XDOM: 확장성 기반의 경량 XML 객체 정보 저장소

(XDOM: The Extensible and Light-Weight XML Object Repository)

오 동 일 † 최 일 환 †† 박 상 원 ††† 김 형 주 ††††
(Dongil Oh) (Ilhwan Choi) (Sang-Won Park) (Hyoung-Joo Kim)

요 약 최근 인터넷의 비약적인 발전과 함께 등장한 XML은 사실상 인터넷 상의 문서 교환과 표현의 표준으로 자리 잡고 있다. 이에 따라 XML 문서를 저장하려는 노력도 많이 진행되고 있다. 하지만 XML 문서는 정형화된 스키마를 가지지 않기 때문에 기존의 데이터베이스 시스템에 저장해서 처리하기에는 많은 문제점이 있다. 본 논문에서는 XML 문서의 구조를 접근하도록 하는 DOM 객체를 저장하는 객체 정보 저장소인 XDOM을 제안한다. XDOM은 기존의 데이터베이스 시스템과는 달리 스키마를 생성할 필요가 없어 정형화되어 있지 않은 문서들도 쉽게 표현할 수 있다. 또한 기존의 응용 프로그램이 메모리가 아닌 저장소 레벨에서 DOM 객체를 투명하게 접근할 수 있게 한다. 본 논문에서는 자바 기반으로 XDOM을 구현하였고 실험을 통해 성능을 비교하였다.

키워드 : XML, DOM, 객체 저장소

Abstract XML is an emerging internet standard for data representation and exchange on the web. Recently, many researches on storing XML documents are in progress. Since XML documents have an implicit semantic schema, it is difficult to store XML data into a DBMS or File System. This paper introduces XDOM, object repositories for the DOM object, which stores an object for persistency support. As XDOM does not require the burdensome design of a fixed schema unlike the legacy DBMS, can store schema-less XML documents. And it can help XML applications to access the DOM object transparently from the object repositories below memory. XDOM introduced in this paper is implemented in pure Java and evaluated using experiments.

Key words : XML, DOM, object repository

1. 서 론

최근 인터넷이 보편화되고 웹이 급성장함에 따라 많은 양의 데이타가 웹을 통해서 교환되고 있고, 대부분의 필요한 정보들은 웹을 통해서 얻을 수 있게 되었다. 하지만 웹 상에 존재하는 데이타에는 구조 정보가 없어 그 의미를 파악하기 힘들고 데이타들이 대부분 표현에만 국한된 HTML(HyperText Markup Language)로 작성되어 있어, 실제로 우리가 원하는 정확한 정보만을 찾기에는 한계가 있다. 이러한 한계를 극복하기 위해 웹 문서 내에 그 문서의 구조 정보를 같이 포함할 수 있게 하는 XML(eXtensible Markup Language)이 등장하게 되었고 웹 기술 표준을 제정하는 W3C(World Wide Web Consortium)에서 인터넷 문서교환의 표준으로 XML을 제정하였다. XML은 새로운 웹 표준 언어로써 기존의 HTML 문서가 가지는 단점을 극복하고 기존에 웹 문서에 구조 정보를 추가시켜 문서의 의미를 쉽게 파악할 수 있는 특징을 갖는다.

XML이 웹의 문서 교환 표준이 되어 가면서 XML 문서를 저장하기 위한 많은 노력들이 진행되고 있다. 특

† 비 회 원 : (주)한국오라클
dioh@korea.com
†† 비 회 원 : 서울대학교 전기컴퓨터공학부
ihchoi@oopsla.snu.ac.kr
††† 종신회원 : 한국외국어대학교 컴퓨터및정보통신공학부 교수
swpark@hufs.ac.kr
†††† 종신회원 : 서울대학교 전기컴퓨터공학부 교수
hjk@oopsla.snu.ac.kr
논문접수 : 2001년 12월 14일
심사완료 : 2003년 3월 7일

히 XML 문서의 저장 방법에 따라 질의 처리 성능에 많은 영향을 주게 되므로 XML 문서의 저장 방법에 대한 다양한 접근 방법이 제시되어 왔다. 대부분의 연구가 파일시스템과 데이타베이스 시스템을 기반으로 진행되어 왔다. 우선 이들에 대해서 간단하게 살펴보도록 한다.

XML 문서를 저장하는 가장 간단한 방법은 각각의 XML 문서를 파일시스템에 텍스트 파일 형태로 저장하는 방법이다. 이 방법의 가장 큰 장점은 구현이 간단하고 데이타베이스 등의 별도의 저장 관리자가 필요없다는 것이다. 또한 XML 원본 문서 전체가 필요한 경우에 별도의 가공 없이 파일시스템으로부터 읽어오기만 하면 된다. 그러나 이러한 파일 시스템 상에 XML 문서를 저장하는 방법에는 몇 가지 중요한 문제점이 존재한다. 먼저, 텍스트 파일로 저장되어 있는 XML 문서는 브라우징, 질의 처리 등 이를 사용하는 경우에 매번 별도의 파싱 과정을 거쳐야 한다는 부담을 갖는다. 뿐만 아니라 질의 처리시 XML 문서를 파싱한 결과가 메모리 상에 언제나 상주해 있어야 한다는 문제점을 갖는다.

XML 문서를 저장하는 다른 방법은 관계형 데이타베이스 시스템에 저장하는 것이다. 기존의 많은 데이타들이 주로 관계형 데이타베이스 시스템에 저장되어 있기 때문에 이 분야에 대한 연구가 더욱 활발히 진행되어 왔다. 하지만 관계형 데이타베이스 시스템을 통한 XML 문서 저장은 구조가 정형화되지 않은 문서를 저장하는데 있어 스키마를 생성하기가 어려운 문제가 있다. 또한 문서의 갱신 작업이 일어날 경우 복잡한 테이블 연산이 필요하고, XML 질의를 SQL(Structured Query Language)로 질의 변환을 해야 한다는 점에서도 비용이 많이 소요된다.

