내장형 XML 데이터베이스 시스템을 위한 P-DOM의 설계 및 구현

강동완[†] · 제권엽[†] · 홍영표[††] · 한동원[†††] · 강현석[††††] · 배종민[††††]

요 약

이동형 단말 시스템에서 XML 데이터의 역할과 데이터베이스의 지원이 중요해지고 있다. 이에 따라, 내장형 XML 데이터베이스 시스템에 대한 연구가 활발하다. XML 문서를 처리하기 위해서는 DOM API를 지원할 필요가 있는데, 기존의 DOM API는 DOM 트리를 메모리에 유지한다. 이것은 제한된 리소스를 가지는 내장형 시스템을 안정적으로 운용하는데 위협적인 요소이다. 본 논문에서는 내장형 시스템의 부족한 리소스를 고려하여 DOM 트리를 메모리가 아닌 내장형 데이터베이스 시스템인 버클리 DB 위에서 구성되는 P-DOM 트리를 제안하고, 이를 관리하는 DOMdbm 시스템을 개발한다.

Design and Implementation of P-DOM for Embedded XML Database System

Dong-Wan Kang[†], Gweon-Yeop Je[†], Yeong-Pyo Hong[††], Dong-Won Han[†††], Hyun-Syug Kang[††††] and Jong-Min Bae[††††]

ABSTRACT

The importance of XML data and their database supports in a mobile terminal system is on the increase. It leads to active research for embedded XML database systems. In order to handle XML documents, DOM API should be supported. The existing DOM API is not suitable for the embedded system with limited resource because the DOM tree requires a large space on main memory. Considering poor resource of embedded systems, we present persistent DOM which is implemented on Berkeley DB and DOMdbm which manages it.

Key words: 내장형 시스템, 내장형 데이터베이스, XML, Berkeley DB, DOM

1. 서 론

컴퓨터와 무선 통신 기술의 발전으로 이동 컴퓨팅 응용들이 주목을 받고 있다. 즉, 컴퓨터와 무선 통신 기술이 효과적으로 연계되어 언제, 어디서든지 이동 환경에서 정보 교환 및 처리가 가능하게 되었다. 최근에는 이러한 이동 통신 환경에서 활용되고 있는 정보 단말들의 성능이 크게 향상되고 있다. 이에 따라 다양한 무선 인터넷 응용들이 가능해지면서, 이러한 응용들의 요구에 부응하기 위해 정보 단말 자체에도 데이터베이스 시스템이 탑재되고 있다[1].

한편, 인터넷 환경에서 XML 문서의 사용이 크게 늘어나면서, XML 문서 데이터베이스에 대한 연구가 활발하다[2-4]. 이때 주로 XML 문서는 관계형 데이터베이스[5,6], 객체 지향 데이터베이스[7], 전용 파일 시스템[8] 등을 이용하여 관리된다.

최근에는 이러한 추세들이 결합되어 PDA 등 소규모 정보 단말에서도 독자적으로 XML 문서를 관리하고, 경우에 따라 서버에서 관리되는 데이터를 가져와

접수일 : 2002년 8월 28일, 완료일 : 2002년 11월 19일
[†] 준회원, 경상대학교 컴퓨터과학과 대학원
[††] 진주국제대학교 컴퓨터정보통신과 교수
[†††] 종신회원, 한국전자통신연구원
[††††] 정회원, 경상대학교 컴퓨터과학과 교수

처리하려는 요구가 커지고 있다. 따라서, 이러한 요구에 적절히 대응하기 위해서는 내장형 XML 데이터베이스 시스템의 개발이 필요하다.

그런데, 현재 PDA와 같은 내장형 시스템을 무선 환경에서 안정적으로 사용하기 위해서는 몇 가지 문제가 있다. 무엇보다 유선 통신 환경보다 속도가 느리고 접속이 불안하여 안정적인 데이터 전송이 보장되지 않는다. 그리고 내장형 시스템은 소형, 경량, 저전력, 저용량 메모리와 저성능의 처리 장치에서 소프트웨어를 구동해야 한다. 이러한 제약을 갖는 내장형 시스템에서 내장형 XML 데이터베이스 시스템이 사용될 수 있도록 하기 위해서는 높은 이식성, 적은 시스템 리소스 사용, 유연한 확장성, XML 문서 관리에 용이한 구조의 지원 등이 요구된다.

일반적으로, XML 문서를 처리하기 위해서는 DOM API의 사용이 필수적이다. 그런데, DOM API를 사용하면 XML 문서를 구성하는 특정 요소를 다루기가 용이하지만, DOM 트리 전체를 메모리에 유지해야 하기 때문에 메모리 공간을 많이 필요로 한다. 이러한 특성은 PDA와 같은 저용량의 메모리와 저성능의 처리 장치를 갖는 내장형 시스템에 큰 부담이 된다.

본 연구에서는 이러한 문제점을 보완하기 위해서 DOM 트리의 모든 노드(Node)들을 보조 기억 장치에 저장하고, 필요할 때만 메모리에 올려 사용할 수 있도록 하는 저장-DOM(P-DOM) 트리를 제안하고, 이를 관리하는 DOMdbm을 개발하였다. 우리는 DOMdbm의 개발을 효과적으로 수행하기 위해 내장형 데이터베이스 시스템으로 개발된 버클리 DB 시스템을 활용하였다. 따라서, DOMdbm에서는 버클리 DB 시스템의 모든 데이터베이스 기능을 이용할 수 있다. 이와 함께, DOMdbm은 이동형 단말기에서의 운용에 적합함으로, XML 문서에 대한 질의 처리 기능과 문서 관리 기능이 추가되면, 이동형 단말기를 위한 XML 데이터베이스 시스템으로 사용할 수 있다.

개발된 DOMdbm의 특징은 크게 두 가지이다. 첫째, 메모리를 효율적으로 사용한다. XML 문서의 일부를 사용할 때는 DOM 트리 전체를 메모리에 유지하지 않고 현재 연산에 참여하는 노드들만을 메모리에 유지한다. 둘째, 사용하기가 쉽다. 사용자는 자신이 이미 익숙해 있는 기존의 DOM API를 사용하는 방법 그대로 시스템을 사용할 수 있다.

