PDM/ODB: XML 데이타의 ODMG 표준 객체지향 데이타베이스로의 사상

(PDM/ODB: Mapping XML to ODMG-Compliant Object-Oriented Database)

고 봉 수[†] 박 상 원[††] 민 경 섭[†††] 김 형 주[††††]

(Bong-Su Ko) (Sangwon Park) (Kyung-Sub Min) (Hyoung-Joo Kim)

요 약 전자 문서로서의 XML의 가치는 날로 증대하고 있다. XML 문서는 반구조적인 특징을 가지고 있으며, 객체지향 데이타베이스는 이를 모델링하기에 적합하다. 기존 데이타베이스 시스템에 저장하기 위하여 XML 문서의 DTD 정보를 이용하여 스키마를 생성하는 방법은 많이 연구되어 왔다. 본 논문에서는 XML 문서의 의미적인 정보를 보존하는 방법으로 XML 문서를 객체지향 데이타베이스에 저장하였다. 이를 통하여 OQL, C++ 바인딩과 같은 객체지향 데이타베이스의 기능을 활용하여 XML 문서에 대한 검색 등이 가능하게 되었으며, 기존 객체지향 데이타베이스 응용 프로그램을 그대로 사용할 수 있게 하였다.

Abstract The value of XML as electronic documents is increasing nowadays. The XML document has properties of semistructured data. It can be modeled as object-oriented model which can be easily adapted by object-oriented database. For storing XML documents to conventional database system, extracting schema information from the DTD of a XML document has been studied for several years. In this paper we store XML documents into object-oriented database, which preserve the semantics of the documents. We can store and query by OQL and make applications by C++ binding which is the access method of object-oriented database. Therefore, existing database applications can be used without modification.

1. 서 론

웹은 하이퍼텍스트 마크업 언어(Hypertext Markup Language : HTML)의 등장으로 1990대 중반부터 폭발적으로 성장해 왔다. HTML문서는 누구나 사용할 수 있을 정도로 간단하고 보편적이며 특별한 데이타 타입이 없는 단순한 텍스트이므로 이식성과 사용 면에서 편리하다. 하지만 전자 문서는 텍스트 자체로서의 의미 외에 데이타의 새로운 태그와 속성을 추가할 수 있는 확장성, 정확한 검색 결과, 자신의 데이타를 기술할 수 있는 기능을 요구하게 되었다. 이러한 기능에 비해 HTML은 작성된 문서의 논리적인 구조와 그 내용에 대하여 문서 자체가 자신을 표현하는 능력이 부족하고, 대량의 문서들로부터 원하는 정보를 검색하는데 드는 비용적 손실이 매우 크다. 이를 해결하기 위한 언어로서 SGML(Standard Generalized Markup Language)이 있지만 인터넷에 대한 고려가 부족한 언어를 웹 상에서 그대로 적용할 수 없기 때문에 SGML 언어의 장점을 최대한 수용한 부분 집합인 XML[1]이 등장하게 되었다.

XML 문서는 크게 선언부(Declaration Part), 문서형 정의부(Document Type Definition:DTD) 및 문서 실체부(Document Instance:DI)로 구성되어 있다. 선언부를 SGML과 비교해 보면 상당히 축소되어 있고, DTD는 SGML의 형태를 많이 보존하고 있으며 문서 실체부에 있어서는 SGML과 비교해서 태그 최소화나 태그 생략

· 본 논문은 1999년도 BK 21 정보기술분야 사업에서 부분적으로 지원받았음.

† 비 회 원 : 한국 오라클 연구원
BongSoo.Ko@oracle.com
†† 비 회 원 : 서울대학교 컴퓨터공학부
swpark@oopsla.snu.ac.kr
††† 비 회 원 : 서울대학교 인지과학과
ksmin@oopsla.snu.ac.kr
†††† 종신회원 : 서울대학교 컴퓨터공학부 교수
hjk@oopsla.snu.ac.kr

논문접수 : 1999년 12월 13일
심사완료 : 2001년 6월 23일

이 적용되지 않는 점이 다르다.

XML 문서를 저장하는 방법은 화일 단위로 저장하거나 문서의 구조를 반영하여 쪼개어진 형태로 저장할 수 있다[2]. 화일 단위로 저장하면 간단하지만 매번 파싱(parsing)을 해야하고, 검색 시스템을 사용자가 직접 구현해야 하는 단점을 가지고 있어서 비효율적이다. 따라서 저장, 검색, 관리하는 특성을 모두 가지고 있는 데이타베이스 시스템을 이용하는 방법이 대두되었다. 이러한 데이타베이스로 [3,4]와 같이 새로운 시스템을 만드는 것과, 기존 데이타베이스 시스템을 이용하는 방법이 있다. 새로운 시스템을 만드는 것은 그 모델을 잘 지원할 수 있는 장점이 있으나 안정성있는 시스템을 구축하는 것이 쉽지 않다[5]. 기존 데이타베이스 시스템으로는 관계형, 객체지향, 객체 관계형 데이타베이스 시스템이 있다. 위의 각각의 방법에 대한 비교는 관련 연구에서 자세히 설명한다.

