韓國 컴퓨터情報學會 論文誌
第10卷 第6號, 2005. 12.

2005-10-6-1-12

내장형 XML 문서 데이터베이스 관리 시스템의 설계 및 구현

안 병 태*, 서 익 진**

Design and Implementation Embedded XML Document DataBase Management System

Byung-Tae Ahn*, Ik-Jin Seo**

요 약

PDA와 같은 내장형 컴퓨팅 기술이 발달하면서 PIMS와 같은 다양한 응용들이 개발되었다. 그리고, 이들 응용 시스템들간 XML 문서를 통한 데이터 공유와 정보 교환이 활발히 이루어지고 있다. 이에 따라 데이터 처리량이 점점 증가하게 되어 내장형 시스템에서도 XML 문서 데이터베이스 관리 시스템의 필요성이 점점 대두되고 있다. 그런데 하드웨어적으로 많은 제약을 갖는 대부분의 내장형 시스템에서는 XML 문서를 처리하기 위하여 자체 파일 시스템을 이용하고 있다. 이로 인해 타 응용과의 효율적인 데이터 공유와 정보 교환, 그리고 효율적인 데이터 처리에 많은 애로사항이 발생하고 있다. 본 논문에서는 이러한 내장형 시스템 응용의 XML 문서를 통한 데이터 공유와 정보 교환을 효율적으로 지원하기 위한 내장형 XML 문서 데이터베이스 관리 시스템을 설계 및 구현하였다.

Abstract

With embedded computing technology makes progress, various applications have been developing in PDA. And information exchanging through XML document is underway actively between applications in internet environment. But most of the embedded system has poor hardware resource. And embedded system uses own file system for XML document management. For reason of, it occurs to many of difficulties of data share and information exchange among other systems. Therefore, an importance of the XML Document Database Management System is on the rise in the embedded system. In this paper, we design and implement Embedded XML Document Database Management System(EXDMS) for efficiently management of XML document and supporting data sharing and information exchanging through XML document in embedded system. As compared with file system, EXDMS shows results that more excellent performance about expandability and compatibility of application. Also, EXDMS has an excellent data processing performance in poor system environment.

▸ Keyword : 데이터베이스(Database), XML, 내장형 시스템(Embedded System)

• 제1저자 : 안병태
• 접수일 : 2005.10.15, 심사완료일 : 2005.10.29
* 밀양대학교 정보통신공학부, ** 경남대학교 경제무역학부

Ⅰ. 서 론

최근 내장형 컴퓨터 기술의 눈부신 발달로 PDA에서도 데스크탑 컴퓨터 수준의 다양한 응용들이 개발되고 있다[1, 2, 3]. 그리고 이러한 응용들은 인터넷 환경에서 일반적으로 XML 문서 형태로 정보를 교환하고 있다. 그런데 하드웨어적으로 많은 제약을 갖는 대부분의 내장형 시스템들은 XML 문서의 관리를 위해 파일 시스템을 사용하고 있다. 따라서 효과적으로 XML 문서를 다루는데 많은 애로를 겪고 있다.

내장형 시스템에서 XML 데이터를 효율적으로 관리하기 위해서는 XML 문서의 특성을 잘 고려하여야 한다. 무엇보다 XML 문서의 논리적인 계층 구조와 엘리먼트의 속성을 통한 검색 등이 지원되어야 한다. 하지만 상용의 관계형 데이터베이스는 이러한 XML 문서의 특성을 잘 반영하지 못하고 있다. 비록 XML 문서의 특성을 잘 반영하고 있더라도 대부분은 제한적인 시스템 자원을 갖는 내장형 시스템을 위해 개발된 것이 아니다. 이를 극복하기 위해서는 내장형 시스템에 적합하도록 XML 문서를 관리하는 데이터베이스 관리 시스템이 필요하다.

본 논문에서는 PDA와 같은 내장형 시스템에서 XML의 특성을 고려하여 XML 문서를 효율적으로 관리하면서도 응용 시스템 간에 데이터 공유와 정보 교환을 용이하게 하는 내장형 XML 문서 데이터베이스 관리 시스템 즉, EXDMS(Embedded XML document Database Management System)를 설계 및 구현하였다. EXDMS는 기존의 파일 시스템과 비교하여 응용의 확장성과 호환성 면에서 뛰어난 성능을 보였으며, 낮은 시스템 환경에서도 좋은 데이터 처리 성능을 보였다.

본 논문의 구성은, 2장에서 관련 연구들을 살펴보고, 3장에서는 EXDMS의 문서 정보 처리 방법을 설명한다. 4장에서는 EXDMS의 설계를, 5장에서는 EXDMS의 구현을 설명한다. 6장에서는 개발된 EXDMS의 성능을 기존의 파일 시스템과 비교한다. 끝으로 7장에서는 결론과 향후 과제를 기술한다.

Ⅱ. 관련 연구

최근 데이터베이스 시스템을 이용하여 XML 문서를 효과적으로 관리하려는 연구가 활발하다[4, 5, 6, 21, 22]. 하지만 이러한 연구들은 대부분 일반 데스크탑 컴퓨터에서 작동하는 시스템으로, 이동 환경의 PDA와 같은 내장형 시스템에 그대로 적용하기는 힘들다. 그렇지만 XML 문서를 데이터베이스로 관리하는 기본적인 방법과 원리는 동일하므로, 데스크탑 컴퓨터에서 XML 문서를 처리하는 방법론과 원리들은 내장형 시스템에서도 그대로 적용할 수 있다. 단지, 시스템 자원의 제약에 따라 데이터 관리를 위한 최소 기능과 이동 환경에 적합한 데이터 처리가 고려되어야 한다

2.1 XML 문서 관리 시스템

XML 문서를 관리하는 시스템은 크게 4가지 형태로 분류할 수 있다[7, 8, 9, 10]. 즉, (1) XML 전용 관리 시스템, (2) 관계형 데이터베이스 시스템을 이용하는 시스템, (3) 객체지향 데이터베이스 시스템을 이용하는 시스템, (4) 운영체제에서 지원하는 파일 시스템을 기반으로 하는 시스템으로 분류할 수 있다.

XML 전용 관리 시스템은 XML 문서 관리를 위한 전용 시스템으로서 XML 문서 처리가 매우 효율적이다. 하지만 XML 문서에 대한 전체 개념 모델이 부족하고, XML 질의 언어의 표준이 미흡하여 질의 처리를 위한 여러 정보들을 복합적으로 구성함으로써 데이터 모델의 정의를 조잡하게 하고 있다. 따라서 이러한 형태의 시스템을 위해서는 트리와 같은 XML 문서의 전체 형태를 잘 설명할 수 있는 개념 모델의 정의가 필요하고, XML 문서의 주요 구조 정보와 필수 구문에 대한 명확한 정의가 필요하다. 또한 질의 처리를 위한 효율적인 메커니즘과 간단한 형태의 데이터 모델을 정의할 수 있는 기능이 필요하다. 대표적인 시스템으로는 Astoria[11], Tamino[12], Timber[13] 등이 있다.

관계형 데이터베이스 시스템을 이용하는 시스템은 기존의 관계형 데이터베이스 시스템이 가지고 있는 강력한 데이터베이스 관리 기능들을 그대로 이용할 수 있다. 하지만 XML 문서를 데이터베이스에 구성하기 위해 테이블 단위로

분해해야 한다. 따라서 DTD 정보를 이용하여 데이터의 내부 구조를 설명하는 데이터베이스 스키마를 생성해야 하고, 생성된 스키마를 이용하여 XML의 전체 데이터 구조를 명확히 정의해야 한다. 또한 XML 문서의 데이터와 DTD 정보가 테이블에 개별적으로 관리되어야 하며, SQL을 통한 직접적인 접근이 가능해야 한다. 대표적인 시스템으로는 XML 저장 관리 시스템[4], Oracle8i[14], X-Database[15] 등을 들 수 있다.

