으로 재구성 과정에서 노드의 Null 값이 많아지고 깊이가 깊어질수록 노드의 변화가 커진다는 단점이 있다. 본 연구는 SCL 모델과 K-ary 모델의 단점을 보안한 ID 부여 방식을 제안한다.

1) 엘리먼트 ID 부여

XML 문서의 구조정보를 표현하기 위해 본 연구에서 사용하는 구조정보는 엘리먼트 이름을 식별할 수 있는 ID, 부모와 자식 엘리먼트간의 계층정보, 동일한 부모 엘리먼트를 갖는 자식 엘리먼트들의 순서 정보가 있다.

먼저, EID(Element ID)는 각 엘리먼트에 대한 고유 식별값을 나타낸다. EID 부여 방식은 2 바이트를 사용하여 표현하는데 각 바이트는 숫자 '0'으로 시작을 하

여 대문자 'A'에서 소문자 'z'까지 순으로 문자를 사용하여 ASCII 코드의 순서를 따르고 있다. 그리고 0 단계(Root)는 숫자 '0'으로 시작으로 하여, 1 단계(Depth)는 영문으로, 2 단계는 숫자, 3 단계는 다시 영문으로 부여하는 반복(Rotation)패턴의 방식이다. 반복패턴을 사용하였기 때문에 별도의 단계(Depth) 데이터를 저장하지 않아도 되기 때문에 SCL 모델의 단점을 보안하였다. 그리고 자식 엘리먼트 ID는 부모 엘리먼트 ID와 자신의 위치 정보 값을 조합하여 생성한다.

2) 속성 ID 부여

속성 ID 값은 속성과 속성의 값으로 나누어서 저장을 한다. 속성 ID의 부여는 그에 해당하는 엘리먼트 ID값을 자신의 위치 정보와 조합하여 생성하고, 속성 값은 속성 ID에 '@'와 자신의 위치 값을 조합하여 생성한다.

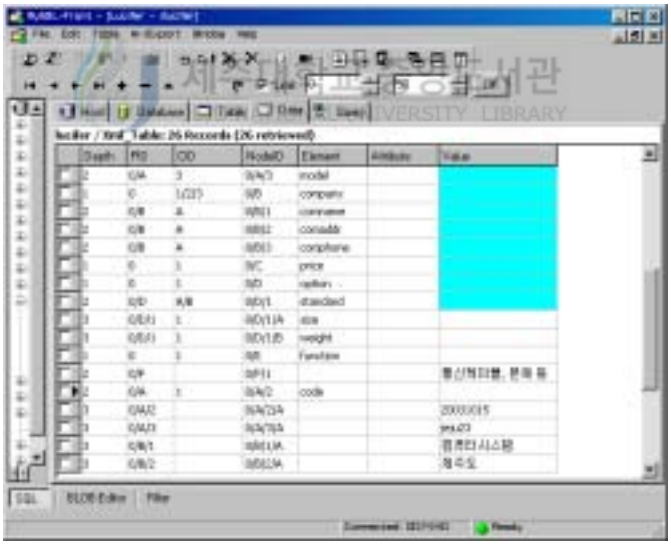


Fig. 14 Before information table

3) ID 조합의 예

XML 문서에서는 동일한 엘리먼트들이 반복적으로 나타날 수 있는데 반복적인 엘리먼트들과 엘리먼트들 간의 순서정보를 표현하기 위해 부모 엘리먼트 ID(PID)

와 자식 엘리먼트 ID(CID)를 사용한다. 예를들면, 〈company〉 엘리먼트는 깊이가 2 단계이고, 3개의 자식노드를 가지고 있다. 부모노드(Root) 위치정보 '0'과 자신의 위치정보를 조합한 '0/B'로 나타낸다. 그림 14는 ID 조합을 통해 얻어진 결과를 데이터베이스에 저장한 결과이다.

4. 정의된 테이블 스키마 구조

XML 문서를 분석 후 생성된 사전 정보 테이블정보는 파일의 타입를 저장하는 테이블(FT), 문서를 저장하는 테이블(DT), 엘리먼트의 목록을 저장하는 테이블(ELT), 엘리먼트의 위치를 저장하는 테이블(NT), 엘리먼트를 저장하는 엘리먼트 테이블(ET), 속성을 저장하는 테이블(AT), 문서의 내용을 저장하는 테이블(CT)로 이동되며 이를 저장하기 위한 정의된 테이블 스키마 구조는 다음 Fig. 15와 같다.

