한국산학기술학회논문지
Vol. 10, No. 10, pp. 2696-2707, 2009

XML 문서 변경 탐지 기능을 갖는 통합 리파지토리 시스템

박성진[1*]
[1]한신대학교 컴퓨터공학부

An Integrated Repository System with the Change Detection Functionality for XML Documents

Seong-Jin Park[1*]
[1]**Division of Computer Engineering, Hanshin University**

요 약 비록 많은 DBMS 업체들이 XML을 지원하기 위해 기존 제품들을 확장하고 있지만 이와는 별도로 DBMS 종류와 플랫폼에 독립적인 경량의 XML 리파지토리 시스템 개발이 요구되고 있다. 본 논문에서 다음과 같은 기능들을 지원하는 XML 통합 리파지토리 시스템의 설계 및 구현에 관해 기술하였다. 구현된 XML 리파지토리 시스템은 XML DTD로부터 XML 문서 저장에 필요한 스키마 구조를 생성하고 데이터베이스 테이블에 저장한 뒤 XMLQL(XML Query Language)를 통해 자유롭게 XML 문서를 생성할 수 있으며 중복된 XML 문서들을 동기화시킨다. XML 리파지토리에는 동일한 데이터가 다양한 XML 문서에 중복될 수 있기 때문에 중복된 XML 문서들의 일관성 유지를 위한 효율적인 변경 탐지 기법이 요구된다. 논문에서는 메시지 다이제스트 기반의 변경 탐지 기법을 제안함으로써 클라이언트 XML 문서와 리파지토리 안의 XML 데이터간의 일관성을 유지하도록 하였다.

Abstract Although, a number of DBMS vendors are scrambling to extend their products to handle XML, there is a need for a lightweight, DBMS and platform-independent XML repository as well. In this paper, we describe such an XML integrated repository system, that solves the following functions : generating relational schema from XML DTDs for storage of XML documents, importing data from XML documents into relational tables, creating XML documents according to a XMLQL(XML Query Language) from data extracted from a database and synchronizing the replicated XML documents. In the XML repository systems, the efficient change detection techniques for XML documents is required to maintain the consistency of replicated XML data because the same data in the repository can be replicated between so many different XML documents. In this paper, we propose a message digest based change detection technique to maintain the consistency of replicated data between client XML documents and a XML data in XML repository systems.

Key Words : XML, Repository, Message Digest, Change Detection

1. 서론

플랫폼에 상관없이 문서 정보의 전송과 교환이 용이하고 문서 내용과 의미를 함께 표현할 수 있다는 장점을 지닌 XML[21]은 1998년 W3C(World Wide Web Consortium)에서 차세대 인터넷 표준으로 채택된 이후, 인터넷 기반 기술로서 인터넷을 기반으로 하는 모든 정보시스템에 널리 사용되고 있다. XML은 웹상에서의 데이터 교환을 위해 제안된 표준 언어로 사용자가 문서의 구조를 자유롭게 정의할 수 있으며 문서 안에 문서의 구조에 대한 정보도 함께 포함한다. 이러한 XML의 특성은 모든 형식의 데이터가 XML 형태로 기술될 수 있도록 해주며, 웹에서 운용되는 모든 데이터가 통합되어 저장 및 처리될 수 있는 기반을 제공한다.

현재 많은 정보시스템들이 XML을 기반으로 구축되어 다양한 XML 정보들이 인터넷을 통해 제공되고 있다. 특

이 논문은 한신대학교 학술연구비 지원에 의하여 연구되었음.
*교신저자 : 박성진(sjpark@hs.ac.kr)
접수일 09년 07월 28일 수정일 09년 10월 08일 게재확정일 09년 10월 14일

히, 소형 데이터 저장소로서의 XML 문서의 활용도가 증가하고 있으며 이에 따라 이러한 XML 문서를 효율적으로 저장하고 관리하는 저장소 즉, 리파지토리 시스템이 요구된다. 즉, XML의 구조적 특성을 효율적으로 활용하면서 대용량의 XML 데이터를 안정적으로 다룰 수 있는 저장 구조와 검색 기능을 갖는 통합 리파지토리 시스템이 필요하다[8].

한편, 통합 리파지토리에 저장된 XML 데이터는 여러 개의 XML 복사본으로 생성되어 다양한 정보환경에서 활용되며 이 과정에서 각 본사본마다 새로운 데이터의 추가와 기존 데이터의 수정이나 삭제와 같은 XML 문서의 변경이 발생할 수 있다. 이 경우, 변경되어 일시적인 불일치성(temporary inconsistency)이 발생한 XML 문서 복사본들을 일관성있게 저장할 수 있는 리파지토리의 버전 관리 기능이 필요하다. 즉, XML 문서들 간의 변경을 탐지하고 이를 리파지토리안에서 통합 관리하는 기능이 요구된다. 그러나, 기존 변경 탐지에 관한 연구들은 문서 단위의 XML 변경 탐지에만 집중되어 있으며 리파지토리와 연계한 변경 탐지 기법들은 제시되고 있지 않다.

본 논문에서는 XML 문서를 RDB로 임포트(import)하고 다시 RDB 데이터를 XML 문서로 엑스포트(export)하는 DBMS 독립적인 리파지토리 시스템과 이 과정에서 효율적으로 동작하는 XML 문서 변경 탐지 기법을 개발하였다. 개발된 XML 통합 리파지토리 시스템은 XML 문서와 RDB 테이블 구조를 중재함으로써 대량의 XML 문서의 저장 및 검색뿐만 아니라 문서에 대한 적절한 버전 관리도 할 수 있다. 제안한 기법은 계층형 XML 문서와 리파지토리안의 관계형 XML 데이터 사이의 이질적인 모델간의 동기화를 가능케하는 새로운 변경 탐지 기법으로 변경 탐지를 위한 데이터의 양을 최소화하기 위하여 엘리먼트 단위로 메시지 다이제스트 알고리즘을 적용하였다.

논문의 구성은 다음과 같다. 1장과 2장에서 기존 연구 내용들을 분석하고 3장에서는 변경탐지를 위한 논문에서 제시한 XML 리파지토리 스키마와 탐지 알고리즘에 대하여 설명하였다. 4장에서는 구현한 리파지토리 시스템의 예를 들어 설명하였으며 5장에 결론을 기술하였다.

2. 관련 연구

기존 연구 중 XML 문서 저장 리파지토리 시스템 구축 방법에 관한 첫 번째 유형으로 새로운 XML 전용 데이터베이스에 기반을 둔 저장 시스템[6]을 들 수 있는데 이 경우, 기존 데이터를 활용하기 위해서는 별도의 데이터 변환 작업 등이 필요하다는 문제점이 있다.

두 번째 방법은 XML-확장 데이터베이스(XML-Extended Databases)로 기존 데이터베이스 안에 XML 문서를 저장 관리할 목적으로 확장 모듈을 추가한 데이터베이스 시스템이다. XML-확장 데이터베이스들은 독립된 저장구조를 갖는 형태부터 기존 저장구조에 변환 기능을 새롭게 추가한 형태까지 다양하며 DB2, Informix, Oracle9i, Microsoft SQL-Server 등[9, 12]이 대표적인 예이다. 이 경우 기존 데이터베이스의 저장구조를 활용할 수 있는 이점이 있지만 데이터베이스마다 서로 다른 인터페이스를 제공하기 때문에 혼란을 줄 수 있다.

세 번째 방법은 기존 데이터베이스 시스템을 이용하여 XML 데이터를 저장하고 질의를 수행하는 방법[3, 4]이다. XML 문서를 저장하기 위한 적절한 테이블이나 클래스를 생성한 다음 미들웨어(middleware)나 랩퍼(wrapper)를 이용하여 XML에 대한 질의를 데이터베이스에 대한 질의어로 변환하여 원하는 결과를 얻는 방법으로 새로운 시스템을 구현하지 않고 원하는 결과를 생성할 수 있는 장점이 있다. 이 경우, 다시 관계형 데이터베이스 기반[1,2,7], 객체-관계형 데이터베이스 기반[11,14], 객체형 데이터베이스 기반[5]의 3가지 방식이 존재한다.