객체지향 데이타베이스 시스템의 경우를 보면 데이타 모델이 XML 특징을 자연스럽게 반영할 수 있고, XML에 대한 질의어를 기존의 객체 질의어를 확장하여 지원할 수 있다는 장점을 가지고 있다는 점에서 최근 연구가 활발해지고 있다. 하지만 관계형 데이타베이스 시스템의 경우와 마찬가지로 XML 질의를 OQL(Object Query Language)로 바꾸는 질의 변환에 드는 비용이 소요된다. 또한 문서의 갱신이 빈번할 경우 객체의 크기가 커지고 저장 공간의 최적화가 힘들어지는 단점이 있다. 그밖에 객체지향 데이타베이스 시스템 자체가 대용량의 데이타에 대한 질의 처리에 성숙되어 있지 않다는 단점도 지적되고 있다.

최근에는 파일 시스템이나 기존의 데이타베이스 시스템이 아닌 새로운 XML 전용 저장장치에 관한 연구가 많이 진행되고 있다[1~8]. 이들 XML 전용 저장장치는 XML의 특징을 살려 특별한 저장 형식을 만들고 XML 문서를 그 형식에 맞추어 쉽게 저장하기 위한 여러 소프트웨어들을 지원하고 있다. 이러한 XML 전용 저장장치는 비록 성능 면에서는 좀더 나은 결과를 내기도 하지만 아직까지 기존의 파일시스템과 데이타베이스 시스템을 장악할만한 성숙한 단계에 이르지 못하고 있고, 대부분의 경우 기존의 저장 기법에서 크게 벗어나지 못한 면을 보이고 있다. 이에 본 논문에서는 DOM(Document Object Model)[9]을 기반으로 한 새로운 저장장치에 대한 방법을 제안하고자 한다.

DOM은 문서의 각 요소를 객체로 만들어 접근하기 위해서 제안된 기술로서, XML 문서를 트리 형태의 자료 구조로 접근할 수 있도록 하는 인터페이스를 제공한다. 질의 처리기를 포함한 많은 XML 응용 프로그램들은 DOM을 사용해서 XML 문서에 접근할 수 있다. 또한 DOM은 정형화되어 있지 않은 문서에서 스키마를 추출할 필요가 없으므로 기존 데이타베이스 시스템에서의 스키마 생성 문제를 고려하지 않아도 된다.

제안하는 시스템은 미리 XML 문서를 파싱하여 저장함으로써, XML 응용 프로그램이 문서를 파싱하는 과정 없이 직접 데이타에 접근하여 보다 효율적으로 데이타를 다룰 수 있도록 한다. 그리고 영속적인 DOM 객체 지원을 보장함으로써, 문서의 크기가 매우 큰 경우에도 필요한 데이타 부분만 메모리에 상주하도록 하여 일정한 크기의 메모리만을 사용하여 응용 프로그램이 수행될 수 있다. 또한 모든 구현을 자바로 함으로써 확장성, 이식성을 높였다. 이런 특징들로 인하여 제안하는 시스템은 보편적인 XML 문서 저장소의 역할뿐만이 아니라, 확장성을 기반으로 큰 시스템의 일부분으로 들어가 서브 시스템으로써의 여러 역할을 수행할 수 있다. 또한 최근 들어 각광을 받고 있는 개인용 무선 단말기(PDA)나 케이블 TV 등과 같은 내장형 시스템에서의 저장 장치로도 사용될 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 XML저장장치에 대한 기존의 연구들을 살펴본다. 3장에서는 제안하는 시스템인 XDOM에 대한 구조와 내부 동작 원리 및 특성 등을 살펴본다. 4장은 제안하는 시스템과 다른 시스템과의 성능비교 결과를 보여준다. 마지막으로 5장은 결론 및 향후 연구 과제에 대해 다루기로 한다.

2. 관련 연구

최근에 효율적인 질의처리와 많은 양의 XML 데이타 관리를 위한 XML 저장 시스템에 대한 연구가 많이 진행되고 있다. 특히 관계형 데이타베이스 시스템 분야에

서 XML 문서를 저장하려는 하는 연구는 더욱 활발히 진행되어 왔다[4,5,6]. 주어진 DTD(Document Type Definition)를 통해서 관계형 데이타베이스의 스키마를 생성하는 방법으로 XML 데이타를 저장하는 연구[4,5]와 DTD없이 XML 데이타를 관계형 데이타베이스에 저장하는 연구[6]가 수행되었다. 그리고 Lore[2]에서는 객체지향 데이타베이스 시스템인 O2[7]에 XML 데이타를 저장하는 연구가 수행되었고 객체지향 데이타베이스 시스템 관점에서 설계되었지만 객체의 단위를 XML 파일 하나로 취하는 연구도 수행되었다[8]. 그리고 파일 시스템을 통해서 텍스트 파일을 저장하려는 연구도 진행되었다[10].

현재 많은 관계형 데이타베이스 시스템 판매업체들이 XML을 지원하려는 노력을 하고 있는데 기존의 데이타베이스 시스템에서 관리하는 데이타를 추출할 때 XML로 변환시켜 주는 작업이나 XML 데이타를 저장하기 위한 시스템을 고안하고 있다. IBM의 DB2 XML Extender[11]는 전체 XML 문서를 하나의 LOB 형태로 저장하거나 DTD를 통해서 XML 문서를 테이블의 집합으로 나누어 저장하는 방법을 사용한다. Microsoft SQL Server 2000의 OPENXML은 특정 메타정보를 통해서 테이블에 레코드를 삽입하고 갱신하는 방법을 보여준다[12]. Oracle의 XML SQL 유틸리티[13]도 다른 관계형 데이타베이스 시스템과 비슷한 방법을 취하고 있다. 그밖에 객체지향 데이타베이스 시스템을 이용한 POET[14]나 Excelon[15] 등도 널리 알려진 상용 XML 저장 시스템들이다.

많은 관련 연구들에서 XML 문서를 파일 시스템 혹은 데이타베이스 시스템에 저장하려는 노력들이 진행되었으나 실제 상용 시스템에 이런 것들이 적용된 사례는 극히 드물다. 관계형 데이타베이스 시스템에 XML을 저장하는 기본적인 기법 중에는 DTD를 이용한 스키마 생성 방법[5], 모든 XML 데이타의 간선(Edge)을 테이블로 표현하는 간선 테이블[6] 방법, 간선 테이블의 레코드를 수평분할해서 여러 테이블로 나누는 애트리뷰트 방법[6] 등이 연구되었으나, 실제 상용 시스템[11~13]에서는 이런 기법들을 적용하지 않고 있다.