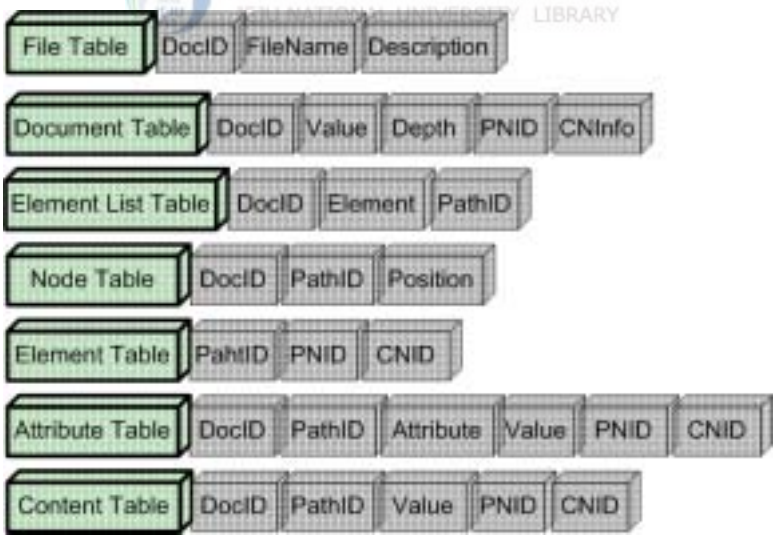


Fig. 15 Database schema structure

5. 저장구조 지원을 위한 프로그램 구현

1) 정형화(Well-Form) 프로그램

Fig. 16는 정형화된 XML 문서를 체크를 하기 위한 프로그램이다.

WellForm 알고리즘

```
public void checkWellFormedness () {
  try {
            parser.setFeature (
            "http://xml.org/sax/features/validation", false);
            parser.setFeature (
            "http://apache.org/xml/features/validation/schema", false);
            parser.parse(new InputSource(new
            StringReader(editor.getText())));
            out.setText("잘 정의된 XML 문서입니다.");
            }
  catch(Exception e) {
            out.setText("잘 정의되지 않은 문서입니다\n");
            out.append(e.toString());
            }
}
```

Fig. 16 Well-Form program

2) 공백문자 제거 알고리즘

Fig. 17은 DOMParser 객체를 생성하고 XML 문서에서 공백문자를 제거하는 알고리즘이다. XML 파일을 파싱하고, Document 노드를 루트노드로 지정한다.

공백 제거 알고리즘

```
public DOMTreeModel (String file) {
  this((Node)null);
  try {
            DOMParser parser = new DOMParser();
            parser.setFeature(
            "http://apache.org/xml/features/dom/include-ignorable-
            whitespace", false);
            parser.parse(file);
            Document doc = parser.getDocument();
            setRootNode(doc);
      }
  catch(Exception e) { }
 }
```

Fig. 17 Space letter of clearing program

3) 트리생성 알고리즘

Fig. 18은 자식 노드들을 for 문을 이용해서 트리로 추가하고 자식 노드들의 다시 재귀적으로 트리를 만들도록 하는 알고리즘이다.

트리생성 알고리즘

```
private void buildTree (Node node, MutableTreeNode where ) {
  if (node == null) {
          return;
  }
          MutableTreeNode treeNode  = insertNode (node, where);
          NodeList nodes  = node .getChildNodes ();
          int len = (nodes  != null) ? nodes .getLength () : 0;
          for (int i = 0; i < len; i++) {
                  Node child = nodes .item (i);
                  buildTree (child, treeNode );
          }
}
```

Fig. 18 Creating of tree program

4) 속성값과 자식원소 출력

Fig. 19는 속성값과 자식 원소를 출력하기 위한 알고리즘이다. XML 문서에 대한 노드에 타입을 얻고, 문자열을 출력한다.

5) 속성값 배열 알고리즘

Fig. 20는 NamedNodeMap에 포함된 속성들을 Attr의 배열 형태로 리턴한다.