논문의 구성은 다음과 같다. 2장에서는 관련 연구를 살펴보고, 3장에서는 버클리 DB 시스템에 대해 설명한다. 4장에서는 P-DOM을 구성하는 요소와 메모리에서 P-DOM 트리가 어떻게 구성되는지 알아본다. 5장에서는 P-DOM 트리를 버클리 DB에서 관리하기 위한 방법과 구현된 내용을 설명하고, 6장에서는 구현된 DOMdbm의 성능을 분석하고 문제점을 짚어본다. 7장에서는 결론과 함께 내장형 XML 데이터베이스 시스템으로 발전시키기 위한 고려 사항을 검토해 본다.

2. 관련 연구

XML 데이터베이스와 관련된 연구는 세계의 여러 곳에서 활발하게 이루어지고 있다. 현재 이 분야의 연구들은 주로 기존 데이터베이스 기술을 XML 관리에 활용하는 것들이다. 그런데, XML 문서를 데이터베이스에 저장하기 위한 방법은, 크게 관계형 데이터베이스 시스템을 이용하는 방법, 객체지향 데이터베이스 시스템을 이용하는 방법, 파일 시스템을 이용하는 방법, 저장소 관리기 기반의 방법이 있다.

관계형 데이터베이스를 이용하는 방법[5,6]은 XML 데이터를 테이블에 저장하고, XML 질의어를 SQL로 변환한 뒤 관계형 데이터베이스 시스템이 처리하는 것이다. 현재 XML 문서의 데이터베이스 관리에는 이 접근이 가장 많이 사용되고 있다. 하지만 XML이 갖는 계층적 문서 구조는 테이블 형태의 관계형 자료 구조와 일치하지 않아, 문서 내용이 여러 테이블들에 분산 처리되면서 비효율적이고, 의미가 왜곡되는 문제가 있다.

객체지향 데이터베이스를 이용하는 방법[7]은, 객체지향 이론이 부각되고 다양한 멀티미디어 데이터가 늘어남에 따라 점차 주목을 받고 있다. 객체지향 데이터베이스는 XML 문서의 계층적 구조를 모델링하기 쉽고, 객체 단위로 효과적으로 데이터를 관리할 수 있게 한다. 하지만 객체지향 데이터베이스 기술은 아직 성숙된 기술이 아니고 표준도 제정되어 있지 않아 대개의 경우 현재는 실험실 수준에 머물러 있는 실정이다.

파일 시스템을 이용하는 방법[8]은 각 XML 문서를 그대로 하나의 ASCII 파일로 저장하고, 질의를 하기 위해 XML 파일에 접근할 때마다 XML 파서를 사용해서 DOM 구조를 만들어 질의를 처리하는 방법이다. 이 방법은 구현이 단순하고 쉽다는 장점이 있지만, 질

의 처리 시마다 XML 파일을 파싱해야 하고, 반환되는 DOM 트리가 상대적으로 크며, 질의 처리를 위해 이 DOM 트리를 메모리에 계속 유지해야 하는 부담이 있다.

저장소 관리기 기반의 방법[9]은 XML 문서에서 나타나고 있는 구성 요소를 저장소 관리기가 지시하는 데이터 구조로 저장하고, 질의에 대한 접근 시에도 저장소 관리기를 통해서 반환되어 진다. 저장소 관리기 기반의 방법은 XML 질의를 처리하는 엔진과 저장소 관리기를 사용자가 스스로 개발해야 하는 부담이 있지만, XML의 특성을 고려하여 설계된 질의어 처리 엔진에 의해서 좋은 성능을 얻을 수 있다.

그런데, 이러한 XML 데이터베이스 연구들은 대개 리소스가 제한된 내장형 시스템을 고려한 것들은 아니다. 그런데, 최근에 이러한 내장형 환경을 고려해 Infonyte DB라는 XML 데이터베이스 시스템의 개발이 발표되었다. Infonyte DB[10]는 자바 기반의 XML 문서 저장 시스템이다. 이는 파일 시스템에 XML 문서를 DOM 형태로 저장하고, XML 질의어인 XQL을 사용하여 질의를 수행한다. Infonyte DB에서는 독립적인 XML 문서 저장 구조를 갖고, 저장된 XML 문서를 DOM API를 통해 조작할 수 있게 한다. 그런데, 이 시스템은 XML 문서를 저장하기 위해서 XML 문서의 DOM 트리를 메모리에 생성한 후, 이를 파일 시스템에 저장한다. 그러므로, 리소스가 부족한 이동형 단말기에 사용하기에는 아직도 적합하지 않다.

본 논문에서는 이러한 Infonyte DB의 문제점을 극복하는 방법으로 DOMdbm을 개발하였다. 따라서, DOMdbm은 Infonyte DB와 매우 유사한 방법으로 XML 문서를 관리하는 시스템이다. 그러나, DOMdbm은 파일 시스템이 아닌 버클리 DB에 XML 문서를 저장하고 관리한다. 따라서 버클리 DB의 모든 데이터베이스 기능을 이용할 수 있다. 그리고, 문서 전체의 DOM 트리를 버클리 DB에 유지한 상태로 연산에 참여하는 일부 노드만 메모리에 올린다. 따라서 리소스가 제한된 이동형 단말기에 적합하다. 즉, DOMdbm은 최소한의 데이터베이스 시스템 기능을 가진 버클리 DB를 기반으로, 저장소 관리기의 개념과 파일 시스템의 개념을 혼합한 형태를 갖는다. 현재 DOMdbm은 질의 처리 기능과 문서 관리 기능이 구현되지 않았지만, 향후 이 기능들을 추가하여 완전한 내장형 XML 데이터베이스 시스템을 개발할 것이다.

3. 버클리 DB 시스템

버클리 DB 시스템[11,12]은 소스가 공개된(Open Source) 내장형 DB 엔진으로 Window NT, Unix, Linux 등 대부분의 OS에 이식이 가능하고, 작고 빠르며 사용자가 이해하고 사용하기 쉬운 신뢰성있는 시스템이다. 이 시스템은 키/데이터 값의 쌍으로 고성능의 병행 저장과 검색을 요구하는 응용에 사용할 수 있도록 응용 프로그램과 직접 연결될 수 있는 라이브러리로 제공되고 있다. 현재 C, C++, Perl, Tcl, Java 등의 다양한 프로그램 언어들에서 사용될 수 있도록 해당 인터페이스 접속 장치를 제공하고 있다.

