ODYSSEUS/XMLStore: 오디세우스 객체관계형 DBMS를 위한 XML 저장 시스템

(ODYSSEUS/XMLStore: An XML Storage System for the ODYSSEUS Object-Relational DBMS)

이 기 훈 † 한 욱 신 †† 김 민 수 ††† 이 종 학 †††† 황 규 영 †††††
(Ki-Hoon Lee) (Wook-Shin Han) (Min-Soo Kim) (Jong-Hak Lee) (Kyu-Young Whang)

요 약 XML 문서가 웹에서 많이 사용됨에 따라, 기존의 DBMS를 이용하여 XML 문서를 저장하고 관리하는 XML 저장 시스템에 대한 연구가 활발히 진행 중이다. 그러나 지금까지의 연구는 대부분 풍부한 모델링 기능을 제공하는 객체관계형 DBMS보다는 관계형 DBMS에 기반하여 이루어져 왔다. 본 논문에서는 오디세우스 객체관계형 DBMS를 위한 XML 저장 시스템인 ODYSSEUS/XMLStore를 설계하고 구현한다. 첫째, XML 문서 구조에서 관계형 및 객체관계형 데이타베이스 스키마로의 매핑에 대해 분석한다. 둘째, 분석된 매핑을 기술하는 방법을 표준 언어인 XML Schema를 활용하여 제안한다. 셋째, 사용자가 명시한 매핑 정보를 데이타베이스에 저장하는 저장 구조를 제안한다. 넷째, 사용자가 정의한 매핑 정보에 의거하여 XML 문서를 관계형 및 객체관계형 데이타베이스에 저장하는 세부 알고리즘들을 제안한다.

키워드 : XML 저장 시스템, 객체관계형 DBMS

Abstract As XML documents become popular in the World Wide Web, a lot of research is being done on XML storage systems that store and manage XML documents using existing DBMSs. However, most of them have been done in the context of relational DBMSs rather than object-relational DBMSs, which have more powerful modeling capability than relational ones. In this paper, we design and implement an XML storage system, ODYSSEUS/XMLStore, for the ODYSSEUS object-relational DBMS that has been under development at KAIST. First, we analyze the mapping from the structure of XML documents to the relational or object-relational database schema. Second, we propose a method for describing the mapping analyzed using a standard language, XML Schema. Third, we propose a storage structure for storing the mapping information specified by the users in the database. Fourth, we propose detailed algorithms for storing XML documents in the relational or object-relational database based on the mapping information provided by the users.

Key words : XML Storage System, Object-Relational DBMS

1. 서 론

XML 문서는 구조 정보를 추가할 수 있는 문서로서, 웹의 새로운 표준 문서로 많이 사용되고 있다[1]. 이에 따라, 대량의 XML 문서들을 효율적으로 저장하고 관리하는 XML 저장 시스템에 대한 필요성이 증가하고 있다.

XML 저장 시스템을 구현하는 방법들 중에서 기존의 관계형 DBMS를 이용하는 방법에 대한 연구가 활발히 진행 중이다. 즉, 관계형 DBMS를 이용하여 XML 문서를 데이타베이스의 레코드로 어떻게 변환하여 저장하느냐에 대한 연구가 활발히 진행 중이며[2, 3, 4], 상용 DBMS에서도 이러한 XML 문서 저장 방법을 제공하고 있다[5, 6].

† 학생회원 : 한국과학기술원 전자전산학과
drlee@mozart.kaist.ac.kr
†† 비 회 원 : 경북대학교 컴퓨터공학과 교수
wshan@knu.ac.kr
††† 비 회 원 : 한국과학기술원 전자전산학과
mskim@mozart.kaist.ac.kr
†††† 종신회원 : 대구가톨릭대학교 컴퓨터정보통신학부 교수
jhlee11@cuth.cataegu.ac.kr
††††† 종신회원 : 한국과학기술원 전산학과 교수
kywhang@mozart.kaist.ac.kr
논문접수 : 2002년 3월 18일
심사완료 : 2002년 10월 1일