한편, XML 미들웨어는 XML 문서와 다양한 데이터베이스 상호간에 데이터를 전송하기 위해 사용하는 독립된 소프트웨어 모듈로 데이터베이스와 XML 문서 사이의 양방향 데이터 전송이 가능한 XML-DBMS[15]와 데이터베이스에서 XML 문서로의 전송만 가능한 DB2XML[20] 등이 대표적이다.

본 논문에서는 DTD 종속적인 스키마 구조와 엘리먼트 분할 방식에 기반을 둔 새로운 XML 리파지토리 시스템을 통하여 기존 관계형 데이터베이스를 활용하여 XML 문서로 효율적으로 저장하고 검색할 수 있도록 하였다. 특히, XML 문서를 저장하기 위해서 특별히 설계된 스키마 구조를 가지고 있어서 XML 문서의 내용뿐만 아니라 문서의 구조 정보 등도 같이 저장할 수 있으며 이를 바탕으로 XML 문서의 내용 검색과 입력, 수정, 삭제뿐만 아니라 변경 탐지도 효율적으로 할 수 있다.

한편, XML 문서가 웹 환경에서 데이터의 교환, 저장소로서 널리 사용됨에 따라 이러한 XML 문서들의 변경 탐지에 대한 많은 연구들이 수행되었는데 대표적인 기존 연구들로는 트리 구조에 관한 변경 탐지 알고리즘[4, 15, 16, 23] 등이 있다.

XyDiff[4]은 가장 먼저 제시된 알고리즘 중의 하나로 데이터웨어하우스안의 다량의 데이터를 위해 시간과 공간 측면에서 효율적인 순서화된 트리 모델에 기반을 둔 알고리즘을 제안하였다. X-diff[23]는 비순서 트리에 기반

을 둔 알고리즘을 제시하였고. diffX[16]는 순회를 통한 노드 쌍을 찾는 대신에 XML 트리안의 독립적인 단편들 간의 일치를 찾는데 집중하였다. KF-Diff[15]은 트리의 각 노드에 대해 루트부터의 유일한 경로에 기반하여 검색하는 방법을 연구하였다.

계층적 데이터에 대한 변경 탐지에 관한 대부분의 연구들[4,15,16,23]은 XML 문서를 순서 혹은 비순서 트리(ordered or unordered tree)로 표현한 다음 트리로 표현된 두 문서간에 서로 대응되는 노드들을 결정하고, 이에 대한 편집 연산 스크립트를 생성하여 새로운 문서 버전을 생성하는 다양한 형태의 트리구조와 휴리스틱 알고리즘을 제시하고 있다[17]. 경로매칭 알고리즘을 사용하여 서브트리의 이동, 복사와 같은 구조적 변화를 효율적으로 탐지하는 연구[10]도 이에 속한다.

그러나, 기존 연구들은 개별적인 XML 문서 파일의 계층형 트리 구조를 대상으로 하는 변경 탐지에만 집중하고 있어 관계형 모델에 기반한 XML 리파지토리 시스템과 연계하여 적용하기에 부적합하다. 또한, XML 문서들을 모두 트리 구조로 메모리 공간에 함께 올려놓고 변경 탐지가 이루어지므로 대용량의 XML 문서간 비교가 어려운 등의 확장성 문제점도 갖고 있다.

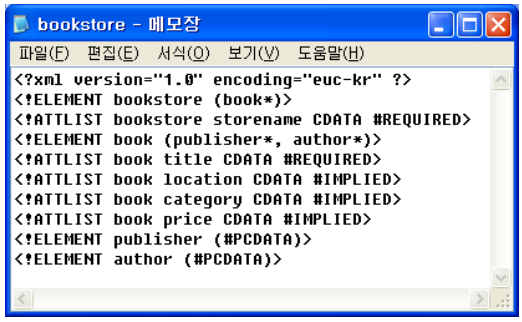
본 논문에서는 계층적 XML 문서를 리파지토리와 동일한 관계형 모델로의 변환을 통해 일괄적으로 변경 탐지를 함으로써 서로 다른 모델 구조를 갖는 XML 문서와 리파지토리 저장소 데이터간의 동기화를 일원화하여 수행하도록 하였고 RDB를 사용함으로써 메모리 제약의 문제점을 해결하였다.

3. 변경탐지 XML 리파지토리 시스템 설계

3.1 XML 리파지토리 스키마 구조

리파지토리안에 XML 문서들을 분할하여 저장하기 위하여 DTD 분석 결과에 따라 관계형 데이터베이스 안에 스키마 구조를 생성한다. DTD 기반 기존 스키마 생성 방식들[12,22]을 참고하여 제시한 리파지토리 스키마 구조는 XML 문서가 갖는 계층적인 구조적 특성을 관계형 데이터 모델의 외래키(foreign key)를 사용하여 엘리먼트간의 상하(직계) 관계를 일대 다의 참조 관계로 변환하여 표현하였다. 특히, XML 데이터의 수정이 가능하도록 문서를 엘리먼트 단위로 쪼개어 저장하는 분할저장 방식을 사용하였으며 엘리먼트를 관계형 데이터베이스의 테이블에 대응시키고, 애트리뷰트는 엘리먼트 테이블의 칼럼으로 매핑시켜 손쉽게 XML 문서간의 변경 탐지가 가능하도록 하였다.

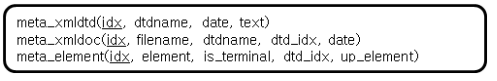
XML 문서를 관계형 데이터베이스에 저장할 때 XML 문서에 대한 메타 데이터와 엘리먼트 속성들에 대한 정보도 함께 저장할 수 있도록 그림 1과 같은 DTD 구조를 갖는 XML 문서에 대하여 그림 2, 그림 3, 그림 4와 같은 스키마 구조를 사용하여 저장하였다.



[그림 1] XML 문서의 DTD 예

- **메타 테이블 스키마**

우선, XML 문서에 대해 메타 정보를 입력할 수 있도록 메타 테이블을 생성한다. 그림 2에서 meta_xmldtd 테이블은 DTD 문서별로 고유한 id값과 DTD 문서의 이름, 입력시간, DTD 문서 전체 내용을 저장한다. meta_xmldoc 테이블의 경우, 입력한 XML 문서에 대한 고유한 id값, XML 문서의 파일명, 참조하는 DTD 인덱스 값과 DTD 문서명, 입력시간 등이 저장된다. 이를 통해 어떤 문서가 언제 입력되었는지 알 수 있으며, XML 문서를 복원하는 경우에도 사용된다. meta_element 테이블의 경우, 엘리먼트에 대한 구조 검색이 용이하도록 엘리먼트들 간의 구조정보를 저장하며 여기에는 각 엘리먼트의 고유한 id값과 엘리먼트명, 단말 노드 유무(단말이면 '1', 아니면 '0'), 참조하는 DTD 문서의 인덱스, 상위 엘리먼트에 대한 인덱스값을 저장한다.