버클리 DB 시스템은 처음에 새로운 해쉬(Hash) 접근 방법을 구현할 목적으로 1990년에 Seltzer와 Yigit가 Hash라는 패키지로서 만들었다[12]. 이 이후 중복 데이터 항목을 다룰 수 있는 중요한 확장과 함께 병행 접근과 트랜잭션을 지원하고 롤백(Rollback) 기능도 지원하게 되었다. 그리고, 이 시스템은 아주 작은 족적(Footprint)을 가지고 있으며, 새로운 레코드를 삽입하는데 불과 몇 줄의 코드만으로 수행할 수 있게 하고, 많은 병행 사용들에 대해서도 정확하게 동작한다. 심지어 하드디스크의 손실과 같은 심각한 오류가 발생하더라도 올바르게 동작하도록 설계되었다.

버클리 DB 시스템에서 모든 데이터는 확장자가 .db인 파일명을 갖는 파일로 관리된다. 이때 모든 기본적인 파일 구조는 고정 및 가변 길이 레코드 형식을 가지며, 이를 B+_트리와 해쉬 구조로 사용할 수도 있다. 무엇보다 버클리 DB 시스템은 데이터를 저장하기 위해서 단순히 키/데이터의 쌍으로 구성되어 있는 레코드를 자료 구조로 사용하고 있다. 즉, 관계형 데이터베이스에서의 테이블, 객체지향 데이터베이스에서의 클래스와 같은 중간 데이터 모델이 없는 단순한 자료 구조를 가진다. 그림 1은 버클리 DB 시스템의 레코드 구조이다.

여기서 키(Key) 필드는 레코드를 유일하게 식별할 수 있는 값으로 시스템이 관리하게 되며, 데이터(Data) 필드는 저장하고자 하는 값으로 길이에 제한이 없으며 어떠한 내용도 가능하다. 따라서 이러한 버클리 DB 시스템을 이용하여 XML 데이터를 저장하고 관리하기

그림 1. 버클리 DB의 레코드 구조

위해서는 사용자가 적합한 자료 구조를 단순 키/데이터의 쌍으로 이루어진 레코드 구조 위에 직접 개발하여 사용해야 한다.

4. 저장-문서 객체 모델(Persistent-DOM)

DOM은 XML 문서를 다루기 위해 W3C에서 제정한 표준 문서 객체 모델이다. DOM을 이용하여 프로그래머는 문서를 생성할 수 있고, 그 문서의 구조에 따라 탐색할 수 있으며, 엘리먼트와 문서 내용 등을 추가, 수정 또는 삭제할 수 있다. DOM은 트리 형태를 가지는데, DOMdbm은 이러한 DOM 트리를 메모리가 아니라 버클리 DB 파일을 이용하여 지속적으로 유지되도록 구성한다. 그리고 기존의 DOM API를 사용하는 방법과 같은 방법으로 지속적으로 유지되는 DOM 트리를 사용할 수 있도록 한다.

4.1 DOM API의 구성

W3C 사양[13]에서 제안하고 있는 DOM의 목적 중 하나는 XML 문서를 다루는 표준화된 프로그래밍 인터페이스를 제공함으로써 환경과 응용 프로그램에 독립적으로 문서를 다룰 수 있게 하는 것이다. 현재 DOM 사양은 Level 3까지 제안된 상태이다. 본 논문에서는 DOM Level 1 API를 사용하였다.

그림 2는 DOM API 인터페이스를 나타낸 것이다. 그림에서 트리 노드 각각은 DOM에 명시된 인터페이스들의 종류들이다. 트리 구조의 관점에서 보면, 모든 노드는 부모 노드가 하나이고 0개 이상의 자식 노드를 가진다. 그리고 각 인터페이스는 여러 개의 필드와 메소드들을 정의한다. 그 중 가장 기본이 되는 인터페이스는 Node 인터페이스이며 거의 모든 인터페이스가 이 Node 인터페이스의 필드와 메소드들을 기본적으로 상속받는다.

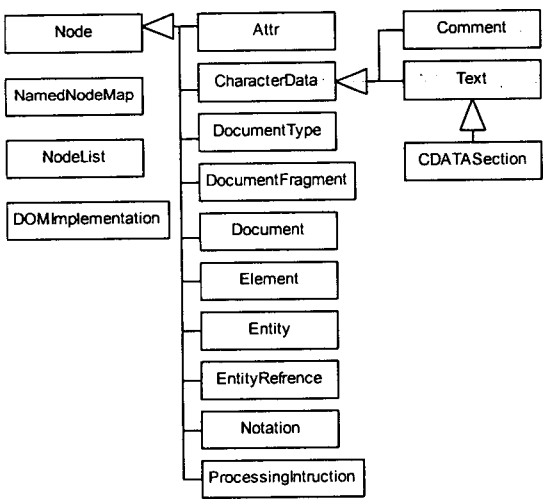

그림 2. DOM Level 1의 인터페이스 계층

4.2 저장-DOM 트리의 표현

기존의 메모리에 구축되는 DOM 트리는 노드 객체들의 참조가 연결되어 구성된다. 이때 참조는 메모리 상의 객체를 식별할 수 있는 정보이다. 그런데, 이러한 DOM 트리를 데이터베이스에 그대로 구성하기 위해서는 노드 객체들 사이의 참조 정보를 알아야 한다. 이를 위하여, 본 논문에서는 모든 객체에 고유한 ID를 부여하고, 이것을 그 객체에 대한 식별 정보로 사용한다. 우선 객체가 메모리에서 만들어 질 때 데이터베이스에 저장하기 위한 ID를 부여받고, 객체 내부의 데이터가 변경될 때마다 이 ID를 통해 데이터베이스를 갱신한다.

