XML 문서 저장관리 시스템을 위한 효율적인 버전닝 기법

An Efficient Versioning Method for XML Document Repository System

손 충 범* 배 양 석** 유 재 수***
Chung-Beom Son Yang-Suk Bae Jae-Soo Yoo

요 약

XML 문서 저장관리 시스템은 손실없이 XML 문서를 저장, 수정하고 관리하는 수직과 수평 버전들을 관리할 수 있어야 한다. 그러나 대부분의 기존 XML 문서 저장관리 시스템들은 버전닝 기법을 지원하지 않고 있다. 일부 버전닝을 지원하는 시스템들은 XML 문서들의 수직적인 버전들만을 관리한다. 수직 버전닝은 문서의 변경 이력만을 유지하는 반면에, 수평 버전닝은 한 문서를 여러 개의 버전들로 분기하게 함으로써 사용자들은 보다 쉽게 원본 문서로부터 새로운 문서들을 생성할 수 있고 다른 의미를 갖는 문서로 편집할 수 있다. 이 논문에서는 수직 및 수평 버전닝을 효과적으로 지원하기 위한 새로운 버전 번호 부여 방법을 제안한다. 또한, 버전닝을 지원하며 XML 문서의 특징인 정보 구조화의 패러다임을 유지하는 스키마를 설계한다.

Abstract

XML document repository system(XDRS) should be able to manage vertical and horizontal versions of documents to store, update and manage XML documents without loss of information. However, most of existing XDRSs do not support a versioning method. Although a few systems support versioning method, they only manage vertical versions of XML documents. While the vertical versioning preserves the update history of documents, the horizontal versioning branches a document to many other versions of documents so that users can easily create new documents from the original version and edit them to have different meanings. In this paper, we propose a new version numbering scheme to support both vertical and horizontal versioning efficiently. We also design a schema that supports versioning and preserves the paradigm of structure information.

1. 서 론

XML[1]을 활용한 분야에서 적용되는 정보 시스템을 구축하기 위해서 가장 중요한 기능이 정보검색과 정보관리를 위한 저장관리 시스템(Repository System)기능이다. 이는 XML로 표현된 구조화된 정보의 특성을 반영하여 가장 적절하게 구현하고 기능을 제공하는 시스템으로 XML의 특징인 정보 구조화의 패러다임을 그대로 유지시킬 수 있는 정보검색이 가능하여야 한다. 또한 XML 문서에 표현할 수 있는 모든 정보에 대한 사용자의 요구에 부응하기 위해서는 XML 문서 내에 포함된 모든 정보를 손실 없이 저장하고 수정하며 관리할 수 있어야 한다. 즉, 일반 텍스트 문서에 비해 XML 문서가 가지고 있는 내용은 단순한 텍스트뿐만 아니라 문서의 구조정보를 이용하여 문서를 효율적으로 관리해야 된다. 또한, XML 문서에 대한 버전 관리는 특허 문서 관리, 소프트웨어 설계와 시스템 매뉴얼, 사용자 매뉴얼 같은 전자 매뉴얼의 응용과 같이 수정된 기존의 문서들이 관리되어야 하는 분야에서 절실한 기능이지만 대부분의 시스템들은 이에 적절히 대응하지 못하고 있다[2]. 특히, 문서의 버전닝을 지원하는 기존의 XML 저장

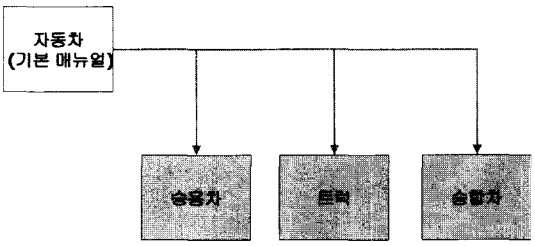
* 준 회 원 : 충북대학교 정보통신공학과 졸업(박사)
cbson@trut.chungbuk.ac.kr
** 비 회 원 : (주)인프라웍스 기술연구소
heart21c@netdb.chungbuk.ac.kr
*** 정 회 원 : 충북대학교 정보통신공학과 부교수
yjs@cbucc.chungbuk.ac.kr

관리 시스템은 단지 수직적인 버전만을 지원함으로 문서를 보다 융통성 있게 확장할 수 있는 효율적인 수평 버전닝에 대한 연구는 전무한 상태이다.

기본적인 사양을 가지는 자동차 매뉴얼의 문서에서 승용차, 트럭, 승합차 등 차종이 다른 매뉴얼들로 문서를 변경할 때, 기존의 수직 버전닝 방법으로 문서를 관리하면 각각의 매뉴얼이 복잡하게 혼합되므로 어느 하나의 매뉴얼 문서를 다시 변경하려면 해당하는 매뉴얼의 이전 버전닝 문서를 찾아서 변경을 해야하고 다시 또 다른 매뉴얼의 변경의 필요성이 생기면 마찬가지로 문서 관리가 더욱 복잡하게 된다. 그림 1처럼 수평 버전닝에 의해 문서를 보다 확장할 수 있고 융통성 있게 관리 할 수 있을 뿐만 아니라 각각의 수평 버전닝된 문서들은 버전닝 방법만으로 독립적이고 새로운 문서를 생성할 수 있다.

이에 이 논문은 새로운 버전 번호부여 방법으로 수직 버전닝 뿐만 아니라 보다 융통성 있는 확장된 버전닝을 지원하고 문서를 관리할 수 있는 수평 버전닝 방법을 제안하고 XML의 특징인 정보 구조화의 패러다임을 유지키면서 버전닝된 XML 문서를 보다 효율적으로 저장하기 위한 스키마를 설계하고 버전닝된 문서를 효율적으로 검색하기 위한 검색 알고리즘을 제안한다.

본 논문의 구성은 2절에서 XML의 개념과 문서의 버전 관리에 대한 관련 연구를 설명하고, 3절에서는 제안하는 XML 문서 버전닝 기법을 제시하고, 4절에서는 실제 제안하는 버전닝 기법을 적용한 시스템의 구현과 성능 평가를 다루고 5절에서는 결론을 기술하였다.



(그림 1) XML 문서의 확장

2. 관련연구

XML 문서를 생성하고 개발하는 동안 문서 내의 구성 요소는 변경되며, 이러한 변경의 작업으로 구성요소의 한 버전들을 다음 버전으로 변환시키는 이력 과정(history step)을 버전 관리라 한다[3,4]. 한 문서의 구성요소는 시간이 지나고 문서가 개발되는 동안 변화되고 진화(evolve)되는데, 이 변경 결과의 구성요소들을 버전이라고 한다. 버전들은 XML 문서 저장관리 시스템 환경에서 많은 목적들을 제공한다. 기본적으로 특정 문서, 또는 문서의 버전을 요구할 수 있다. 또한 버전들에 질의를 하거나 예전의 문서를 재구성할 수 있다.

한 문서의 버전 관리는 오류 수정, 사용자 요구 사항의 변경, 내용 정정, 문서의 기능 향상을 위한 기능 정정이나 다른 기능 추가, 다른 사용을 위한 분기하는 문서 발생 등의 이유로 문서를 생성하여 시간이 지남에 따라 변화하는 상황에서 발생되는 버전들을 관리한다.

문서의 버전은 수직 버전과 수평 버전으로 나눌 수 있다. 문서의 수직 버전은 문서 개발 과정에서 발생하는 작은 정정이나 일부 내용 추가나 삭제가 원인으로 전 버전들에 비해 문서가 진화되고 보다 정교해지는 특성을 가지고 있다. 문서의 수평 버전은 문서에 다른 기능을 첨가하거나 다른 사용을 위해 분기하는 문서를 제공하려 할 때 발생한다.