객체지향 데이터베이스 시스템을 이용하는 시스템은, XML 문서의 모델링에 적합하고, 각 요소들의 구조를 정의하기 쉬울 뿐만 아니라, 상속과 같은 객체지향의 특성도 쉽게 사용할 수 있다. 하지만 내용 검색을 위해서 XML 문서의 모든 영역에 대한 인덱스가 생성되어야 하고, 객체에 대한 관리가 이루어져야 한다. 이와 함께 XML 문서에 대한 무결성이 보장되어야 한다. 즉, 저장 전의 XML 문서와 저장 후에 다시 복원한 XML 문서가 동일하도록 지원해야 한다. 대표적인 시스템으로는 eXcelon[16], Ozone[17] 등이 있다.

운영체제에서 지원하는 파일 시스템을 기반으로 하는 시스템은 데이터베이스 시스템이나 저장 관리자와 같은 추가 모듈이 필요하지 않고, 사용하기 쉬우며, 원본 데이터 재생성과 같은 추가 부담이 존재하지 않는다. 또한 순서 정보와 같은 정보의 손실없이 XML 데이터를 관리할 수 있다. 하지만 질의 처리 성능이 다른 방식에 반하여 상당히 떨어지고, XML 데이터의 일부만을 필요로 하는 경우에도 항상 전체 XML 문서를 파싱해야 하는 단점이 있다. 또한 XML 문서를 파싱하여 생성된 중간 결과물은 질의를 처리하는 동안 메모리 상에 상주시켜야 하며, 운영체제에서 지원하는 파일 시스템 자체가 지니는 동시성 제어, 보안, 파손 회복 등에 문제를 그대로 지닌다. 대표적인 시스템으로는 Trolltech 사 QTOPIA[18]가 있다.

본 논문에서 제안하는 EXDMS는 어떠한 데이터 모델도 가지고 있지 않은 내장형 데이터베이스를 이용하여 트리 형태의 XML 구조 정보와 XML 문서 내용 정보를 함께 구성하여 관리한다. 그리고 구성된 구조 정보를 바탕으로 스키마 정보를 생성하여 데이터 관리와 정보 검색 기능을 실현한다. 즉, XML 데이터 처리를 위하여 데이터의 표현은 관계형 테이블의 형태를 빌어 표현하고, 이를 키와 데이터 값의 쌍으로 구성된 레코드로 관리한다. 이러한 방법은 XML 문서를 엘리먼트 단위로 데이터베이스에 구성케 함으로써 XML 문서의 내용 추출시 발생되는 노드들의 통합을 크게 줄일 수 있다.

2.2 XML 문서 저장 방법

XML 문서를 하부 저장 구조에 저장하는 방법은 크게 3가지로 분류할 수 있다[4]. 즉, (1) XML 문서를 분할하여 저장하는 분할 저장 모델, (2) XML 문서를 그대로 저장하는 비분할 저장 모델, (3) 분할 저장 모델과 비분할 저장 모델을 혼합한 혼합 모델이다.

분할 저장 모델은 XML 문서를 엘리먼트 단위로 분할하여 저장하는 방법으로 문서의 일부 내용이 변경되어도 관련된 부분만을 수정하면 되기 때문에 문서의 편집과 관리가 용이하다. 그리고 동일한 내용을 가지는 노드들을 공유할 수 있는 장점이 있다. 하지만 문서의 내용을 추출할 때, 모든 노드들을 순회하면서 통합해야 하는 단점이 있다.

이에 반하여 비분할 저장 모델은 XML 문서 전체를 한꺼번에 저장하고, 가지거리(Offset) 정보를 이용하여 각 단말 노드에 접근한다. 따라서 XML 문서의 내용을 추출할 때 통합과정이 필요없고 빠른 문서 참조가 가능하다. 하지만 문서의 일부만을 수정하여도 문서 전체를 다시 재구성해야 하는 문제점이 있다.

혼합 모델은 위의 두 모델을 혼용하여 사용하는 모델로 각 모델의 단점을 보완하기 위해 상대 모델의 특성을 포함한다. 따라서 두 모델의 장점을 동시에 가질 수 있다. 하지만 이 모델은 두 모델이 유지하는 데이터 정보들을 모두 포함해야 하므로 많은 저장 공간을 필요로 한다.

본 논문에서는 XML 문서를 저장하기 위하여 분할 저장 모델을 사용한다. 이는 내장형 시스템이 지니는 제약적인 시스템 환경에서 XML 문서를 관리할 때, 타 방법보다 시스템 자원을 덜 사용하면서도 효율적인 처리를 할 수 있기 때문이다.

III. XML 문서 정보의 구성

XML 문서는 트리 형태의 구조를 가지고 있어 일반적으로 이를 처리하는 방법에 따라 XML 문서 관리 시스템의 처리 효율이 크게 달라진다. 본 논문에서는 XML 문서 처리를 위하여 우선 DTD 문서를 분석하여 DTD 트리를 생성하여 데이터베이스에 저장한 후, 저장된 DTD 트리를 이용하여 스키마 정보를 생성한다. 그리고 생성된 스키마 정보

와 XML 문서를 분석하여 XML 문서 트리를 생성하고 이 정보를 데이터베이스에 저장한다. 이 때, 생성되는 트리의 각 노드들은 효율적인 데이터 검색을 위하여 자신만의 노드 ID를 가지며, 데이터베이스는 기본적으로 이 노드 ID를 이용하여 모든 노드들을 관리한다.

3.1 문서 트리의 표현

XML DTD는 트리로 표현할 수 있다. 이에 따라 트리의 노드들을 효과적으로 다룰 수 있도록 ID를 부여할 수 있다. 그러나 이러한 방법은 트리 구조의 변경시 트리상의 모든 노드 ID들을 재구성해야 하는 단점을 가지고 있다[4]. 그래서 본 논문에서는 기존 방법의 장점을 가지면서도, 트리 구조의 변경에 대처할 수 있는 새로운 노드 ID 부여 방법을 제안한다. 즉, 본 논문에서 제안하는 노드 ID는 특정 노드에 대한 접근을 용이하게 하고, 부모 노드나 자식 노드를 추적할 수 있게 한다.

3.1.1 노드 ID의 구성

다음 (그림 1)은 우리가 제안하는 노드 ID의 기본 구성이다. 즉, 노드 ID는 네 개의 성분으로 구성되며, 구분자로 도트(.)를 사용한다.

그림 1. 노드 ID의 구성
Fig 1. Composition of Node ID

(그림 1)에서 Parent(부모) 성분은 부모 노드의 이름이다. 따라서 부모 노드나 형제 노드를 검색할 때, 이 Parent 값을 이용하면 된다. 그런데 트리에서 부모 노드와 동일한 이름을 가진 다른 노드가 존재할 수 있다. 이 때에는 Depth(깊이) 성분과 Group(그룹 번호) 성분을 이용하여 식별할 수 있다. 여기서 Depth 성분은 루트 노드로부터의 깊이 차를 나타내는데 항상 부모 노드의 깊이는 자식 노드의 깊이보다 1 작고 형제 노드들은 같은 값을 가진다. 따라서 Parent 성분 값이 동일한 여러 노드가 존재할 때도 부모 노드와 형제 노드를 별도로 식별할 수 있다. 만약 Parent 성분 값과 Depth 성분 값이 같은 노드가 존재한다면 이 때에는 Group 성분으로 노드를 식별한다. Sibling 성분은 형제 노드들에서 자신의 순서를 나타내며 형제 노드가 식별될 때마다 1씩 증가한다. Group 성분은 노드가 속한 서브트리의 그룹을 나타내는데 Group 성분 값은 특정 노드에서 자신의 형제 노드나 자식 노드를 인식할 때마다 혹은 형제 노드가 추가될 때마다 1씩 증가한다.

3.1.2 노드 ID의 부여 방식

노드 ID를 부여하는 순서를 정하는 방법은 크게 너비 우선 방식과 깊이 우선 방식으로 구분할 수 있다. 이 두 방식의 원리는 동일하지만 트리를 순회하는 순서가 달라지기 때문에 노드 ID의 Group 성분 값을 부여하는데 차이가 난다. 따라서, 너비 우선 방식은 파서가 DTD의 구성 요소들(ELEMENT, ATTLIST 등)을 파싱하는 순서와 같지만, 깊이 우선 방식은 XML 문서의 엘리먼트를 파싱하는 순서를 따르게 된다. 그런데, 제안하는 EXDMS에서는 DTD와 XML 문서 트리를 모두 너비 우선 방식으로 구성한다. 너비 우선 방식이 XML 문서 트리를 생성할 때 파서에 의해 파싱되는 엘리먼트 노드의 순서로 Group 성분 값을 부여하기 때문에 깊이 우선 방식과 유사한 형태로 가능하기 때문이다. 그리고 DTD와 XML 문서 트리의 노드 ID 부여 방식이 통일됨으로써 추가적인 처리 모듈을 줄일 수 있었다.