기본적으로 W3C 사양에 따라 DOM 트리를 구성하기 위해서는 부모(P) 노드, 첫 번째 자식(FC) 노드, 마지막 자식(LC) 노드, 이전 형제(PS) 노드, 다음 형제(NS) 노드에 대한 참조 정보를 각 노드 객체에서 유지하면 된다. 그림 3은 DOM 트리를 구성하는 일부 노드들과 이들 노드가 메모리에서 표현되는 모습을 보여주고 있다. 그림 3에서 보는 것처럼 트리가 구성되어 있을 경우 메모리에서는 각 노드들의 참조가 유지된다. 따라서, 한 노드 객체에 대한 연산이 수행되면 그 노드의 부모 및 자식, 형제 노드에 대한 참조 값이 변경된다.

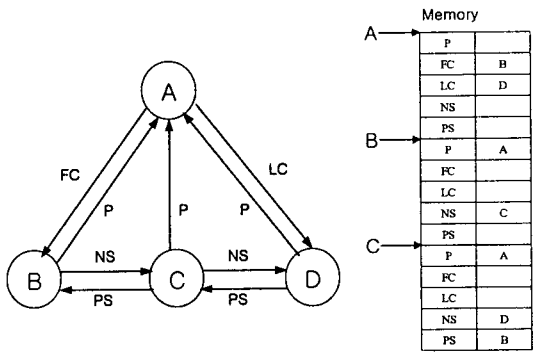

그림 3. 메모리상의 DOM 트리 표현

앞에서 언급한 바와 같이 버클리 DB 시스템은 키/데이터의 쌍으로 데이터를 저장하고 검색할 수 있는 낮은 수준의 API를 제공한다. 이 때문에, 메모리에 생성된 노드 객체를 버클리 DB 파일에 저장하기 위해서는 객체에 부여된 ID를 키로, 객체의 내부 데이터를 값으로 하여 저장해야 한다.

그림 4는 그림 3의 DOM 트리를 버클리 DB 파일에 저장했을 때의 모습인 P-DOM(Persistent DOM) 트리를 나타내고 있다. 이렇게 DB 파일에 저장되어 있는 노드 정보를 사용하기 위해서는, 메모리의 객체로 다시 복원되어 져야 한다. 이를 위해 P-DOM 트리에 대한 연산의 구현은 먼저 연산하고자 하는 노드와 연산에 참여하는 노드들이 메모리의 객체로 복원되고, 연산이 수행된 후 다시 DB 파일로 저장되도록 하는 방법을 사용하였다.

이러한 P-DOM 트리를 버클리 DB 시스템에 구성하는 기본적인 기능이 P-DOM API의 Node 인터페이스에 정의되어 있다. 본 논문에서, Node 인터페이스는 NodeImpl 클래스에서 구현한다. 이 클래스는 그림 3에서 나타난 부모 노드, 자식 노드, 형제 노드의 정보를 가지고 있으며, 이 데이터는 바이트 배열이다. DOM 트리를 메모리에 구축하였을 경우 이들 데이터는 해당 노드 객체의 참조(Reference)일 테지만, P-DOM 트리를 버클리 DB 파일에 저장하기 위해서는 노드 객체에 고유한 ID를 부여하고 이를 노드 객체에 접근하는 정보로 사용해야 한다. 그런데, P-DOM API에서 해당하는 노드를 얻기 위의 메쏘드를 호출하였을 때는 Node 인터페이스를 구현한 객체에 대한 참조를 반환해야 함으로, 이들 메쏘드에서는 해당하는 객체의 ID를 이용하여 버클리 DB 시스템으로부터 객체의 데이터를 읽어와서 메모리에 노드 객체를 생성한 후, 이 객체의 참조를 반환하게 된다. 즉, 내부적으로 NodeImpl 클래스의 인스턴스가 생성되고, 이 인스턴스의 데이터가 버클리 DB 파일에 저장되며, 이 데이터를 통해 다른 노드를 참조할 수 있게 된다.

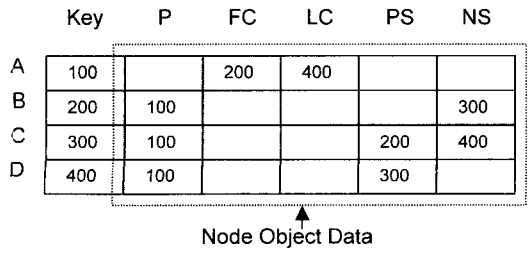

그림 4. 버클리 DB에 저장된 P-DOM 트리 노드

한편, NodeImpl 클래스는 자바의 Serializable 인터페이스를 구현하고 있다. 이는 객체를 데이터베이스에 저장할 때 객체의 데이터를 바이트 배열로 변환하기 위해 자바의 직렬화 메커니즘을 이용하기 위해서이다. 객체를 바이트 배열로 변환하는 것에 대해서는 5.2.2절의 객체 변환기를 기술하는 부분에서 상세히 설명하기로 한다.

5. 저장-DOM의 구현

앞 장에서 살펴본 P-DOM 트리 표현을 기반으로 XML 문서를 버클리 DB 시스템을 이용해 저장하고, 관리하는 DOMdbm을 개발하였다. 이 장에서는 이의 구현을 알아본다.

5.1 DOMdbm의 시스템 구조

DOMdbm의 시스템 구조는 그림 5와 같다. 여기서 기반이 되는 저장소 관리기(Storage Manager)는 P-DOM 트리를 버클리 DB 파일에 저장하는 부분인데, 키 생성기(Key Generator)와 객체 변환기(Marshaler)를 포함하고 있다. 키 생성기는 객체의 ID를 생성하며, 객체 변환기는 객체를 바이트 배열로 변환하는 역할을 한다. 저장소 관리기는 키 생성기를 이용하여 객체의 ID를 얻고, 객체 변환기를 이용하여 저장할 객체를 바이트 배열로 변환하여, 버클리 DB 파일에

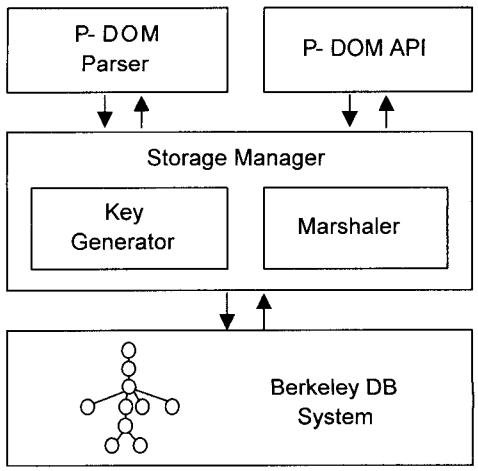

그림 5. DOMdbm의 시스템 구조

삽입/수정/삭제하고 검색하는 기능을 수행한다. 버클리 DB 시스템에 P-DOM API를 구현한 해당 클래스들의 인스턴스 객체들이 저장되며, 이 객체들의 P-DOM 트리가 구성된다. P-DOM 파서는 XML 문서를 분석하여 P-DOM 트리를 구성하며, 이는 저장소 관리기와 P-DOM API를 이용하여 이루어진다. P-DOM 파서는 노드 객체를 생성하고, 이 객체들이 저장소 관리기를 통해서 버클리 DB 시스템에 저장된다. P-DOM 파서는 저장된 노드 객체의 ID를 반환받게 되는데, 이를 이용하여 P-DOM 트리를 구성한다.