이렇게 생성된 다중 버전들에게 버전 번호를 부여함으로써 버전 관리를 보다 용이하게 한다[8]. 버전 번호는 버전들 사이에서 순서를 결정할 수 있고 버전 번호를 검사함으로써 한 버전이 다른 버전의 조상인지, 자손인지를 쉽게 결정할 수 있다. 또한, 두 버전들 사이에서 발생하는 변경들의 기록을 비교함으로써 변경된 차이점을 체크할 수 있다. 버전 관리를 위한 버전 번호는 루트에서 이어지는 트리로 버전 계층(version hierarchy)을 구성한다.

기존의 전통적인 문서 버전 제어 시스템으로

RCS[5], SCCS[6] 등이 있는데, 이들 버전 제어 시스템들은 텍스트 중심의 버전 관리 툴로써 텍스트 라인의 변경 사항을 변경 이력으로 버전을 유지하고 필요시 기존의 버전을 이용하여 문서의 버전닝을 하는 도구로 개발되었다. 그러나 이들은 XML 문서들의 논리적인 구조를 유지하지 못하는 중대한 단점을 가지고 있다[6,7,8].

구조화된 XML 문서를 효율적으로 데이터베이스에 저장하고 검색하기 위한 데이터 모델링 방법으로서 XML 문서 전체를 저장하는 비분할 저장 모델과 XML 문서를 엘리먼트 단위로 분할하여 저장하는 분할 저장 모델이 있다[9,10,11,12,13]. 분할 모델은 XML 문서의 실제 내용을 가지고 있는 각각의 단말 엘리먼트 안에 문서의 내용이 나뉘어 저장된다. 따라서 문서의 구조적 정보나 엘리먼트만 수정하면 되므로 문서의 버전 관리가 쉽고 동일한 내용을 갖는 노드들을 공유할 수 있다는 장점이 있다. 그러나 XML 문서 전체를 추출하고자 하는 경우, 각 단말 노드들을 순회하며 통합하는 과정에서 많은 시간이 소요하게 된다. [11]에서는 RDBMS에 XML 문서를 저장하고 검색할 수 있는 저장관리 시스템을 분할 모델을 기반으로 구현하였다. 이 시스템에서는 분할 모델을 사용하여 문서의 수정을 용이하게 지원하지만 멀티미디어에 대한 저장관리를 지원하지 못하며, XML 문서의 구조적인 특성을 고려한 다양한 구조검색에 대한 고려는 전혀 없다.

3. 제안하는 XML 문서 버전관리 기법

이 절에서는 XML 문서의 특징인 구조 정보를 반영하여 제안하는 버전닝 기법들에 대해서 기술한다. 제안하는 XML 문서 버전닝 기법들은 문서의 변경 기록들을 구별할 수 있는 버전 번호를 부여하여 해당 버전의 XML 문서를 저장관리할 수 있는 데이터 모델과 스키마를 설계하며 버전닝된 문서를 검색하는 방법에 대해서 기술한다.

3.1 버전 번호 할당 방법

제안하는 문서 버전닝 기법의 첫 번째 목적은 버전들에 번호를 할당하는 기법을 정하여 한 문서의 변경 기록들이 그것들과 관련된 버전 번호에 의해 구분하여 정렬될 수 있고 모든 변경 기록들은 버전 번호들에 의해서 변경된 데이터를 순서적 탐색에 의해 알아낼 수 있다.

(1) 버전 번호 구분

버전 번호는 릴리스 버전 번호와 변경 버전 번호로 구분된다. 릴리스 버전 번호는 원본 문서나 문서 개발 중 릴리스 된 문서에 부여하는 버전 번호의 첫 번째 필드의 번호로 이 릴리스 버전 번호만으로 릴리스 문서를 생성한다. 변경 버전 번호는 릴리스 문서가 변경이 발생하여 수직 작업과 수평 작업의 버전닝을 통해 제안하는 방법으로 할당된 버전 번호로써 수직 변경 버전 번호와 수평 변경 버전 번호로 나뉘게 된다. 이들 변경 버전 번호로 한 문서에서 수직, 수평 버전닝 문서를 생성한다. 예로, 버전 번호가 1.2.0일 때, 첫 번째 필드인 1이 릴리스 번호가 되고 첫 번째 필드를 제외한 2.0이 변경 버전 번호가 된다.

(2) 제안하는 버전 번호 할당 방법

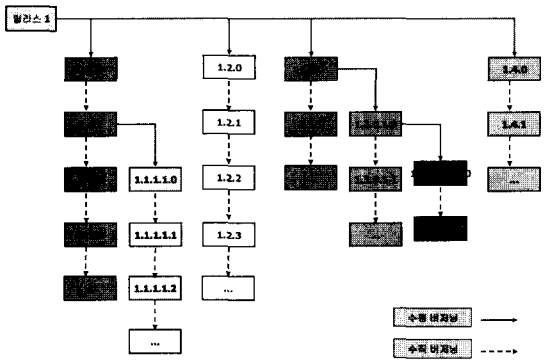
제안하는 버전 번호 할당 방법의 버전 계층은 트리로 구성되며 다음과 같은 방법으로 버전 번호를 부여한다.

한 문서가 버전닝이 되면 버전 번호 할당은 릴리스 버전 번호에 더불어 수평 버전 번호와 수직 버전 번호의 순서대로 변경 버전 번호의 두 번호를 조합하여 할당하며 각각의 버전 번호의 구별은 도트로 구분한다.

(그림 2) 제안하는 버전 번호 할당 규칙

하나의 릴리스 문서가 처음 변경되면 수평 변경 작업이 된다. 이것은 이후에 같은 릴리스 문서에 또 다른 수평 변경 작업이 발생해도 기존의 버전 번호와 독립적으로 수평 작업의 버전 번호가 부여될 수 있다. 수평 변경 작업시 버전 번호는 두 변경 버전 번호가 조합되어 부여하는데, 수평 변경 번호만이 부여되며 수직 변경 번호는 수직 변경 작업이 발생하지 않았음으로 단지 0을 부여함으로써 수평 변경임을 나타낸다. 수직 변경은 수직 번호가 하나 증가하여 할당하며 수평 번호는 변하지 않는다. 이후 각각의 수평 작업 영역에서 다시 발생하는 수평 작업들은 분기하는 버전 번호에 두 번호를 조합하여 덧붙여서 할당되며 마찬가지로 수직 번호는 0을 부여받는다. 그러므로 제안하는 방법으로 할당된 버전 번호의 끝이 0으로 끝나는 작업들은 모두 수평 변경 작업임을 뜻하게 되며 수평 변경이 발생할 때마다 두 개의 번호씩 덧붙여지게 된다. 또한 이후의 수직 변경 작업이 발생하면 동일하게 수직 번호가 증가된다.

그림 3에서 릴리스 문서인 버전 1에서 처음 수평 변경이 발생하면 릴리스 버전 번호에 두 개의 변경 버전 번호가 첨가되고 끝 번호 값이 0인 수평 버전 번호인 1.1.0의 문서가 생성되고 이후의 수직 변경이 발생한 문서들은 수직 변경 번호가 증가되면서 버전 번호를 부여하여 수직 버전의 문서인 버전 번호 1.1.1, 1.1.2, 1.1.3, ...의 문서가 생성된다. 해당 릴리스 문서에서 다시 수평 변경이



(그림 3) 제안하는 버전 번호 할당 방법

발생하면 수평 변경 번호가 증가하여 1.2.0, 1.3.0, 1.4.0, ...을 부여하게 된다. 이후 다시 발생하는 수평 변경은 두 개의 변경 버전 번호가 분기할 때의 버전 번호에 추가되어 변경 버전 번호의 수가 4개 이상인 수평 버전닝 문서를 생성하게 된다. 버전 번호 1.1.1.0은 버전 1.1.1인 문서에서 분기하여 수평 변경 작업으로 버전닝을 하면 두 개의 번호가 추가되어 버전 번호 1.1.1.1.0을 부여하게 된다. 마찬가지로 버전 번호 1.3.0.1.0인 문서도 1.3.0인 문서에서 수평 변경 작업으로 수평 버전닝된 문서를 생성하게 된다.