3.2 문서 데이터의 구성

EXDMS에서는 DTD 정보를 다루기 위해서 SAX 파서를 이용하여 DTD 문서에서 XML 문서의 구조 정보를 추출한 후, 이 정보를 트리 형태로 변환하고, 다시 이 트리를 이용하여 스키마 정보를 생성하여 DTD 데이터베이스에 구성한다. 그런데 XML 문서의 엘리먼트 정보를 생성할 때는 필요한 구조 정보를 추출하기 위해서 기존에 구성된 DTD 트리를 순회하면서 처리해야 한다. 이는 특정 노드에 대한 접근을 수시로 발생시키게 된다. 따라서 이를 효율적으로 처리하기 위해서 특정 노드의 접근을 효율적으로 처리하는 방법이 필요하다. 따라서 본 논문에서는 앞에서 설명한 바와 같이 데이터베이스에 저장되어 있는 각 노드마다 고유한 노드 ID를 부여하고, 이 노드 ID와 노드의 이름을 이용하여 인덱스를 생성하여 사용할 수 있도록 한다.

3.2.1 DTD 정보의 구성

DTD 정보의 구성은 먼저, SAX 파서를 이용하여 DTD 트리를 생성한다. 그리고 생성된 DTD 트리를 이용하여 DTD 정보를 구성한다. 이 때, 구성되는 정보는 엘리먼트 정보와 애트리뷰트 정보로 구분되어 DTD 데이터베이스에 저장된다.

다음 (그림 2)는 책(Book)에 대한 예제 DTD로서, 트리 형태로 표현하면 (그림 3)과 같다. EXDMS는 (그림 2)와 같은 DTD 문서에서 XML 구조 정보를 추출하여 (그림 4)와 같은 DTD 노드 정보, 즉 스키마 정보와 DTD 인덱스 정보를 DTD 데이터베이스에 생성한다. (그림 4)와 같은 테이블 구조 형태의 관리는 EXDMS의 하부 EDBM이 제

공하는 레코드 형태의 데이터 관리 기능을 이용하게 된다.

```
<!ELEMENT   book (booktitle, author*)>
<!ELEMENT   booktitle (#PCDATA)>
<!ELEMENT   author (name, gender, address)>
<!ATTLIST   author id ID #REQUIRED>
<!ATTLIST   author passwd CDATA #REQUIRED>
<!ELEMENT   name (#PCDATA)>
<!ELEMENT   gender EMPTY>
<!ATTLIST   gender person (m|f) "m">
<!ELEMENT   address (#PCDATA|city)>
<!ELEMENT   city (#PCDATA)>
```

그림 2. 책(Book)에 대한 예제 DTD
Fig 2. Example DTD instead of Book

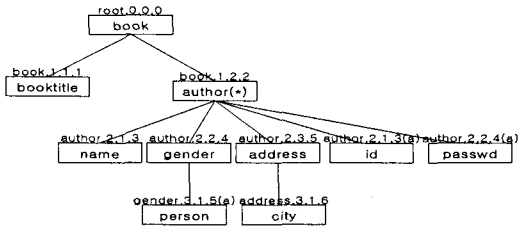


그림 3. 예제 DTD의 트리 구조 표현
Fig 3. Tree Structure Presentation of Example DTD

(그림 3)은 (그림 2)의 예제 DTD에 나타난 XML 구조 정보를 트리 형태로 나타낸 것으로, 각 노드는 엘리먼트와 애트리뷰트(노드의 노드 ID 값 뒤에 (a) 표시) 노드로 구분되며, 각각 노드 ID 값(Node_Value)을 가지게 된다. 이 때 노드 ID 값은 부모 노드가 그 자식 노드의 노드 ID 값을 유추할 수 있는 정보를 가지고 있어 특정 자식 노드의 탐색을 용이하게 한다.

N_Value	P_Point	E_Name	Content_Model	A_List
root.0.0.0		book	booktitle, author*	
book.1.1.1	root.0.0.0	booktitle	PCDATA	
book.1.2.2	root.0.0.0	author	name, gender, address	id, passwd
author.2.1.3	book.1.2.2	name	PCDATA	
author.2.2.4	book.1.2.2	gender	EMPTY	person
author.2.3.5	book.1.2.2	address	PCDATA|city	
address.3.1.6	author.2.3.5	city	PCDATA	

(a) 엘리먼트 노드의 정보

N_Value	P_Point	A_Name	Type	Mode	D_Value
author.2.1.3	book.1.2.2	id	ID	REQUIRED	
author.2.2.4	book.1.2.2	passwd	CDATA	REQUIRED	
gender.3.1.5	author.2.2.4	person	m|f		m

(b) 애트리뷰트 노드의 정보

그림 4. 데이터베이스에 저장되는 DTD 트리의 노드 정보
Fig 4. Node Information of DTD Tree in Stored Database

(그림 4)는 (그림 3)의 DTD 트리 노드 정보를 데이터베이스에 저장하는 모습을 나타낸 것으로 XML 문서 정보를 추출하여 데이터베이스를 구성할 때 스키마 정보로 사용된다. 여기서 DTD 트리의 노드 정보는 크게 엘리먼트 정보와 애트리뷰트 정보로 나누어지기 때문에 이를 구분하기 위해 데이터베이스를 엘리먼트와 애트리뷰트 두 가지 테이블 형태로 관리하도록 하였다. (그림 4a)에서 엘리먼트 노드의 정보는 노드 ID 값(N_Value)과 부모의 노드 ID 값(P_Point), 노드의 이름(E_Name), 내용 모델(Content_Model), 그리고 리스트 형식의 애트리뷰트의 이름(A_List)으로 구성된 레코드 집합으로 저장된다. 그리고 (그림 4b)의 애트리뷰트 노드의 정보는 노드의 노드 ID 값(N_Value)과 부모의 노드 ID 값(P_Point), 노드의 이름(A_Name), 애트리뷰트의 데이터 타입(Type), 애트리뷰트의 모드(Mode), 그리고 애트리뷰트의 기본값(D_Value)으로 구성된 레코드 집합으로 저장된다.

3.2.2 XML 문서 데이터의 구성

XML 문서 트리를 처리함에 있어서 EXDMS는 XML 문서의 내용을 엘리먼트 단위로 분리한 다음, 이를 인스턴스 데이터베이스에서 관리한다. 다음 (그림 5)는 (그림 2)의 DTD에 유효하게(Valid) 발생된 XML 문서 인스턴스이며, (그림 6)은 이 문서를 엘리먼트 단위로 분리한 후, 인스턴스 데이터베이스에 저장하는 모습을 보여준다.

```
<book>      <booktitle> XML </booktitle>
          <author id="253" passwd="XML12">
            <name> Je </name>
            <gender person="m">
            <address> Jin-ju </address>
          </author>          <author id="48" passwd="c48">
            <name> Choi </name>
            <gender person="m">
          <address>          <city>Ma-san</city> </address>
          </author>
</book>
```

그림 5. (그림 2)의 DTD를 만족하는 XML 문서의 예
Fig 5. Example of XML Document satisfactorying DTD of Fig 2

NO	E_Value	N_Value	Element_Name	PCDATA	Attribute
1	root.0.0.0	root.0.0.0	book		
2	book.1.1.1	book.1.1.1	booktitle	XML	
3	book.1.2.2	book.1.2.2	author		id 253 passwd XML12
4	author.2.1.3	author.2.1.3	name	Je	
5	author.2.2.4	author.2.2.4	gender		person m
6	author.2.3.5	author.2.3.5	address	Jin-ju	
7	book.1.3.6	book.1.2.2	author		id 48 passwd c48
8	author.2.1.7	author.2.1.3	name	Choi	
9	author.2.2.8	author.2.2.4	gender		person m
10	author.2.3.9	author.2.3.5	address		
11	address.3.1.10	address.3.1.6	city	Ma-san	

그림 6. 인스턴스 데이터베이스에 저장된 XML 문서 데이터
Fig 6. XML Document Data Stored in Instance Database

(그림 6)에서 보는 것과 같이 인스턴스 데이터베이스에서는 엘리먼트의 이름(Element_Name), PCDATA 값(PCDATA), 그리고 애트리뷰트 (Attribute) 정보를 저장한다. 이와 함께 해당 엘리먼트의 DTD 정보로 DTD 트리의 노드 ID 정보(N_Value) 등을 함께 저장한다. 그리고 엘리먼트의 노드 ID(E_Value)와 입력된 순서 값(NO)을 가지고 있어 엘리먼트를 무작위 혹은 순차적으로 처리할 수 있게 한다. 그런데 애트리뷰트는 하나의 항목이 아닌 여러 항목들을 동시에 나타내고 있는데, 이러한 정보들은 다른 항목과는 다르게 이름과 값 형태의 문자열로서 구성된다.