XML 문서를 데이타베이스의 레코드로 변환하여 저장하기 위해서는, XML 문서 구조가 데이타베이스 스키마에 어떻게 매핑되는지를 정의하고, 정의된 매핑에 의거하여 XML 문서를 저장하는 기능이 필요하다[5, 7]. 이와 같은 기능을 지원하기 위해 상용 DBMS에서는 XML 문서 구조에서 데이타베이스 스키마로의 매핑을 정의하기 위한 매핑 언어를 제공하고 있다. 그리고 매핑 언어로 작성한 매핑 정보를 이용하여 XML 문서를 데이타베이스의 레코드로 변환하여 저장하는 기능을 제공하고 있다.

그러나 기존의 관계형 DBMS에 기반한 매핑에서는 객체관계형 DBMS 에서 제공하는 참조(reference) 타입과 컬렉션(collection) 타입을 활용한 매핑을 제공하지 못하는 단점이 있다. 또한, 상용 DBMS에서는 표준이 아닌 독자적인 매핑 언어를 사용하여 매핑을 기술하게 하므로 사용자에게 부담을 주는 단점이 있다.

본 논문에서는 오디세우스 객체관계형 DBMS[8]를 위한 XML 저장 시스템인 ODYSSEUS/XMLStore를 설계하고 구현한다. ODYSSEUS/XMLStore는 객체관계형 DBMS에서 제공되는 참조 타입과 컬렉션 타입을 활용한 매핑을 제공하므로, 트리(tree) 구조를 가지는 XML 문서를 보다 자연스럽게 표현할 수 있는 장점이 있다. 또한, 컬렉션을 사용하므로 주어진 엘리먼트 E의 자식 엘리먼트들을 불필요하게 여러 객체에 나누어 저장하지 않고, 엘리먼트 E와 대응되는 데이타베이스 객체의 컬럼 값으로 저장할 수 있으므로, 저장 및 검색에서 유리한 장점이 있다. 그리고 표준 언어인 XML Schema를 활용하여 매핑을 기술하므로, 상용 DBMS에 비해 사용자가 좀 더 쉽게 매핑을 기술할 수 있는 장점이 있다.

본 논문에서는 먼저 XML 문서 구조에서 관계형 및 객체관계형 데이타베이스 스키마로의 매핑에 대해 분석한다. 다음으로 분석된 매핑을 기술하는 방법을 표준 언어인 XML Schema를 활용하여 제안한다. 다음으로 사용자가 명시한 매핑 정보를 데이타베이스에 저장하는 저장 구조를 제안한다. 마지막으로 XML 문서를 데이타베이스에 저장할 때에 고려해야 되는 이슈들을 분석하고, 사용자가 정의한 매핑 정보에 의거하여 XML 문서를 관계형 및 객체관계형 데이타베이스에 저장하는 세부 알고리즘들을 제안한다.

본 논문의 구성은 다음과 같다. 제 2절에서는 관련 연구로서 XML 문서 저장에 관한 기존연구, XML Schema, 그리고 상용 DBMS의 XML 문서 저장 기능에 대해서 설명한다. 제 3절에서는 ODYSSEUS/XML Store의 설계로서 시스템 아키텍춰, 매핑 파일의 설계, 그리고 XML 문서 저장의 설계에 대해 설명한다. 제 4절에서는 제 3절의 설계를 바탕으로 ODYSSEUS/XML Store의 구현에 대해 설명한다. 마지막으로 제 5절에서는 결론을 내린다.

2. 관련 연구

본 절에서는 관련 연구로서 XML 문서 저장에 관한 기존 연구, XML Schema, 그리고 상용 DBMS의 XML 문서 저장 기능에 대해 설명한다. 제 2.1절에서는 XML 문서 저장에 관한 기존 연구를 두 가지 측면에서 분석하여 설명한다. 제 2.2절에서는 XML 문서 구조를 기술하는 표준 언어인 XML Schema에 대해 설명하고, 제 2.3절에서는 상용 DBMS에서 제공하는 XML 문서 저장 기능에 대해서 설명한다.