속성 리턴 알고리즘

```
public void print(Node node) {
  int type = node.getNodeType();
    switch(type) {
          case Node.DOCUMENT_NODE:
          System.out.println("DOC: "+node.getNodeName());
          Document d = (Document)node;
          print(d.getDocumentElement());
    break;
    case Node.TEXT_NODE:
      if( n == 0) {
          System.out.println("TEXT: "+node.getNodeValue());
          }
    break;
    case Node.ELEMENT_NODE:
          System.out.println("ELEMENT: "+node.getNodeName());
          // 속성 값 출력하기
          NamedNodeMap attrs = node.getAttributes();
            int len = attrs.getLength();
            for(int i = 0; i < len; i++) {
                print(attrs.item(i));
            }

          // 자식 원소 출력하기
          NodeList children = node.getChildNodes();
            if(children != null) {
                    int n = children.getLength();
                    for(int j = 0 ; j < n ; j++) {
                            print(children.item(j));
                    }
            }
    break;
```

Fig. 19 Print Attribute value and Element

속성값 리턴 알고리즘

```
protected Attr[] sortAttributes(NamedNodeMap attrs ) {
  int len = (attrs != null) ? attrs.getLength() : 0;
  Attr array[] = new Attr[len];
  for (int i = 0; i < len; i++) {
          array[i] = (Attr)attrs.item(i);
  }
  for (int i = 0; i < len - 1; i++) {
          String name  = array[i].getNodeName();
          int index = i;
          for (int j = i + 1; j < len; j++) {
                  String curName = array[j].getNodeName();
                  if (curName.compareTo(name) < 0) {
                          name  = curName;
                          index = j;
                  }
          }
          if (index != i) {
                  Attr temp = array[i];
                  array[i] = array[index];
                  array[index] = temp;
                  }
          }
          return array;
}
```

Fig. 20 Array Attribute value program

Ⅳ. 구현 결과 및 검증

1. 정형화(Well-Form)된 XML 문서 알고리즘 적용 결과

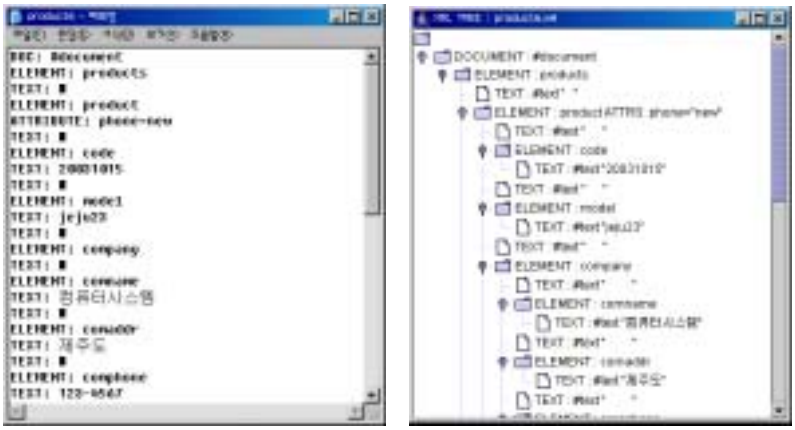
그림 21 (a)는 예제 XML 문서이고, 그림 21 (b)는 정형화 알고리즘을 이용한 XML 문서가 정형화 되었는가를 체크한 결과이다.

(a) (b)

Fig. 21 Well-Form XML document

2. DOM을 이용한 계층 구조

Fig. 22 (a)는 예제 XML 문서에서 공백 문자 제거 알고리즘을 이용하여, DOM 파서에 의해 처리된 것을 products.txt 파일로 저장을 했고, 그 결과를 나타낸 것이다. 그림 22 (b)는 DOM을 이용한 트리형태의 결과이다.



(a) (b)

Fig. 22 Class structure

3. 엘리먼트 추가, 삭제, 갱신 처리

1) 엘리먼트 추가, 삭제, 갱신

질의문을 처리하기 위해, 먼저 ELT(Element List Table)에서 product 엘리먼트의 PathID의 값을 알아보고 ET(Element Table)에서 PNID의 값을 가져온다. 그리고 number의 엘리먼트 값을 생성하고 product의 값과 number의 값을 결합하여 엘리먼트 ID를 부여하고, number 엘리먼트를 추가한다.

Table 2 Add element

- 질 의 문 : /product//, Add → number. - 질의 의미 : product 엘리먼트 자식 노드를 추가.