버클리 DB 파일에 저장되어 있는 P-DOM 트리에 접근하기 위해서, 응용 프로그램에서는 P-DOM API의 Document 인터페이스를 구현한 클래스의 인스턴스를 이용하여 P-DOM 트리를 사용하게 되는데, 버클리 DB 파일에 저장되어 있는 노드 객체를 사용하기 위해서, 저장소 관리기가 해당 객체의 ID를 키로 하여 데이터베이스에서 객체의 데이터를 꺼내온 후, 바이트 배열을 객체로 변환하여 생성시키게 된다.

5.2 저장소 관리기

저장소 관리기는 P-DOM API를 구현한 객체를 버클리 DB 파일에 저장하고 복원하는 역할을 담당한다. 즉, 생성된 ID와 변환된 객체 데이터를 버클리 DB 파일에 저장하는 역할을 담당한다. 이를 위해 저장소 관리기는 객체의 고유한 ID를 생성하는 키 생성기, 그리고 객체의 메모리 내부 데이터를 바이트 배열로 변경하여 데이터베이스에 저장하거나 반대로 데이터베이스에 저장되어있는 바이트 배열을 다시 객체로 변환시키는 역할을 하는 객체 변환기를 내부에 포함하고 있다.

저장소 관리기는 노드 객체를 버클리 DB 파일에 삽입/수정/삭제 및 검색하는 기능을 가지고 있다. 노드 객체를 데이터베이스에 저장하기 위해 먼저 키 생성기로부터 객체 ID를 생성하고, 객체 변환기를 이용하여 저장할 노드 객체를 바이트 배열로 변환한 후, ID를 키로 하고 변환한 바이트 배열을 값으로 하여 버클리 DB 파일에 저장하게 된다. 저장된 노드 객체를 검색하기 위해서는 검색하려는 노드 객체의 ID를 이용한다. 객체의 수정과 삭제도 마찬가지이다. 그림 6은 저장소 관리기가 노드를 삽입/수정/삭제 및 검색하는 알고리즘이다.

```
byte[] insertNode(Node node) {
  byte[] key
  byte[] data = marshaler.marshaling(node);
  while(true) {
    key = = keyGenerator.generate();
    berkeryDB.put(key, data);
    if (!KEYEXIST) break;
  }
  return key;
}

void updateNode(Node node) {
  byte[] key = node.getKey();
  byte[] data = marshaler.marshaling(node);
  berkeryDB.put(key, data);
}

void removeNode(Node node) {
  byte[] key = node.getKey();
  berkeryDB.del(key);
}

Node retriveNode(byte[] key) {
  byte[] data;
  berkeleyDB.get(key, data);
  Node node = marshaler.unmarshaling(data);
  return node;
}
```

그림 6. 저장소 관리기의 알고리즘

5.2.1 키 생성기

키 생성기는 주어진 크기의 바이트 배열을 임의로 생성하여 반환하는 데, 그 값이 유일하지 않다. 그러나 객체의 ID는 고유해야 하기 때문에, 객체를 버클리 DB 파일에 저장하는 과정에서 이 문제를 보완하고 있다. 객체의 ID를 생성하는 방법은 다양하기 때문에 KeyGenerator 인터페이스에서 그 기능을 정의하고, 다른 클래스에서 실제로 구현할 수 있도록 하였다.

5.2.2 객체 변환기

객체 변환기에서 객체를 바이트 배열로 변환할 때, 그 형식은 다양하게 구현될 수 있다. 변환을 수행하기 위해서는 변환될 객체의 내부 데이터에 접근해야 한다. 그러나, 객체의 필드에 직접 접근하는 방법은 캡슐화(Encapsulation)를 해치기 때문에 좋은 방법이 아니다. 그리고, 메쏘드를 통해 데이터에 접근하기 위해서

는 새로운 인터페이스가 필요하다. 이는 다른 방법으로 자바의 리플렉션(Reflection)을 이용할 수도 있다. 그러나, 새로운 인터페이스를 선언하는 전자의 방법은 객체마다 저장될 데이터가 다르므로 어려움이 있다. 그리고, 후자인 리플렉션을 사용하는 방법은 보안상 객체의 전용 필드에 접근할 수 없는 문제점이 있다.

본 논문에서는 변환 방법으로 자바의 직렬화 메커니즘을 사용하였다. 직렬화란 자바 가상 머신이 메모리상의 객체를 외부 저장 장치에 저장하기 위해 바이트 배열로 변환시켜 주는 것을 말한다. 이 방법은 객체를 복원하기 위해서 역시 자바에서 제공하는 메커니즘을 사용해야 한다. 따라서, 이는 저장된 데이터를 다른 언어에서 접근하거나 또는 데이터의 일부분만을 엑세스하는 데에는 적합하지 않다. 하지만 본 논문에서는 현재 P-DOM API를 사용하는 언어로 자바(Java)를 사용하였기 때문에 이 방법을 사용하였다.

5.3 P-DOM API

P-DOM 트리는 P-DOM API를 구현하는 클래스의 인스턴스들로 구성된다. 그러므로 각 노드의 실제 클래스가 다를 수 있으므로 공통의 데이터와 기능을 한 곳으로 모으고, 동일한 데이터 타입으로 접근할 수 있도록 하기 위해 최상위 클래스가 필요하다. 이 클래스는 Node 인터페이스를 구현하며, 노드에 대한 연산을 수행하는 기본 타입이 된다. XML 문서를 구성하는 각 객체들은 종류에 따라 Node 인터페이스를 구현한 클래스를 확장하여 부가적인 기능이 추가된다. 그림 7은 DOMdbm에서 P-DOM API를 구현한 클래스의 계층도이다.