제안하는 버전 번호 할당 방법으로 번호를 부여함으로써, 기존 버전들의 버전 번호는 변하지 않고 확장된 새로운 버전 번호를 추가할 수 있고 버전 번호로 버전의 순서를 결정 가능하며 해당 버전의 직속 조상이나 후손을 알 수 있다. 또한, 한 문서의 변경 작업시 수직 작업 문서의 버전을 관리할 수 있을 뿐만 아니라 수평적이고 독립적으로 생성되는 문서들의 버전을 생성, 관리가 가능하다. 그러므로 수평 작업으로 생성된 새로운 버전들 즉, 버전 번호가 0으로 끝나는 버전들은 그것을 조상으로 하는 독립적인 버전들을 형성할 수 있으며 자체로 새로운 문서를 이루게 할 수 있다. 수평 영역에서 생성된 버전 번호로 변경을 수행한 수평 작업의 횟수와 수직 작업의 횟수를 알 수 있으며 그 합으로 변경된 총 횟수를 알 수 있다. 예로, 버전 1.3.1.1.0.1.1은 변경 버전 번호의 수가 6개이므로 수평 작업이 3번이고 수직 작업이 2번이므로 총 5번의 변경 작업을 수행한 것이다.

3.2 스키마 설계

본 논문에서는 문서 변경시 변경된 엘리먼트만을 반영할 수 있고 동적으로 버전 기능을 효율적으로 제공하는 분할 모델을 사용하여 문서를 저장한다. 분할 모델은 전체 문서 검색시 나뉘어진 엘리먼트들을 병합하기 위해 많은 시간이 요구되는 단점이 있는데, 이를 개선하여 보다 빠른 병합

을 함으로써 보다 효율적인 문서 검색을 위한 스키마를 설계한다.

XML 문서의 변경 사항을 쉽게 저장, 관리하기 위해 엘리먼트 테이블과 변경 테이블의 스키마를 설계하였다. 엘리먼트 테이블은 문서를 엘리먼트 단위로 분할하여 문서 전체를 저장하게 된다. 변경 테이블은 버전닝되는 문서의 변경된 엘리먼트들을 저장하게 된다.

(1) 엘리먼트 테이블

엘리먼트 테이블은 다음과 같이 7개의 필드로 구성된다.

- DID(DocumentID)

DID는 저장되는 모든 XML 문서들을 구별할 수 있는 문서들의 유일한 식별자이다.

- VERSION

VERSION은 제안하는 버전 번호 할당 방법으로 부여된 버전 번호가 된다. 이 버전 번호는 릴리스 버전 번호, 수평 변경 버전 번호, 수직 변경 버전 번호로 구분된다.

- EORD(Element ORDer)

EORD는 한 문서 내에서 엘리먼트들을 구별할 수 있는 유일한 식별자로써 번호가 부여된다. 이 번호는 트리 구조를 이루는 엘리먼트들을 중위 순회하여 얻어진 순서 번호로, 한 문서 상에서 순서대로 편집된 엘리먼트들의 순서를 뜻한다.

- LEVEL

LEVEL은 엘리먼트들의 트리 구조에서 해당 엘리먼트가 위치한 트리의 레벨값이 된다. 트리의 레벨은 그 자체가 트리의 계층 구조를 표현하기 때문에 XML 문서의 구조를 표현하는데 매우 적합하다.

DID	VERSION	EORD	LEVEL	ELEMENT	XMLNAME	CONTENTS

(그림 4) 엘리먼트 테이블의 스키마

- ELEMENT

ELEMENT는 문서 내의 해당 엘리먼트의 이름이다.

- XMLNAME

XMLNAME은 저장하는 문서의 이름이다.

- CONTENTS

CONTENTS는 한 엘리먼트가 포함하고 있는 내용이 된다. 엘리먼트의 태그를 포함하여 엘리먼트와 엘리먼트가 가지는 내용을 포함하게 된다.

(2) 변경 테이블

변경 테이블은 버전닝되는 XML 문서의 변경 사항을 반영하게 되며 엘리먼트 테이블에서 엘리먼트들을 구별할 수 있는 필드인 DID, VERSION, EORD, LEVEL, ELEMENT, CONTENTS와 같은 필드에 변경 플래그인 U-FLAG를 하나 추가한다. 이 U-FLAG는 문서의 변경 사항을 표시하는 플래그로 엘리먼트의 추가를 표시하는 A(Add), 엘리먼트의 삭제를 표현하는 D(Delete), 엘리먼트의 내용 변경을 표시하는 U(Update)의 플래그를 가지게 된다.

DID	VERSION	EORD	LEVEL	ELEMENT	U-FLAG	CONTENTS

(그림 5) 변경 테이블의 스키마

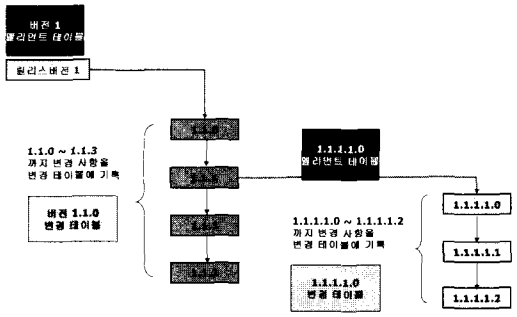
3.3 XML 문서 저장

실제 XML 문서의 저장은 릴리스 문서와 수평, 수직 변경 작업에 의해 버전닝되는 문서를 저장한다. 릴리스 문서는 문서 전체를 엘리먼트 단위로 분할하여 엘리먼트 테이블에 저장한다. 릴리스 문서를 한 번 수평 변경한 문서는 문서의 변경 부분을 변경 테이블에 저장하게 된다. 이후의 수직 작업으로 버전닝되는 문서는 마찬가지로 변경 테이블에 변경 사항을 저장한다. 여러 번 수평 변경한 문서 즉, 변경 버전 번호의 수가 4개 이상이고 번호의 끝값이 0인 문서는 수평 작업 자체로

새로운 문서로 관리될 수 있기 때문에 엘리먼트들을 재구성하여 엘리먼트 테이블에 새롭게 저장한다. 이것은 수평 버전닝으로 문서를 분기하여 이 문서를 조상으로 하는 독립된 새로운 의미를 갖는 문서를 생성할 수 있기 때문이다. 또한 수평 작업으로 분기할 때, 변경 사항은 변경 테이블에 유지하게 되는데 이는 단지 분기시의 변경 사항만을 알고 싶은 경우를 위한 것이다. 이후의 수직 작업은 변경 부분을 변경 테이블에 유지하면 된다.