Ⅳ. 시스템의 설계

내장형 시스템에서 XML 문서를 효과적으로 관리하기 위해서는 기본적으로 XML DTD를 바탕으로 문서의 구조 정보를 추출하여 스키마 정보를 생성하고 이를 이용하여 XML 문서 인스턴스들을 관리하고 검색하는 기능이 제공되어야 한다. 이를 위해 EXDBS는 우선 문서 전체에 대한 구조 정보를 트리 형태로 유지하고 관리한다. 그리고 이 구조 정보는 데이터베이스 스키마 정보로 활용되어 XML 문서를 데이터베이스에 구성하여 관리할 때 사용하게 된다. 하지만 관계형 데이터 모델이나 객체지향 데이터 모델과 같은 전형적인 데이터 모델은 가지고 있지 않다. 따라서 EXDMS에서는 데이터의 구성을 관계형 모델의 테이블 형식을 빌어 구성하며, 내장형 데이터베이스 시스템이 인식할 수 있는 간단한 키와 값의 쌍 형식으로 레코드를 관리한다. 한편, XML 문서를 분할 저장 방식으로 저장 관리함으로써 엘리먼트 단위의 문서 관리와 공유를 용이하게 한다. 그리고 XML 문서를 탐색하면서 구조 정보와 내용 정보를 함께 데이터베이스에 저장함으로써 문서를 재구성하고 통합하기 위한 오버헤드를 크게 줄인다.

4.1 EXDMS의 구조

EXDMS는 SAX 파서를 이용하여 DTD와 XML 문서의 내용을 추출하고, 추출된 내용 정보를 데이터베이스에서 관리한다. 그리고 효율적인 데이터 처리를 위하여 EXDMS는 DTD와 XML 문서의 내용 정보만을 추출하고, 추출한 내용 정보의관리는 하부의 내장형 데이터베이스 시스템인 EDBM[19]을 이용한다. 다음 (그림 7)은 EXDMS의 구조를 보여준다.

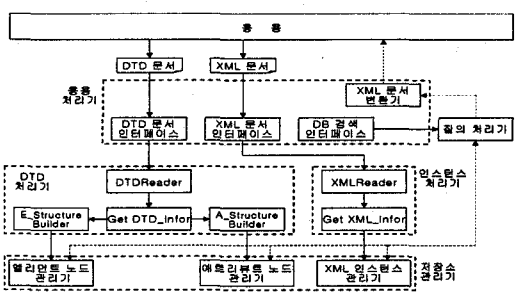


그림 7. EXDMS의 구조
Fig 7. Structure of EXDMS

4.2 EDBM

EDBM은 PDA와 같은 내장형 시스템의 제한적인 하드웨어 환경하에서 데이터를 데이터베이스 형태로 관리해 주는 내장형 데이터베이스 시스템으로, 어떠한 데이터 모델도 가지지 않으면서도 응용 시스템에 대한 확장성과 강력한 DBMS의 기능을 제공한다. EDBM은 효율적인 데이터 처리를 위하여 자신은 데이터에 관한 처리 기능만을 담당하고, 내부에 버클리 DB[1]라는 다른 내장형 데이터베이스 시스템 엔진을 이용하여 데이터를 DB 형태로 저장/관리한다. 이 버클리 DB는 대부분의 운영체제에 이식이 가능하고, 작고 빠르며, 쉬운 사용성을 가지고 있으며, 라이브러리(Library) 형태로 제공되어 응용 프로그램과 직접 연결하여 사용할 수 있는 장점을 가지고 있다.

4.3 응용 처리기

응용 처리기는 상위의 응용에서 요구하는 명령과 데이터를 하위 모듈에 전달하거나, 데이터베이스에 저장된 데이터를 응용에 되돌린다. 따라서 응용 처리기는 응용을 지원하기 위한 명령어 인터페이스만을 가진다. 인터페이스의 구성은 DTD 문서 인터페이스, XML 문서 인터페이스, DB 검색 인터페이스, XML 문서 변환기의 네부분으로 되어 있다. DTD 문서 인터페이스와 XML 문서 인터페이스는 각각 DTD 문서와 XML 문서를 DTD 처리기와 XML 처리기에 전달하여 문서 정보를 추출하게 한다. 그리고 DB 검색 인터페이스는 응용의 데이터 검색 요구시 저장소 관리기에 직접 접근하여 XML 구조 정보와 인스턴스 정보를 검색하고, 그 결과를 XML 문서 변환기에 전달한다. XML 문서 변환기는 DB 검색 인터페이스에서 전달되는 XML 문서 인스턴스 정보를 XML 문서로 변환하여 응용에 전달한다.

4.4 DTD 처리기

DTD 처리기(DTD Processor)는 SAX 파서를 이용하여 DTD 문서를 분석하여 XML의 구조 정보를 추출하고, 추출한 구조 정보들을 엘리먼트 및 애트리뷰트의 정보로 구분하여 구성한 후, 그 내용을 DTD 데이터베이스에 저장한다. DTD 처리기의 구성은 DTDReader, Get DTD_Infor, E_Structure Builder, A_Structure Builder의 네 부분으로 구성되어 있다. DTDReader는 SAX 파서를 이용하여 DTD 문서에서 XML의 기본적인 구조 정보를 얻는다. Get DTD_Infor는 DTDReader에서 얻은 XML 구조 정보를 엘리먼트 및 애트리뷰트 정보 단위로 추출하고, 추출된 XML 문서의 구조 정보를 엘리먼트와 애트리뷰트 정보로 구분하여 DTD 데이터베이스에 저장한다. 이 때, 추출된 XML 문서의 구조 정보는 트리 형태로 구성되는데 이 트리의 각 노드들은 엘리먼트와 애트리뷰터로 구분되며, E_Structure Builder와 A_Structure Builder는 이러한 엘리먼트와 애트리뷰트의 정보를 구분하여 저장소 관리기를 통해 DTD 데이터베이스에 저장한다.

4.5 인스턴스 처리기

인스턴스 처리기(Instance Processor)는 SAX 파서를 이용하여 XML 문서의 유효성(Validation)을 검사하고, DTD 처리기에서 기 처리한 DTD 정보를 이용하여 XML 문서의 엘리먼트 정보를 추출한 후, 인스턴스 데이터베이스에 저장할 수 있는 데이터 형태로 재구성한다. 인스턴스 처리기는 XMLReader, Get XML_Infor의 두 부분으로 구성되어 있으며, XMLReader는 SAX 파서를 이용하여 XML 문서의 유효성을 검사하고, 엘리먼트의 기본적인 데이터 정보를 얻는다. Get XML_Infor 모듈은 XMLReader 모듈에서 추출된 정보와 DTD 처리기에서 기 처리한 DTD 정보를 이용하여 노드의 위치값과 순서 정보 등 추가적인 정보를 구성하여 인스턴스 데이터베이스에 저장할 수 있는 형태의 레코드로 변환한다.

4.6 저장소 관리기

저장소 관리기(Storage Manager)는 EDBM의 API 모듈을 이용하여 응용 처리기나 DTD 처리기, 인스턴스 처리기가 요청하는 데이터를 저장(Store)하거나 적재(Load)하는 작업을 수행한다. 이 때, DTD 처리기와 인스턴스 처리기에서 처리하는 데이터를 DTD 데이터베이스와 인스턴스 데이터베이스로 구분하여 관리하며, 이들 데이터베이스 외에도 데이터베이스를 관리하기 위한 추가적인 정보들을 관리한다.