객체지향 데이타베이스는 관계형 데이타베이스 시스템에 비해 계층적 트리구조를 갖는 XML 문서를 모델링하기 쉽고, 트리를 탐색할 때 죠인 연산의 부담이 적으며 전후종속 관계를 클래스 구조로 쉽게 지원할 수 있다[4,6]. 객체지향 데이타베이스에 XML 문서를 저장하는 방법은 파싱된 DOM(Document Object Model)[7]을 저장하는 방법과 엘리먼트(element) 태그의 의미를 유지하면서 저장하는 방법이 있다. 전자와 같이 DOM을 저장하는 방법은 XML 문서의 의미적인 정보를 잃어버리게 된다. 즉 사람을 나타내는 person과 같은 태그의 의미를 가지고 있는 클래스나 애트리뷰트가 데이타베이스 내에 나타나지 않기 때문에 저장된 데이타를 분석하기 전에는 그 데이타가 의미하는 바를 알아내기가 힘들다. 또한 객체지향 데이타베이스에서 사용하는 OQL을 사용할 때 의미있는 질의를 생성하기가 어렵게 된다. 즉 사용자의 측면에서 저장된 데이타를 잘 사용하기 위해서는 XML에 나타나는 태그의 의미적인 정보를 보존하여 저장하는 것이 필요하다.

본 연구에서는 엘리먼트 태그의 의미를 유지하여 객체지향 데이타베이스에 저장하는 방법을 제안한다. 이때 고려해야 할 부분은 반구조적(semistructured) 성질을 갖는 XML 문서의 특성과 구조적 성질을 갖는 데이타베이스 특성의 차이를 없애는 것이다. 반구조적 성질은 정규 문법을 따르고 있어서 순서가 있고 반복적이며 선택적이다. 반면에 구조적 성질은 폐쇄 성질을 따르고 있어서 무순서 성질을 가지고 있고 비선택적이다. 이러한 성질의 차이는 XML 문서를 데이타베이스에 저장하는데 해결되어야 한다. 본 논문에서는 객체지향 데이타베이스의 DTD 정보에서 정적인 반복, 선택 성질을 파악하여 객체지향 데이타베이스 스키마를 생성하고, 실제 XML 문서에서는 동적인 반복, 선택 성질을 파악하여 자료의 손실을 없이 객체지향 데이타베이스 객체로 저장하는 시스템을 설계, 구현하였다. 즉 본 논문은 XML 전자 문서의 논리적인 구조를 기술하는 DTD 정보를 관리하기 위하여 이를 객체지향 메타 스키마로 사상하는 방법을 제안하고, 이 메타 스키마를 이용하여 실제 XML 문서를 객체지향 데이타베이스에 저장하는 방법을 제안하고자 한다.

2절에서는 XML을 저장 관리하는 현재 유사 연구 분야들에 대해 간단히 살펴보고, 3절에서는 XML을 ODMG[8] 표준을 따르는 객체지향 데이타베이스에 저장관리하기 위한 여러 가지 고려사항을 알아본다. 4절에서는 PDM[9]의 설계 및 구현에 대해 살펴보고, PDM이 지닌 장, 단점을 제시한다. 5절에서는 결론 및 향후 연구 계획에 대해 살펴보기로 한다.

2. 관련 연구

2.1 XML의 저장 매체 유형

XML을 저장, 관리하는 저장 매체를 유형별로 나누어 보면 화일 시스템, 관계형 데이타베이스 시스템 및 객체지향 데이타베이스 시스템으로 나눌 수 있다. 먼저 화일 시스템을 XML 저장 매체로 사용하는 경우 기존의 HTML은 정의 가능한 구조가 필요 없으므로 단일 블록으로 저장하는 것이 간단하고 편리하지만, XML은 HTML과는 달라서 화일 시스템에 단일 블록으로 저장하면 XML에 대한 정보를 추출하기 위하여 매번 파싱을 해야 한다. 그러므로 XML 응용 프로그램이 문서전체를 저장하지 않고 소량의 요소들 만을 저장하고자 하는 경우에 비효율적이 된다. 그리고 다양한 연결 구조와 데이타의 의미를 유지하기 위해서 복잡한 형태의 XML을 파싱하려는 시도는 화일 시스템의 한계를 벗어난 것이다. 물론 화일 시스템에서도 XML 문서와 그 의미를 BLOB으로 저장하고 매번 사용될 때마다 파싱하여 사용할 수 있다. 하지만 이러한 접근 방식은 항상 사용자가 저장, 검색 시스템을 직접 구현해야 하는 단점이 있다.

관계형 데이타베이스 시스템을 XML 저장 매체로 사용할 경우는 테이블 기반의 데이타 모델이 계층적이고 상호 연결된 구조를 갖는 XML에 적합하지 않은 단점을 가지고 있다[10,11]. 그리고 가변적인 크기의 자료나 BLOB을 다루는데 최적의 시스템이 아니므로, 관계형 데이타베이스 시스템의 테이블은 XML 트리 구조를 효과적으로 표현하기에 부적합하다. 즉 관계형 데이타베이

스 시스템은 XML 문서를 테이블 형태로 매핑하는 과정에서 XML 문서의 구조와 의미를 잃어버리게 되고 데이타베이스 설계과정에서도 중복 문제를 초래하게 된다. 또한 관계형 데이타베이스 시스템은 객체 단위의 로킹(locking)을 다룰 수 없으며, 많은 테이블로 변환된 XML 문서에 대한 검색은 많은 죠인 연산을 강요할 수 있으므로 검색 시간이 길어지게 된다. 데이타베이스 시장을 대부분 차지하고 있는 것이 관계형 데이타베이스 시스템이므로 위와 같은 단점에도 불구하고 아직까지는 이를 XML 저장 매체로 사용하고 있다[4].