2) 엘리먼트 삭제

질의문을 처리하기 위해, 엘리먼트 추가와 동일한 방법으로 PathID 값이 찾아온다. 그

리고 NT(Node Table), ET, CT(Content Table) 테이블의 PathID 값을 삭제하고, PNID와 CNID값을 조합하여 DT(Document Table)에 있는 PNID의 값과 동일 값을 찾아서 삭제를 하면서 마친다.

Table 3 Delete element

- 질 의 문 : /product//number, Del → number. - 질의 의미 : product의 자식 엘리먼트 number 삭제.

3) 엘리먼트 갱신

질의문을 처리하기 위해, 추가 처리에서 보았던 방식으로 CT에 있는 테이블을 참조한 PathID의 값을 갱신하고 NT 테이블에서 Position 값을 찾아서 갱신을 하면서 마친다.

Table 4 Updating element

- 질 의 문 : /product//number, Renew → No.FDD75. - 질의 의미 : number 값을 No.FDD75로 갱신.

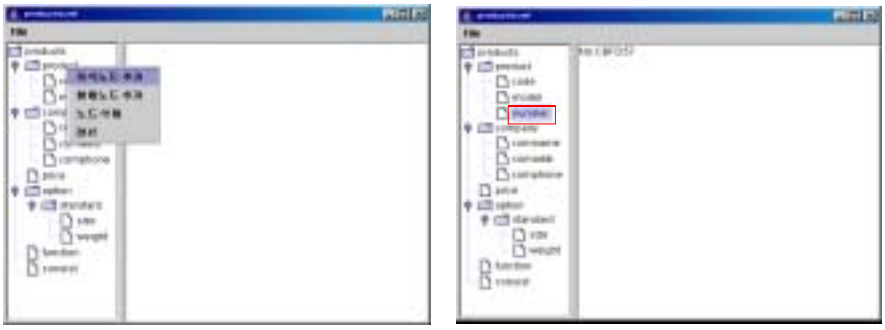
4) 엘리먼트 추가 실행 결과

Fig. 23 (a)는 〈product〉엘리먼트에 자식 엘리먼트인 〈number〉를 추가하고, 엘리먼트 값을 No. CBFD57를 입력하고 문서를 저장을 하였다. Fig. 23 (b)는 엘리먼트 추가 실행 결과를 보여주는 것이다.

5) 엘리먼트 삭제 실행 결과

Fig. 24 (a)는 product. xml 문서에서 〈number〉엘리먼트를 삭제 명령을 실행하는 모습이

다. 그리고 Fig. 24 (b)는 노드를 삭제 하고 보여지는 실행 결과이다.



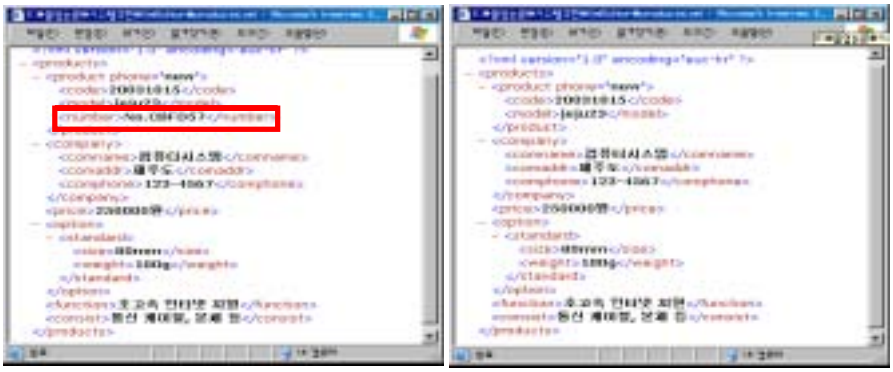
(a) (b)

Fig. 23 Add element

(a) (b)

Fig. 24 Delete element

Fig. 25 (a)는 XML 문서에서 〈number〉엘리먼트 추가를 하고 웹 브라우저에서 실행결과를 나타낸 것 이고, Fig. 25 (b)는 〈number〉엘리먼트 삭제를 웹 브라우저에서 실행한 결과를 나타내었다.