그림 6은 버전닝된 XML 문서를 저장하는 엘리먼트 테이블과 변경 테이블을 나타낸다. 버전 1의 릴리스 문서와 여러 번 수평 변경한 문서는 엘리먼트 단위로 분할하여 엘리먼트 테이블에 저장하게 되는데, 버전 1.1.1.1.0은 문서를 재구성하여 저장하게 된다. 한번 수평 변경한 문서와 이후의 수직 변경한 문서인 버전 1.1.0에서 1.1.3까지는 문서들의 변경된 부분만을 반영한 변경 테이블에 저장하게 된다. 마찬가지로 여러 번 수평 변경한 문서를 이후에 수직 변경한 버전 1.1.1.1.0에서 1.1.1.1.2인 문서들의 변경 사항을 변경 테이블에 저장하게 된다.

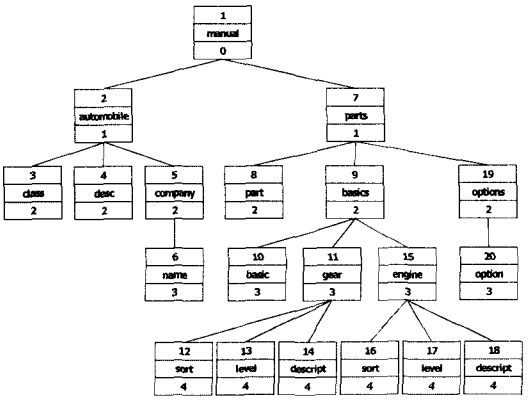
그림 7은 자동차의 기본 사양에 대해 설명하고 있는 자동차 매뉴얼에 대한 간단한 XML 문서의 예인 manual.xml을 보이고 있다. 또한 그림 8은 예를 들고 있는 manual.xml의 계층 구조를 트리로 표현을 하고 있는데, 각각의 노드는 XML 문서를 구성하는 엘리먼트들의 EORD 값, 엘리먼트의 이름, 그리고 트리의 레벨을 뜻하는 레벨 값을 나타내고 있다.



(그림 6) 버전닝된 XML 문서의 저장

```
<manual status="public">
  <automobile>
    <class> 자동차 메뉴얼 </class>
    <desc> 이 매뉴얼은 자동차의 사양에 대한 설명으로... </desc>
    <company>
      <name> 자동차 회사 </name>
    </company>
  </automobile>
  <parts>
    <part> 자동차 사양 </part>
    <basics>
      <basic> 기본 사양 </basic>
      <gear>
        <sort> 기어 </sort>
        <level> 변속 </level>
        <descript> 기어에 대한 설명 </descript>
      </gear>
      <engine>
        <sort> 엔진 </sort>
        <level> 마력 </level>
        <descript> 엔진에 대한 설명 </descript>
      </engine>
    </basics>
    <options>
      <option> 선택 사양 </option>
    </options>
  </parts>
</manual>
```

(그림 7) 간단한 XML 문서의 예



(그림 8) XML 문서의 계층 구조

1	1	1	0	manual	manual	<manual status="public"></manual>
1	1	2	1	automobile	manual	<automobile></automobile>
1	1	3	2	class	manual	<class>자동차 메뉴얼</class>
1	1	4	2	desc	manual	<desc>이 매뉴얼은 ···</desc>
1	1	5	2	company	manual	<company></company>
1	1	6	3	name	manual	<name>자동차 회사</name>
1	1	7	1	parts	manual	<parts></parts>
1	1	8	2	part	manual	<part>자동차 사양</part>
1	1	9	2	basics	manual	<basics></basics>
1	1	10	3	basic	manual	<basic>기본 사양</basic>
1	1	11	3	gear	manual	<gear></gear>
1	1	12	4	sort	manual	<sort>기어</sort>
1	1	13	4	level	manual	<level>변속</level>
1	1	14	4	descript	manual	<descript>기어에 대한 설명</descript>
1	1	15	3	engine	manual	<engine></engine>
1	1	16	4	sort	manual	<sort>엔진</sort>
1	1	17	4	level	manual	<level>마력</level>
1	1	18	4	descript	manual	<descript>엔진에 대한 설명</descript>
1	1	19	2	options	manual	<options></options>
1	1	20	3	option	manual	<option>선택 사양</option>

(그림 9) 릴리스 문서를 저장한 엘리먼트 테이블

그림 9는 실제 릴리스 문서인 manual.xml을 엘리먼트 단위로 분할하여 엘리먼트 테이블에 저장하고 있다.

3.4 XML 문서 변경

XML 문서에 대한 변경은 부분 삽입이나 부분 삭제 연산에 의해 발생한다. 그러나 기존의 문서와는 다른 구조화된 특성을 갖는 XML 문서는 내용만을 삽입하거나 삭제할 수도 있지만 문서의 구조가 변하면서 내용이 추가되거나 삭제되는 경우도 있다. 그렇기 때문에 XML 문서에 대한 변경은 크게 엘리먼트의 삽입이나 삭제와 같은 구조 변경과 내용 변경을 동시에 고려하여야 한다.

(1) 엘리먼트의 삽입

XML 문서에 엘리먼트들의 삽입은 문서의 중간과 마지막에 삽입할 수 있다. 엘리먼트의 삽입 시, 우선 삽입할 해당 위치의 EORD값과 LEVEL값을 구하고 변경 테이블의 변경 플래그에 엘리먼트의 추가 표시를 하고 삽입하는 엘리먼트의 내용을 변경 테이블에 저장하면 된다. EORD값은 다음과 같은 방법으로 부여한다.

1) 문서의 중간에 삽입
   한 문서 내에서 엘리먼트와 엘리먼트 사이로 삽입되는 경우로, EORD값의 부여는 삽입될 위치의 양쪽 엘리먼트의 EORD값을 더하여 2로 나눈 값으로 EORD를 부여한다.
2) 문서의 마지막에 삽입
   문서의 제일 마지막에 삽입하는 경우로, 마지막 엘리먼트의 EORD값에 1을 더하여 부여한다.

(2) 엘리먼트의 삭제

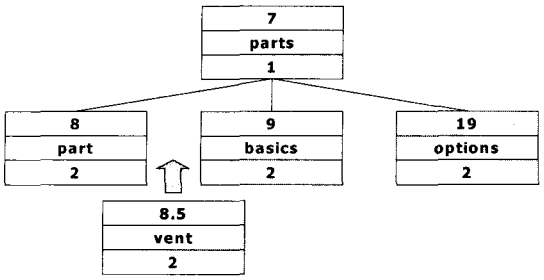
문서 내에 사용된 엘리먼트의 삭제가 발생할 경우에 삭제하려고 하는 엘리먼트의 LEVEL값과 바로 다음의 엘리먼트, 즉 EORD값이 바로 다음으로 큰 값을 가지는 엘리먼트의 LEVEL값을 비교하여 값이 작으면 자식 엘리먼트가 없는 엘리먼트로써, 바로 삭제를 하고 변경 테이블에 변경 플래그를 삭제 표시를 하면 된다. 그렇지 않고 비교한 값이 해당 엘리먼트의 LEVEL값보다 크면 자식 엘리먼트가 존재하는 것이므로 모든 하위 엘리먼트들을 찾아서 삭제를 해주어야 한다. 모든 하위 엘리먼트들을 찾기 위해서는 자신의 LEVEL값보다 같거나 작은 엘리먼트가 나타날 때까지의 사이에 있는 큰 LEVEL값을 갖는 모든 엘리먼트가 해당된다. 그러므로 해당 엘리먼트와 모든 하위 엘리먼트들을 다 삭제함으로써 자식 엘리먼트가 있는 엘리먼트의 삭제 연산을 하게 된다. 또한 변경 테이블의 변경 플래그는 모든 삭제되는 엘리먼트들에 삭제 표시를 한다.