이렇듯 기존의 연구방법과 상용 시스템에서는 데이타베이스 시스템을 통한 XML 데이타 저장에 관련하여 기존의 데이타베이스 시스템이 가지고 있는 기술을 XML 데이타 저장에는 충분히 활용하지 못했다. 또한 XML 데이타에 대한 질의 처리 시 질의 변환을 위한 비용이나, 데이타베이스 시스템에 저장되어 있는 XML 데이타에 대한 갱신 작업에 소요되는 비용의 문제를 간과하고 있다. 이렇듯 연구 방법과 상용 시스템 사이의 괴리는 기존의 데이타베이스 시스템에 XML 데이타를 저장하여 관리하는 방법이 부적절하다는 것을 반증해주는 예로 볼 수 있다.

최근에는 파일 시스템 혹은 데이타베이스 시스템을 통한 저장 방법에서 벗어나 XML 전용 저장소에 관련된 연구들이 진행되고 있다[1~8]. [1]에서는 XML 문서를 직접 저장하지 않고 DOM 객체를 저장하여 DOM의 영속성을 보장하는 저장 방법이 제안되었다.

본 논문에서 제안하는 XDOM은 객체 저장소[3]를 기반으로 하여 영속적인 DOM 객체를 저장하기 위한 시스템이다. 이는 DOM 객체를 바이너리 파일 형태로 저장하여 영속성을 보장하는 PDOM[1]과 비슷한 모델로 구현되었다. PDOM은 영속적인 DOM 객체에 대한 지원에만 주안점을 두고 연구되었기 때문에 기존의 저장소들과는 개념적으로 다르다. 단지 XML 문서를 응용 프로그램이 쉽게 사용할 수 있는 바이너리 파일로 변환하는 것에만 초점이 맞춰진 것이다. 이로 인하여 PDOM은 멀티쓰레드를 지원하지 않고 DOM 객체가 아닌 다른 객체의 저장을 허용하지 않음으로써 확장성이 부족하다는 단점을 갖는다. 본 논문에서 제안하는 XDOM은 PDOM의 기본 모델을 기반으로 PDOM이 가지고 있는 문제점을 해결하여 좀 더 일반적인 XML 전용 저장소를 구현하였다. XDOM이 갖는 확장성은 [16,17]에 잘 보여준다. 각각 시그너처[16]와 XLink 시맨틱 정보[17]를 담은 객체를 XDOM에 저장하여 사용하고 있다.

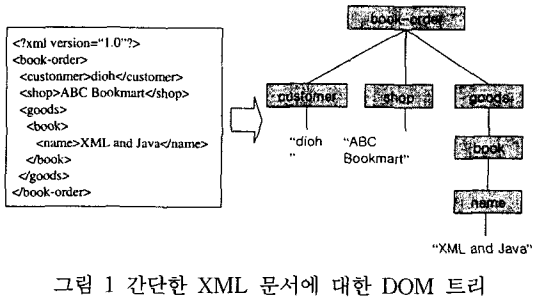

그림 1 간단한 XML 문서에 대한 DOM 트리

3. XDOM 시스템

3.1 DOM의 개요

XML 문서는 트리 형태의 데이타 구조를 가지고 있다. 이와 같은 형태의 자료 구조를 접근하기 위해서 W3C에서는 DOM이라는 인터페이스를 채택하였다. 현재 대부분의 XML 응용 프로그램들은 DOM API를 XML 문서 조작의 표준 인터페이스로 사용하고 있다.

그림 1은 간단한 XML 문서에 대한 DOM 트리를 보여준다. 예를 들어, 사용자가 현재 book-order 노드에 해당하는 객체를 참조하고 있다면 이 객체의 getFirstChild() 라는 함수를 통해서 customer노드에 해당하는 객체를 얻을 수 있다. 이렇듯 DOM 인터페이스는 트리 조작에 필요한 각종 함수들을 제공함으로써 응용 프로그램이 쉽게 노드 탐색을 하도록 도와주며, XML 문서의 구조 변경등의 다양한 트리 조작이 가능하도록 해준다. DOM은 저장소에 저장되어 있는 XML 문서들을 파싱해서 그 결과로 생성되는 객체들을 메모리 상에 상주시키고, 응용 프로그램은 이들을 DOM 표준 인터페이스를 통해서 참조하여 사용한다. 이런 방식의 문제는 메모리상의 DOM 객체와 기존의 저장장치에 있는 XML 파일 사이의 일관성 결여에 있다. 이는 많은 응용 프로그램이 DOM 객체 사용을 통한 문서의 조작을 수행할 경우 변경된 내용이 즉시 저장소에 반영되지 않고 응용 프로그램이 종료되기 직전에 일괄적으로 반영하기 때문이다. 또한 매번 번거로운 파싱 작업을 수행해야 한다는 부담을 갖는다. 뿐만 아니라, 대용량의 XML 파일의 경우는 메모리에 모든 객체가 상주하기 힘든 경우도 발생한다. 이러한 문제점들은 대부분의 저장소들이 DOM 객체를 지속적인 관리 대상으로 보지 못하고 단순히 XML 문서를 접근하기 위한 인터페이스로만 보기 때문에 발생한다.

제안하는 시스템에서는 기존의 메모리 상에 만들어지는 DOM 객체를 파일 상에서 영구적으로 보전되는 XDOM 객체와 구분하여 인-메모리 DOM 객체로 명명한다.

3.2 XDOM의 구조 및 특징

XDOM은 DOM 객체 저장소로서 객체 캐쉬(Object Cache)를 통해 응용 프로그램에 필요한 DOM 객체를 제공한다. 그림 2는 XDOM에 대한 구조를 간단하게 보여준다. 여기에서 IO Thread는 파일 매니저 역할을 수행하여 파일에 저장되어 있는 DOM 객체에 대한 접근을 관리하면서 메모리의 객체 캐시에 DOM 객체를 업로드를 한다. 객체 캐쉬는 파일에 저장된 DOM 객체와 사용자 쓰레드에서 요구하는 객체 사이의 중간 역할을 수행하며 두 가지 경로를 통해 사용자에게 객체에 대한 접근을 허용한다. 하나는 DOM API이고 다른 하나는 XDOM 시스템을 관리할 수 있는 특별한 API인 Extended XDOM API이다. 사용자 쓰레드에서는 파일에 어떠한 형태로 저장되어 있는지는 알 필요 없이 DOM API를 통해서 XML 문서에 대한 요청만 하게 되므로 사용자 레벨에서는 투명성을 보장하게 된다.