2.1 XML 문서 저장에 관한 기존 연구

본 절에서는 XML 문서 저장에 관한 기존 연구에 대해 설명한다. 제 2.1.1절에서는 하부 저장 시스템에 따른 기존 연구에 대해 설명하고, 제 2.1.2절에서는 매핑 정의 방법에 따른 기존 연구에 대해 설명한다.

2.1.1 하부 저장 시스템에 따른 기존 연구

XML 문서를 저장하기 위해 사용되는 하부 저장 시스템으로는 관계형 DBMS, 객체지향 DBMS, 독자적인 저장 시스템, 그리고 객체관계형 DBMS 등이 있다. 각 경우에 대해서 간략하게 특징을 설명한다.

먼저 관계형 DBMS를 하부 저장 시스템으로 사용하는 방법[4, 9]에서는 XML 문서의 구조를 관계형 데이타베이스의 테이블 형태로 구성하여 저장한다. 이 방법에서는 기존의 관계형 DBMS가 가지는 우수한 성능을 그대로 활용할 수 있는 장점이 있다. 그러나 XML 문서는 트리 구조를 가지고 관계형 데이타베이스의 테이블은 평면(flat) 구조를 가지므로, XML 문서의 트리 구조를 자연스럽게 표현하기가 힘든 단점과, 질의 처리 시에 고비용의 조인(join) 연산이 많이 발생하는 단점이 있다.

다음으로 객체지향 DBMS를 하부 저장 시스템으로 사용하는 방법[10]에서는 객체지향 개념을 이용하여 XML 문서의 트리 구조를 표현하므로, 관계형 DBMS에 비해 모델링 측면에서 보다 자연스러운 장점이 있다. 그러나, 참고문헌 [10]에서는 XML 문서 구조와 동일한 구조의 스키마만을 생성한다. 따라서, 입력 XML 문서의 구조가 저장이나 검색에 효율적인 형태가 아니라면 저장 및 검색 성능이 떨어지는 단점을 가진다.

다음으로 관계형 DBMS와 객체지향 DBMS를 사용하는 방법 이외에 독자적인 시스템을 개발하여 XML 문서를 저장하는 방법[11]이 있다. 이 방법에서는 XML

문서를 저장하기 위해 독자적으로 개발된 시스템을 사용하므로 XML 문서가 가지는 특징들을 잘 지원해줄 수 있으나, 현재까지 연구되어온 방대한 데이타베이스 관련 기술들을 활용하기 어려운 단점이 있다.

마지막으로 관계형 DBMS와 객체지향 DBMS의 장점들을 수용한 객체관계형 DBMS를 하부 저장 시스템으로 사용하는 방법[12]이 있다. 이 방법에서는 XML 문서의 트리 구조를 보다 자연스럽게 표현할 수 있는 장점, 대용량 데이타에 대한 복잡한 질의를 안정성 있게 처리할 수 있는 장점, 그리고 현재까지 연구되어온 방대한 데이타베이스의 관련 기술들을 활용할 수 있는 장점이 있다. 따라서 본 논문에서는 이러한 장점들을 가지는 객체관계형 DBMS를 하부 저장 시스템으로 사용한다. 참고문헌 [12]에서는 객체관계형 DBMS를 활용한 XML 색인에 대해 집중적으로 다루고 있다. 이에 반해 본 논문에서는 객체관계형 DBMS를 활용한 효율적인 XML 문서 저장에 대해 집중적으로 다룬다.

2.1.2 매핑 정의 방법에 따른 기존 연구

XML 문서 구조에서 데이타베이스 스키마로의 매핑을 정의하는 방법에는 시스템이 자동적으로 매핑을 정의하는 방법과, 사용자가 수동적으로 매핑을 정의하는 방법이 있다. 먼저 시스템이 자동적으로 정의하는 방법에 대해 설명하고, 다음으로 사용자가 수동적으로 정의하는 방법에 대해 설명한다.