Node 인터페이스는 NodeImpl 클래스에서 구현되었다. NodeImpl 클래스는 노드의 이름과 값을 저장할 수 있고, P-DOM 트리를 구성하는데 필요한 데이터,

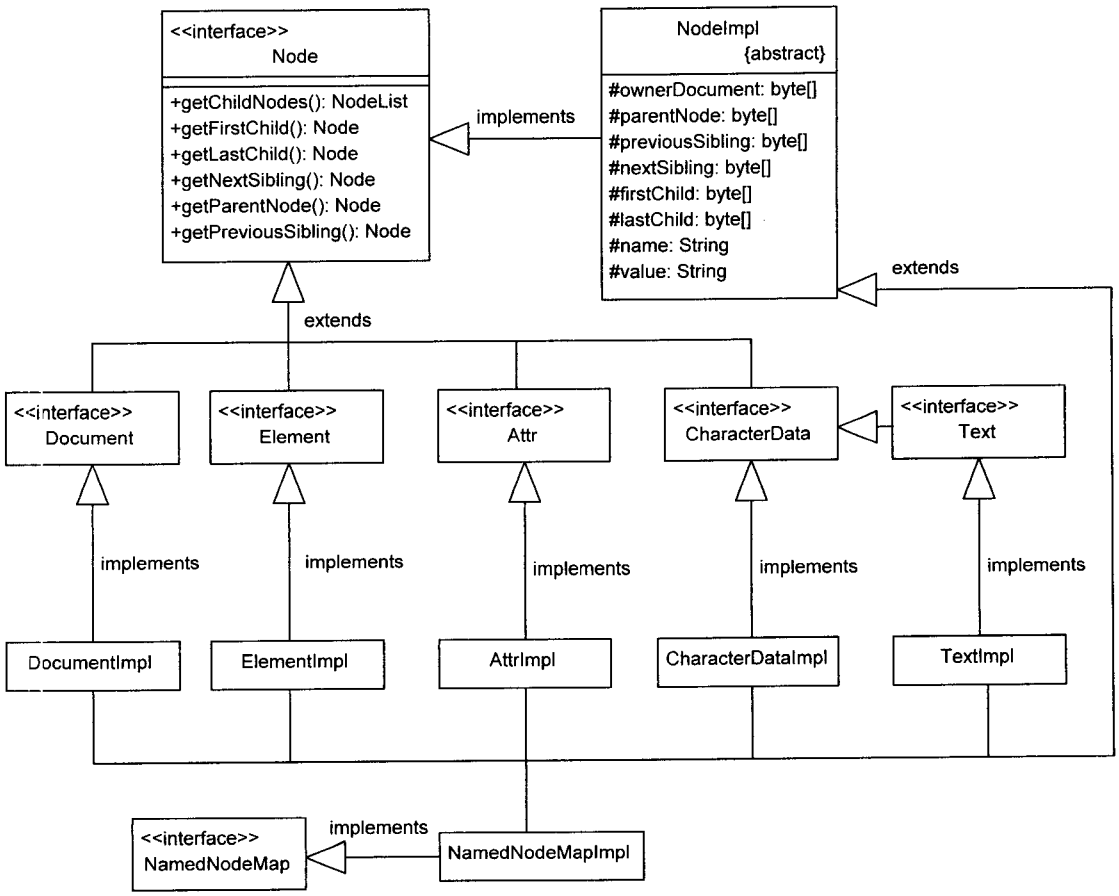

그림 7. P-DOM API를 구현한 클래스의 다이어그램

즉 부모 노드 객체의 ID, 첫 번째 자식 노드 객체의 ID, 마지막 자식 노드 객체의 ID, 이전 형제 노드 객체의 ID, 그리고 다음 형제 노드 객체의 ID를 가지고 있다. 그리고 노드에 대한 연산을 수행하기 위해서는 자기 자신의 객체 ID가 필요한 경우가 있기 때문에 자신의 ID도 가지고 있다. 이러한 객체 ID는 모두 바이트 배열이며, 해당 노드 객체에 접근하기 위해서 버클리 DB 파일에 저장되어 있는 객체를 메모리로 환원시킬 때 사용된다. Node 인터페이스에는 팩토리 메쏘드(Factory Method)가 선언되어 있다. 팩토리 메쏘드는 P-DOM 트리를 조작하기 위해, 노드의 추가/수정/삽입 등의 기능이 구현된다.

Document 인터페이스는 DocumentImpl 클래스에서 구현한다. DocumentImpl 클래스는 DocumentType 노드 객체와 Root Element 노드 객체의 ID를 가지고 있다. DocumentImpl 클래스는 NodeImpl 클래스를 확장하고 있다. Document 인터페이스도 팩토리 메쏘드가 선언되어 있는데, P-DOM 트리를 구성하는 노드 객체를 생성하는 기능을 가지고 있다. P-DOM 트리를 구성하기 위해서는 먼저 DocumentImpl 클래스의 인스턴스를 하나 생성한 후, 이 객체 팩토리 메쏘드를 이용하여 해당 노드 객체를 생성한 후, Node 인터페이스의 팩토리 메쏘드를 이용하여 P-DOM 트리를 조작하게 된다.

5.4 P-DOM Parser

P-DOM Parser는 P-DOM 트리를 버클리 DB에 구성한다. 이는 내부적으로 SAX 파서를 이용한다. SAX 파서는 XML 문서에서 태그가 시작되거나 끝날 때, 또는 DTD, 주석, PCDATA 등이 나타날 때 해당 이벤트를 통지해 주는데, SAX 파서에 등록된 Document Handler에서 이러한 이벤트를 받아서 처리하게 된다. 그래서 P-DOM Parser는 SAX 파서에 등록하여 Document Handler로 사용하도록 하였다. 본 연구를 위해 SAX 파서는 SUN에서 제공하는 JAXP[14]를 사용하였다.