XDOM의 특징으로는 논리적 모델과 물리적 모델의

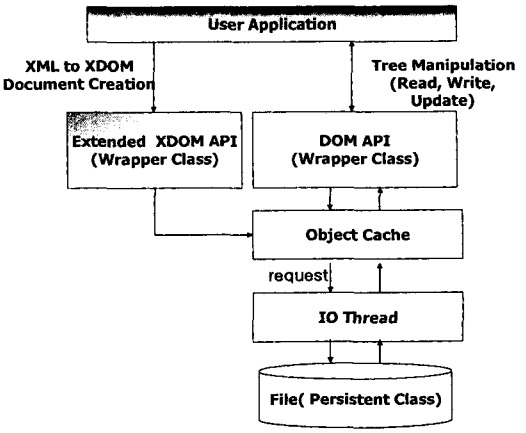

그림 2 XDOM Architecture

분리와 확장성을 들 수 있다. XDOM은 사용자에게 제공되는 객체와 물리적으로 저장되는 객체를 분리하여 논리적 모델과 물리적 모델을 분리한다. 이를 통해 메모리 관리가 용이해진다. 그림 2를 보면 파일에 저장되는 객체와 API에서 제공되는 객체가 다르다는 것을 알 수 있다. 객체의 분리를 통해 객체 캐쉬에 있는 객체를 쉽게 교체할 수 있기 때문에 메모리 관리를 효율적으로 수행할 수 있다. 또한 XDOM은 임의의 객체에 대한 저장을 지원하여 확장이 용이한 구조를 가진다. XDOM은 기본적으로 DOM 객체에 대한 저장뿐만 아니라 다양한 기능을 해줄 수 있는 여러 가지 객체에 대한 저장을 보장한다. 예를 들어, XML 문서에 효율적인 질의 처리를 위한 특정 정보를 저장하는 객체나 XML 문서간의 링크 정보를 담고 있는 XLink의 의미 정보[12]를 담고 있는 객체 등을 저장할 수 있다. 이렇듯 XDOM은 XML 데이타에 대한 저장장치로써 많은 기능을 쉽게 추가할 수 있는 확장성이 뛰어난 구조를 가진다.

3.3 Persistent 객체 & Wrapper 객체

XDOM에서 XML 데이타의 요소들을 DOM 객체로 만들고 저장하여 이를 사용하는데 있어 가장 특징적인 것은 DOM 객체에 대한 물리적 모델과 논리적 모델의 분리 즉, 구현과 인터페이스의 분리이다. 여기서 구현은 DOM 객체가 가지는 데이타를 의미하고 인터페이스는 이런 데이타를 통해서 제공되는 여러 가지 로직을 의미한다. 물리적 모델과 논리적 모델이 분리되어 있기 때문에 물리적 모델의 변경이 논리적 모델에 영향을 미치지 않고 그 반대의 경우도 마찬가지이다. 이를 통해 시스템의 물리적 모델과 논리적 모델의 독립 구현이 가능하다.

이러한 특징을 반영하기 위해 XDOM에서는 기존의

인-메모리 DOM 객체를 두 가지 객체로 분리하여 구현하게 되는데, 그 한 가지는 구현에 해당되는 객체로써 Persistent 객체라 하며 실제 DOM 객체에 대한 중요한 정보인 객체간의 관계와 객체 자신에 대한 정보를 저장해놓는다. Persistent 객체는 실제 저장소에 저장되는 객체이다. 다른 한 가지는 Wrapper 객체로써 DOM 객체에 필요한 정보들을 직접 가지지 않고 인터페이스만을 제공해준다. Wrapper 객체는 저장소에 따로 저장되지 않고 인터페이스만을 제공하는 경량의 객체이다.

그림 3은 그림 1에서 보여준 XML 데이타들에 대한 Persistent 객체를 보여준다. 각각의 노드에 해당하는 Persistent 객체는 자신의 OID(Object Identifier)를 가지고 있으며 다른 노드들간의 관계 정보를 객체의 참조가 아닌 OID 값을 통해 가지고 있다. 그림 4는 Wrapper 객체의 예를 보여주고 있다. Wrapper 객체는 OID 정보만을 가지고서 DOM 객체의 인터페이스를 제공한다.

특정 인터페이스에 필요한 모든 데이타는 Persistent 객체를 통해서 가져오게 된다. 그림 5에서는 이들 두 객체의

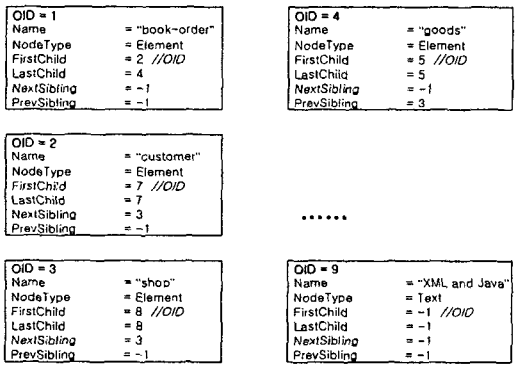

그림 3 Persistent 객체

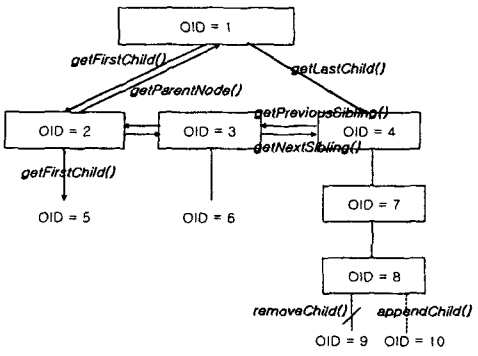

그림 4 Wrapper 객체

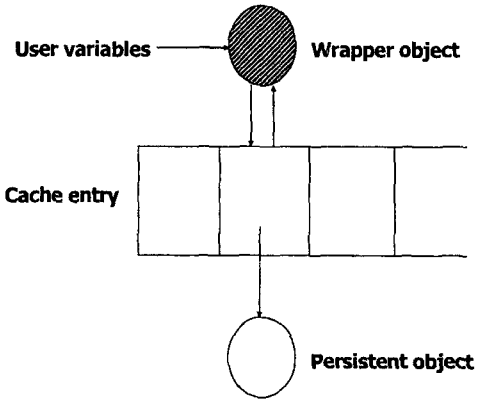

그림 5 Persistent & Wrapper 객체 동작 원리

동작 원리를 간단하게 보여주고 있다. 예를 들어, 사용자 변수가 OID가 1번인 Wrapper 객체의 getFirstChild() 라는 함수를 호출하게 되면 Wrapper 객체의 getFirstChild()는 Persistent 객체로부터 FirstChild값이 2라는 사실을 알게된다. 이를 통해서 getFirstChild()는 사용자 변수에 OID가 2번인 새로운 Wrapper 객체를 생성해서 반환한다.