[그림 2] 메타 테이블 스키마

- **엘리먼트 테이블 스키마**

XML 문서의 DTD를 파싱함으로써 그림 3과 같은 'et_'로 시작하는 엘리먼트 테이블(ET;Element Table)을 생성한다. 먼저, 유일(unique)한 엘리먼트명을 갖는 테이블이 엘리먼트 수만큼 생성되며 각 엘리먼트 테이블은 고유 인덱스

(idx) 컬럼과 상위 엘리먼트와의 연결을 위한 상위 인덱스(up_idx) 컬럼을 갖는다. 또한, XML 문서 정보에 대한 인덱스(xmldoc)와 DTD 정보 참조를 위한 인덱스(xmldtd)가 포함된다. 루트 엘리먼트의 경우, 상위 엘리먼트가 없으므로 상위 인덱스 값을 갖지 않아 별도의 et_rootelement 테이블에 저장된다. 각 엘리먼트의 애트리뷰트는 테이블의 컬럼명으로, 엘리먼트의 텍스트는 'text'라는 컬럼명으로 변환 생성되어 XML 문서 내용을 컬럼값으로 저장한다.

et_rootelement(idx, xmldoc, xmldtd)
et_*bookstore*(idx, up_idx, *storename*, xmldoc, xmldtd)
et_*book*(idx, upidx, *title*, *location*, *category*, *price*, xmldoc, xmldtd)
et_*publisher*(idx, up_idx, text, xmldoc, xmldtd)
et_*author*(idx, up_idx, text, xmldoc, xmldtd

[그림 3] 엘리먼트 테이블 스키마

- **다이제스트 테이블 스키마**

엘리먼트 테이블당 'dt_'로 시작하는 하나의 메시지다이제스트 테이블(DT;Digest Table)을 각각 생성한다. 여기에는 노드 다이제스트값 nMDV, 하위노드 메시지다이제스트값 nSDV, 문서 전체의 다이제스트값 tMDV, 노드와 하위노드들의 변경여부를 나타내는 nF, sF 플래그 칼럼이 포함된다.

dt_rootelement(idx, xmldoc, xmldtd, nMDV, tMDV, nF, tF)
dt_*bookstore*(idx, up_idx, xmldtd, xmldoc, nMDV, sMDV, nF, sF)
dt_*book*(idx, up_idx, xmldtd, xmldoc, nMDV, sMDV, nF, sF)
dt_*publisher*(idx, up_idx, xmldtd, xmldoc, nMDV, sMDV, nF, sF)
dt_*author*(idx, up_idx, xmldtd, xmldoc, nMDV, sMDV, nF, sF)

[그림 4] 다이제스트 테이블 스키마

3.2 다이제스트 테이블

XML 문서안의 데이터 즉, 엘리먼트 값들의 변화를 알기 위해서 엘리먼트 테이블의 각 튜플과 애트리뷰트를 모두 식별하여 직접 비교하지 않고도 메시지 다이제스트 결과값 하나만으로 엘리먼트 튜플 단위의 비교가 가능하다. 이 과정에서 다이제스트 테이블이 필요하다. MD4[14]는 보안 관련 표준 알고리즘의 하나로 일종의 해시 함수이다. 입력으로 다양한 크기의 메시지를 받아서 출력으로 유일한 128비트의 지문(fingerprint)형 메시지 다이제스트를 생성한다. 즉, 생성된 메시지 다이제스트 결과 값이 동일하면 원본 입력 데이터도 동일한 데이터로 간주할 수 있다. 이는 메시지 다이제스트 함수가 생성하는 결과 값이 입력 값에 따라 유일하기 때문이며 이러한 다이제스트 방법은 대량의 데이터가 소량의 데이터로 변경됨으로, 데이터의 내용 변형은 고려하지 않고 데이터의 변화 여부를 판단하는 방법으로 유용하다. 일반적으로는 보안을 위해 전송 데이터간의 변조 여부 판별을 위해 사용되지만 본 논문에서는 이 방법을 XML 엘리먼트 튜플 단위의 변경 탐지를 위한 메시지 다이제스트 테이블 생성에 적용하였다.

다이제스트 테이블은 리파지토리 안의 튜플들(XML 문서의 엘리먼트 정보들)을 하나의 메시지로 보고 메시지 다이제스트를 적용한 결과 값들을 저장하기 위한 저장소이다. 변경 탐지는 두 XML 문서의 동일한 DT 테이블 사이의 불일치 탐지를 의미한다. 먼저, X 엘리먼트를 위한 DT 테이블과 ET 테이블의 스키마는 다음과 같다.

<DT 테이블>
dt_*엘리먼트명*
(idx,up_idx,xmldtd,xmldoc,nMDV,sMDV,nF,sF)
<ET 테이블>
et_*엘리먼트명*(idx,up_idx,*애트리뷰트명1, 애트리뷰트명2,..., 애트리뷰트명n*,xmldoc, xmldtd)

이때, DT 테이블의 nMDV은 ET 테이블의 인덱스 관련 애트리뷰트를 제외한 나머지 애트리뷰트값들(*애트리뷰트명1, 애트리뷰트명2, ..., 애트리뷰트명n*)을 문자열 결합한 뒤, 이 값에 대해 MD4 알고리즘을 적용하여 얻은 128비트 해시값을 32바이트 문자열로 표현한 값으로 그림 5와 같이 정의할 수 있다.

$tMDV = hexstr(MD4(tMDV_1 \oplus \ldots \oplus tMDV_n))$
$nMDV = hexstr(MD4(av_5 \oplus av_6 \oplus \ldots \oplus av_m))$
$sMDV = hexstr(MD4(sMDV_1 \oplus \ldots \oplus sMDV_n))$
m : 애트리뷰트의 개수(릴레이션 ET의 degree)
av_k : k번째 애트리뷰트값
n : 자식 엘리먼트의 개수
$tMDV_k$: 루트 엘리먼트의 k번째 자식 엘리먼트의 서브트리 다이제스트값
$sMDV_k$: k번째 자식 엘리먼트의 서브트리 다이제스트값
$\oplus$: 문자열 결합 연산자
MD4(m) : 메시지 m을 다이제스트하는 함수
hexstr(h) : 128비트 해시값 h를 16진수 32바이트로 변환 문자열 함수

[그림 5] 메시지 다이제스트를 통한 엘리먼트 테이블 생성

nMDV들은 이전 동기화 이후 각 EDT 테이블간의 튜플 단위의 불일치 여부를 탐지하기 위해 사용된다. 한편, EDT 테이블 사이의 nMDV 비교시 불필요한 하위 노드들에 대한 비교횟수를 줄이기 위하여 변경이 발생한 하위 엘리먼트들에 대해서만 불일치 유무를 검색할 필요가 있다. 이를 위해 특정 엘리먼트 노드의 모든 하위 자손 노드 엘리먼트의 nMDV 값들을 포함하여 한꺼번에 다이제스트한 결과 값인 sMDV를 정의하면 그림 5와 같다.

이때, k번째 자식 엘리먼트의 서브트리 다이제스트값 $sMDV_k$는 다시 k번째 자식 엘리먼트를 부모노드로 하는 또 다른 n개의 손자노드들의 서브트리 다이제스트값 sMDV를 문자열 결합한 뒤 다시 다이제스트한 값이다. 따라서, 특정 엘리먼트들 간의 sMDV값들이 서로 같다면 하위 엘리먼트들 간에는 변경이 없으므로 하위 노드들에 대해서는 더 이상 동기화 여부를 판단하지 않아도 된다. 이 원리는 문서 전체를 다이제스트한 tMDV값들에도 동일하게 적용된다.

제시한 메시지 다이제스트 방법은 밀접한 연관성을 갖는 속성 노드들을 그룹핑하여 엘리먼트 단위로 다이제스트함으로써 개별 애트리뷰트 노드들의 해싱 비용을 줄이면서 하위 엘리먼트의 다이제스트값이 상위 엘리먼트에 재귀적으로 내포하도록 함으로써 변경 없는 하위 엘리먼트의 불필요한 비교도 방지하였다.

또한, 사용한 메시지 다이제스트 방법은 메시지 변조 방지에 사용되는 해시 알고리즘으로 서로 다른 엘리먼트끼리 같은 해시값을 가질 확률이 상대적으로 극히 적다. 여기에 엘리먼트 하위의 애트리뷰트 중에는 RDB의 기본키에 해당하는 유일(unique)한 칼럼 값이 반드시 포함되어 다이제스트 결과 값의 유일성을 더욱 강화하여 별도의 비교키 값이 없어도 대응하는 엘리먼트간의 변경 탐지가 가능하다.