(3) 엘리먼트의 내용 변경

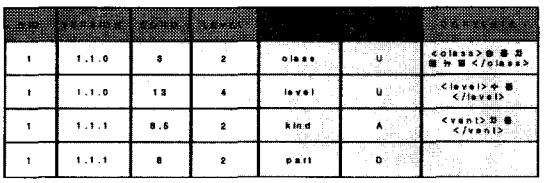
엘리먼트의 내용 변경이 발생하면 변경 테이블의 변경 플래그를 내용 변경으로 표시를 하고 변경된 내용을 테이블에 저장한다.

(4) XML 문서 변경

1) 처음 수평 변경 작업
   그림 7의 manual.xml인 자동차 기본 매뉴얼에서 승용차 매뉴얼로 문서를 변경하면 처음 수평 변경한 문서를 생성하게 된다. 문서를 EORD가 3인 class 엘리먼트가 승용차 매뉴얼인 내용 변경과 EORD가 13인 level 엘리먼트를 수동으로 내용을 변경하면 처음 수평 변경 작업으로 버전 1.1.0의 문서를 생성하게 된다. 또 버전 1.1.0의 수평 변경 문서에 그림 3처럼 EORD가 8과 9 사이에 차종을 뜻하는 vent 엘리먼트를 추가하고 EORD가 8인 part 엘리먼트를 삭제하게 되면 버전 1.1.1의 수직 변경되는 문서를 생성하게 된다. 이때 삽입되는 엘리먼트의 EORD 값은 삽입될 위치의 양쪽 엘리먼트의 EORD 값을 더하여 2로 나눈 값인 8.5가 된다.



(그림 10) XML 문서 변경



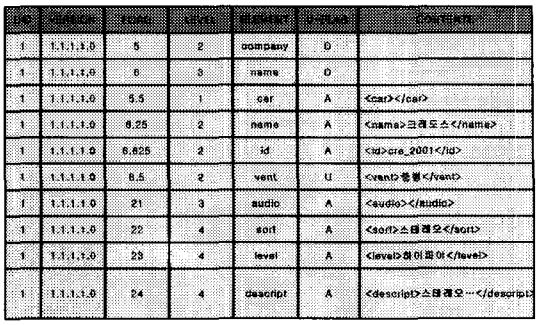
(그림 11) XML 문서의 변경 반영

그림 11은 manual.xml 문서를 처음 수평 변경한 후에 수직 변경한 문서를 생성할 때, 변경된 사항들을 기록한 변경 테이블이 된다.

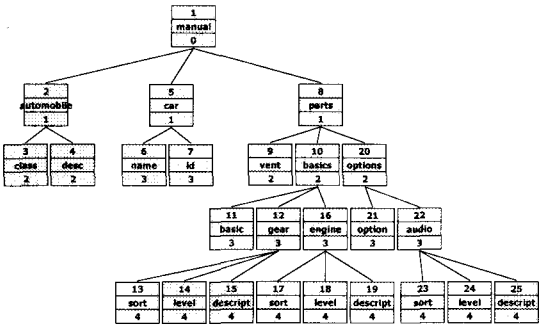
2) 여러 번 수평 변경 작업

버전이 1.1.1인 승용차 매뉴얼에서 차종이 크레도스인 매뉴얼로 수평 변경이 다시 발생하면 변경 번호 자리 수가 두 개 증가된 버전 1.1.1.1.0의 여러 번 수평 변경한 문서를 생성하게 된다. company 엘리먼트가 삭제되고 해당 차종을 선택하는 car 엘리먼트와 선택 사양으로 audio 엘리먼트를 추가하면 버전 1.1.1과 1.1.1.1.0 사이의 변경 사항들을 그림 12의 변경 테이블에 반영하게 된다.

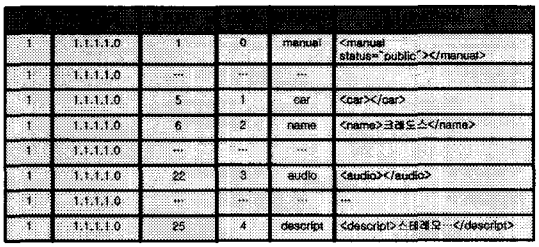
여러 번 수평 변경이 발생하여 생성되는 문서는 그 문서를 조상으로 하는 새로운 문서로써 독립적으로 관리될 수 있으므로 문서 안의 엘리먼트들을 재구성하여 엘리먼트 테이블에 저장한다. 문서가 여러 번 변경됨으로써 기존의 XML 문서의 구조가 많이 변경되었고 내용 또한 많은 변경이 발생하였으므로 엘리먼트들을 재구성하여 EORD 값을 주고 내용을 엘리먼트 테이블에 반영하게 된다. 이것은 수평 변경 작업으로 분기되는 문서들의 보다 빠른 검색과



(그림 12) XML 문서의 여러 번 수평 변경 사항 반영



(그림 13) 재구성한 문서의 계층 구조



(그림 14) 문서를 재구성한 엘리먼트 테이블

이후의 세분화되는 수직 변경 작업들을 보다 쉽게 관리할 수 있다는 장점을 가지게 되며 독립적인 문서로써 새로운 의미가 부여되는 문서들을 생성할 수 있다. 그림 13과 그림 14는 자동차 매뉴얼에서 여러 번 수평 버전닝을 한 문서로써 차종이 크레도스라는 매뉴얼로써 의미를 가지는 독립적인 문서를 생성하고, 보다 쉽게 문서를 관리하기 위해서 버전 1.1.1의 승용차 매뉴얼의 문서를 재구성한 계층 구조와 엘리먼트들을 재구성하여 테이블에 반영한 것이다.

3.5 버전닝된 XML 문서 검색

버전닝된 XML 문서의 검색은 해당 문서의 DID와 VERSION으로 검색을 수행한다. 버전닝된 문서는 엘리먼트 테이블과 변경 테이블에 버전 별로 엘리먼트들이 분할되어 저장되어 있기 때문에 문서의 해당 DID와 VERSION으로 엘리먼트들을 추출하고 하나의 문서를 이루기 위해 추출된 엘리먼트들을 병합하는 과정이 필요하다.

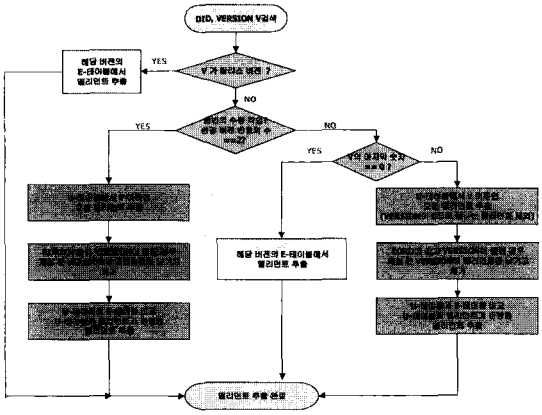
(1) 엘리먼트의 추출

버전닝된 문서를 병합하기 위해서는 엘리먼트를 추출해야 한다. 우선 변경 버전 번호의 수가 2개인 한 번의 수평 변경 작업과 이후의 수직 변경 작업에서 엘리먼트를 추출하는 방법은 다음과 같다.

변경 테이블에서 해당 VERSION보다 작거나 같은 모든 엘리먼트들을 추출한다. 그리고 EORD로 추출된 엘리먼트들을 정렬하여 EORD가 같고 VERSION이 틀린 경우, 제일 큰 VERSION의 엘리먼트만 남기고 나머지는 제거한다. 변경 테이블에서 추출한 엘리먼트들과 엘리먼트 테이블의 엘리먼트들을 비교하여 EORD가 같으면 변경 테이블에서 추출한 엘리먼트들만 사용하고 EORD가 다른 엘리먼트들은 변경 테이블과 엘리먼트 테이블에서 추출하면 검색하려는 문서의 모든 엘리먼트들이 추출된다.