구현과 인터페이스의 분리를 통해서 XDOM에서는 DOM 객체를 용이하게 저장할 수 있고 메모리의 효율적인 관리가 가능하며 객체 캐쉬에서의 객체 교체 작업을 수월하게 진행할 수 있다. Persistent 객체를 저장할 때 Persistent 객체는 다른 객체에 대한 참조 값을 데이타로 가지지 않으므로 객체를 저장할 때 있어 객체의 직렬화가 쉽게 보장된다. 그리고 사용자 쓰레드에서 사용자 변수들이 많은 Wrapper 객체를 참조해도 Wrapper 객체 자체는 메모리를 크게 차지하지 않는 객체이므로 일정한 메모리 사용이 보장될 수 있다. 또한 Persistent 객체는 사용자 변수나 Wrapper 객체에 의해 참조되지 않으므로 객체 캐쉬에서 교체 작업이 쉽게 이루어질 수 있다. 다시 말해 객체 캐쉬에서 희생물로 선정된 객체들은 특정한 사용자 변수에 의해서 참조되는 경우가 없기 때문에 쉽게 메모리에서 쓰레기 수집이 될 수 있다는 것이다.

3.4 객체 캐쉬

객체 캐쉬(그림 2 참조)는 사용자 쓰레드로부터 Persistent 객체로의 임의 접근을 배제하고 Wrapper 객체를 통해서만 접근을 허용한다. 객체 캐쉬는 인터페이스의 요청에 대한 결과도 Wrapper 객체를 통해서 하기 때문에 사용자 수준에서 기존의 인-메모리 DOM 객체의 사용과 동일하게 보이도록 하위레벨을 감추어 주는 역할을 한다. 또한 객체 캐쉬는 XDOM의 사용 환경에

따른 캐쉬 엔트리 크기도 쉽게 조정이 가능하여 리소스가 적은 환경에서나 범용 저장소로 XDOM이 쉽게 적용될 수 있도록 한다.

객체 캐쉬는 객체 분리를 통한 메모리 사용 조절과 동시성 제어를 가장 큰 특징으로 한다. 객체 캐쉬는 경량의 Wrapper 객체와 저장소에 저장되어 있는 Persistent 객체를 분리하여 Persistent 객체가 응용 프로그램의 사용자 변수로부터 직접 참조되지 않도록 한다. 따라서 Persistent 객체는 메모리에서 쉽게 쓰레기 수집이 되고 이를 통해 XDOM은 메모리 관리를 용이하게 한다.

객체 캐쉬를 통한 동시성 제어는 로킹(locking)을 통해서 수행이 된다. 사용자 쓰레드는 두 가지 로크 모드에서 수행된다. 일반적으로 사용자 쓰레드는 사용자 모드에서 일을 수행하는데 캐쉬에 객체를 요구할 때 캐쉬 전체에 로크를 걸고 캐쉬 모드로 진입한다. 이는 OS에서 커널에 시스템 콜을 하는 것과 비슷한 형태로 동작을 한다. 캐쉬 모드로 진입을 하였을 경우, 캐쉬에 로크를 잡고 있는 쓰레드는 신속히 일을 처리하고 다시 사용자 모드로 돌아간다. 이때 객체에 대한 I/O 작업이 요구되는 경우는 IO Thread에 해당 작업을 위임하고 사용자 모드로 돌아간다. 이러한 일련의 로킹 작업을 통해 멀티 쓰레드 환경에서 동시성 제어를 수행할 수 있다.

이 외에도 객체 캐쉬는 메모리 상에서 캐쉬 엔트리에 정해진 개수만큼의 Persistent 객체를 유지하여 일정한 메모리 사용을 보장한다. 또한 파일 시스템에 저장되어 있는 Persistent 객체와 사용자 쓰레드의 사용자 변수에 의해서 참조되는 Wrapper 객체 사이에서 두 객체 사이의 동기화를 보장한다. 뿐만 아니라 메모리에 객체를 캐슁함으로써 디스크의 I/O를 줄이는 역할을 한다.

그림 6은 객체 캐쉬의 구조와 간단한 동작 원리를 보여주고 있다. 특정 쓰레드의 Wrapper 객체가 Persistent 객체의 데이타를 얻기 위해 객체 캐쉬에 Persistent 객체를 요청하게 되면 일단 해당 Persistent 객체가 캐쉬

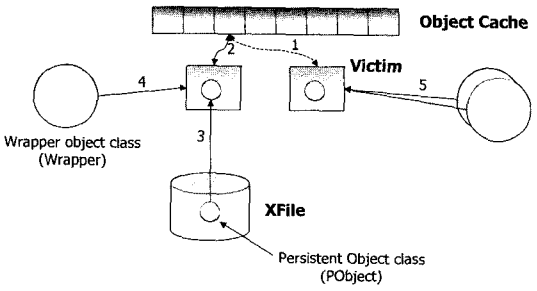

그림 6 객체 캐쉬

엔트리에 있는지를 판단한다. 만약 존재한다면 다른 쓰레드에 의해서 사용 중인지를 판단하고 사용 중이면 다른 쓰레드에 의해서 로크가 풀릴 때까지 기다린다. 만일 캐쉬 엔트리에 해당 Persistent 객체가 존재하지 않는다면 디스크로부터 객체를 가져오는 작업을 수행한다. 이때 캐쉬 엔트리의 빈 공간이 없을 경우는 캐쉬 엔트리에서 적당한 교체작업을 통해서 빈 공간을 확보한 다음 그 공간에 디스크로부터 읽은 Persistent 객체를 올린다. 이때 이뤄지는 교체 정책은 LRU(Least Recently Used)를 따른다. 이렇게 확보된 Persistent 객체의 데이타를 가지고 Wrapper 객체의 특정 함수는 원하는 작업을 수행한다. 객체 캐쉬는 Persistent 객체를 가져오는 작업뿐만 아니라 새로운 Wrapper 객체를 생성해서 반환하거나 Persistent 객체의 데이타를 갱신하는 작업도 수행한다.