본 논문에서 사용할 저장 매체인 객체지향 데이타베이스 시스템을 XML 저장 매체로 사용할 경우 다루어지는 단위가 객체이므로 계층적 XML 트리 구조를 쉽게 객체지향 데이타베이스로 모델링할 수 있다. 이는 객체지향 데이타베이스 시스템이 계층적 트리 항해(navigation)와 다양한 연결 검색 기능을 제공하기 때문이다. 관계형 데이타베이스 시스템의 경우에서는 여러 번의 죠인 연산을 해야하는 것에 비해 객체지향 데이타베이스에서는 경로식(path expression)으로 표현되기 때문에 훨씬 나은 성능을 가질 수 있음을 알 수 있다[6]. 또한 XML 문서를 객체지향 데이타 모델로 변환하는 과정이 관계형 데이타 모델에서의 XML 객체 데이타와 테이블 형식의 데이타 간에 발생하는 부조화(impedance mismatch)문제가 없다.

2.2 XML의 저장 형태

XML을 선택한 저장 매체에 저장할 때, XML을 데이타와 함께 스키마까지 모두 한 덩어리로 저장하는 방법과 본 연구에서 제시하는 방법인 스키마를 먼저 구성한 후에 스키마대로 데이타를 재구성한 후에 저장하는 방법으로 나눌 수 있다. 반구조적 데이타와 스키마를 함께 저장하는 시스템 중 대표적 시스템인 Lore[3]와 TSIMMIS[12]에서는 XML 데이타는 그래프로 저장되고 스키마는 그래프에서 에지를 레이블화한 속성으로 저장된다. 이 방법은 XML 같은 반구조적인 데이타에 요구되는 융통성을 제공한다. 예를 들어 새로운 데이타가 적재되면 바로 그 구조에 상관없이 이전의 구조가 변했더라도 잘 적용된다. 하지만 기존의 데이타베이스가 아닌 새로운 데이타베이스 시스템을 이용하는 것은 시스템의 안정성을 확보하는데 문제가 있다[5].

반면에 기존의 데이타베이스 시스템에는 어떤 데이타들의 공통 구조로서 데이타의 구조적 관계와 의미적 관계를 정의한 스키마를 이용함으로써 데이타에 대한 구조적 접근이 가능해진다[6]. 스키마는 질의를 정형화하는 바탕이 되며, 스키마를 기반으로 하여 데이타에 대한 질의를 수행하거나 데이타 연산을 수행함으로써 질의를 최적화 시킬 수 있다. 또한 데이타들의 특성을 반영하는 개요로서 데이타와 데이타들 간의 관계에 대한 구조적 관점을 제공한다. 이런 스키마를 먼저 구성한 후에 실제 데이타 객체들을 데이타베이스에 저장하는 방식[10]은 일단 DTD를 파싱하여 그 결과를 데이타 모델의 스키마로 사상시킨 후에 실제 XML 문서를 파싱하여 XML 객체들을 저장하는 형태를 취하고 있는데, 이런 방식을 사용하면 XML은 데이타베이스의 특성을 이용할 수 있게 된다. 하지만 반구조적 성질의 데이타를 데이타베이스의 구조적 성질로 변환함에 따른 부조화 문제를 해결하는 것이 문제이다. 이런 문제가 해결된다면 데이타와 스키마를 동시에 저장하는 방식에 비해 한번만 데이타베이스에 저장하면 되어 시간, 공간 낭비를 줄일 수 있다는 장점이 있다. 본 논문은 이와 같은 방법으로 먼저 DTD를 이용하여 스키마를 생성하고 스키마에 맞도록 XML 문서 객체들을 데이타베이스에 저장하는 방식을 연구한다.

3. XML의 저장 관리시 고려 사항

본 논문의 구현은 크게 DTD로부터 객체지향 데이타베이스 스키마로 사상하는 부분과 사상된 스키마에 맞도록 XML 문서의 객체들을 객체지향 데이타베이스에 저장하는 부분으로 나누게 된다. 이때 XML 문서 데이타의 무손실을 보장하도록 객체지향 데이타베이스가 설계되어야 한다. 그러므로 DTD가 없으면 스키마를 만들 수 없으므로 데이타베이스에 저장할 수 없다. 하지만[13]에서와 같이 데이타로부터 스키마 정보를 추출할 수 있으므로 DTD 정보는 항상 있다고 가정하고 있다.