시스템이 자동적으로 매핑을 정의하는 방법[4, 9, 10]에서는 시스템이 미리 정의된 몇 가지의 매핑 규칙(예: 인라이닝(inlining) 규칙[4])들을 적용하여 XML 문서 구조에 대응되는 데이타베이스 스키마를 생성한다. 이 방법에서는 매핑이 자동적으로 이루어지므로 사용자가 XML 문서 구조에서 데이타베이스 스키마로의 매핑에 대해 알지 못하더라도 쉽게 시스템을 이용할 수 있는 장점이 있다. 그러나, 미리 정의된 몇 가지의 매핑 규칙들만을 사용하여 매핑하므로, 사용자의 요구에 맞는 최적화된 매핑을 찾아내기 힘든 단점이 있다.

사용자가 수동적으로 매핑을 정의하는 방법[5, 6]에서는 XML 문서의 구조를 결정하는 엘리먼트와 애트리뷰트가 데이타베이스의 테이블과 컬럼으로 각각 어떻게 매핑되는지를 사용자가 직접 정의한다. 이 방법은 사용자가 자신의 요구에 맞는 최적화된 매핑을 정의할 수 있는 장점을 가진다. 본 논문에서는 상용 시스템에서와 마찬가지로 XML 문서 구조와 데이타베이스 스키마에 대한 지식이 있는 사용자가 시스템을 이용한다고 가정하여 수동적으로 매핑을 정의한다.

2.2 XML Schema

XML Schema는 XML 문서 구조를 기술하는 XML 형식의 표준 언어이다[13, 14, 15]. XML Schema는 XML 문서 구조를 기술하기 위하여 엘리먼트를 선언하는 *element* 엘리먼트, 애트리뷰트를 선언하는 *attribute* 엘리먼트, 엘리먼트의 타입을 정의하는 *complexType* 엘리먼트, 추가적인 정보를 기술하는 *annotation* 엘리먼트 등을 제공한다.

element 엘리먼트는 엘리먼트를 선언하기 위해 사용된다. *element* 엘리먼트는 엘리먼트의 이름과 타입을 나타내기 위해 *name*과 *type* 애트리뷰트를 가지며, 문서 상에서 엘리먼트가 나타날 수 있는 최소 횟수와 최대 횟수를 나타내기 위해 *minOccurs*와 *maxOccurs* 애트리뷰트를 가진다.

attribute 엘리먼트는 애트리뷰트를 선언하기 위해 사용된다. *attribute* 엘리먼트는 애트리뷰트의 이름과 타입을 나타내기 위해 *name*과 *type* 애트리뷰트를 가진다.

complexType 엘리먼트는 자식 엘리먼트나 애트리뷰트를 가지는 엘리먼트의 타입을 정의하기 위해 사용된다. XML Schema에서는 자식 엘리먼트나 애트리뷰트를 가지는 엘리먼트의 타입을 복합 타입(complex type)이라 하고, 자식 엘리먼트와 애트리뷰트를 모두 가지지 않는 엘리먼트의 타입과 애트리뷰트의 타입을 단순 타입(simple type)이라 한다.

annotation 엘리먼트는 XML Schema에 주석과 응용 프로그램에 필요한 정보를 추가적으로 기술하기 위해 사용된다. *annotation* 엘리먼트는 *documentation* 엘리먼트와 *appinfo* 엘리먼트를 자식 엘리먼트로 가지는데, *documentation* 엘리먼트는 주석을 기술하기 위해서 사용되고, *appinfo* 엘리먼트는 응용 프로그램에 필요한 정보를 기술하기 위해서 사용된다.

그림 1은 도서 정보에 관한 XML 문서 구조를 기술하는 XML Schema를 나타낸다. 그림에서 도서를 나타내는 *book* 엘리먼트는 도서명을 나타내는 *title* 엘리먼트와 도서의 저자를 나타내는 *author* 엘리먼트를 자식 엘리먼트로 가지고, 각 도서에 고유하게 부여되는 번호를 나타내는 *id* 애트리뷰트를 가진다. 그리고 *author* 엘리먼트는 저자의 이름을 나타내는 *name* 엘리먼트와 저자의 이메일 주소를 나타내는 *email* 엘리먼트를 자식 엘리먼트로 가진다.