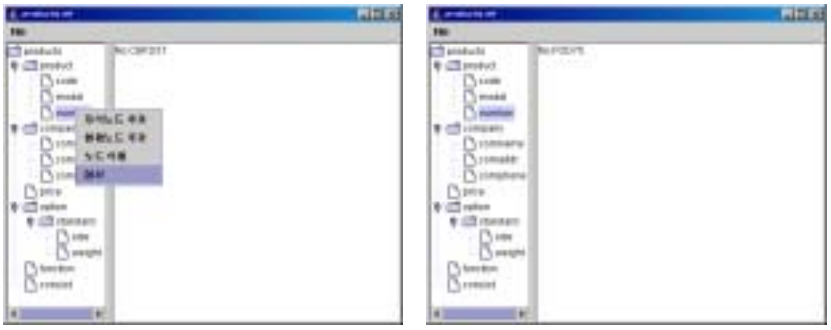


(a) 엘리먼트 추가 (b) 엘리먼트 삭제

Fig. 25 Element add and delete

6) 엘리먼트 갱신 실행 결과

Fig. 26 (a)는 product.xml 문서에서 〈number〉엘리먼트의 값 No.CVFD57을 No.FDD75의 값으로 갱신 명령을 실행하는 모습이다. 그리고 그림은 26 (b)는 〈number〉엘리먼트를 갱신 하고 나서 보여지는 실행 결과이다.



(a) (b)

Fig. 26 Updating element

4. 성능 평가

본 장에서는 효율적인 XML 문서를 위해 기존 모델인 분할 모델과 가상 분할 저장 모델

을 기준으로 평가한다. 비교를 위한 항목으로는 XML 문서의 저장 시간과 검색 시간을 두었다.

Fig. 27은 문서의 크기와 저장시간에 대한 성능평가 그래프이다. X축 변수는 문서의 크기(K)를 나타내고 Y축 변수는 저장시간(Sec)나타낸다. X축 변수 문서크기는 10K를 처음으로 하여 5의 배수 만큼 증가하고 Y축의 단위 저장시간은 소수점 2째 자리까지 나타내었다. 처음 문서의 크기가 작은 10K와 50K를 저장하는데 걸리는 시간은 가상분할과 분할 그리고 제안시스템 모두가 거의 일정하다고 생각이 된다. 하지만 문서의 크기 200K가 되면 조금씩의 성능차이를 그래프를 통하여 알 수가 있다.

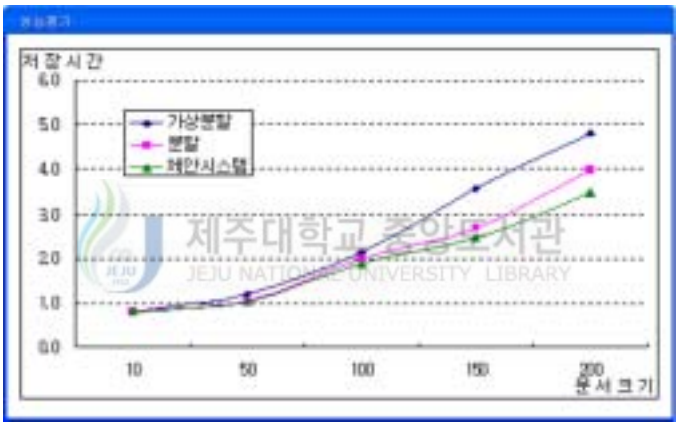


Fig. 27 Capability rating for document size

Fig. 28은 노드길이와 검색시간에 대한 성능평가 그래프이다. X축 변수는 노드길이를 나타내고 Y축 변수는 검색시간(Sec)를 나타낸다. X축 변수는 엘리먼트에 대한 자식 엘리먼트에 대한 수를 말하며, 10을 시작하여 5의 배수 만큼 증가를 한다. Y축 변수는 그림 26의 저장시간과 동일하다. 노드의 길이가 10일 때는 가상분할과 분할모델은 거의 동일하지만, 제안 시스템은 약간의 성능차이를 보이고 있다. 그리고 노드의 길이가 켜질수록 성능차이가 크다는 것을 그래프를 보고 알 수 있다.

Table 2와 Table 3은 문서크기와 노드길이에 대한 성능평가를 나타낸 것이다. 동일한 문서조건을 10회 측정 후 평균값을 표로 나타내었다.