본 연구에서는 객체지향 데이타베이스로 ODMG 표준을 따르는 SOP[14]를 사용하였다. 이를 위해서는 먼저 XML의 DTD를 해석해서 객체지향 데이타베이스 스키마를 생성해야 하는데, 이 과정에서 고려해야 할 점은 크게 순서(ordering)와 선택(selection) 문제이다. 반구조적 성질을 갖는 XML을 구조적 성질을 갖는 객체지향 데이타 모델로 변환했을 때, 엘리먼트의 순서정보를 잃어버리는 문제를 어떻게 해결할 것인가가 고려대상이 된다. 그림 1과 같이 XML의 DTD는 엘리먼트라는 논리적 요소들로 구성되어져 있다. 엘리먼트의 이름은 XML 문서에서 태그로 사용된다. 엘리먼트의 구조는 다른 엘리먼트와 #PCDATA, EMPTY등과 같은 기본형들 그리고 연결자(connector)들로 구성되는데, XML이 반구조적 성질을 갖는 이유는 이러한 연결자들 때문이다.

```
<?xml encoding="US-ASCII"?>
<!ELEMENT personnel (person)+>
<!ELEMENT person (name,email*,url*,link?)>
<!ATTLIST person id ID #REQUIRED>
<!ELEMENT family (#PCDATA)>
<!ELEMENT given (#PCDATA)>
<!ELEMENT name (#PCDATA|family|given)*>
<!ELEMENT email (#PCDATA)>
<!ELEMENT url EMPTY>
<!ATTLIST url href CDATA #REQUIRED>
<!ELEMENT link EMPTY>
<!ATTLIST link manager IDREF #IMPLIED
subordinates IDREFS #IMPLIED>
```

그림 1 DTD의 예 : personnel.dtd

연결자에는 순서를 나타내는 연결자(“,”), 선택을 나타내는 연결자(“|”), 옵션을 나타내는 연결자(“?”) 및 반복을 나타내는 연결자(“+”, “*”)[1] 등이 있다.

위의 연결자들 중 반복을 나타내는 연결자(“+”, “*”)는 SOP에서 List형을 지원하기 때문에 SOP 스키마로 사상하는데 고려할 필요가 없지만, 순서를 나타내는 연결자(“,”)를 통해 XML 데이타 객체들에게 순서를 부여하는 점이 고려되어야 한다. 그림 1의 예에서 <!ELEMENT person (name,email*,url*,link?)>는 name 객체가 나열된 후에 email 객체가 0번 이상, url 객체가 0번 이상, 그리고 link 객체가 올 수도 있고 그렇지 않을 수도 있음을 의미한다 이렇듯 XML 데이타의 객체들은 그 순서를 가지고 있지만, 데이타베이스는 객체들이 순서를 가지지 않도록 정의되어 있다. 따라서, 이 점이 고려되어야 XML 문서에서 객체지향 데이타 모델로 사상할 때 데이타의 무손실을 보장할 수 있다. 또한 XML 문서는 선택을 나타내는 연결자("|")를 통해 XML 객체들이 선택적으로 올 수 있게 한다. 예를 들어 (family|given)와 같은 XML 문서 유형이 있으면 family 객체 혹은 given 객체가 나오도록 XML 문서가 구성된다. 즉 family 객체가 나온다면 given 객체는 나오지 않도록 해야 한다. 이러한 유형을 객체지향 스키마로 사상하려면 태그를 사용하거나 객체지향 데이타베이스에서 UNION을 지원해야 한다. 만약, 그렇지 않은 경우라면 쓸모 없는 NULL 값들이 저장되게 된다.

1) SOP의 List는 객체의 갯수를 0개 이상 저장할 수 있지만, 1개 이상으로 강제할 수는 없다. 그러므로 +와 *는 동일하게 List로 매핑된다.

4. PDM/ODB의 설계 및 구현

우리는 웹 상의 데이타 교환과 저장에 대한 표준인 XML을 이용한 중계자 시스템인 XWEET[15]를 제안하였다. 이것은 XML 데이타를 처리하기 위한 미들웨어 시스템으로서 기존의 래거시 시스템으로부터의 XML 형식의 데이타 접근과 중계자 시스템 수준에서의 XML 데이타 저장과 접근 방법에 대한 연구를 데이타베이스 처리 관점에서 수행한 것이다. XWEET 시스템 중 특히 데이타베이스에 XML 데이타를 저장하는 모듈로 PDM(Persistent Data Manager)[9]이 있다. PDM은 관계형 데이타베이스인 SRP[14,16]를 이용한 PDM/RDB, 랩퍼를 이용하여 접근하는 PDM/Wrapper 및 본 논문에서 제안한 객체지향 데이타베이스인 SOP를 이용한 PDM/ODB로 구성되어 있다. PDM/RDB는 XML 문서를 XML 질의어를 이용하여 XML-뷰를 정의할 수 있도록 하였다. 이렇게 함으로서 XML 데이타를 관계형 데이타베이스의 테이블과 동일한 관점에서 SQL로 질의를 작성할 수 있도록 하였다. PDM/Wrapper는 랩퍼를 통하여 XML 데이타를 접근할 때 유연한 확장을 위하여 적절한 접근 경로를 제공하도록 하였다. PDM/ODB는 XML 객체를 DTD를 이용하여 분석한 후 엘리먼트의 의미를 보존하는 적절한 이름의 클래스의 인스턴스로 저장하여 데이타를 ODMG C++/CLOS[17]/Java 바인딩이나 OQL로 데이타를 접근하는 방법을 이용하였다. 본 논문에서는 PDM/ODB에 관하여 설명한다.