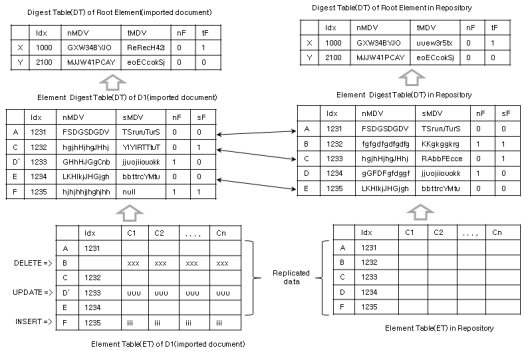
3.3 변경 탐지 알고리즘

제안하는 XML 문서 변경 탐지의 기본적인 전략은 엑스포트된 XML 문서들에서 수행된 데이터 변경을 탐지하여 임포트시에 변경된 내용들을 통합 리파지토리안에 동기화하여 저장하는 것이다.

리파지토리에서 XML 문서의 변경 탐지는 중복된 XML 문서들 간의 변경 여부를 탐지하여 특정 시점에서 통합된 하나의 XML 문서 버전을 생성해야 한다[19]. 예를 들어, 리파지토리안의 통합 A.XML 문서의 일부 내용들이 A1.XML, A2.XML, A3.XML에 중복저장되어 있는 경우, 이후에 변경된 A1'.XML, A2'.XML, A3'.XML 문서와의 동기화를 통해서 최신의 갱신 데이터를 갖는 A'.XML 문서를 생성할 수 있어야한다. 기존 연구들은 구조가 동일한 XML 문서의 변경 탐지를 통한 버전 관리에 집중되어 있으며 변경내용을 반영하기 위한 연산 스크립트 생성 등 복잡하고 추가적인 절차를 따라야 한다.

기존 방식대로 엘리먼트, 텍스트와 동일하게 애트리뷰트를 노드로 표현할 경우, 많은 노드로 인해 계층구조가 복잡해진다. 또한, 연관된 애트리뷰트 노드들을 독립적으로 변경탐지 비교를 하는 것은 엘리먼트 중심의 애트리뷰트들 간의 연관 관계를 충분히 활용하지 못해 비교연산이 불필요하게 증가한다. 이는 RDB안의 데이터 처리를 엔터티 단위를 고려하여 튜플 단위로 처리하지 않고 단순한 테이블 행과 열의 구조만을 고려하여 개별 셀의 컬럼값 단위로 처리하는 경우와 마찬가지이다. 따라서, 제안한 비교탐지 기법에서는 애트리뷰트를 단독 노드로 간주하지 않고 엘리먼트 서브 노드로 간주하여 엘리먼트는 튜플로, 엘리먼트 애트리뷰트는 칼럼으로 표현하는 엘리먼트 테이블(ET)을 사용하여 엘리먼트 단위에서만 비교가 이루어지도록 하였다. 이는 계층형 문서 모델과 관계형 문서 모델간의 자연스러운 변환이 가능하도록 하며 불필요한 애트리뷰트 노드간의 비교연산을 줄일 수 있다.

리파지토리안의 변경 탐지 알고리즘은 그림 6과 같은 ET 및 DT 테이블들을 대상으로 적용할 수 있다. 그림 6에서 DT는 그림 7의 DT 테이블 구축 단계를 통해 각 엘리먼트 튜플의 nMDV와 sMDV값이 하위 엘리먼트 테이블부터 유도되었고 상위 부모 엘리먼트의 sMDV값이 1(change)로 설정되어 변경내용이 있음을 가정하였다. EDT 테이블에는 추가로 2개의 nF, sF 비트 플래그 칼럼을 두어 XML 문서의 노드 데이터의 변경과 서브트리 노드 데이터의 변경 유무를 표시(0: unchanged, 1: changed)하였다.



[그림 6] ET와 DT테이블에 대한 변경 탐지 예

그림 6은 클라이언트 XML 문서에 3가지 변경(B노드는 삭제, D노드는 D'노드로 변경, F노드는 추가)이 발생한 경우를 반영한 예이다. 임포트된 XML 문서와 리파지토리안의 XML 데이터간에 동일한 DT 테이블의 nMDV값이 일치하면 해당 노드에 변경이 없음을, 추가로 sMDV값까지 일치하면 모든 하위 노드들에도 변경이 없음을 의미한다. 따라서, nF가 1이면 변경이 있음을 sF가 1이면 하위 노드 중에 변경이 있음을 의미한다. 각 튜플의 메시지 다이제스트 값 nMDV값이 동일하면 두 튜플의 엘리먼트 데이터는 동일하다.

1) 전처리(preprocess) 단계

상향식으로 최하위 단말 노드 Xi, Yi의 ET 테이블로부터 DT 테이블 생성을 시작한다. 각 DT 튜플의 nMDV값을 구하고 자식 DT 튜플들의 nMDV값을 합산하여 다이제스트하여 sMDV값을 생성하고 자식 DT가 없는 단말 DT 테이블 튜플의 경우에는 sMDV값으로 null값을 설정한다. 이 과정을 최상위 루트 노드 Xi, Yi DT 테이블까지 진행한다.

2) 변경탐지(change detection) 단계

클라이언트 쪽의 동기화 대상 문서를 D1, 리파지토리 쪽의 원본 문서를 D2라 하면 D1, D2의 최상위 Xi, Yi DT 테이블부터 하향식으로 비교를 시작한다. 두 문서 D1, D2에 속한 Xi, Yi DT 테이블의 각 튜플에 대해서 만약 부모 DT 튜플이 없다면 Xi와 Yi DT 테이블 튜플사이에 일치하는 nMDV값이 존재하는지 비교한다. 만약, 있다면 각 Xi와 Yi 튜플의 nF에 0으로 설정(노드 비변경)하고 Xi와 Yi DT 테이블 튜플 사이에 일치하는 tMDV값이 존재하는지 비교하여 있다면 각 Xi와 Yi 튜플의 tF에 0설정(비변경)하고 없다면 각 Xi와 Yi DT 테이블 튜플의 tF에 1로 설정(변경)한다. 한편, 일치하는 nMDV값이 존재하지 않는다면 각 Xi와 Yi 튜플의 nF에 1로 설정(변경)하고 Xi와 Yi DT 테이블 튜플 사이에 일치하는 tMDV값이 존재하는지 비교하여 있다면 각 Xi와 Yi 튜플의 tF에 0설정(비변경)하고 없다면 각 Xi와 Yi 튜플의 tF에 1설정(변경)한다. 이 경우, 클라이언트 새로운 엘리먼트 자료가 삽입되었거나 혹은 서버에 기존 엘리먼트가 삭제된 경우이다.

만약 부모 DT 튜플이 있다면 부모 DT 튜플의 sF가 0이면 nF와 sF를 0으로 설정(서브트리 비변경)한다. 그리고, 부모 DT 튜플의 sF가 1이면 Xi와 Yi 튜플사이에 일치하는 nMDV값이 존재하는지 비교한다. 만약 존재한다면 각 Xi와 Yi 튜플의 nF에 0으로 설정(노드 비변경)하고 Xi와 Yi DT 테이블 튜플 사이에 일치하는 sMDV값이 존재하는지 비교하여 있을 경우에는 각 Xi와 Yi 튜플의 sF에 0으로 설정(비변경)하고 없을 경우에는 Xi와 Yi 튜플의 sF에 1로 설정(변경)한다.