다음으로 변경 버전 번호의 수가 4개 이상인 여러 번의 수평 변경 작업은 VERSION의 끝값이 0으로 끝나므로 엘리먼트 테이블에 버전닝된 문서를 재구성하여 저장해 놓았기 때문에 엘리먼트 테이블에서 해당 문서의 DID와 VERSION을 가지는 모든 엘리먼트들을 추출하면 된다. 그 외, 이후의 수직 변경 작업은 변경 테이블에서 VERSION이 0으로 끝나는 엘리먼트들을 제외하여 추출한 엘리먼트들을 가지고 한 번의 수평 변경 작업과 같은 방법으로 엘리먼트들을 추출하면 된다.

그림 15는 버전닝된 XML 문서의 엘리먼트를 추출하는 순서도를 표현하고 있다.



(그림 15) 엘리먼트들의 추출 순서도

(2) 문서 병합

한 문서의 병합은 해당 문서의 DID와 VERSION으로 추출한 엘리먼트들을 EORD, LEVEL, CONTENTS를 이용하여 통합을 하면된다. 엘리먼트들의 EORD값은 한 문서 내의 엘리먼트들의 편집된 순서이기 때문에 EORD 순서대로 통합을 하면 하나의 XML 문서가 바로 병합이 되기 때문에 보다 빠른 병합을 할 수 있다는 장점을 가지고 있고, 그러므로 보다 빠른 문서를 검색할 수 있다. 이것은 기존의 분할 모델이 가지고 있는 단점인 문서 검색시, 문서 병합 시간이 오래 걸리는 것을 보완하게 된다.

엘리먼트들을 통합하는 과정은 EORD 순서로 엘리먼트들을 정렬을 한 후에 다음의 방법을 이용한다. 루트 엘리먼트는 XML 선언 부분 및 DTD 선언 부분과 시작 태그를 기록한다. 하위 EORD 값을 가지는 엘리먼트로 이동하여 시작 태그를 기록한다. 그다음 하위 엘리먼트들의 LEVEL값들을 비교하여 값이 작거나 같은 엘리먼트가 있을 때까지 그 사이에 있는 엘리먼트들과 CONTENTS에 있는 내용을 기록하고 닫는 태그를 기록한다. 다음 EORD값 순서대로 추출된 엘리먼트들을 위의 방법을 수행하여 모든 엘리먼트들의 내용을 기록한다. 마지막으로 루트 엘리먼트의 닫는 태그를 기록하면 하나의 문서 병합을 완료하게 되고 해당 DID와 VERSION을 갖는 문서의 검색을 마치게 된다.

3.6 구조 검색

본 논문에서는 XML 문서의 특징인 구조적 정보를 이용한 엘리먼트 단위의 구조 검색을 지원한다. 구조 검색은 문서의 논리적인 구조에 기반한 질의로서, 엘리먼트들의 계층간의 관계, 같은 계층 내에서의 관계 등을 고려하여야 한다. 계층간의 관계는 부모/자식/조상/자손 관계가 있으며, 같은 계층 내의 관계는 형제 엘리먼트 간의 순서가 있다. 전체 엘리먼트 집합에서 특정 엘리먼트를 유일하게 구분해줄 수 있는 DID, VERSION, EORD를 추출하는 것을 최종 목적으로 한다.

(1) 부모 검색

우선 찾고자하는 부모 엘리먼트의 기준이 되는 자식 엘리먼트의 EORD를 구한다. 이것의 LEVEL 값을 가지고 LEVEL 값이 하나 작은 엘리먼트를 역순 검색하여 처음으로 나오는 EORD값을 가진 엘리먼트가 바로 부모 엘리먼트가 된다.

(2) 자식 검색

부모 엘리먼트의 EORD값 보다 크고 LEVEL값이 같거나 작은 엘리먼트가 있을 때까지 검색하여 레벨 값이 하나 더 큰 엘리먼트들 모두가 자식 엘리먼트가 된다.

(3) 형제 검색

형제 검색에서는 기준 엘리먼트로부터 좌우 형제 엘리먼트에 대한 검색 질의를 처리한다. 기준이 되는 엘리먼트의 EORD값에서 역순 검색하여 기준 위치의 LEVEL값보다 낮은 LEVEL값을 가진 EORD부터 순차 검색하여 LEVEL값이 다른 엘리먼트가 나올 때까지의 LEVEL값이 같은 모든 엘리먼트들의 EORD를 구하면 형제 엘리먼트들이 검색된다.

(4) 같은 이름의 형제 검색

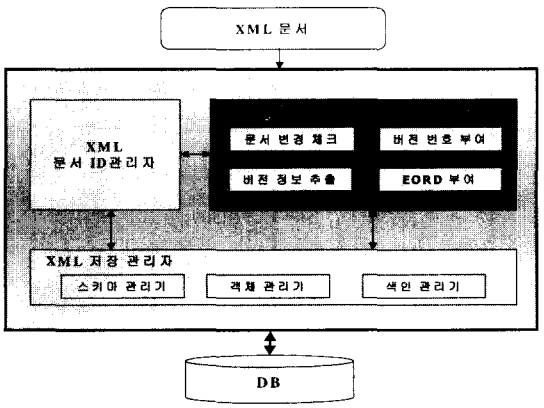
형제 검색으로 추출된 엘리먼트들 중에서 ELEMENT 이름이 같은 엘리먼트들의 EORD를 구하면 같은 이름의 형제 검색이 된다.

4. 구현 및 성능 평가

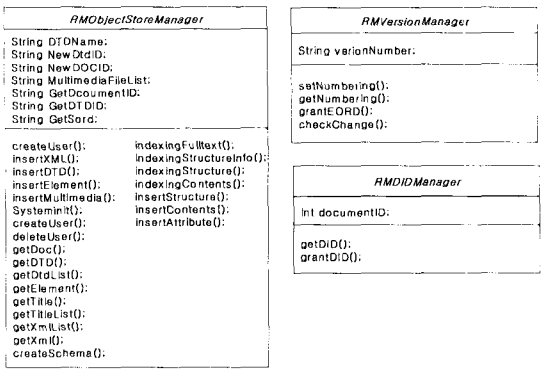
4.1 제안하는 버전닝 기법을 적용한 저장관리 시스템 구조

본 논문에서 제안한 버전닝 기법을 적용하여 구현한 XML 문서 저장관리 시스템은 자바로 구현하였으며, 하부 저장 시스템으로 RDBMS의 하나인 ORACLE 8.0을 사용하였다. 그림 16은 제안하는 XML 문서 버전닝 기법을 적용하여 구현한 XML 문서 저장관리 시스템 구조를 보여준다. 구현한 XML 문서 저장관리 시스템은 XML 문서 ID 관리자, XML 버전 관리자, XML 저장 관리자로 구성된다. XML 문서 ID 관리자는 저장하려는 XML 문서의 유일한 식별자인 문서의 ID를 부여하고 관리하는 모듈이고 XML 저장 관리자는 실제 DBMS에 XML 문서를 저장하고 관리하는 모듈로써 데이터베이스 스키마를 관리기하는 스키마 관리기, 문서들을 저장하고 읽어오는 객체 관리기, 검색을 위해 색인 정보를 관리하는 색인 관리기로 구성된다.