4.1 PDM/ODB의 설계 및 구현

PDM/ODB가 동작하는 과정은 그림 2와 같이 DTD에서 객체지향 스키마를 추출하고, 추출해낸 스키마 구조대로 XML 문서에서 각 객체들을 뽑아내어 SOP에 저장한다. 그리고 이기종의 데이타베이스나 결과도

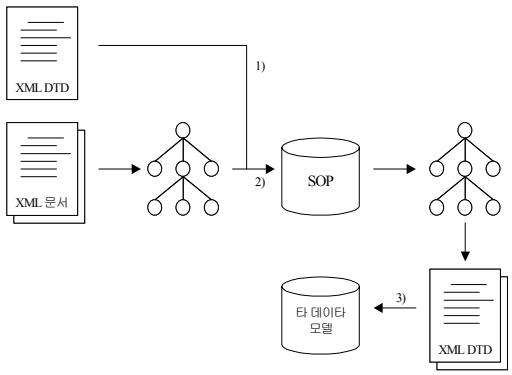

그림 2 PDM의 동작 과정

XML 문서로 SOP에서 재 생성해야 한다. 물론 이 SOP에서 재생성된 XML 문서는 SOP에 저장되기 전의 XML 문서와 비교해서 데이타의 손실이 없도록 해야 한다. SOP에서 스키마 정보는 중요한 의미를 가진다. 객체 내에 담겨있는 정보를 해석하는데 없어서는 안되는 중요한 정보들을 제공하기 때문이다. 객체라 함은 실제로 디스크 또는 메모리 내에 존재하는 일정 영역을 가리킬 뿐이고 이에 사용자들이 사용할 수 있는 의미있는 정보를 담기 위해서는 스키마 정보의 도움을 받아야 한다. 스키마 정보란 클래스의 이름, 클래스의 상위 클래스 정보, 클래스의 하위 클래스 정보, 클래스 내에 있는 속성들의 이름과 각 속성의 크기, 타입과 객체내에서의 위치, 클래스에 속한 메소드의 입출력 인자 타입과 그 메소드에 해당하는 함수를 호출하는 방법등을 포함한다. 이런 정보들은 컴파일시에는 C++ 헤더 화일내에서 모두 구할 수 있는 정보들이다. 하지만 SOP에서는 C++ 컴파일러없이 작동되는 런타임 모듈들도 존재하므로 이 모듈들이 원하는 정보를 구할 수 있도록 C++ 헤더 화일로부터 위의 정보들을 스키마 관리자에게 등록해 주어야 한다[18]. 즉, 스키마 등록과 객체 등록은 SOP 임포트 툴(import tool)을 이용하게 된다. SOP 임포트 툴의 역할은 C++ 헤더 화일로부터 필요한 정보들을 추출하여 스키마 관리자에 등록하는 것이다. 또한 XML 문서에서 추출한 객체들도 SOP에 저장한다.

따라서 구현 방법은 DTD를 분석하여 C++ 헤더 화일을 생성해 내고 XML 문서를 분석하여 추출된 객체들을 SOP에 저장시키는 C++ 소스를 생성해 내는 것이다. 이렇게 생성된 C++ 헤더와 C++ 소스를 컴파일하여 실행시키면 XML 문서가 저장되는 것이다. 이때 XML 데이타의 순서 정보를 유지하기 위하여, 생성된 모든 XML 클래스들은 상위 클래스로 XMLObject를 두었다. 이 클래스는 내부 애트리뷰트로 Ref<List<Ref<RefAny>>> _elm_order를 가지고 있다. _elm_order는 객체들의 OID를 순서대로 유지하고 있는 것으로서 응용 프로그램에서 객체의 순서 정보가 필요하면 이 애트리뷰트를 이용하여 순서를 알아낸다. 즉 XMLObject 클래스는 _elm_order 애트리뷰트와 이를 조작하는 메쏘드를 가지고 있다.

4.2 DTD로부터 스키마 생성

그림 3은 PDM에서 DTD로부터 객체지향 스키마를 생성하는 시스템이다. 이는 DTD 입력을 파싱하여 그 정보를 통해서 C++헤더 화일을 생성해 내는 것이다. 이 부분을 설계할 때 앞서 언급한 고려사항에서 순서 연결자(“,”)와 선택 연결자(“|”)에 해당되는 DTD 부분을 객체

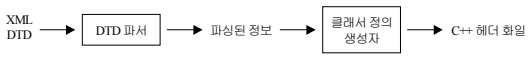

그림 3 DTD-to-Schema 시스템 구성도

지향 스키마로 사상할 때 데이타의 손실이 발생하지 않도록 해야 한다는 점을 중심으로 설계한다. 먼저 DTD를 요소별로 살펴보면 <?...?>형태와 <!DOCTYPE...>형태 그리고 인스턴스의 내용부분으로 나눌 수 있다. <?...?>형태는 XML의 서두 부분으로 버전, 인코딩, DTD를 선택적으로 저장할 수 있는 기능 등으로 구성되고 <!DOCTYPE...>형태는 문서명을 지정하기 위한 예약어(reserved keyword)이며 인스턴스의 내용부분은 크게 ELEMENT, ATTRIBUTE, ENTITY로 나눌 수 있다. 객체지향 스키마로의 사상에 관련된 형태는 인스턴스의 내용 부분이므로 인스턴스의 내용 부분인 ELEMENT, ATTRIBUTE, ENTITY를 어떻게 사상했는지에 관련된 설계과정은 알고리즘 1과 같다.