만약, 일치하는 nMDV값이 존재하지 않는다면 각 Xi와 Yi 튜플의 nF에 1을 설정(변경)하고 Xi와 Yi DT 테이블 튜플 사이에 일치하는 sMDV값이 존재하는지 비교한다. 만약, 있다면 각 Xi와 Yi 튜플의 sF에 0으로 설정(비변경)하고 없다면 각 Xi와 Yi 튜플의 sF에 1로 설정(변경)한다.

```
[Definition]
Xi : 문서 D1의 임의의 노드
Yi : 문서 D2의 임의의 노드
ET(Xi), DT(Xi) : 노드 Xi에 대응하는 ET테이블,DT테이블
ET(Yi), DT(Yi) : 노드 Yi에 대응하는 ET테이블,DT테이블
DTj(Xi) : Xi 대응 DT 테이블의 j번째 튜플
DTj(Yi) : Yi 대응 DT 테이블의 j번째 튜플
ParentOf(t) : 튜플 t의 부모 튜플
t.av : 튜플 t의 애트리뷰트 av의 값
ParentOf(n) : 노드 n의 부모 노드
ChildOf(n) : 노드 n의 자식 노드
RootOf(n) : 노드 n의 최상위 부모노드(루트노드)
TerminalOf(n) : 노드 n의 최하위 자식노드(단말노드)

[step1_preprocess]

Xi=TerminalOf(Xi);
Create DT(Xi) from ET(Xi);
while (RootOf(Xi) ≠ Xi)
{
  DTj(Xi).nMDV = hexstr(MD4(⊕n DTj(Xi).avn))
  DTj(Xi).sMDV = hexstr(MD4(⊕j DTj(ChildOf(Xi)).sMDV)
  if (DTj(ChildOf(Xi)) == null)
     DTj(Xi).sMDV = null;
  Xi=ParentOf(Xi);
  Generate DT(Xi) from ET(Xi);
 }
 DTj(Xi).nMDV = hexstr(MD4(⊕n DTj(Xi).avn))
 DTj(Xi).tMDV = hexstr(MD4(⊕j DTj(ChildOf(Xi)).sMDV)
Yi=TerminalOf(Yi);
Create DT(Yi) from ET(Yi);
while (RotOf(Yi) ≠ Yi )
{
   DTj(Yi).nMDV = hexstr(MD4(⊕n DTj(Yi).avn))
   DTj(Yi).sMDV = hexstr(MD4(⊕j DTj(ChildOf(Yi)).sMDV)
   if (DTj(ChildOf(Yi)) == null)
      DTj(Yi).sMDV = null;
   Yi=ParentOf(Yi);
   Create DT(Yi) from ET(Yi);
 }
 DTj(Xi).nMDV = hexstr(MD4(⊕n DTj(Xi).avn))
 DTj(Xi).tMDV = hexstr(MD4(⊕j DTj(ChildOf(Xi)).sMDV)
```

[그림 7] 변경탐지 알고리즘(전처리 단계)

3) 후처리(postprocess) 단계

D1, D2의 최상위 Xi, Yi DT 테이블부터 하향식으로 통합을 시작한다. D2의 Yi DT 테이블 중에서 nF가 1로 설정된 튜플들을 삭제하고 D1의 Xi DT 테이블 중에서 nF가 1로 설정된 튜플들을 Yi DT 테이블에 추가한다. 이와 같은 과정을 하위 자손 DT 테이블에 대해서도 동일하게 튜플 삭제와 추가 과정을 반복 적용한다.

```
[step2_change detection]

Xi=RootOf(Xi); Yi=RootOf(Yi);
while (ChildOf(Xi) ≠ null and ChildOf(Yi) ≠ null)
{
 if (DTj(Parent_Of(Xi)) ≠ null and DTj(Parent_Of(Yi)) ≠ null)
 {
   if (Parent_Of(DTj(Xi)).sF==0 and Parent_Of(DTj(Yi)).sF==0)
      DTj(Xi).nF = 0; DTj(Xi).sF = 0;
      DTj(Yi).nF = 0; DTj(Yi).sF = 0;
  else if (Parent_Of(DTj(Xi)).sF == 1 and
              Parent_Of(DTj(Yi)).sF == 1)
  {
    if (DTj(Xi).nMDV == DTj(Yi).nMDV)
      DTj(Xi).nF = 0; DTj(Yi).nF = 0;
      if (DTj(Xi).sMDV == DTj(Yi).sMDV)
       DTj(Xi).sF = 0; DTj(Yi).sF = 0;
      else
       DTj(Xi).sF = 1; DTj(Yi).sF = 1;
    else
      DTj(Xi).nF = 1; DTj(Yi).nF = 1;
      if (DTj(Xi).sMDV == DTj(Yi).sMDV)
       DTj(Xi).sF = 0; DTj(Yi).sF = 0;
     else
      DTj(Xi).sF = 1; DTj(Yi).sF = 1;
           }
   }
 else if (DTj(Xi).nMDV == DTj(Yi).nMDV)
 {
    DTj(Xi).nF = 0; DTj(Yi).nF = 0;
    if (DTj(Xi).tMDV == DTj(Yi).tMDV)
      DTj(Xi).tF = 0; DTj(Yi).tF = 0;
    else
      DTj(Xi).tF = 1; DTj(Yi).tF = 1;
  }
  else
  {
     DTj(Xi).nF = 1; DTj(Yi).nF = 1;
     if (DTj(Xi).tMDV == DTj(Yi).tMDV)
       DTj(Xi).tF = 0; DTj(Yi).tF = 0;
     else
       DTj(Xi).tF = 1; DTj(Yi).tF = 1;
  }
Xi=ChildOf(Xi); Yi=ChildOf(Yi);
}
```

[그림 8] 변경탐지 알고리즘(변경탐지 비교 단계)

```
[step3_postprocess]

Xi = RootOf(Xi);
Yi = RootOf(Yi);
while (ChildOf(Xi) ≠ null or ChildOf(Yi) ≠ null )
{
    if (DTj(Yi).nF == 1)
        DT(Yi) - {DTj(Yi)};
    if (DTj(Xi).nF == 1)
        DT(Yi) U {DTj(Xi)};
    Xi = ChildOf(Xi);
    Yi = ChildOf(Yi);
 }
```

[그림 9] 변경탐지 알고리즘(후처리 단계)

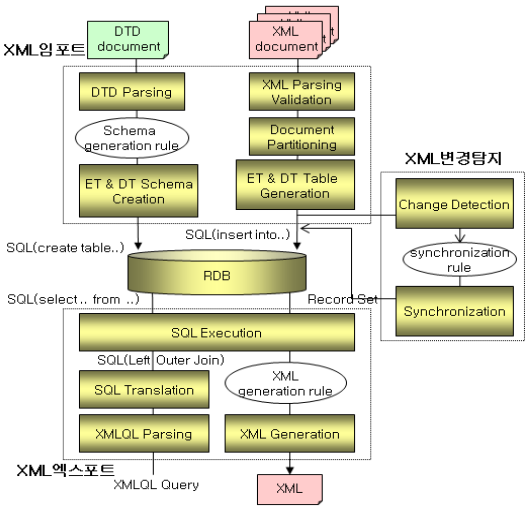
그림 7, 그림 8, 그림 9는 그림 6과 같은 DT간의 불일치 즉, 충돌이 발생했을 경우, 각 DT에 튜플이 새로 추가되거나 기존 튜플이 수정, 삭제되었는지 여부에 대해 순차적으로 테스트하고 메시지 다이제스트값을 사용하여 변경을 탐지하고 동기화시켜가는 과정들을 보여준다. 제안한 변경탐지 알고리즘에서 ET와 DT사이의 동기화는 실질적인 동기화 작업의 사전 준비 과정에 해당함으로 동기화 작업과 분리되어 실행시킬 수 있어 동기화 운영의 융통성을 제공하고 동기화 시간을 단축시킬 수 있다.

4. 변경 탐지 XML 리파지토리 시스템 구현

4.1 시스템 구조

본 논문에서 설계 및 구현한 XML 리파지토리 시스템은 DBMS에 독립적으로 어떠한 관계형 데이터베이스에도 분할 저장가능하고 다시 자유로운 검색질의를 통해 검색 결과를 완전한 XML 문서 형태로 반환할 수 있다. 즉, 관계형 데이터베이스 안에 저장된 XML 문서 내용을 다시 기존의 XML 파일로 복원할 수 있어 XML 파일과 데이터베이스 간에 자유로운 상호 변환이 가능하다.