저장소 관리기는 엘리먼트 노드 관리기, 애트리뷰트 노드 관리기, XML 인스턴스 관리기의 세부분으로 구성되어 있으며, 각각 DTD 문서의 엘리먼트 정보와 애트리뷰트 정보, 그리고 XML 문서 정보를 저장/관리한다.

4.7 질의 처리기

일반적인 데이터베이스 응용에서 데이터베이스에 저장되어 있는 데이터에 대한 질의는 필수적인 요소이다. 그런데 이러한 응용들이 XML 문서를 이용한다면, 질의를 위해서 XQuery와 같은 질의어를 사용해야 한다. 하지만 내장형 시스템의 제한된 성능 때문에 이러한 질의어를 실행하기 힘들

다. 따라서 EXDMS에서는 SQL이나 XQuery와 같은 XML 질의어 없이 특정 엘리먼트나 애트리뷰트를 쉽게 검색할 수 있도록 간단한 메커니즘을 제공한다. 다음 (그림 8)은 (그림 5)의 예제 XML 문서에서 PCDATA 값이 "Choi"인 엘리먼트와 그 DTD 정보를 검색하는 메커니즘을 SQL로 보여준다.

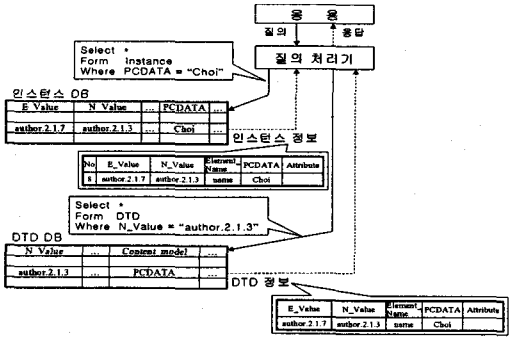


그림 8. 질의 처리 메커니즘
Fig 8. Query Processing Mechanism

(그림 8)에서 보는 것과 같이 질의 처리기는 먼저 인스턴스 DB에 접근하여 PCDATA 컬럼의 값이 "Choi"인 레코드를 검색한다. 그리고 검색된 레코드의 N_Value 컬럼 값을 이용하여 다시 DTD DB에서 해당 엘리먼트의 구조 정보를 검색한 후, 기 검색된 엘리먼트 정보와 구조 정보를 객체로 변환하여 응용에 응답한다.

V. 시스템의 구현

EXDMS의 구현은 레드햇 리눅스 기반의 와우 리눅스 7.3 운영체제와 g++ 2.95 버전의 C++ 언어를 이용하였다. 이러한 결정은 타 버전의 운영체제와 프로그래밍 언어에 비해 안정성과 효율성이 뛰어나기 때문이었다. 그리고 SAX 파서로는 W3C[20]에서 권고한 XML 표준을 따르면서도 문서의 파싱 속도가 빠른 아파치 Xerces-c 2.0 버전을 사용하였으며, 내장형 시스템의 낮은 성능 효율과 제한된 시스템 환경을 고려하여 펜티움II-400Mz 프로세스에 128Mbyte의 RAM을 가진 컴퓨터 시스템에서 구현하였다.

5.1 EXDMS의 모듈 구성

EXDMS은 크게 응용처리기, DTD 처리기, 인스턴스 처리기, 저장소 관리기의 4개 모듈로 구현되었으며, DTD 처리기와 인스턴스 처리기 모듈은 다시 여러개의 서브 모듈들로 구현되었다. 다음의 (그림 9)는 구현된 EXDMS의 전체 구성을 보여준다.

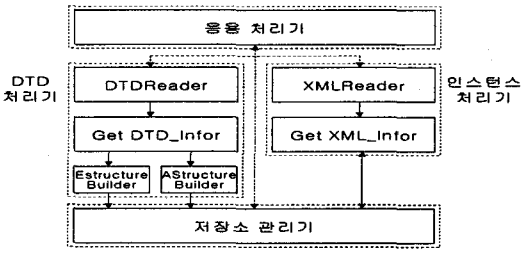


그림 9. EXDMS의 구성
Fig 9. Composition of EXDMS

5.2 응용 처리기

응용 처리기는 응용에서 요구하는 명령들과 데이터를 하부의 DTD 처리기와 인스턴스 처리기, 그리고 저장소 관리기에 전달하고, DB에 구성되어 있는 XML 문서 인스턴스 정보를 XML 문서로 재구성한다. 따라서 응용 처리기는 데이터 처리가 아닌 데이터 처리 명령에 대한 인터페이스만을 가지고 있다. 다음 <표 1>은 응용 처리기의 구현객체와 함수들이다.

표 1. 응용 처리기의 구현 객체와 함수
Table 1. Implementation Object & Function of Application Processing Tool

구현 객체	구현 함수	기 능
exdms	dtdSave	DTD 문서 정보를 DTD 데이터베이스에 저장
	instanceSave	XML 문서를 인스턴스 데이터베이스에 저장
	searchAllNode	모든 엘리먼트 노드 객체를 검색
	searchNodeData_Name	노드 이름으로 엘리먼트 노드 객체 검색
	searchNodeData_NodeValue	노드 ID로 엘리먼트 노드 객체 검색
	searchAllAttribute	모든 애트리뷰트 노드 객체를 검색
	searchAttributeData_Name	노드 이름으로 애트리뷰트 노드 객체 검색
	searchAttributeData_NodeValue	노드 ID로 애트리뷰트 노드 객체 검색
	searchAllInstance	모든 엘리먼트 객체 검색
	searchInstance_Name	엘리먼트 이름으로 엘리먼트 객체 검색
	searchInstance_NodeValue	노드 ID로 엘리먼트 객체 검색
	searchInstance_Pcdata	PCDATA 값으로 엘리먼트 객체 검색
	xmlViewer	엘리먼트 객체를 XML 문서로 표현
	delData	엘리먼트 객체 삭제

5.3 DTD 처리기

DTD 처리기는 SAX 파서를 이용하여 DTD 문서를 파싱하는 DTDReader 모듈, DTD 구조 정보를 구성하는 Get DTD_infor 모듈, 그리고 엘리먼트 노드와 애트리뷰트 노드 정보를 EDBM에 저장할 수 있는 레코드 형태로 변환하는 EStructureBuilder와 AStructureBuilder 모듈로 구성되어 있다. 그런데 EXDMS는 XML 구조 정보를 엘리먼트 정보와 애트리뷰트 정보로 구분하여 관리한다. 따라서 DTD 처리기는 XML 구조 정보를 처리하기 위하여 엘리먼트와 애트리뷰트를 구분된 알고리즘으로 처리하며, 이 두 알고리즘은 상호 보완적으로 동작한다.

5.3.1 엘리먼트 정보 처리 모듈

DTD 처리기의 엘리먼트 정보 처리 모듈은 DTDReader 모듈의 SAX 파서가 엘리먼트에 대한 이벤트(Event)를 발생시킬 때, 보고되는 XML 구조 정보 중에서 엘리먼트 정보들을 처리한다. 이 때, 엘리먼트 정보들은 객체로 변환되어 처리되며, 데이터베이스에 저장될 때에도 객체로 저장된다.