XML 문서 저장관리 시스템의 가장 핵심적인 버전닝을 담당하는 XML 버전 관리자는 저장하려는



(그림 16) XML 문서 저장관리 시스템 구조



(그림 17) 클래스다이어그램

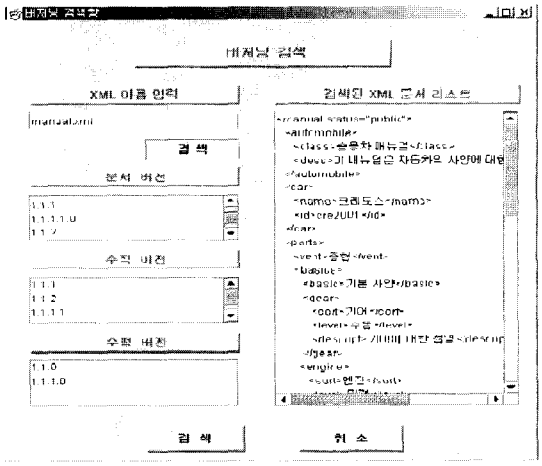
XML 문서가 변경된 문서인지를 체크하는 모듈, 변경된 XML 문서에 제안하는 버전 번호할당 방법으로 버전 번호를 부여하는 모듈, 문서의 버전 정보를 추출하고 엘리먼트들의 정보와 구조 정보를 추출하는 모듈, 그리고 XML 문서 내에서 엘리먼트들의 순서 정보를 나타내는 EORD를 부여하는 모듈로 구성된다.

그림 17은 구현한 시스템의 클래스 다이어그램을 나타낸다. RMObjectStoreManager는 XML 저장관리자의 기능을 수행하는 클래스를 나타내며, RMVersionManager는 XML 버전관리자의 기능을 수행하는 클래스이다. RMDIDManager는 XML 문서 ID 관리자의 기능을 수행하는 클래스이다.

4.2 버전닝 문서 검색 인터페이스

사용자가 쉽게 버전닝된 XML 문서에 대한 검색을 하고 검색 결과를 볼 수 있도록 사용자 인터페이스를 구현하였다. XML 문서를 문서 이름으로 검색을 하면 해당 문서의 수평 버전닝과 수직 버전닝이 통합된 한 문서의 버전이 창에 나타남과 동시에 수직 버전과 수평 버전으로 나뉘어진 문서의 버전을 볼 수가 있다. 수직 버전닝된 문서의 검색은 수직 버전 창에서 검색을 하고 수평 버전닝된 문서의 검색은 마찬가지로 수평 버전 창에서 검색을 하면 해당 버전의 문서 검색이 이루어진다.

그림 18은 구현한 버전닝된 문서의 검색 인터



(그림 18) XML 문서 검색 인터페이스

페이스를 보여주고 있으며, manual.xml 문서를 검색하여 해당 문서의 통합 버전 뿐만 아니라 문서의 수평 버전과 수직 버전을 보여주고 있다.

4.3 성능 평가

본 절에서는 기존의 XML 문서 저장 방식인 분할 저장 모델[11]과 RDBMS와 IRS의 장점만을 이용하여 설계하고 구현한 분할 모델의 XML 저장관리시스템[2]과 분할 모델로 데이터 모델을 설계하여 제안하는 버전닝 기법을 적용한 XML 문서 저장관리 시스템 모델을 비교하여 성능을 평가한다. 성능을 비교하기 위한 평가 항목으로는 XML 문서의 버전닝을 위한 저장 공간과 버전닝된 문서의 검색 시간으로 평가하였다. 기존의 XML 문서 저장 방식인 분할 저장 모델을 분할 모델 1으로 표기하고, RDBMS와 IRS의 장점만을 이용하여 설계하고 구현한 분할 모델의 XML 저장관리시스템을 분할 모델 2로 표기한다.

XML 문서의 분할 저장 모델은 문서의 실제 내용을 가지고 있는 각각의 엘리먼트로 나누어서 저장함으로써 문서의 구조적 정보나 엘리먼트만 수정하면 되므로 문서의 버전관리가 쉽다는 장점을 가지고 있어서 제안하는 버전닝 기법의 성능 평가를 비교하는 대상으로 설정하였다.

저장하는 XML 문서는 5K 바이트(byte) 크기를 가지는 문서로 설정하였고 다음과 같이 문서의 버전닝을 위한 요소를 정하여 실험을 하였다.

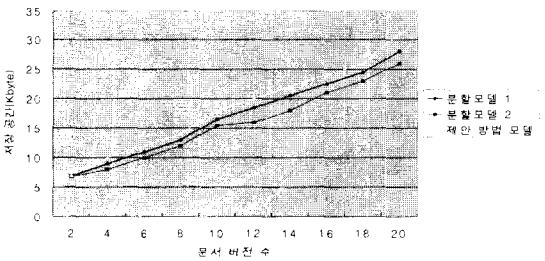
- 기존의 XML 문서의 버전닝은 수평 버전닝에 대한 연구는 전무한 상태이기 때문에 제안하는 수평 버전닝 방법을 분할 모델에서는 수직 버전닝처럼 취급을 한다.
- 각각의 버전닝 문서는 이전 버전에 비해 수직 버전닝은 20% 변경이 발생했고 수평 버전닝은 50% 변경이 발생한 문서를 생성한다.

그림 19는 버전닝된 XML 문서의 저장 공간에 대한 성능 평가이다. 버전닝된 XML 문서를 저장시, 문서의 변경은 총 50번의 버전닝을 하는 것으로 하며 분할 모델에서 수직 버전닝의 5번째마다 수평 버전닝이 발생하는 것으로 하여 실험을 하고 측정하였다.

제안하는 버전닝 기법을 적용한 모델은 기존의 분할 저장 모델에 비해서 상대적으로 약간 많은 저장 공간을 차지한다. 제안하는 버전닝 기법은 수평 버전닝을 할 때, 독립적으로 생성될 수 있는 새로운 의미를 갖는 문서를 생성하기 위해서 변경 사항만을 저장하는 분할 모델에 비해 문서 전체를 재구성하여 모든 엘리먼트들을 저장하기 때문이다.

(표 1) 버전닝된 XML 문서의 저장공간 비교

문서 버전수	2	10	20	30	40	50
분할모델1	7K	16.5K	28K	39.5K	51K	62.5K
분할모델2	6.8K	15.5K	26K	35.5K	47K	59.5K
제안방법	7K	18K	31K	44K	57K	70K



(그림 19) 버전닝된 XML 문서의 저장 공간

그림 20은 버전닝된 XML 문서의 검색 시간에 대한 성능 평가이다. 버전닝된 XML 문서의 검색시, 문서의 변경은 총 20번의 버전닝을 하는 것으로 하며 제안하는 버전닝 기법에서 설계한 스키마를 분할 모델에 적용하여 검색 시간을 측정하고 제안하는 버전닝 기법인 수평 버전닝과 스키마 모두를 적용한 검색 시간으로 나누어서 성능 평가를 하였다. 또한, 분할 모델에서 수직 버전닝의 4번째마다 수평 버전닝이 발생하는 것으로 하여 실험을 하였다.

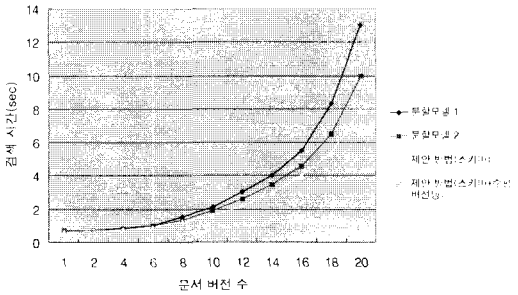
분할 저장 모델은 문서의 버전 관리가 쉽다는 장점을 가지고 있으나 문서의 변경된 부분만을 저장하므로 버전닝된 XML 문서를 검색할 때, 병합을 하는데 많은 시간을 소비하는 단점을 가지고 있다. 이에 비해 제안하는 버전닝 기법중 스키마를 분할 모델에 적용하면 보다 검색 시간을 줄일 수 있다. 이것은 문서 검색시, 문서의 엘리먼트 순서인 EORD 값의 순서대로 엘리먼트를 추출하여 문서를 병합하기 때문에 보다 나은 검색 시간을 얻을 수 있는 것이다.