XML 리파지토리 시스템은 그림 10과 같이 구성된다. 먼저, XML 문서를 데이터베이스 안에 분할 저장하기위해 적절한 스키마를 생성한 뒤, XML 문서 내용을 임포트하는 XML 임포트 모듈과 분할 저장된 XML 문서에 대한 검색 질의를 처리하고 그 결과를 새로운 XML 문서로 엑스포트하는 XML 엑스포트 모듈, 동기화를 수행하는 변경탐지 모듈로 구성된다.



[그림 10] XML 리파지토리 시스템의 구성

XML 문서 임포트 모듈은 다양한 XML 어플리케이션으로부터 생성된 XML 문서 및 DTD를 리파지토리로 입력하는 역할을 담당한다. 특히, 기본적으로 DTD와 XML 문서

의 로딩(loading) 기능뿐만 아니라 XML 문서에 대한 파싱(parsing)과 적합성(validation) 검사도 수행한다. 이는 여러 엘리먼트로 구성된 XML 문서를 파싱하고 엘리먼트 단위로 문서를 처리해 주기 위해서 필요한 기능이며, 추가적으로 적합한 문서를 검증해주기 위해서 필요하다. 파싱에는 XML 문서의 파싱뿐만 아니라 DTD 문서의 파싱 기능도 있어야 한다. XML 임포트 모듈은 DOM 파서를 사용하여 XML 문서 파일을 해석하고 적절한 테이블에 매핑하여 XML 엘리먼트 값들을 저장한다.

임포트모듈의 핵심은 XML 문서의 분할 저장 기능이다. 이는 저장소의 기본적인 기능으로 추후 검색 요청이나 문서의 변경요청이 있을 경우 이를 처리해 주기 위해 XML 데이터를 저장하는 기능이다.

먼저, 임의의 XML 문서를 파싱한다. 우선 메타 정보를 메타 테이블(meta_xmldoc, meta_xmldtd, meta_element)에 입력하고 메타 정보 입력 후 문서 구조에 맞는 각각의 엘리먼트 테이블에 XML 문서 내용을 분할하여 저장한다. 이를 위해서는 저장되어질 스키마 구조가 데이터베이스 안에 미리 생성되어 있어야 하는데 그림 3에서 보는 바와 같이 각 엘리먼트는 별개의 테이블로 생성된다. 실제 엘리먼트 테이블간의 구조적인 연결은 없지만 구조적인 정보가 meta_element 테이블에 존재하므로 이를 이용하여 각 테이블에 매핑되어 저장되어진다. XML 문서의 내용 정보는 내부적으로는 관계형 데이터베이스 언어인 SQL의 insert 구문의 형태로 변환하여 저장하게 되는데 동일한 DTD 구조를 갖는 여러 개의 XML 문서는 동일한 테이블 내에 저장이 가능하고, 다른 DTD를 갖는 XML 문서가 있 경우에만 새로운 구조를 추가적으로 만들어 사용한다.

XML 문서 엑스포트 모듈은 데이터베이스에 저장된 XML 문서를 다시 복원하는 기능을 한다. 이는 XML 문서를 파싱하는 과정에서 생성된 메타 정보를 사용하여 저장했던 원래의 XML 문서를 복원할 수 있을 뿐만 아니라 적절한 질의를 통해 일부만을 별도의 XML 문서로 생성할 수 있음을 뜻한다.

임포트할 당시에 명세한 XML 문서명을 지정해주면 임포트하기 이전의 XML 문서 파일 형태와 동일하게 복원할 수 있다. 이때, 관계형 데이터베이스안의 분할 저장되었던 엘리먼트 단위의 문서 정보들은 모두 삭제된다. 즉, 저장소로부터 XML 문서를 추출해내는 기능을 수행한다.

XML 변경 탐지 모듈은 3.3절의 변경 탐지 동기화 알고리즘을 DT에 적용함으로써 통합 리파지토리에 임포트하는 XML 문서의 변경 유무를 식별하고 리파지토리안의 XML 데이터에 변경 내용을 반영하는 동기화 기능을 수행한다.

4.2 XMLQL

기존 XML 질의어인 XQL 등은 XML 문서를 대상으로 선택, 추출, 재구성, 조합 등의 다양한 검색 기능과 풍부한 질의 기능을 지원하는 대신 상대적으로 복잡한 질의 처리를 요구하고 있으며 기존 SQL 사용자들에게는 익숙하지 않다. 본 논문에서는 리파지토리 인터페이스로서 기존 SQL과 유사하면서도 질의 처리가 복잡하지 않고 XML 데이터에 대한 CRUD(Create Read Update Delete)까지 지원하는 XMLQL(XML Query Language)를 제안하였다.

XMLQL은 엘리먼트, XML 문서 등의 내용(content) 부분에 대한 검색 기능, 부모/자식/형제/조상/후손 등의 구조 정보를 이용한 검색 기능, 속성의 이름 및 값에 대한 검색 기능 등을 제공하며 이외에도 저장된 DTD의 삭제, 특정 XML 문서의 삭제, 엘리먼트와 애트리뷰트의 추가, 삭제, 수정도 가능하다. 이 모듈에서는 show절에 의해 명세된 검색 요청을 처리하고 그 결과를 원하는 형태의 XML문서로 만들어 주는 기능을 한다.

XMLQL 처리 모듈은 분할되어 데이터베이스에 들어있는 XML 문서 정보들에 대해 그림 11과 같은 show절

```
<query specification> ::= <show clause> <on clause> [ <inxml clause> | <indtd clause> ]
<show clause> ::= SHOW <element list>
<element list> ::= <element sublist> [ { <left paren><element sublist><right paren> } . . . ]
<element sublist> ::= <element columnlist> [ { <comma> <element columnlist> } . . . ]
<element columnlist> ::= <element name> <period> [ <asterisk> | <element column> ]
<element column> ::= <attribute name> | <text>
<on clause> ::= ON <search condition> [ { <and> <search condition> | <or> <search condition> } ]
<search condition> ::= <element name> <period> <element column> <predicate> <value>
<predicate> ::= > | < | >= | <= | <> | = | LIKE
<inxml clause> ::= INXML <document name>
<indtd clause> ::= INDTD <document name>
```

[그림 11] XMLQL 구문 정의(show구문 일부)

로 시작하는 XMLQL 언어를 처리한다.

그림 11에서 정의한 show절은 출력할 엘리먼트 정보들을 명세하며 on절은 찾고자 하는 엘리먼트에 대한 검색 조건을, inxml절과 indtd절은 각각 검색하고자 하는 문서를 명세함으로써 저장된 문서중 특정 문서로만 검색을 제한할 수 있다. 그림 12와 같은 XMLQL 질의 구문을 실행하면 그림 17과 같은 XML 문서가 생성된다.

```
Show    극장.*( 영화.영화명( 영화정보.*)),극장위치.*( 주소.*, 교통.text))
On      영화.영화명='집으로'
InDTD   영화정보
```

[그림 12] XMLQL 사용 예

XMLQL은 리파지토리가 RDB에 구축되어 있어 내부적으로는 SQL로 변환되어 처리되어야하는데 분할되어 저장된 XML 데이터들을 하나의 XML 문서로 통합하여 엑스포트하기 위해서는 엘리먼트 테이블간의 외래키들을 이용한 죠인을 통해서만 가능하다. 그림 13은 그림 12와 같은 XMLQL 질의 구문이 처리되기 위하여 내부적으로 여러 Left Outer Join을 사용하는 여러 개의 서브 SQL 질의 구문으로 변환됨을 보여준다.