앞서 살펴보았듯이 객체 캐쉬는 기존의 캐시의 역할인 디스크 I/O를 줄이는 것 뿐만 아니라 Wrapper 객체와 Persistent 객체의 중간 매개체 역할을 한다. 또한 다음 절에 소개될 IO Thread와 연동하여 동시성 제어를 수행한다.

3.5 IO Thread

XDOM의 IO Thread(그림 2 참조)는 사용자 쓰레드와 별도로 수행되면서 디스크 상의 입출력 작업을 전담하는 쓰레드이다. 멀티 쓰레드 환경에서 다수의 사용자 쓰레드가 저장소로부터 독립적으로 I/O 작업을 수행할 때 동시성 제어와 같은 문제를 일으키게 된다. 제안하는 시스템에서는 객체 캐쉬가 이런 동시성 제어를 처리하나 객체 캐쉬가 I/O 작업까지 처리하게 되면 동시성 제어의 효율이 떨어지게 된다. 따라서 객체 캐쉬와 파일 사이에서 IO Thread가 사용자 쓰레드와 동시에 수행되면서 모든 I/O 작업을 처리한다. 사용자 쓰레드는 I/O 작업이 수행할 때 캐쉬 모드로 객체 캐쉬에 대한 사용권한을 획득한 후에 바로 IO Thread에 I/O 작업을 넘겨준다. IO Thread는 GZIP을 통해서 파일 시스템에 Persistent 객체를 압축하여 저장하고 압축된 객체를 가져오기 때문에 I/O 작업 속도의 효율을 높이게 된다.

3.6 DOM API

XDOM에서는 DOM 레벨 1의 모든 인터페이스를 구현하였다[1]. 그림 7에서는 이를 위해 구현한 클래스의 구조를 보여주고 있다. 그림의 구조는 DOM API에서 제공하는 인터페이스의 상속 구조와 비슷하나 XDOM에

1) DOM 레벨 1 API의 목록 등의 세부 설명은 [9]를 참고하기 바란다.

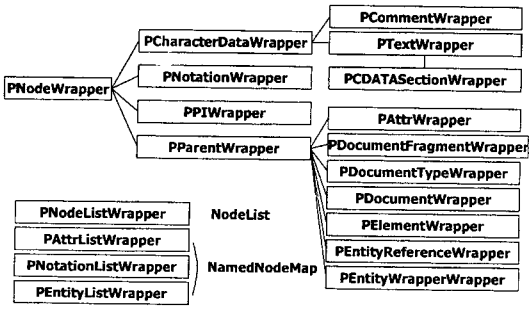

그림 7 DOM API

서는 몇 가지 차이점을 가지고 구현하였다. 먼저 PParent Wrapper를 클래스를 만들어 자식이 있는 노드와 없는 노드를 분리해서 구현하였다. 이는 자식에 대한 정보를 가질 필요가 없는 클래스(Comment, Text CDATA Section)에 대해서는 저장소 안에 자식 정보를 가지게 할 필요가 없도록 하기 위함이다. 그리고 Named NodeMap과 같은 경우에는 이 클래스를 이용하는 클래스들(Attirbute, Notation, Entity)이 서로 다른 방법으로 동작을 하게 되기 때문에 분리 구현하였다. 그리고 NodeList나 NamedNodeMap을 구현한 클래스는 영구적인 정보를 가지지 않고 상황에 맞추어 만들어지므로 저장소에는 저장될 필요가 없는 클래스들이다.

XDOM에서 구현한 DOM API는 다음에 설명되는 특징을 가지고 있다. 실제 DOM 객체는 저장소에 저장되어 있고 이것을 객체 캐쉬를 통해서 사용자 쓰레드가 접근한다는 것은 3.4절에서 언급한 바 있다. 이러한 구조를 통해서 사용자 쓰레드는 DOM 객체를 요구하지만 실제 사용자 쓰레드의 사용자 변수는 객체 캐쉬에 있는 DOM 객체를 참조하지 않고 객체 캐쉬에서 만들어주는 DOM 객체를 얻어오게 된다. 이렇게 객체 캐쉬에서 만들어진 객체는 데이타를 직접 가지지 않고 다른 객체를 직접 참조하지 않으므로 아주 경량의 객체가 된다는 사실이다. 이런 구조는 사용자 쓰레드가 DOM API를 통해서 많은 객체를 요구해서 메모리에 상주시킨다고 해도 메모리에 많은 부분을 차지하지 않게 되고 쓰레기 처리도 용이하게 해준다.

3.7 확장 XDOM API

확장 XDOM API는 DOM을 이용하는 응용 프로그램을 위해 제공되는 API가 아니라 시스템 관리 응용프로그램이나 XML 데이타를 관리하는 응용프로그램 등이 사용을 하게 되는 API를 제공한다. 그림 8은 여러 가지 간단한 API의 예제를 보여주고 있다. XDOM은 바이너

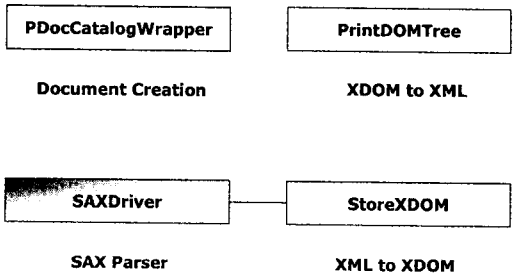

그림 8 확장 XDOM API 예제

리 파일 하나에 여러 가지 문서를 저장하게 되므로 이것을 효율적으로 관리할 수 있도록 카탈로그를 두고 이를 통해서 문서를 접근할 수 있도록 한다. 카탈로그는 문서를 리스트 형태로 가지고 사용자의 요청에 의해서 문서의 도큐먼트 노드(Document Node)를 반환하게 된다. 카탈로그에는 문서를 검색할 수 있는 API도 제공하고 있고 삽입, 삭제 등은 물론 다양한 API를 포함한다.