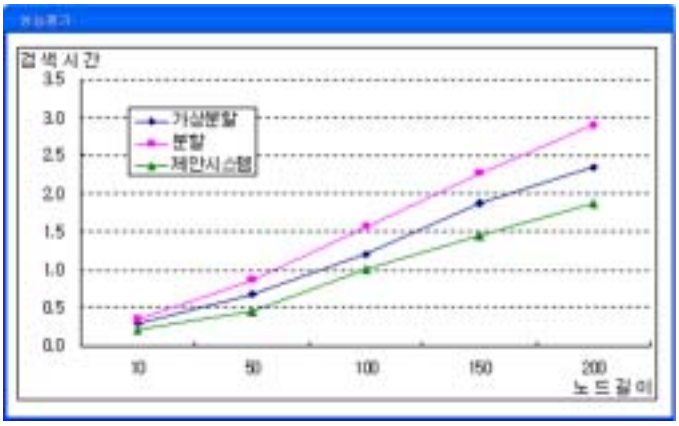


Fig. 28 Capability rating for node length

Table 5 Capability rating for document size

	10	50	100	150	200
가상분할	0.79	1.18	2.12	3.56	4.81
분할	0.78	1.05	1.98	2.69	3.99
제안시스템	0.77	1.01	1.88	2.45	3.49

Table 6 Capability rating for node length

	10	50	100	150	200
가상분할	0.29	0.69	1.21	1.87	2.35
분할	0.35	0.86	1.56	2.26	2.89
제안시스템	0.21	0.45	1.01	1.45	1.87

Ⅴ. 결론 및 향후 연구

본 연구는 XML 문서를 효율적으로 검색하고 변경을 지원하는 저장 구조를 위해 가상 분할의 가지는 문서 검색에 대한 효율성과 분할 모델이 장점인 동적인 문서 변경을 수용하여 본 시스템을 제안하였다. 이를 위하여 저장될 XML 문서를 구조정보 추출기를 통해 엘리먼트, 속성과 컨텐츠로 나누어 구조정보를 매핑 테이블로 보내고, 매핑 테이블은 각 엘리먼트, 속성과 컨텐츠를 분석하였다. 분석되어진 값들은 인덱스 관리자에 ID를 부여한다. 그리고 인덱스 관리자에 의해 부여된 ID 값들은 7개의 테이블 스키마에 저장하였으며, 또한 사전 테이블을 생성하여 엘리먼트의 부모-자식 간의 상속 ID를 부여토록 하였다.

그 결과 엘리먼트, 속성, 구조, 내용 색인 및 추가 및 갱신이 가능함을 보였으며, 특히 엘리먼트 추가나 삭제 시 위치 정보 변경 발생을 피하기 위하여 제안된 부모-자식간의 엘리먼트의 ID 상속 값을 사용하여 위치 정보 변경 없이 갱신이 가능함을 보였다.

향후 연구 과제로는 ID를 가변적으로 부여함으로 인해 기존에 부여되었던 ID는 재사용이 불가능하다. 따라서, 이를 재사용 측면을 고려한 유동적인 부여 방식에 대한 추가 연구가 필요하다.

〔참고문헌〕

[1] Tim Bray, Jean Paoli et. al (ed.). Extensible Markup Language(XML) 1.0(Third Edition), World Wide Web Consortium(W3C), 2004. (http://www.w3c.org/TR/2004/REC-xml-20040204)

[2] 연제원, 조정수, 이강찬, 이규철, "XML 문서 구조검색을 위한 저장 시스템 설계",『한국정보과학회 학술 발표논문집(B)』, 제26권 제1호, 1999, pp. 3-5.

[3] D. W. Shin, "BUS:An Effective Indexing and Retrieval Schema in Structured Documents", in Proc. Digitial Libraries, 1998.

[4] Brian Lowe, Justin Zobel, Ron Sacks-Davis, "A Formal model for Databases of Structured Text", Proceedings of the Fourth International Conference on Database Systems for Advanced Applications(DASFAA '95), 1995, pp. 449-456.

[5] Toung Dao, "An Indexing Model for Structured Document to Support Querues on Content, Structured and Attributes", Proceedings of ADL '98, 1998, pp. 88-97.

[6] 박종관 외, "XML 문서의 효율적인 구조 검색을 위한 색인 모델",『한국정보과학회 논문지』, 제 8-D권, 2001, pp451-460.