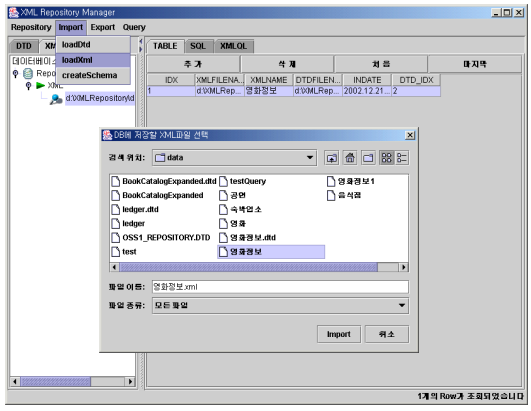
```
SELECT distinct 극장.*, 영화.영화명, 영화정보.*, 극장위치.*, 주소.*, 교통.text
FROM ((((
  극장 LEFT OUTER JOIN 영화       ON 극장.idx=영화.up_idx  )
       LEFT OUTER JOIN 영화정보 ON 영화.idx=영화정보.up_idx )
       LEFT OUTER JOIN 극장위치  ON 영화.idx=극장위치.up_idx
       LEFT OUTER JOIN 주소       ON 극장위치.idx=주소.up_idx )
      LEFT OUTER JOIN 교통       ON 극장위치.idx=교통.up_idx )
Where 영화.영화명='집으로'
```

[그림 13] XMLQL의 내부 변환처리 예

4.3 시스템 사용 예

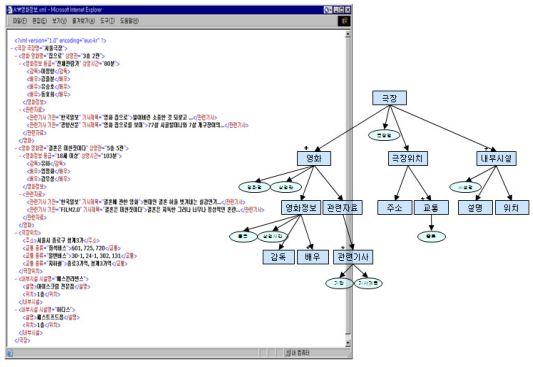
XML 리파지토리 시스템의 구현 환경은 윈도우즈 환경에서 자바 언어를 사용하여 개발되었으며 JAXP XML 파서와 JDBC 미들웨어 그리고, 오라클 DBMS를 사용하여 개발되었다. 개발된 시스템은 크게 클라이언트 패키지와 서버 엔진 패키지로 구성되었다.

클라이언트 패키지를 구성하는 클래스 모듈들은 DBExplorer, JDBEMenuListener, DBList, DBNode, DBTreeCellRenderer, SQLPanel, TablePanel, TreePanel들이며 이들 클래스들을 통해 관리자가 관계형 데이터베이스내의 XML 문서를 관리하고 각 문서에 대한 메타 정보를 확인하거나 설정할 수 있다. 기본적인 검색 기능과 저장소 관리를 위한 인터페이스를 제공한다. 서버 패키지를 구성하는 클래스 모듈은 XMLSQL, XMLImport, XMLQuery, XMLExport, toDOC 등이다. 이들 클래스들은 oraxml이라는 서버 패키지를 구성하여 클라이언트로부터 XML 문서의 저장 및 검색 요청을 처리한 뒤 그 결과를 XML 문서로 생성하여 제공한다.



[그림 14] XML 임포트 실행 화면

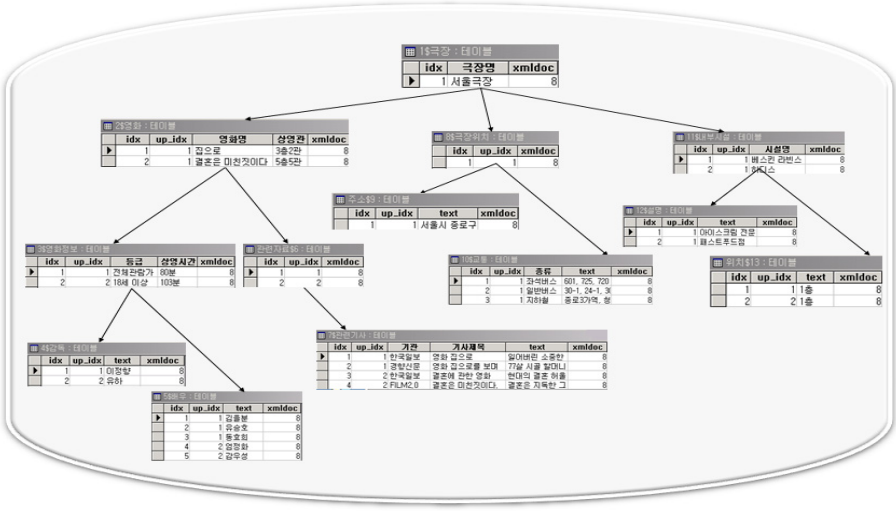
그림 14는 engine 클래스에 있는 XMLimport를 실행하는 화면으로 임포트 모듈을 구현한 기능을 보여준다.



[그림 15] '영화정보.xml'의 내용 및 구조

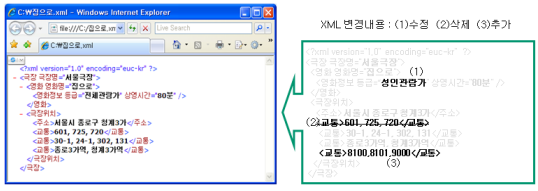
그림 15와 같은 문서 내용과 계층적 구조를 갖는 '영화정보.xml' 파일을 그림 14와 같이 통합 리파지토리에 임포트하면 그림 16과 같이 리파지토리안의 미리 생성된 엘리먼트 테이블에 엘리먼트 단위로 분할되어 저장되게 된다.

그림 16과 같이 리파지토리 스키마안에 저장된 영화정보.XML 문서에 대하여 그림 12와 같이 특정 영화에 대한 부분 정보를 검색하여 엑스포트하게 되면 그림 17에서 보는 바와 같이 '집으로'영화와 관련된 정보만을 갖는 XML 문서가 생성되게 된다. 이후, 이와 같이 리파지



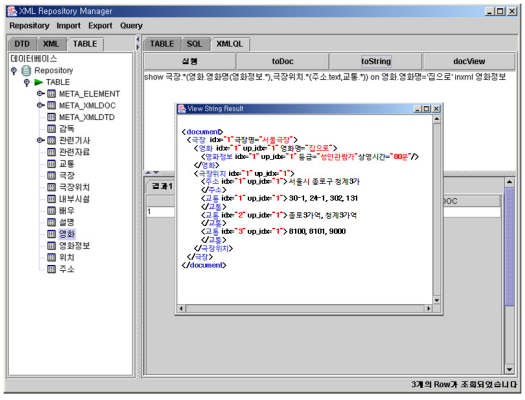
[그림 16] '영화정보.xml'의 리파지토리 저장 구조

토리로부터 엑스포트된 XML 문서에 대해서 그림 17과 같이 수정이나 삭제 혹은 추가와 같은 XML 문서의 내용을 변경할 수 있다.



[그림 17] 검색 결과로 생성된 XML 문서와 변경 내용

내용이 변경된 '집으로.XML'을 임포트하게 되면 변경 탐지 모듈이 작동하여 임포트되는 문서에 대한 메시지 다이제스트를 통한 DT 테이블을 생성하게 된다. 새로운 DT 테이블들은 기존의 리파지토리안의 원본 XML 데이터에 대한 다이제스트값들과의 변경탐지 과정을 통하여 변경된 데이터가 있는지를 확인하게 되고 변경된 데이터에 대해서는 동기화가 수행된다. show 검색 구문을 사용하여 해당 결과를 XML 문서로 받아보기 위해서는 export 기능을 선택하여 저장될 XML 문서명을 지정하면, 이를 바탕으로 XML 문서를 생성하게 된다. XMLexport를 실행한 그림 18을 통해서 그림 17과 같은 '집으로.xml' 문서를 생성했던 그림 12와 같은 동일한 XMLQL을 다시 실행한 결과가 이전과는 달라졌음을 보여준다. 즉, 새롭게 임포트된 '집으로.xml'의 변경 내용을 적절히 탐지하고 동기화함으로써 리파지토리안에 새롭게 변경된 '영화정보.xml' 버전이 저장되어 관리되고 있음을 알 수 있다.