5.3.1.1 엘리먼트 노드 객체

DTD 처리기에서 엘리먼트 정보를 구성하여 DTD 데이터베이스에 저장하기 위해서는 엘리먼트 노드 객체라는 데이터 구조를 생성하여야 한다. 엘리먼트 노드 객체는 DTD 데이터베이스에 저장하는 엘리먼트 노드의 구조 정보를 가지고 있으며, DTDReader 모듈이나 Get DTD_Infor 모듈에서 생성된다. 그리고 EStructrureBuilder 모듈에 의해 레코드 형태로 재표현되어 DTD 데이터베이스에 저장된다. 다음의 (그림 10)은 엘리먼트 노드 객체의 데이터 구조를 보인다.

```
1 Class Element {                    // 엘리먼트 노드 객체
2 element_node_value      // 엘리먼트 노드의 노드 ID
3 parent_node_value       // 부모 노드의 노드 ID
4 node_name                  // 노드(엘리먼트)의 이름 정보
5 content_model           // 엘리먼트의 내용모델 정보
6 attribute_list          // 애트리뷰트 리스트 정보
7 } e_node ;
```

그림 10. 엘리먼트 노드 객체
Fig 10. Object of Element Node

5.3.1.2 엘리먼트 정보 처리

엘리먼트 정보 처리는 DTDReader 모듈에서 생성된 엘리먼트 노드 객체에 엘리먼트의 노드 ID와 부모 노드의 노드 ID를 구성한다. 그리고 엘리먼트의 내용 모델을 분석하여 자식 엘리먼트의 노드 객체를 생성하고, 엘리먼트의 노드 ID 값과 부모 노드의 노드 ID 값 그리고 노드(엘리

먼트) 이름 정보를 구성하여 DTD 데이터베이스에 저장한다. 다음 [알고리즘 1.]은 Get DTD_Infor 모듈의 엘리먼트 정보 처리 절차를 보여준다.

알고리즘 1. DTD 상의 엘리먼트 정보 처리
Algorithm 1. Elemnet Information Processing based in DTD

```
1  Algorithm buildElement (Element e_node) 엘리먼트 정보 트리를 구성
2  int group = 0          // 위치 정보 중 자신의 그룹 정보
3  // 루트 노드일 경우 노드 ID와 부모 노드의 노드 ID값을 초기화하고
4  // 루트 노드가 아닐 경우 기 저장되어 있는 노드의 노드 ID값을 가져옴
5      if (e_node is Root) then
6          e_node.element_node_value = "root.0.0.0"
7          e_node.parent_node_value = NULL
8      else
9      e_node.element_node_value = getNodeValue(e_node.node_name)
10 end if
11// 자식 노드를 가지는 엘리먼트일 경우 자식 노드의 노드 객체를 생성하고
12// 이 객체의 노드 ID값과 부모 노드의 노드 ID값, 그리고 노드 이름을
13     // 부여한 후, DTD 데이터베이스에 저장
16     loop (e_node have the Child Node)
17         makeChildRecord(e_node)
18     end loop
19 // 루트 노드일 경우 구성된 노드 객체를 레코드 형태로 변환하여 DTD
20 // 데이터베이스에 저장하고, 그렇지 않은 일반 노드일 경우에는 기존
21     // 저장된 노드 객체 값에 내용 모델 값을 추가
22     if (e_node is Root) then
23         node_record = EStructureBuilder(e_node)
24         saveNode(node_record)
25     lese
26         updateNode(e_node.content_model)
27     end if
29 return
```

[알고리즘 1.]에서 라인2의 group 변수는 노드의 위치 정보 중에서 Group 성분을 나타내며 새로운 노드가 추가될 때마다 1씩 증가한다. 라인5에서 라인10까지는 엘리먼트의 노드 ID 값과 부모의 노드 ID 값을 설정하는 부분으로 루트일 경우(라인6, 7), 노드 ID값을 "root.0.0.0"으로 설정하여 루트 노드임을 나타내고, 부모의 노드 ID 값을 NULL로 초기화한다. 만약 루트 노드가 아닐 경우(라인9) 기 저장되어 있는 엘리먼트 노드 객체에서 현재 노드의 노드 ID 값을 얻는다. 라인16에서 라인18은 내용 모델을 검사하여 자식 엘리먼트 노드가 존재할 경우 새로운 노드 객체를 생성하고 DTD 데이터베이스에 저장한다. 라인22에서 라인27까지는 노드 객체를 DTD 데이터베이스에 저장하는 부분으로, 루트일 경우(라인23, 24)에는 노드 객체를 DTD 데이터베이스에 저장 가능한 레코드 형태로 변환한 후 저장하고, 루트 노드가 아닐 경우(라인26)에는 기 저장되어 있는 노드 객체 값에 내용 모델 값을 추가한다.

5.3.2 애트리뷰트 정보 처리 모듈

DTD 처리기의 애트리뷰트 정보 처리는 엘리먼트 정보 처리와 같이 DTDReader 모듈의 SAX 파서가 애트리뷰트 정보에 대한 이벤트(Event)를 발생시키고 XML 구조 정보를 보고할 때, 보고되는 애트리뷰트 정보들을 처리한다.

5.3.2.1 애트리뷰트 노드 객체

애트리뷰트 노드 객체는 엘리먼트 노드 객체와 같이 애트리뷰트 노드의 데이터 구조를 가지고 있으며, DTDReader 모듈에서 생성되어 DTD 처리기의 애트리뷰트 정보 처리 과정에서 애트리뷰트의 모든 정보를 구성하게 된다. 그리고 AStructureBuilder 모듈에 의해 레코드 형태로 재표현되어 DTD 데이터베이스에 저장된다. 다음의 (그림 11)은 애트리뷰트 노드 객체의 데이터 구조를 보인다.

```
1 Class Attribute {          // 애트리뷰트 노드 객체
2 attribute_node_value     // 애트리뷰트 노드의 노드 ID
3 parent_node_value        // 부모 노드의 노드 ID
4 parent_node_name           // 부모 노드의 이름
5 attribute_name           // 애트리뷰트의 이름 정보
6 type                     // 애트리뷰트의 타입 정보
7 mode                     // 애트리뷰트의 모드 정보
8 value                    // 애트리뷰트의 기본 값
9 } a_node ;
```

그림 11. 애트리뷰트 노드 객체
Fig 11. Object of Attribute Node

5.3.2.2 애트리뷰트 정보 처리

애트리뷰트 정보 처리는 DTDReader 모듈에서 생성된 애트리뷰트 노드 객체에 애트리뷰트의 노드 ID 값과 보모의 노드 ID 값을 구성하고, 애트리뷰트 노드 객체를 DTD 데이터베이스에 저장한다. 그리고 부모에 자신의 애트리뷰트 정보를 추가한다. [알고리즘 2.]는 Get DTD_Infor 모듈의 애트리뷰트 정보 처리 절차를 보여준다.

알고리즘 2. DTD 상의 애트리뷰트 정보 처리
Algorithm 2. Attribute Information Processing based in DTD

```
1 Algorithm buildAttribute (Attribute a_node)
2 // 부모 노드의 노드 객체를 불러옴
3 parent_node = getNode(e_node.parent_node_name)
4 // 부모 노드의 노드 객체를 이용하여 애트리뷰트 노드의 노드 ID값 설정
5 a_node.attribute_node_value = setAttributeNodeValue(parent_node)
6 // 애트리뷰트 노드 객체에 부모 노드의 노드 ID값을 부여
7 a_node.parent_node_value = parent_node.element_node_value
8 //부모 노드의 attribute_list 값에 현재 애트리뷰트 노드 객체의 정보를 추가
9 appendNodeAList(parent_node.attribute_list)
10 // 애트리뷰트 노드 객체를 DTD 데이터베이스에 저장
11 a_node_record = EStructureBuilder(a_node)
12 saveAttribute(a_node_record)
13 return
```

[알고리즘 2.]에서 라인3의 getNode() 함수는 부모의 엘리먼트 노드 객체를 불러와 애트리뷰트 노드 객체의 부모 노드

ID값을 설정한다. 또한 라인5의 setAttributeNodeValue() 함수를 이용하여 자신의 노드 ID값을 구하고, 자신의 애트리뷰트 정보를 부모의 엘리먼트 노드 객체에 추가한다(라인9). 마지막으로, 구성된 애트리뷰트 노드 객체를 EDBM의 레코드 형태로 재구성하여 DTD 데이터베이스에 저장한다(라인11, 12).

5.4 인스턴스 처리기

인스턴스 처리기는 XML 문서를 파싱하는 XMLReader 모듈, XML 문서의 내용 정보를 구성하는 Get XML_Infor 모듈로 구성되어 있다. XMLReader 모듈은 내부에 SAX 파서를 포함하고 있으며, SAX 파서는 XML 문서에서 엘리먼트가 발견되면 이벤트를 발생시키고, 엘리먼트의 내용 정보를 보고한다. Get XML_Infor 모듈은 XMLReader에서 이벤트가 발생되면 보고되는 정보와 기 처리되어 있는 DTD 문서 정보를 이용하여 엘리먼트 정보들을 처리한다.