제안하는 버전닝 기법을 모두 적용하여 XML 문서를 검색시, 상당히 검색 시간이 좋은 것으로 나타났다. 기존의 분할 모델의 수직 버전닝만을 사용하면 문서의 변경하는 횟수가 증가할수록 생성되는 버전의 수가 많아지고 그만큼의 병합을 하는 시간이 상당히 오래 걸려서 문서 검색 시간이 엄청나게 증가하게 된다. 그러나 제안하는 수평 버전닝 기법을 적용하면 수평 버전닝시 문서 전체를 재구성하여 전체를 저장하므로 문서를 병합하는데 걸리는 시간이 거의 없어지고, 이후의 수직 버전닝도 문서의 버전이 증가하더라도 수평 버전닝되는 문서까지의 변경된 엘리먼트들만을 추출하여 문서가 병합되므로 검색 시간을 상당히 줄일 수가 있다. 또한 처음 수평 버전닝을 하는 문서가 많을수록, 그리고 이후의 여러 번 수평 버전닝을 하는 문서가 많이 발생할수록 버전닝되는 XML 문서의 검색 시간은 상당히 단축되게 된다.

성능 평가의 결과, 제안하는 버전닝 기법을 적용한 모델이 분할 저장 모델들에 비해 약간의 저장 공간을 차지하나 수평 버전닝에 의해 문서를 검색시 보다 빠른 검색을 지원하는 우수한 성능을 보인다.

(표 2) 버전닝된 XML 문서의 검색시간 비교

문서 버전수	1	4	10	16	20
분할모델1	0.7초	0.85초	2.13초	5.5초	13.02초
분할모델2	0.68초	0.83초	1.9초	4.6초	10초
제안방법 (스키마)	0.65초	0.79초	1.64초	4초	7.9초
제안방법(스키마 +수평버전닝)	0.65초	0.64초	0.85초	0.7초	0.71초



(그림 20) 버전닝된 XML 문서의 검색 시간

5. 결 론

본 논문에서는 기존의 XML 저장관리 시스템을 위한 수직적인 버전닝뿐만 아니라 수평적인 버전닝을 할 수 있는 확장된 버전닝 기법들을 제안하였다. 수평 버전닝을 위한 보다 확장된 버전 번호 할당 방법을 제안하여 수평 버전닝에 의한 독립적이고 새로운 의미를 갖는 문서를 생성하고 관리가 가능하게 하였다. XML 문서의 변경 사항을 쉽게 관리하는 데이터 모델과 스키마를 설계하였으며 버전닝된 XML 문서의 보다 빠른 병합을 지원하여 상당히 많은 검색 시간을 단축하였다. 또한 XML 문서의 특징을 반영한 구조 검색을 지원하게 하였다.

향후 연구 방향으로 제안하는 버전닝 기법을 적용하여 보다 다양한 성능 평가를 수행하여 우수성을 확인하며 DTD의 변경이 반영되고 보다 다양한 검색을 지원하며 멀티미디어 처리를 위한 XML 문서 저장관리 시스템을 설계 및 구현하는 것이다.

참 고 문 헌

[1] Extensible Markup Language(XML) 1.0, http://www.w3.org/TR/1998/REC-xml-19980210.

[2] 강형일, "효율적인 구조검색을 위한 XML 저장관리 시스템 설계", 박사학위논문, 2001. 2.

[3] 최동운, 김수용, 송행숙, "객체 지향 소프트웨어 개발 환경을 위한 지역 버전관리자", 한국정보처리학회 논문지 제5권 제12호, 1998.

[4] 오상엽, 김홍진, 장덕철, "버전 제어를 위한 소프트웨어 구성요소의 검색 시스템", 한국정보처리학회 논문지 제3권 5호, 1996.

[5] W. F. Tichy, "RCS-A System for Version Control", Software-Practice and Experience, Vol.15, No.7, 1986.

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

[7] A. Marian, et al., "Change-centric management of verions in an XML warehouse", In Proc. VLDB 2001, Roma, Italy, Sept., 2001.

[8] Gregory Cobena, Serge Abiteboul and Amelie Marian, "Detecting Changes in XML Documents", 2001.

[9] G. Copeland and S. Khoshafian, "A Decomposition Stroage Model", Proceedings of the ACM SIGMOD Conference, 1985.

[10] Daniel Poulin, Guy Huard, and Alain Lavoie, "The Other Formalization of Law : SGML Modeling and Tagging", ICAIL'97 ACM, 1997.

[11] Daniela Florescu and Donald Kossmann, "Storing and Querying XML Data using an RDBMS", IEEE Data Engineering Bulletin, 22(3), 1999.

[12] Francois, "Generalized SGML repositories: Requirements and modelling", Computer Standards & Interfaces, pp. 11~24, 1996.

[13] Takeyuki Shimura, Masatoshi Yoshikawa, and

Shunsuke zUemura, "Storage and Retrieval of XML Documents Using Object-Relational Databases", DEXA, 1999.

[14] Fabo Vitali and David G. Durand, "Using versioning to support collaboration on the WWW", 4th WWW Conference, 1995.

[15] A. Dix, T. Rodden and I. Sommervile, "Modeling versions in collaborative work", IEE Proc-Softw. Eng., Vol144, No 4, 1997.

[16] Arthur M. Keller and Jeffrey D. Ullman, "A Version Numbering Scheme with a Useful Lexicographical Order", IEEE, 1995.

[17] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener, "The Lorel Query Language for Semistructured Data", International Journal on Digital Library, 1(1), 1997.

[18] Albrecht Schmidt, Martin L. Kersten, Menzo Windhouwer, Florian Waas, "Efficient Relational Storage and Retrieval of XML Documents", WebDB (Informal Proceedings), 2000.

[19] Donald Kossmann and Daniela Florescu, "A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database", Rapport de recherche, 1999.

◐ 저 자 소 개 ◑

손 충 범
1997년 충북대학교 정보통신공학과 졸업(공학사)
1999년 충북대학교 대학원 정보통신공학과 졸업(공학석사)
2002년 충북대학교 대학원 정보통신공학과 졸업(공학박사)
관심분야 : XML, 데이터베이스 시스템, 정보검색 프로토콜, 분산 객체 컴퓨팅 분야 etc.
E-mail : cbson@trut.chungbuk.ac.kr

배 양 석
1997년 충북대학교 컴퓨터공학과 졸업(공학사)
2002년 충북대학교 대학원 정보통신공학과 졸업(공학석사)
2002년~현재 : (주)인프라웍스 기술연구소 근무
관심분야 : 데이터베이스, XML
E-mail : heart21c@netdb.chungbuk.ac.kr

유 재 수
1989년 전북대학교 컴퓨터공학과 졸업(공학사)
1991년 한국과학기술원 전산학과 졸업(공학석사)
1995년 한국과학기술원 전산학과 졸업(공학박사)
1995년~1996년 8월 목포대학교 전산통계학과 전임강사
1996년 8월~현재 : 충북대학교 정보통신공학과 부교수
관심분야 : 데이터베이스 시스템, XML, 멀티미디어 데이터베이스, 분산객체 컴퓨팅, etc.
E-mail : yjs@cbucc.chungbuk.ac.kr