P-DOM Parser는 P-DOM 트리를 구성하는 데 필요한 정보들을 유지하고 있다. P-DOM 트리를 구성하기 위해서는 Document 객체가 필요하고, Document 객체의 Factory 메쏘드를 사용해서 노드를 생성하게 된다. 노드를 추가하기 위해서는 추가될 노드의 부모 노드 객체가 필요하다. P-DOM Parser에서는 스택을 이용하여 부모 노드의 ID를 유지하고 있다. XML 문서가 계층형으로 구성되어 있으므로 스택의 최상(Top) 위치에 놓여 있는 노드 객체는 현재 노드의 부모 노드이다. 현재 노드를 P-DOM 트리에 추가하는 방법은 스택의 최상 위치에 있는 부모 노드의 ID를 이용하여 DB로부터 부모 노드 객체를 복원한 후, 노드 추가 연산을 수행하는 것이다. 연산의 결과는 다시 버클리 DB 시스템에 저장된다.

6. 성능 평가 및 분석

6.1 실험 환경

제안된 DOMdbm을 구현하여 펜티엄III PC (Windows 2000) 환경에서 실험하였다. 버클리 DB는 버전 4.1을 사용하였고, 구현언어는 JDK 1.4이다. 성능 분석을 위해서 Infonyte DB[10]와 실험 결과를 비교하였다. 비교 항목은 노드의 개수, 수행 시간, DB 파일의 크기, 메모리 사용량인데, 하나의 XML 문서를 파싱하여 DB에 저장하는 것을 기준으로 한다. 실험을 위해 4개의 XML문서를 사용하였는데, 각각 3KB(A), 10KB(B) , 161KB(C), 1178KB(D)의 크기를 가진다.

6.2 실험 결과

표 1은 Infonyte DB와 DOMdbm의 성능을 비교한 것이다.

표 1에서 보는 것처럼, DOMdbm은 XML 문서의

표 1. Infonyte DB와 DOMdbm의 성능 비교

	노드 개수 (개)		수행 시간 (ms)		DB 파일 크기 (KB)		메모리 사용량 (KB)	
	Infonyte	Domdbm	Infonyte	Domdbm	Infonyte	Domdbm	Infonyte	Domdbm
A	62	94	411	430	2	80	548	372
B	429	446	412	1260	9	280	602	380
C	11546	12382	822	16103	156	6100	1868	1584
D	33698	49453	5800	82339	393	25040	6336	1538

구조적인 정보와 객체를 저장하기 위한 부가적인 데이터가 필요하기 때문에 XML 문서를 구성하는 노드의 개수가 늘어날수록 데이터베이스 파일의 크기가 크게 증가하는 단점이 있다. 노드 하나의 크기가 평균 500byte 정도이고, 노드의 개수가 많기 때문이다. 그리고 노드의 조작을 위해서는 버클리 DB에 저장되어 있는 노드가 메모리에 자바 객체로 복원되어 져야하고, 조작 후 다시 바이트 배열로 변환되어 저장되어지기 때문에 수행 시간이 늦다. 그러나 메모리의 사용량 부분에서는 XML 문서의 크기가 커지고, 노드의 개수가 많아지더라도 현재 연산에 참여하는 노드만 메모리에 유지하기 때문에 어느 정도의 일정한 수준에서 메모리가 사용되는 것을 알 수 있다.

6.3 실험 결과 분석

표 2는 Infonyte DB와 DOMdbm 간의 기능 비교표이다. 표 2에서 보듯이, 현재 DOMdbm은 Infonyte DB에 비해 기능이 부족하다. 그러나 DOMdbm은 효율적인 메모리 사용으로 내장형 시스템에 적합하다. 향후, 질의 처리 기능과 문서 관리 기능을 추가하고, DB 파일의 크기를 줄이면 내장형 시스템을 위한 XML 데이터베이스 시스템으로 적합할 것이다.

데이터베이스 파일의 크기를 줄이기 위해서는 노드의 개수와 노드의 크기를 줄여야 한다. 노드의 개수를 줄이기 위해서 P-DOM 트리를 구성하는 노드 중에 실제 데이터를 가지지 않는 노드들을 제거하고, 연결될 수 있는 Text 노드들을 하나의 노드로 통합할 수 있다. 현재, 노드 하나의 크기가 큰 이유는 저장된 구조가 자바의 직렬화 메커니즘에 의해 바이트 배열로 구성되어 있기 때문이다. 노드의 크기를 줄이기 위해서는 자바의 직렬화 기법을 사용하지 않는 다른 방법이 필요하다. 이를 위해서는 저장해야 할 노드의 정보를 얻을 수 있는 인터페이스를 추가하고 이를 통해 노드의 정보를 획득하면 효율적인 구조의 바이트 배열을 구성할 수 있을 것이다. 그리고, 저장되기 전에 바이트 배열을 압축을 하여 저장하면 노드의 크기를 크게 줄일 수 있을 것이다.

표 2. Infonyte DB와 DOMdbm의 기능 비교

	Infonyte DB	DOMdbm
저장 시스템	파일 시스템	버클리 DB
DB 파일의 크기	작음	큼
수행 속도	빠름	느림
질의 처리	지원	지원안함
문서 관리	지원	지원안함
메모리 사용	비효율적	효율적
내장형 시스템의 적합성	부적합	적합

낮은 수행 속도의 문제를 해결하기 위해서 노드에 대한 연산을 수행할 때 자주 엑세스되는 노드를 메모리에 캐쉬(Cache)를 할 수 있다. 그리고, 노드가 저장되기 전에 버퍼링(Buffering)을 하여 보조 기억 장치의 엑세스를 줄이는 방법이 있다. 그리고, 향후 추가될 질의 처리를 고려하면, 버클리 DB의 검색 기능을 이용할 수 있다.

7. 결론 및 향후 과제

본 논문에서는 XML 문서의 DOM 트리를 메모리에 구성하여 조작하는 대신에, 버클리 DB에 구성하여 조작할 수 있는 시스템인 DOMdbm을 저장-DOM(P-DOM) 트리로 개발하였다. DOMdbm은 XML 문서의 일부분을 사용하기 위해 전체 문서의 DOM 트리를 메모리에 유지할 필요가 없고, XML 문서의 저장과 조작 시 현재 연산에 참여하는 노드만이 메모리에 유지되므로 효율적으로 메모리를 사용한다. 그리고, W3C에서 제정한 DOM API를 그대로 따르는 P-DOM API를 구현하였으므로 사용자 입장에서는 기존의 DOM API를 사용하는 것과 다르지 않다. 또한 버클리 DB의 모든 기능을 이용할 수 있고, 효율적으로 메모리를 운용하기 때문에 시스템의 자원이 부족한 이동형 단말기에서의 운용에 적합하다.