확장 XDOM API에는 카탈로그뿐만 아니라 실제 문서를 저장하기 위한 특별한 API도 제공되고 있다. 그림 8에서는 SAX 파서를 통해서 파싱을 하고 이를 XDOM 파일에 저장하는 API인 StoreXDOM API를 제공하는 예제로 보여주고 있다. DOM 파서는 파싱하는 과정에서 많은 리소스를 차지할 뿐 아니라 속도 면에서 비효율적이게 되므로 SAX(Simple API for XML) 파서를 사용하고 있다. 파싱을 통해 XML 문서를 스캔하면서 스택을 통해 문서를 트리 구조로 만들고 이를 XDOM에 저장하게 된다.

이 외에도 확장 XDOM API에서는 PrintDOMTree와 같이 XDOM에 저장되어 있는 XML 문서를 텍스트로 바꾸어 주는 기능을 하는 API도 제공하고 있다.

4. 성능 평가

본 장에서는 XDOM에 대한 성능 평가를 보인다. 우선 2장 후반부에서도 언급한 바 있는 PDOM[1]과 기존의 인-메모리 DOM 방식과의 XML 데이타 처리 능력을 비교한다. 그다음으로 멀티 쓰레드 환경을 지원하는 부분에서 PDOM과의 수행 차이를 보임으로써 제안하는 시스템의 우수성을 보이기로 한다.

4.1 XML 데이타 처리 능력 비교

표 1은 XML 데이타를 탐색[2]하는데 소요되는 시간과 메모리 사용량을 실험한 결과이다. 실험은 펜티엄 Celeron

2) XML 데이타의 탐색은 TreeWalker 인터페이스[18]를 사용하여 수행하였다.

표 1 XDOM과 PDOM 그리고 인-메모리 DOM 성능평가

	nasa (cost/heap size)	department	shakespeare
XDOM	1209ms/600K	952ms/300K	2030ms/500K
PDOM	1176ms/800K	901ms/450K	2720ms/900K
인-메모리 DOM	2738ms/2000K	1001ms/900K	3297ms/1300K

566MHz, RAM 256MB의 환경에서 수행하였다. 실험에 사용한 XML 데이타로는 nasa[19], department[20], shakespeare[21] 데이타의 3가지 데이타 집합을 사용하였다. nasa 데이타는 복잡한 데이타를 표현하기 때문에 트리의 깊이가 깊고 분기도 많은 특징을 갖는다. department 데이타는 학교 내의 부서를 표현하는 XML 문서로 트리의 깊이는 얕으나 분기가 많고, 같은 형태의 데이타가 반복되는 특징을 갖는다. shakespeare 데이타는 연극대본으로 전체적으로 트리의 깊이가 얕고 텍스트 노드가 많은 부분을 차지한다.

수행 시간을 비교해보면 XDOM과 PDOM이 거의 비슷한 시간으로 수행하였고 인-메모리 DOM의 경우는 현저하게 떨어지는 모습을 볼 수 있다. 이는 인-메모리 DOM은 수행 속도는 빠르지만 파싱하는데 걸리는 시간이 전체 소요 시간에서 고정적으로 많은 부분을 차지하기 때문이다.

메모리 크기의 경우에도 인-메모리 DOM의 경우가 XDOM이나 PDOM에 비해 상대적으로 매우 큰 것을 볼 수 있다. 이는 PDOM과 XDOM은 파일에 영속적인 DOM 객체를 저장하는 방식을 통해 일정한 크기의 메모리 공간을 차지하는 반면 인-메모리 DOM의 경우 메모리의 크기가 파일 크기에 비례해서 커지기 때문이다.

4.2 멀티 쓰레드 환경의 수행 능력 비교

표 2는 XDOM과 PDOM의 멀티쓰레드 수행능력을 비교한 결과로 각각에 5개의 쓰레드를 가지는 응용 프로그램을 수행하여 테스트를 하였다.

PDOM은 하나의 같은 문서에 여러 개의 쓰레드가 동시에 접근하는 것과 서로 다른 여러 문서들을 여러 개의 쓰레드가 병렬적으로 접근하는 두 가지 멀티 쓰레드 환경에서 교착상태에 빠져 프로그램이 중단되는 단점을 보였다. 이에 비해 XDOM은 멀티 쓰레드 환경에서도 정상적으로 수행됨을 볼 수 있다.

표 2 멀티 쓰레드 환경에서의 XDOM과 PDOM

	모두 같은 XML 문서들	서로 다른 XML 문서들
5개 쓰레드(PDOM)	X	X
5개 쓰레드(XDOM)	O	O

5. 결 론

정보의 저장에 있어서 파일 시스템이나 데이타베이스 시스템에 저장하는 방법은 가장 보편적이지만 XML과 같은 데이타 모델을 저장하고 관리하는데 있어 여러 가지 문제점이 노출되었다. 기존의 저장소는 XML 데이타의 모델에 적합하지 않고 정형화된 구조가 없는 문서를 저장할 때 많은 문제점이 있다. 이에 본 논문에서는 영구적인 DOM 객체를 기반으로 하는 새로운 저장소인 XDOM을 통해 문제의 해결방안을 제시했다.

XDOM은 기존의 데이타베이스 시스템과는 달리 스키마를 생성할 필요가 없어 정형화되어 있지 않은 문서들도 쉽게 표현할 수 있다. 또한 XDOM은 사용자에게 제공되는 논리적 객체와 저장소에 저장되는 물리적 객체를 분리하였다. 사용자에게는 논리적 객체만 제공되므로 XDOM 시스템 내부 구조를 감추고 DOM 객체 사용을 투명하게 보장한다. 논리적 객체는 데이타를 가지지 않고 인터페이스만을 제공하기 때문에 경량의 객체로 메모리에 상주하기 때문에 메모리 관리를 용이하게 한다. 물리적 객체는 데이타를 가지고 파일에 저장되는 객체이다. 물리적 객체는 객체 캐쉬를 통해서만 관리가 되므로 쉽게 메모리에서 교체가 될 수 있어 쓰레기 수집을 용이하게 한다. 이런 논리적 물리적 기법을 통해서 XDOM은 일정한 메모리 사용을 유지하고 메모리 관리를 용이하게 한다. 사용자와 저장소 사이에 객체 캐시는 두 개로 분리되어 있는 객체사이의 연결을 원활하게 수행할 수 있게 해주고 캐쉬를 통해서 시스템의 성능을 향상 시켰다. XDOM은 사용자 쓰레드와 파일 I/O 쓰레드를 분리하여 멀티 쓰레드 환경에 적합하도록 설계가 되어있고 DOM 객체뿐만 아니라 다양한 임의의 객체에 대한 저장도 원할히 수행될 수 있도록 되어 있어 확장성을 보장하고 있다.