그림 1에서 엘리먼트와 애트리뷰트는 각각 *element*와 *attribute* 엘리먼트를 사용하여 선언하고 있다. 그리고 자식 엘리먼트나 애트리뷰트를 가지는 *book*과 *author* 엘리먼트의 타입은 *complexType* 엘리먼트를 사용하여 정의하고 있으며, 자식 엘리먼트와 애트리뷰트를 모두

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
    <xsd:documentation>
      Book schema for www.book.com.
      Copyright 2001 www.book.com. All rights reserved.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="author" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="name" type="xsd:string"/>
              <xsd:element name="email" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:integer"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

그림 1 XML Schema의 예

가지지 않는 *title*, *name*, *email* 엘리먼트의 타입과 *id* 애트리뷰트의 타입은 integer, string과 같은 단순 타입을 사용하여 정의하고 있다. 마지막으로 저작권(copyright)과 같은 추가적인 정보는 *annotation* 엘리먼트를 사용하여 기술하고 있다.

2.3 상용 DBMS의 XML 문서 저장 기능

본 절에서는 상용 DBMS에서 제공하는 XML 문서 저장 기능에 대해 설명한다. 제 2.3.1절에서는 IBM DB2의 XML 문서 저장 기능에 대해 설명하고, 제 2.3.2절에서는 Microsoft SQL Server의 XML 문서 저장 기능에 대해 설명한다. 그리고 제 2.3.3절에서는 Oracle의 XML 문서 저장 기능에 대해 설명한다.

2.3.1 IBM DB2

IBM DB2는 XML Extender를 통하여 XML 문서를 데이타베이스에 저장하는 기능을 제공하고 있다[5]. XML Extender는 XML 문서 구조에서 데이타베이스 스키마로의 매핑을 기술하기 위해 자체 매핑 언어로서 DAD(Document Access Definition)를 제공하며, 사용자는 DAD를 사용하여 XML 문서 구조에서 데이타베이스 스키마로의 매핑을 기술한다.

DAD는 XML 문서 구조를 기술하기 위해 *element_node*와 *attribute_node* 엘리먼트를 제공하며, 데이타베이스로의 매핑을 기술하기 위해 *RDB_node* 엘리먼트를 제공한다. *element_node*와 *attribute_node* 엘리먼트는 각각 XML 문서의 엘리먼트와 애트리뷰트를 나타내며, 자식 엘리먼트로 *RDB_node* 엘리먼트를 가진다. 그리고 *RDB_node* 엘리먼트는 자식 엘리먼트로 테이블을 기술하는 *table* 엘리먼트, 컬럼을 기술하는 *column* 엘리먼트, 그리고 테이블 사이의 주 키-외래 키 릴레이션쉽을 기술하는 *condition* 엘리먼트를 가진다.

DAD에서 XML 문서의 엘리먼트는 데이타베이스 테이블이나 컬럼으로 매핑하고, XML 문서의 애트리뷰트는 데이타베이스 컬럼으로 매핑하도록 한다. 그리고 XML 문서의 엘리먼트 사이의 릴레이션쉽은 데이타베이스 테이블 사이의 주 키-외래 키 릴레이션쉽(primary key-foreign key relationship)으로 매핑하도록 한다.

DAD 매핑 파일을 작성하는 절차는 다음과 같다. 먼저 XML 문서 구조를 *element_node*와 *attribute_node* 엘리먼트를 사용하여 기술한다. 다음으로 엘리먼트와 애트리뷰트의 매핑 및 엘리먼트 사이의 릴레이션쉽의 매핑을 *RDB_node* 엘리먼트를 사용하여 기술한다. 이때 루트 엘리먼트에 해당하는 *element_node* 엘리먼트의 *RDB_node* 자식 엘리먼트에 테이블들 사이의 모든 주 키-외래 키 릴레이션쉽을 기술한다. 그리고 루트 엘리먼트를 제외한 *element_node* 엘리먼트와 *attribute_node* 엘리먼트의 *RDB_node* 자식 엘리먼트에 각 엘리먼트와 애트리뷰트가 매핑되는 테이블과 컬럼을 기술한다.