[7] 이규철 외 4명, "효율적인 XML 문서 변경 및 검색을 위한 페이징 기법",『정보과학회 논문지 학술발표논문집(Ⅰ)』, 제26권 1호, 1999, pp3-5.

[8] Daniela Florescu and Donald Kossmann, "Storing and Querying XML Data using an RDBMS", IEEE Data Engineering Bulletin, 22(3), PP 27-34, 1999

[9] E. Damiani, S. Vimercati, S. Paraboachk and p. Samarati, "XML Access Control System: A Component-Based Approach", In Proc. IFIP WG11.3 Working Conference on Database security, The Netherlands, 2000. 8

[10] 정태선 외 3명 "XML 데이터를 위한 객체지향 데이터베이스 스키마 및 질의 처리", 한국정보과학회 논문지 : 데이터베이스, 29(2), 2002

[11] 윤정혜 외 2명 "XML 문서에 대한 효율적인 검색기법", 『한국정보과학회 논문집』, 제 11권 제 1호, 2004, pp11-14

[12] 김성완 외 3명 "XML 문서에서의 엘리먼트 타입을 이용한 구조적 검색 기법의 설계" 한국정보과학회, Vol.30, no.1, pp584-586, 2003

[13] S-Y. Chien, V.J Tsotras, and C. Zaniolo, "Version Management of XML Documents", WebDB 2000 Workshop, Dallas. TX, 2000

[14] Albrecht Schmidt, Florian Waas, Martin Kersten, Michael J. Carey, Ioana Manolescu and Ralph Busse, "Xmark : A Benchmark for XML Data Management", Proc. VLDB, Hong Kong, China, 2002

[15] 민영수 외 5명, "XML 문서를 위한 구조정보 추출기의 설계 및 구현", 한국정보과학회, '99 가을 학술발표논문집(Ⅰ), pp, 81-83, 1999

[16] A. Gabillon and E. Bruno, "Regulation Access to XML Documents", In Proc. IFIP WG11.3 Working Conference on Database Security, 2001

감사의 글

정신없이 달려왔던 대학원 생활을 되돌아보면, 2년이라는 시간이 그리 길지도 짧지도 않았던 것 같습니다. 늘 부족했던 저를 아들처럼 보살핌과 가르침을 주셨던 곽호영 교수님이 있었기에 제가 무사히 졸업을 할 수가 있었던 것 같습니다. 먼저, 지도 교수님이신 곽호영 교수님께 감사하다는 말부터 시작을 할까합니다. 그리고 논문의 완성되기까지 심사와 지도를 해 주신 김장형 교수님, 변상용 교수님, 이상준 교수님, 안기중 교수님, 송왕철 교수님, 변영철 교수님, 김도현 교수님께 감사의 글을 통해 고마움을 전하고 싶습니다.

저를 낳아주시고 지금까지 키워 주신 부모님과 언제나 잘해 주지는 못했지만 형을 믿고 따라준 동생들 그리고 언제나 아들처럼 생각을 해 주시는 장인어른과 장모님 그리고 처형과 형님, 처제들에게도 감사하다는 말을 하고 싶습니다. 제가 힘들 때 마다 격려와 응원을 아끼지 않았던 나의 반려자이자 가장 소중한 보물 1호 부인 김미숙에게 고맙다는 말과 이 논문을 바칩니다.

항상 연구실에 맏형처럼 든든하신 김정희 선배님과 한경복 선배님, 동생같은 동기 홍석건, 후배 권훈, 막내둥이 강봉남 그리고 대학원 생활을 하면서 조언을 많이 해 주신 김영민 선배님, 이정하 선배님, 정은경 선배님, 친형 같은 송재경 선배, 전철승 선배, 송창훈 선배, 멋쟁이 후배인 이경진, 이정윤, 김성수 동생들에게도 이 면을 통해 감사하다는 말을 합니다.

항상 파이팅을 외쳐준 고마운 선배와 나의 베스트 프렌드의 이름을 부르면서 끝내려고 합니다. 고광현 선배, 조형준 선배, 김태언, 김태윤, 김민석, 김승경, 김철환, 한정용 정말 감사합니다.

마지막으로 나의 아내에게 사랑한다는 말을 전하고 감사의 글을 마칩니다.

2004年 12月 아쉬움과 또 다른 힘찬 도약을 하기위해서 ...