개발된 DOMdbm을 완전한 내장형 XML 데이터베이스 시스템으로 확장하기 위해서는 앞으로 질의 처리 기능과 문서 관리 기능이 추가되어야 한다. 우선, XML 문서가 이미 P-DOM 트리로 저장되어 있으므로, 질의를 위해 XML 문서를 다시 파싱할 필요가 없다. 그런데, 질의를 처리하기 위해 P-DOM 트리를 순회하는 것은 보조기억 장치의 특성상 시간이 많이 걸리므로, 버클리 DB의 검색 기능을 사용하여야 한다. 이를 위해서 노드의 내부 정보를 바이트 배열로 변환할 때 구조를 가지는 바이트 배열로 변환할 필요가 있다. 그리고, P-DOM 트리를 구성하는 구조 정보와 문서의 데이터를 별도로 저장하거나, XML 문서의 구성 요소별로 저장할 수 있다. 그러나 저장되는 구조가 어떤 것이더라

도, 저장된 XML 문서는 P-DOM API를 통해서 조작할 수 있어야 한다.

다음으로, 버클리 DB에 저장된 XML 문서를 보다 효과적으로 관리하기 위해서 별도의 카탈로그가 필요하다. 이 카탈로그에는 문서의 이름, P-DOM 트리의 루트 노드의 키 등과 같은 문서 정보가 저장된다. 사용자는 이 카탈로그를 이용하여 저장되어 있는 XML 문서의 루트 노드를 얻은 후, P-DOM API를 이용하여 XML 문서를 사용할 수 있다.

참 고 문 헌

[1] Mizi Research, http://www.mizi.com.

[2] M. Olson, K. Bostic, and M. Seltzer, "Berkeley DB," Proc. of the 1999 Summer Usenix Technical Conference, Monterey, California, June 1999.

[3] G. Bex, "simpleDB: a simple embedded database", http://alpha.luc.ac.be/~gjb/simpleDB/", January 2001.

[4] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom, "Lore : A Database Management System for Semistructured Data," Technical Report, Stanford University, Database Group, Februdary 1997.

[5] D. Florescu and D. Kossmann, "Storing and Querying XML Data using as RDBMS," Bulleting of the Technical Committee on Data Emgineering, Vol. 22, No.3, 1999.

[6] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. DeWitt, and J. Naughon, "Relational Database for Queyring XML Documents : Limitations and Opportunities," Proc. of 25th Int'l Conf. on VLDB, Edinburgh, Scotland, UK, 1999.

[7] R. Bourret, "Mapping DTDs to Databases," http://www.xml.com/pub/a/2001/05/09/dtdtodbs.html".

[8] D. Florescu, D. Kossman, "A Performance Evaluation of Alternatice Mapping Schemes for Storing XML Data in a Relational Database," Rapport de recherche No. 3680, INRIA, Recquencourt, France, May 1999.

[9] M. Carey, D. DeWitt, J. Naughton, M. Solomon, et. al, "Shoring Up Persistent Applications", Proc. of Conf. on ACM SIGMOD, 1994.

[10] Infonyte DB, http://www.infonyte.com.

[11] Berkeley DB, http://www.sleepycat.com/docs/reftoc.html.

[12] M. Seltzer, and O, Yigit, "A New Hashing Package for UNIX," Proc. 1991 Winter USENIX Conference, Dallas, TX, January 1991.

[13] DOM Level 2 Core Recommendation, "http://www.w3c.org/TR/2000/REC-DOM-Level-2-Core-20001113/".

[14] Java API for XML Parsing 1.1, "http://java.sun.com/xml/xml_jaxp.htm".

강 동 완

2002년 경상대학교 컴퓨터과학과 학사

2002년~현재 경상대학교 대학원 컴퓨터과학과 석사 과정

관심분야 : XML 데이터베이스 통합, XML 질의 처리, 프로그래밍 시스템

제 권 엽

2002년 경상대학교 컴퓨터과학과 학사

2002년-현재 경상대학교 대학원 컴퓨터과학과 석사 과정

관심분야 : XML, 내장형 XML DBMS, XML 문서 저장소

홍 영 표

1981년 광운대학교 전자계산학과 학사

1985년 광운대학교 전자계산학과 석사

2003년 경상대학교 컴퓨터과학과 박사

1984년~현재 진주국제대학교 컴퓨터정보통신과 부교수

관심분야 : 내장형 시스템, 데이터베이스, XML 문서관리

한 동 원

1982년 숭실대학교 전자공학과 학사
1992년 한남대학교 전자공학과 석사
1995년 충남대학교 컴퓨터과학과 박사과정 수료
1982~현재 한국전자통신연구원 책임연구원 휴대클라이언트연구팀장
관심분야 : 멀티미디어 휴대정보단말, 웨어러블 컴퓨터, 편재형 컴퓨팅 분야

강 현 석

1981년 동국대학교 전자계산학과 학사
1983년 서울대학교 계산통계학과 석사(전산학)
1989년 서울대학교 계산통계학과 박사(전산학)
1984~1993년 전북대학교 전자계산학과 부교수
1993년~현재 경상대학교 컴퓨터과학과 교수
관심분야 : 객체지향 데이터베이스, 전자문서 관리, 멀티미디어

배 종 민

1980년 서울대학교 수학교육과 학사
1983년 서울대학교 대학원 계산통계학과 석사
1995년 서울대학교 대학원 계산통계학과 박사
1982년~1984년 한국전자통신연구소 연구원
1997년~1998년 Virginia Tech. 객원연구원
1984년~현재 경상대학교 전임강사, 조교수, 부교수, 교수
관심분야 : XML 데이터베이스 통합, 정보검색, 디지털 라이브러리, 데이터마이닝

교 신 저 자

강 현 석 660-701 경남 진주시 가좌동 900번지 경상대학교 컴퓨터과학과