본 논문에서는 객체 캐쉬를 통해서 경량의 DOM 객체 저장소를 구현하였다. 실제 본 논문에서 제안한 시스템의 성능 향상을 위해서는 다양한 전략이 필요하다. 특히 객체 캐쉬 이하에서 다양한 기술 적용이 필요하다. 예를 들어, 캐쉬 교체 전략에 있어 현재 사용중인 LRU 정책을 좀 더 효율적인 LRU-K 교체 알고리듬을 이용해 캐시에서의 속도를 향상시킬 필요가 있고 파일에 저장되는 객체의 사이즈를 효율적으로 압축해서 파일에 저장된 객체를 읽어올 때 좀 더 효율적으로 수행할 수 있도록 할 필요가 있다. 그리고 파일에 저장되어 있는 객체들에 대해 각 객체들과 관련 있는 객체들은 같은

페이지 안에 배치시켜 객체가 캐시에 올라올 때 관련 있는 다른 객체들도 같이 캐시에 올려질 수 있도록 하는 것도 연구할 필요가 있다. 본 논문은 확장성을 기반으로 하고 있기 때문에 앞으로 다양한 시스템과의 연동에 대한 연구도 기대된다.

현재 본 논문에서 제안하는 시스템을 확장하는 XML 데이타의 관리 시스템을 개발 중이다. 본 논문에서는 XML의 저장 측면에 주로 초점을 둔 시스템을 제안하였으나, 개발 중인 시스템에서는 저장 측면 이외의 데이타의 삽입, 삭제, 변경 및 버전 관리, 사용자 관리 등의 각종 데이타 관리 기능을 포함할 계획이다.

참 고 문 헌

[1] Gerald Huck, Ingo Macherius, Peter Fankhauser.. "PDOM: Lightweight Persistency Support for the Document Object Model," *OOPSLA'99*, Denver, 1999.

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

[3] Dallan Quass, Jennifer Widom, Roy Glodman, Kevin Haas, Qingshan Luo, Jason McHugh, Svetlozar Nestorov, Anand Rajaraman, Hugo Rivero, Serge Abiteboul, Jeffrey D. Ullman, Janet L. Wiener: Lore: "A Lightweight Object Repository for Semi-structured Data," *SIGMOD Conf.* 1996: 549.

[4] A. Deutsch, M. f. Fernandez, D. Suciu, "Storing Semi-structured Data with STORED," *SIGMOD Conference* 1999 : 431-442.

[5] Daniela Florescu and Donald Kossmann. "Storing and querying XML data using an RDBMS," *IEEE Data Engineering Bulletin*, 1999.

[6] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, J. F. Naughton, "Relational Database for Querying XML Documents: Limitations and Opportunities," *VLDB* 1999: 302-214.

[7] F. Bancihon, G. Barbedette, V.Benzaken, C. Delobel, S. Gamerman, C. Lecluse, P. Pfeffer, P. Richard, F. Velez. "The design and implementation of O2, and object-oriented database system," *In Proceedings of the second international workshop on object-oriented database*, 1988, ed. K Dittrich.

[8] C. Kanne, G. Moerkotte, "Efficient storage of XML data," *International Conference of Data Engineering 2000*, pp198.

[9] Lauren Wood (Ed.): Document Object Model(DOM) Level 1. W3C Recommendation, 1998.

[10] S Abiteboul, S. Cluet, T Milo. "Querying and updating the file," *VLDB* 1993, pp 73-84.

[11] IBM DB2 XML Extender.http://www4.ibm.com/software/data/db2/extenders/xmlext/

[12] Microsoft SQL Server 2000 Books Online, XML and Internet support

[13] Oracle XML SQL utilities. http://otn.oracle.com/tech/xml/oracle_xsu/

[14] POET content manager suit. http://www.poet.com/

[15] Excelon, the ebusiness information server. http://www.odi.com/excelon/

[16] 박희경, XLink. http://oopsla.snu.ac.kr/rnd2001/xlink.html

[17] 최윤라, 시그너처와 DTD를 이용한 XML 질의 최적화. http://oopsla.snu.ac.kr/rnd2001/

[18] http://www.w3.org/TR/1999/CR-DOM-Level-2-19991210/traversal.html#TreeWalker

[19] NASA, http://xml.gsfc.nasa.gov/guides/

[20] http://www.cs.wisc.edu/nigagra/data/department/department.dtd

[21] http://metalab.unc.edu/bosak/xml/eg/

오 동 일

2000년 서울대학교 컴퓨터공학과 학사
2002년 서울대학교 컴퓨터공학과 석사
현재 (주)한국오라클 근무. 관심분야는 XML, 데이타베이스

최 일 환

1996년 서울대학교 컴퓨터공학과 학사
1998년 서울대학교 컴퓨터공학과 석사
1998년~현재 서울대학교 컴퓨터공학과 박사과정 재학중. 관심분야는 XML, 데이타베이스

박 상 원

1994년 서울대학교 컴퓨터공학과(학사)
1997년 서울대학교 컴퓨터공학과(석사)
1997년~현재 서울대학교 컴퓨터공학부 박사과정. 관심분야는 데이타베이스, XML, Semistructured data, Web

김 형 주

1982년 서울대학교 전자계산학과(학사)
1985년 Univ. of Texas at Asution(석사)
1988년 5월~1988년 9월 Univ. of Texas at Austin. Post-Doc. 1988년 9월~1990년 12월 Georgia Institute of Technology(부교수). 1991년 1월~현재 서울대학교 컴퓨터공학부 교수