5.4.1 XML 문서의 엘리먼트 객체

엘리먼트 객체는 DTD 정보 데이터 구조인 엘리먼트 노드 객체, 애트리뷰트 노드 객체와 같이 XML 문서상의 엘리먼트 데이터 구조를 가지고 있다. 엘리먼트 객체는 XMLReader 모듈에서 생성되어 인스턴스 처리기의 인스턴스 정보 처리 과정에서 엘리먼트의 모든 정보를 구성하며, 레코드 형태로 재표현하여 인스턴스 데이터베이스에 저장된다. (그림 12)는 엘리먼트 객체의 데이터 구조를 보여주고 있다.

```
1 Class Element { // XML 문서 인스턴스를 위한 엘리먼트 객체
2 element_value    // 엘리먼트의 노드 ID
3 dtd_node_value   // DTD 정보인 엘리먼트 노드의 노드 ID
4 element_name     // 엘리먼트의 이름 정보
5 pcdata_information // 엘리먼트의 PCDATA 정보
6 attribute_inofrmation // 엘리먼트의 애트리뷰트 정보
7 } element ;
```

그림 12. 엘리먼트 객체
Fig 12. Elemnet Object

5.4.2 XML 문서의 엘리먼트 정보 처리

XML 문서의 엘리먼트 정보 처리는 XMLReader 모듈에서 생성된 엘리먼트 객체에 엘리먼트의 노드 ID값과 DTD 정보를 가진 엘리먼트의 노드 ID 값을 구성하고, 엘리먼트 객체를 EDBM의 레코드 형태로 변환한 다음, DTD 데이터베이스에 저장한다. [알고리즘 3.]은 인스턴스 처리기 모듈의 XML 문서상의 엘리먼트 정보의 처리 절차를 보여준다.

알고리즘 3. XML 문서상의 엘리먼트 정보 처리
Algorithm 3. Element Information Processing based in XML Document

```
1 Algorithm buildInstance (Element element) // XML 문서 인스턴스 구성
2 // 기 처리된 DTD 정보에서 현재 노드의 DTD 노드 정보(노드 ID)를 가져옴
3  element.dtd_node_value = getNodeValue(element.element_name)
4  // 엘리먼트의 노드 ID값을 생성
5  if (element is not Root Element) then // 루트 엘리먼트가 아닐 경우
6   element.element_value = getElementValue(element.dtd_node_value)
7  else                                  // 루트 엘리먼트일 경우
8   element.element_value = "root.0.0.0"
9   end if
10  // 인스턴스 데이터베이스에 엘리먼트 객체를 저장
11  saveInstance(element)
12  return
```

위의 [알고리즘 3.]에서 라인3는 getNodeValue() 함수를 이용하여 DTD 정보인 엘리먼트의 노드 ID값을 구성한다. 그리고 라인5에서 라인9는 엘리먼트의 노드 ID값을 생성하며, 라인11은 구성된 엘리먼트 객체를 인스턴스 데이터베이스에 저장한다.

5.5 저장소 관리기

저장소 관리기(StorageManager)는 내부에 EDBM의 API를 가지고 응용 처리기와 DTD 처리기, 인스턴스 처리기에서 요구하는 데이터 삽입/삭제/검색 기능을 수행한다. 저장소 관리기는 내부에 DTD 데이터베이스의 엘리먼트 노드 객체를 관리하는 NodeManager 객체와 애트리뷰트 노드 객체를 관리하는 AttributeManager 객체, 그리고 인스턴스 데이터베이스를 관리하는 InstanceManager 객체를 가지고 있으며, 이 객체들의 구현 함수는 다음 〈표 2〉와 같다.

표 2. 저장소 관리기의 구현 객체와 함수
Table 2. Implementation Object & Function of Storing Place Management Tool

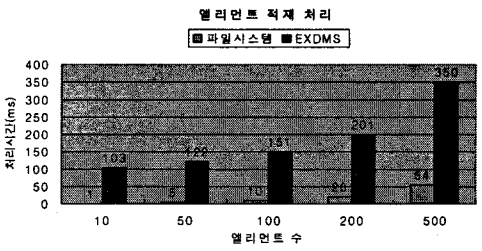
구 현 객 체	구현 함수	기 능
Node Manager	setUpNodeDB	DTD 데이터베이스의 엘리먼트 DB를 초기화
	saveNode	엘리먼트 노드 객체 저장
	getNodeValue	객체의 노드 ID을 호출
	getAllNode	모든 엘리먼트 노드 객체를 호출
	getNodeData	특정 정보로 엘리먼트 노드 객체 호출
	updateNodeContentModel	엘리먼트 노드 객체의 내용모델 정보를 수정
	appendNodeAList	엘리먼트 노드 객체 애트리뷰트 정보 추가
Attribute Manager	setUpAttributeDB	DTD 데이터베이스의 애트리뷰트 DB를 초기화
	saveAttribute	애트리뷰트 노드 객체를 저장
	getAllAttribute	모든 애트리뷰트 노드 객체를 호출
	getAttributeData	특정 정보로 애트리뷰트노드 객체 호출
Instance Manager	setUpInstanceDB	인스턴스 데이터베이스를 초기화
	getInstanceInfor	인스턴스 데이터베이스의 DB 정보 호출
	setInstanceInfor	인스턴스 데이터베이스의 DB 정보 저장
	saveInstance	엘리먼트 객체를 저장
	getAllInstance	모든 엘리먼트 객체를 호출
	getInstanceData	특정 정보로 엘리먼트 객체 호출
	getCount	검색된 엘리먼트 객체의 개수를 구함
	delElement	엘리먼트 객체 삭제

VI. 비교 분석

현재 EXDMS와 유사한 기능을 갖는 내장형 XML 데이터베이스 관리 시스템은 없기 때문에 비교 대상을 찾기가 쉽지 않다. 따라서 이 장에서는 제안한 EXDMS를 일반 파일 시스템과 비교하였다. 우리는 비교대상 시스템으로 전형적인 PDA를 위한 PIMS(Personal Information Management System) 개발 환경인 TROLLTECH사의 QTOPIA 환경[18]에서 개발된 PIMS 응용의 파일 시스템을 선택하였다. 이 파일 시스템은 데이터 처리 형태는 다르지만 XML 문서 형태로 데이터를 표현하여 저장하고 있다. 따라서 비교를 위해 두 시스템의 데이터 처리 모듈을 각각 구현하고, 기본적인 데이터 처리에 대한 처리 효율만을 측정하였다.

EXDMS와 QTOPIA 환경의 파일 시스템은 데이터 처리 방법에 있어 많은 차이가 있다. 우선 EXDMS는 SAX 파서를 이용하여 XML 문서의 엘리먼트 단위로 데이터를 추출하고, 이 정보를 데이터베이스에 저장한다. 때문에 응용 계층에서는 데이터베이스에 저장된 엘리먼트를 단위로 데이터를 처리하게 된다. 이에 반하여 QTOPIA 환경의 파일 시스템은 파일 시스템을 이용하여 데이터를 XML 문서로 저장한다. 때문에 응용 계층에서의 데이터 처리를 위해서는 XML 문서의 모든 데이터를 메모리에 구성해야 한다.

(그림 13)은 EXDMS와 QTOPIA 환경의 파일 시스템에서 기본 데이터 처리 결과를 보여준다.