알고리즘 1 DTD로부터 SOP 스키마 생성

```
IN ← DTD 화일
while DTD 파일의 화일 끝(EOF)이 아니면 do
    IN에서 <"으로 시작해서 >"로 끝나는 문자열 S를 구분한다.
        if S의 종류가 ENTITY인 경우 then
            Hash table H ← ENTITY로 선언된 식별자와 정의 부분
        else if S의 종류가 ELEMENT인 경우 then
            새로운 클래스 C 생성
            H에 있는 식별자가 사용되면 H에서 대체 시키고 괄호가
            없는 동등한 의미로 변환한다. 변환된 결과의 내용명세
            CS를 분석하여 반복적 성질을 갖는 것은 C의 배열 I로
            그렇지 않은 것은 C의 배열 N으로 저장한다.
        else if S의 종류가 ATTRIBUTE이 경우 then
            괄호가 없는 동등한 의미로 변환
            변환된 결과의 내용 명세를 분석하여 반복적 성질 갖는
            것은 C의 I에, 비반복적 성질을 갖는 것은 C의 N에 저장
        end if
end while
C의 I, N을 속성(Attribute)으로 갖는 클래스 C의 스키마 정의를 출력
```

4.2.1 ELEMENT의 경우

ELEMENT의 구문은 다음과 같다.

<!ELEMENT Element-이름 내용-명세 >

엘리먼트 이름은 실제 SOP의 스키마에서 하나의 클래스로 사상된다. 내용 명세(content specification)는 반복적인 성질을 갖는 내용과 비반복적인 성질을 갖는 내용으로 구분해 볼 수 있는데, 이를 명확히 하기 위해서 DTD 파서가 내용 명세를 파싱할 때 다음과 같이 괄호가 없는 형태로 먼저 변경하는 것이 좋다[19].

```
class Email : public XMLObject {
public:
    Ref<d_String> text;
    Email(Ref<d_String>&);
};
class Name : public XMLObject {
public:
    Ref<List<Ref<d_String> > > text;
    Ref<List<Ref<Family> > > family;
    Ref<List<Ref<Given> > > given;
    Name();
 };
class Given : public XMLObject {
public:
    Ref<d_String> text;
    Given(Ref<d_String>&);
};
class Family : public XMLObject {
public:
    Ref<d_String> text;
    Family(Ref<d_String>&);
};
class Url : public XMLObject {
public:
    Ref<d_String> href;
    Url(Ref<d_String>&);
};
class Link : public XMLObject {
public:
    Ref<d_String> manager;
    Ref<List<Ref<d_String> > >
                    subordinates;
    Link(Ref<d_String>&);
};
class Personnel : public XMLObject {
public:
    Ref<List<Ref<Person> > > person,
};
class Person : public XMLObject {
public:
    Ref<List<Ref<Email> > > email;
    Ref<List<Ref<Url> > > url;
    Ref<Name> name;
    Ref<Link> link;
    Ref<d_String> id;
    Person(Ref<Name>&, Ref<Link>&,
    Ref<d_String>&);          };
```

그림 4 생성된 SOP 스키마의 예: personnel.h

$(a \mid b) \rightarrow a?, b?$ $(a, b)* \rightarrow a*, b*$
$(a, b)? \rightarrow a?, b?$ $(a*)* \rightarrow a*$
$(a*)? \rightarrow a*$ $(a?)* \rightarrow a*$
$(a?)? \rightarrow a?$

이렇게 변경하고 나면 내용 명세의 각 요소들은 반복적 성질을 갖는 경우와 그렇지 않는 경우로 분류가 되고 이는 사상된 클래스에 반복적 성질을 갖는 경우는 Ref<List<Ref<a>>>으로, 그렇지 않은 경우는 Ref<a>로 구분되어 속성(attribute)으로 구성되어 질 수 있다. 그림 1의 personnel.dtd 화일을 입력으로 해서 생성된 SOP 스키마 헤더 화일이 그림 4와 같다. 이 때, ELEMENT의 내용 명세에서 고려해야 할 사항이 순서 연결자인데, 이러한 순서 정보가 손실되는 것을 막기 위해 XMLObject 클래스를 둔다.

4.2.2 ATTRIBUTE의 경우

ATTRIBUTE의 구문은 다음과 같다.

```
<!ATTRIBUTE Attribute-이름 AttDef*>
AttDef ::= Name AttType DefaultDecl
```

다음은 AttType에 따라 SOP의 사상관계를 다음과 같다.

- CDATA의 경우
  DTD : <!ELEMENT Article>
  <!ATTLIST Article DATE CDATA #REQUIRED>
  XML 문서 : <Article DATE="January 15, 1999">...
  </Article>
  SOP 스키마 사상 결과 : class Article {
  Ref<d_String> date; ...
- NMTOKEN의 경우
  DTD : <!ATTLIST Part DATE NMTOKEN
  #REQUIRED>
  XML 문서 : <Part DATE="1998-05-04">...</Part>
  SOP 스키마 사상 결과 : class Part {
  Ref<d_String> date; ...
- NMTOKENS의 경우
  DTD : <!ATTLIST Table Name NMTOKEN
  #REQUIRED
  Fields NMTOKENS \#REQUIRED$>$} \\
  XML 문서 : <Table Name="SECURITY" Field=
  "USERID
  PASSWORD DEPARTMENT">... </Table>
  SOP 스키마 사상 결과 : class Table {
  Ref<d_String> name;
  Ref<List<Ref<d_String>>>
  fields; ...
- ID와 IDREF의 경우
  DTD : <!ATTLIST Section My-ID ID #IMPLIED>
  <!ATTLIST Cross-Reference Target IDREF
  #REQUIRED>

XML 문서 : <Section My-ID="Why.XML.Rocks"> </Section>
<$Cross-Reference Target="Why. XML.
Rocks"/>
SOP 스키마 사상 결과 : class Section {
Ref<d_String> my-id;
Ref<Cross-Reference,
"idref"> ra; ...
class Cross-Reference {
Ref<Section, ra> "idref";...