[그림 18] XML 엑스포트 실행 화면

실행 화면을 통하여 본 논문에서 제시한 메시지 다이제스트 알고리즘을 활용한 XML 문서의 변경 탐지 기법이 정확하게 동작함을 확인할 수 있으며 개발한 DBMS 독립적인 리파지토리 시스템이 계층적인 XML 문서의 효율적인 저장과 검색뿐만 아니라 동기화 기능까지도 갖춘 효율적인 통합 리파지토리 시스템임을 확인할 수 있다.

5. 결론

본 논문에서는 클라이언트의 데이터 저장소로서 XML 문서의 사용이 증가함에 따라 대량의 XML 문서를 기존의 관계형 데이터베이스를 활용하여 효율적으로 저장하고 검색할 수 있는 XML 리파지토리 시스템을 설계 및

구현하였다. 개발한 시스템은 XML 문서를 RDB로 임포트(import)하고 다시 RDB 데이터를 XML 문서로 엑스포트(export)하는 기능을 제공한다.

이를 위하여 XML 문서를 엘리먼트 단위로 분할 저장할 수 있는 효율적인 저장 스키마 구조를 제시하였고 제시한 저장 구조는 내용 검색 및 구조 검색이 용이할 뿐만 아니라 본래의 XML 문서로의 복원이 가능하다는 이점을 갖는다. 또한, 분할 저장된 XML 문서에 대해 자유로운 검색뿐만 아니라 추가, 수정, 삭제까지도 가능한 SQL과 유사한 형태의 XMLQL 질의를 독자적으로 제공함으로써 사용자에게 편리하고 효율적인 인터페이스를 제공하였다.

또한, 임포트되는 XML 문서와 리파지토리 데이터와의 동기화 문제가 이슈화되고 있으나 계층적인 XML 문서 구조와 관계형 데이터베이스 테이블의 구조가 상이함에 따라 기존 XML 동기화 기법이나 데이터베이스 동기화 기법을 직접 적용하기 어려운 문제점이 있다. 본 논문에서는 이질적인 두 모델간의 데이터를 동기화시킬 수 있는 엘리먼트 단위의 저장구조를 갖는 DBMS 독립적인 데이터 변경 탐지 모듈을 구현하였다.

구현한 변경 탐지 기법은 튜플 단위의 메시지 다이제스트 결과를 충돌을 탐지하기 위한 정보로 사용함으로써 이전 버전 없이도 변경여부를 탐지하고 동기화시킬 수 있다. 또한, 엘리먼트 단위정보에 기반하여 검색 대상을 축소하고 변경이 없는 하위 엘리먼트 노드들에 대한 불필요한 검색도 줄임으로써 동기화 과정을 단순화하였다.

개발된 XML 리파지토리 시스템은 XML 문서와 RDB 테이블 구조를 중재함으로써 대량의 XML 문서를 관계형 데이터베이스에 저장할 수 있고 저장된 문서에 대한 자유로운 검색이 가능할 뿐만 아니라 검색 결과 역시 XML 형태로 제공하며 DBMS의 종류에 상관없이 독립적으로 실행된다는 특성이 있다. 또한, XML 변경 탐지 모듈을 통하여 다양한 환경에서 중복된 XML 데이터간의 일치를 요구하는 많은 응용 프로그램을 위한 효과적인 데이터 저장소로서 개발한 리파지토리 시스템이 활용될 수 있다.

참고문헌

[1] Daniela Florecu and Donald Kossmann, "A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database," INRIA TR, 1999.

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

[3] Gordana Pavlovic-Lazetic, "Native XML Databases vs. Relational Databases In Dealing With XML Document," Kragujevac J. Math. Vol. 30, pp181−199, 2007.

[4] Gregory Cobena, Serge Abiteboul, Amelie Marian, "Detecting Changes in XML Documents," Proc. of the 18th Int'l conf. on Data Engineering, pp. 41-52, 2002

[5] James Rumbaugh, Object-Oriented Modeling and Design, Prentice Hall, 1991.

[6] Jason McHugh, Serge Abiteboul, Roy Goldman, Dallan Quass and Jennifer Widom, "Lore: A Database Management System for Semistructured Data," SIGMOD Record 26(3), pp. 54-66, 1997.

[7] Jayavel Shanmugasundaram, Eugene Shekita, Jerry Kiernan, Rajasekar Krishnamurthy, Efstratios Viglas, Jeffrey Naughton and Igor Tatarinov, "A General Technique for Querying XML Documents Using a Relational Database System," SIGMOD Record 30(3), pp .20-26, 2001.

[8] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt and Jeffrey F. Naughton, "Relational Databases for Querying XML Documents : Limitations and Opportunities," VLDB 99, pp. 302-314, 1999.

[9] Josephine Cheng, Jane Xu, "XML and DB2," ICDE 2000, pp. 569-573, 2000.

[10] Kyong-Ho Lee, Yoon-Chul Choy, Sung-Bae Cho, "An Efficient Algorithm to Compute Differences between Structured Documents," IEEE Trans. Knowl. Data Eng. Vol.16 No.8, pp. 965-979, 2004

[11] Michael. Stonebraker, Object-Relational DBMSs : The Next Great Wave, Morgan Kaufmann Publishers, 1996.

[12] Microsoft Coroperation, "Microsoft SQL Server 2000," http://www.microsoft.com/sql/default.asp, 2000.

[13] Raihan Al-Ekram, Archana Adma, Olga Baysal: diffX: an algorithm to detect changes in multi-version XML documents. CASCON, pp. 1-11, 2005.

[14] Rivest, R., "The MD4 Message Digest Algorithm," RFC 1320, MIT and RSA Data Security,(1992).

[15] Ronald Bourret, "XML-DBMS," http://www.informatik.tudarmstadt.de/DEVS1/staff/bourret/xmldbms/readme.html, 2000.

[16] S. S. Chawathe and H. Garcia-Molina, "Meaningful Change Detection in Structured Data," In Proc. Of

SIGMOD, pp. 26-37, 1997.
[17] Shu Yao Chien, Vassilis J. Tsotras, Carlo Zaniolo, "XML Document Versioning," ACM SIGMOD Record, Vol. 30, pp. 46-53, 2001.
[18] Takeyuki Shimura, Masatoshi Yoshikawa and Shunsuke Uemura "Storage and Retrieval of XML Documents Using Object-Relational Databases" DEXA 99, pp. 206-217, 1999.
[19] Tancred Lindholm, "A three-way merge for XML documents," Proc. for the 2004 ACM symposium on Document Engineering, pp. 1-10, 2004.
[20] Volker Turau, "DB2XML 1.3," http://www.informatikfhwiesbaden.de/~turau/DB2XML/index.html, 2000.
[21] W3C, "Extensible Markup Language(XML) 1.0," http://www.w3.org/TR/1998/REC-xml19980210.html, 1998.
[22] Yasser Abdel Kader, Barry Eaglestone, “Siobhán North: An Analysis of Relational Storage Strategies for Partially Structured XML,” WEBIST, pp. 165-170, 2008.
[23] Yuan Wang, David J. Eewitt, Jin-yi Cai, "X-Diff : An Effective Change Detection Algorithm for XML Documents," Proc. of the 19th Int'l conf. on Data Engineering, pp. 519-530, 2003.

박 성 진(Seong-Jin Park) **[정회원]**

- 1993년 2월 : 고려대학교 일반대학원 전산과학과(이학석사)
- 1998년 2월 : 고려대학교 일반대학원 전산과학과(이학박사)
- 1998년 3월 ~ 2000년 2월 : 한국전자통신연구원 선임연구원
- 2000년 3월 ~ 현재 : 한신대학교 컴퓨터공학부 부교수

<관심분야>
데이터웨어하우징, 모바일데이터베이스, OLAP 등