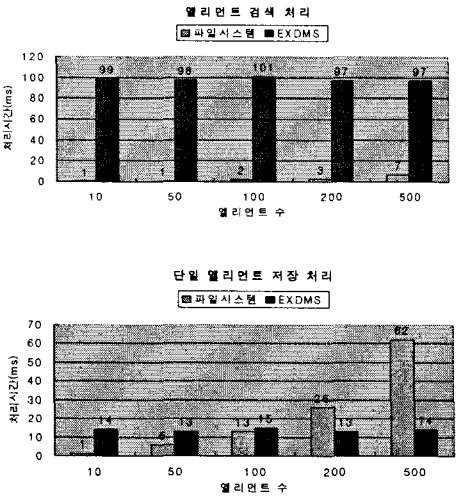


그림 13. EXDMS와 파일 시스템의 데이터 처리 결과
Fig 13. Data Processing Result of EXDMS & File System

(그림 13)의 엘리먼트 적재 처리의 경우, 파일 시스템은 자체 XML 파서를 이용하여 XML 문서를 파싱하고 엘리먼트를 추출한 후 메모리에 구성하지만, EXDMS는 데이터베이스에서 엘리먼트 하나하나를 메모리에 적재함으로써 데이터베이스에 접근하는 오버헤드를 가지게 된다. 따라서 EXDMS는 데이터베이스에 접근하는 시간 때문에 파일 시스템보다 처리 효율이 떨어진다. 엘리먼트 검색처리의 경우는, 파일 시스템은 모든 엘리먼트 정보를 메모리에 구성하여 데이터를 검색한다. 이에 반하여 EXDMS는 검색 데이터를 데이터베이스에 접근하여 탐색한다. 따라서 파일 시스템보다 EXDMS가 좀더 많은 처리 시간을 소요한다. 하지만 파일 시스템의 경우 메모리상의 데이터가 증가할수록 검색 시간이 증가한다. 이에 반하여 EXDMS는 검색 대상 데이터에 상관없이 거의 동일한 처리 시간을 가지고 있다. 마지막으로 단일 엘리먼트 저장 처리의 경우, 파일 시스템은 기 저장되어 있는 데이터가 많을수록 처리 시간이 증가한다. 이에 반하여 EXDMS는 기 저장되어 있는 데이터에 상관없이 일관된 처리 시간을 보여준다. 이는 파일 시스템의 경우 하나의 엘리먼트를 처리하기 위해 저장되어 있는 모든 엘리먼트를 메모리에 구성하고, 엘리먼트 처리가 끝난 후, 다시 모든 엘리먼트를 동시에 저장한다. 이에 반해 EXDMS는 해당 엘리먼트 하나만 메모리에 적재하여 처리한 후, 다시 이 엘리먼트를 데이터베이스에 저장하기 때문에 처리 시간이 일정하다.

위의 비교들을 종합해 보면, 〈표 3〉과 같다.

표 3. EXDMS와 파일 시스템의 비교
Table 3. Comparison of EXDMS & File System

비 고	EXDMS	파일 시스템
XML 문서 처리	비효율적	효율적
단일 데이터 처리	효율적	비효율적
데이터 검색	비효율적	효율적
시스템 리소스 (메모리) 사용량	적음	많음
시스템 확장성	높음	낮음
시스템 호환성	높음	낮음

〈표 3〉에서 보는 바와 같이 대체적으로 파일 시스템이 EXDMS보다 XML 문서에 대한 처리 효율이 뛰어나다. 하지만 이는 파일 시스템이 데이터 처리 효율을 위해 자신만의 데이터 형식을 따르기 때문이며, 여러 시스템에서 사용하는 XML 문서가 아닌 자신의 시스템에서만 사용할 수 있도록 정의한 XML 형식을 따르기 때문에 타 시스템에 대한 확장성과 호환성이 떨어진다. 그리고 모든 데이터를 메모리에 구성해야 하기 때문에 큰 문서를 처리하기 힘든 단점을 가진다. 이에 반해 EXDMS는 비록 파일 시스템보다 XML 문서에 대한 처리 효율은 조금 떨어지지만, 일반적인 모든 XML 문서를 처리할 수 있고, 한 작업을 위해 하나의 데이터만을 메모리에 구성하기 때문에 파일 시스템보다 적은 메모리 용량을 요구한다. 그리고 단일 데이터 처리에 대한 데이터 처리 효율이 파일 시스템보다 뛰어난 결과를 보이는데, 이는 일반 응용에서 데이터를 처리하기 위해서 단일의 데이터만을 요구하는 것을 감안할 때, 데이터 처리 효율이 파일 시스템보다 좀더 뛰어난 결과를 보일 수 있다.

참고문헌

[1] M. Olson, K. Bostic, and M. Seltzer, "Berkeley DB," Proceedings of the 1999 Summer Usenix Technical Conference, Monterey, California, pp. 1-10, June 1999.

[2] F. Lam, N. Lam, and R. Wong, "Efficient Synchronization for Mobile XML Data," Proceedings

of the eleventh international conference on Information and knowledge management, pp. 153-160, Nov. 2002.
[3] 제권엽, 최무희, 안병태, 심명선, 강현석, "내장형 데이터베이스 시스템을 이용한 PDA용 PIMS 응용의 개발", 한국정보과학회 춘계 학술발표회 논문집 A권, pp. 791-793, 2003. 4.
[4] 이종설, 강형일, 손충범, 강승헌, 정연수, 민영수, 박종관, 유재수, "XML 저장관리 시스템의 설계 및 구현," Journal of the Research Institute for Computer and Information Communication, Vol. 7, No. 2 , pp. 85-97, 1999. 11.
[5] I. Varlamis, M. Vazirgiannis, "A System for Generating and Manipulating Relational Databases using Valid XML Documents," Proceedings of the 2001 ACM Symposium on Document engineering, pp. 105-114, Nov. 2001.
[6] 이재호, 신판섭, 김정은, "순환적 DTD 구조 처리를 위한 XML 문서 관리 시스템의 설계," 인천교육대학교 과학교육논총 제 13집, Vol. 13, pp. 291-301, 2001. 2.
[7] A. Salminen, F. Tompa, "Requirement for XML Document Database Systems," Proceedings of the 2001 ACM Symposium on Document engineering, pp. 85-94, Nov. 2001.
[8] D. Suciu, "On Database Theory and XML," ACM SIGMOD Record, Vol. 30, No. 3, pp. 39-45, Sept. 2001.
[9] L. Khan and Y. Rao, "A Performance Evaluation of Storing XML Data in Relational Database Management Systems," Proceeding of the third international workshop on Web information and data management, pp. 31-38, Nov. 2001.
[10] 민준기, 박명제, 안재용, 정진완, "다양한 저장소에서의 효율적인 XML 저장기법에 대한 연구," 데이터베이스 연구, 제 19권, 제1호, pp. 1-12, 2003. 3.
[11] Chrystal Software, "Astoria," "http://www.astoriasoftware.com/products/index.jsp".
[12] H. Schoning, "Tamino - A DBMS Designed for XML," IEEE ICDE, pp. 149-154, April 2001.
[13] H. Jagadish et al, "TIMBER: A Native XML Database," The VLDB Journal, Vol. 11, Issue 4, pp. 274-291, Dec. 2002.
[14] Oracle, "Oracle8i," "http://otn.oracle.co.kr/products/oracle9i/pdf/9i_new_features.pdf".
[15] M. Golfarelli, S. Rizzi and B. Vrdoljak, "Data Warehouse Design from XML Sources," Proceedings of the fourth ACM international workshop on Data warehousing and OLAP, pp. 40-47, Nov. 2001.
[16] ObjectDesign, "eXcelon," "www.datec.co.kr/products/products.asp".
[17] S. Lahiri and J. Widom, "Ozone: Integrating Structured and Semistructured Data," DBPL Conf., pp. 1-20, Sept. 1999.
[18] Trolltech, "QTOPIA," "http://www.trolltech.com".
[19] 제권엽, 최무희, 안병태, 강현석, "개인 정보 단말기의 응용을 위한 내장형 데이터베이스 관리 시스템의 설계 및 구현," 멀티미디어학회 춘계 학술발표 논문집, pp. 134-137, 2003. 5.
[20] W3C, "XML," "www.w3c.org".
[21] 이상태, 임종선, 주경수, "관계형 DBMS를 이용한 XML 스키마 기반의 XML DBMS 설계," 한국컴퓨터정보학회 논문지, 제9권, 제4호, 2004. 12.
[22] 박준범, 박창우, 오수열, "ODMG 객체 모델 기반의 XML 문서 저장 관리 시스템에 관한 연구," 한국컴퓨터정보학회 논문지, 제8권, 제2호, 2003. 6

저 자 소 개

안 병 태

2005년 10월 현재 밀양대학교 정보통신공학부 외래교수
2001~현재 Best Click Computer 대표
〈관심분야〉 Web & DB 연동, XML, Embedded DB

서 익 진

1997년 8월 삐에르망데스 그로노블대학교 경제학박사
1999년~현재 경남대학교 국제무역학부 조교수
〈관심분야〉 디지탈 경제학, 발전경제학