• ENTITY의 경우
DTD : <!ATTLIST BOOK-REF TARGET ENTITY
#REQUIRED>
<!ENTITY another-bookSYSTEM
"http://wwwoopsla.snu.ac.kr">
XML 문서 : <BOOK-REF TARGET="another-book">
...</BOOK>
SOP 스키마 사상 결과 : ENTITY로 선언된 정보는 해쉬 테이블에 저장한다.

4.3 XML 문서에서 SOP 객체 생성하기

그림 5는 PDM에서 XML 문서로부터 객체지향 인스턴스를 생성하는 과정이다. 이와 같이 XML 문서에서 SOP의 객체를 만들기 위해서 먼저 DOM을 이용한다. DOM은 XML 문서의 요소와 내용을 응용과 프로그래밍 언어가 쉽게 액세스하고 조작할 수 있도록 구성된 계층적인 트리이다(그림 6참조). 따라서, 텍스트인 XML 문서를 DOM 구조로 변환한 후에 DOM 트리를 순회해 가면서 SOP 객체를 생성하는 소스 코드를 만들어 주면 된다.

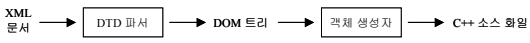

그림 5 XML document-to-Instance 시스템 구성도

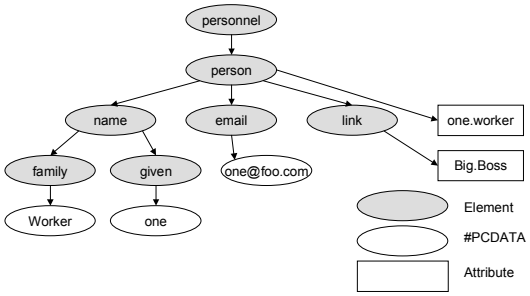

그림 6 그림 8의 DOM 트리

DOM 트리의 탐색은 상향식(bottom-up) 방식으로 하는데, 이유는 단말 노드가 원자 노드(atomic node)이므로 단말 노드부터 객체를 생성하고 생성된 객체의 OID들을 묶어서 위 레벨의 객체를 만들고 그 레벨의 객체 OID들을 모아서 그 위 레벨의 객체를 만들어 가는 식으로 SOP의 객체를 생성해 나가게 설계하였다(그림 7 참조).

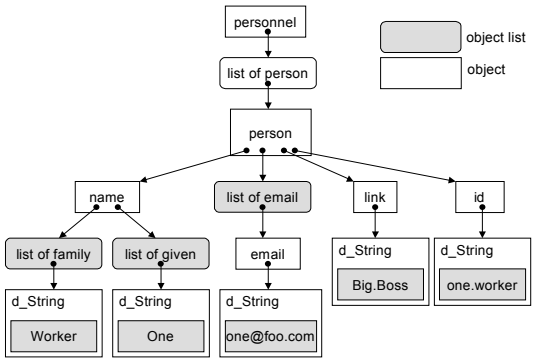

그림 7 그림 8이 SOP에 저장된 형태

알고리즘 2 SOP 스키마 구조의 SOP 객체 생성(C++ 소스 파일)

```
DOC ← XML Document
Make DOM Tree DT by DOC
S1.push(root node of DT, height)      /* Stack S1 */
while {(A ← S1.pop) is not Empty do
      if A is Text Node then
         Make an instance of A
         S2.push(OID and height of A)
                              /* Stack S2 */
            if S2.top() > s1.top() then
                 break
            end if
      else
         S2.Push(OID and height of A)
            NL ← all children of A    /* Array NL */
            S1.push(all elements of NL)
      end if
end wihle
while true do
      B ← S2.pop()
      List X ← B
      if height of B > depth of S2 then
          break
end if
end while
W ← S2.pop()
리스트 X의 모든 요소들의 OID와 속성들의 값을 묶어서
W의 인스턴스를 삽입하는 소스코드 생성
S2.push(W)
```

```
<? xml version="1.0"?>
<personnel>
<person id="one.worker">
<name>
<family>Worker</family>
<given>One</given>
</name>
<email>one@foo.com</email>
<link manager="Big.Boss"/>
</person>
. . .
</personnel>
```

그림 8 XML 문서의 예 (personnel.xml)

이와 같이 저장된 객체들은 모두 XMLObject에서 상속받는다. XMLObject는 XML 문서를 재 생성할 수 있는 순서정보를 가지고 있으며, XML 질의어를 처리할 수 있는 메쏘드들을 포함하고 있다. 또한 의미있는 엘리먼트들로 모델링 하였기 때문에 가상적으로 분할한 방법에서는 힘들었던 기존의 XML 문서가 아닌 객체지향 데이타베이스의 자료들과 함께 질의를 처리하는 것이 가능하게 되었다. 동시에 XMLObject의 메쏘들을 이용하여 XML 문서 질의도 할 수 있게 되었다.

5. 결론 및 향후 연구계획

본 논문에서는 XML 문서를 객체지향 데이타베이스에 효과적으로 관리할 수 있도록 DTD를 객체지향 데이타베이스 스키마로 사상하였다. 또한 이 스키마에 맞도록 XML 문서를 객체지향 데이타베이스에 저장하는 방법론을 제시하였으며 XML 데이타의 손실 없이 저장됨을 보였다. 그렇게 함으로써 데이타베이스 프로그래머나 질의어 사용자로 하여금 XML 질의어를 배울 필요없이 XML 데이타를 데이타베이스의 데이타처럼 사용할 수 있고, 기존의 데이타베이스에 산재해 있는 자료들과 함께 결합하여 질의어를 작성할 수 있는 잇점이 있다. 또한 XML을 교환하기 위해 데이타베이스에서 XML을 재 생성해 낼 수도 있다.

하지만 이 방법의 단점은 일단 XML 데이타와 스키마를 한꺼번에 저장하는 방식에 비해 융통성이 떨어진다는 단점이 있다. 즉 DTD는 데이타베이스의 스키마가 고정적인 성격을 갖는 것과는 달리 자주 변할 수 있다. 즉, 이렇게 자주 변하는 DTD를 융통성있게 고정적 성격을 갖는 데이타베이스에 반영하기가 힘들다는 점이다. 그리고, 클래스의 크기와 개수, 클래스 내 속성의 크기와 개수 등을 제한하여 시스템 성능 향상을 할 수 있는 의미적 정보 분석이 필요하다. 이는 데이타 마이닝 기술[10]을 반영한다면 앞으로 향상된 성능을 가질 수 있을 것이다.

참 고 문 헌

[1] *Extensible Markup Language (XML) 1.0. W3C recommendation*, http://www.w3.org/TR/REC-xml, 1998.

[2] GMD-IPSI, GMD-ISPI XQL Engine, *http://xml.darmstadt.gmd.de/xql*, 2000.

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

[4] POET, XML The Foundation for the Future, *http://www.poet.com*.

[5] Michael Stonebraker and Paul Brown, *Object-Relational DBMSs Tracking The Next Great Wave*, Morgan Kaufmann, 2 edition, 1999.

[6] Vassilis Christophides, Serge Abiteboul, Sophie Cluet, and Michel Scholl, From Structured Documents to Novel Query Facilities, *SIGMOD*, 1994.

[7] W3C, Document Object Model (DOM), *http://www.w3.org/DOM*, Feb. 2000.

[8] R.G.G. Cattell and Douglas K. Barry, editors, *The Object Database Standard: ODMG 2.0*, Morgan Kaufmann Publisher, Inc., 1997.

[9] 박상원, 민경섭, 김형주, XML 데이타베이스 지원을 위한 통합 환경, *한국정보과학회 논문지(CP)*, 6(6), 2000.

[10] Alin Deutsch, Mary~F. Fernandez, and Dan Suciu, Storing Semistructured Data with STORED, *SIGMOD*, 1999.

[11] R. Goldman, J. McHugh, and J. Widom, From Semistructured Data to XML: Migrating the Lore Data Model and Query Language, *WebDB '99*, June 1999.

[12] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom, Object Exchange Across Heterogeneous Information Sources, *ICDE*, 1995.

[13] Minos Garofalakis, Aristides Gionis, Rajeev Rastogi, S. Seshadri, and Kyuseok Shim, XTRACT: A System for Extracting Document Type Descriptors from XML Documents, *SIGMOD*, 2000.

[14] 안정호, 김형주, SRP에서 SOP까지, *한국정보과학회지*, 4 1994.

[15] Jae-Mok Jeong, Sangwon Park, Tae-Sun Chung, and Hyoung-Joo Kim, XWEET: XML DBMS for Web Environment, *The First Workshop on Compu-*

ter Science and Engineering 2000, Seoul, Korea, Jun. 2000, http://oopsla.snu.ac.kr/xweet/xweet-eng.ps.
[16] 이강우, 안정호, 김형주, 확장용이 클라이언트-서버 RDBMS의 설계 및 구현, *한국정보과학회 SIGDB*, 1994.
[17] 정태선, 조은선, 김형주, Sopclos: 객체지향 데이타베이스 관리 시스템을 위한 CLOS 인터페이스, *한국정보과학회 논문지(B)*, 24(9), 1997.
[18] 김형주 외 SOP 팀, *서울대에서 만든 SOP 이야기*, 마이트 프레스, 2000.
[19] Jayavel Shanmugasundaram, Kristin Tufte, Gang He, Chun Zhang, David DeWitt, and Jeffrey Naughton, Relational Databases for Querying XML Documents: Limitations and Opportunities, *VLDB*, 1999.

고 봉 수
1991년 ~ 1997년 동국대학교 컴퓨터공학 학사 졸업. 1997년 ~ 1998년 과학기술원 정보통신 공학석사. 1998년 ~ 2000년 서울대학교 컴퓨터공학 석사 졸업. 2001년 한국 오라클 모바일 연구소

박 상 원
정보과학회논문지 : 컴퓨팅의 실제
제 7 권 제 3 호 참조

민 경 섭
정보과학회논문지 : 컴퓨팅의 실제
제 7 권 제 3 호 참조

김 형 주
정보과학회논문지 : 컴퓨팅의 실제
제 7 권 제 1 호 참조