예를 들어, 그림 1과 같은 XML 문서 구조를 그림 2와 같은 관계형 데이타베이스 스키마로 매핑하기 위해 사용되는 DAD는 그림 3과 같다. 그림에서 보듯이 루트 엘리먼트인 *book* 엘리먼트의 *RDB_node* 자식 엘리먼트에 매핑에 사용되는 book과 author 테이블, 그리고 book과 author 테이블 사이의 주 키-외래 키 릴레이션쉽이 기술되어 있다. 다음으로 *id* 애트리뷰트, *title*, *name*, *email* 엘리먼트의 *RDB_node* 자식 엘리먼트에 각 엘리먼트와 애트리뷰트가 매핑되는 테이블과 컬럼이 기술되어 있다.

book 테이블

이름	id	title
타입	integer	varchar(100)

author 테이블

이름	bookId	name	email
타입	integer	varchar(100)	varchar(100)

그림 2 관계형 데이타베이스 스키마의 예

```
<element_node name="book">
    <RDB_node>
        <table name="book" key="id"/>                          ] book과 author
        <table name="author"/>                                 ] 엘리먼트 사이의
        <condition> book.id=author.bookId </condition>         ] 릴레이션쉽의 매핑 정보
    </RDB_node>
    <attribute_node name="id">
        <RDB_node>
            <table name="book"/>                               ] id
            <column name="id" type="integer"/>                 ] 애트리뷰트의
        </RDB_node>                                            ] 매핑 정보
    </attribute_node>
    <element_node name="title">
        <text_node>
            <RDB_node>
                <table name="book"/>                           ] title
                <column name="title" type="varchar(100)"/>     ] 엘리먼트의
            </RDB_node>                                        ] 매핑 정보
        </text_node>
    </element_node>
    <element_node name="author" multi_occurrence="YES">
        <element_node name="name">
            <text_node>
                <RDB_node>
                    <table name="author"/>                     ] name
                    <column name="name" type="varchar(100)"/>  ] 엘리먼트의
                </RDB_node>                                    ] 매핑 정보
            </text_node>
        </element_node>
        <element_node name="email">
            <text_node>
                <RDB_node>
                    <table name="author"/>                     ] email
                    <column name="email" type="varchar(100)"/> ] 엘리먼트의
                </RDB_node>                                    ] 매핑 정보
            </text_node>
        </element_node>
    </element_node>
</element_node>
```

그림 3 DAD의 예

그러나 XML Extender에서 매핑 언어로 사용되는 DAD는 관계형 DBMS에 기반한 매핑만을 제공하므로, 객체관계형 DBMS에서 제공하는 참조 타입과 컬렉션 타입을 활용한 매핑을 지원하지 못하는 단점이 있다. 또한 XML 문서 구조에서 데이타베이스 스키마로의 매핑을 기술하기 위해 독자적인 매핑 언어를 사용하므로, 사용자에게 부담을 주는 단점이 있다.

2.3.2 Microsoft SQL Server

Microsoft SQL Server는 XML Bulk Load 유틸리티를 통하여 XML 문서를 데이타베이스에 저장하는 기능을 제공하고 있다[6]. XML Bulk Load는 XML 문서 구조에서 데이타베이스 스키마로의 매핑을 기술하기 위해 자체 매핑 언어로서 annotated XDR Schema를 제공하며, 사용자는 annotated XDR Schema를 사용하여 XML 문서 구조에서 데이타베이스 스키마로의 매핑을 기술한다.

그러나, annotated XDR Schema도 IBM DB2 XML Extender의 DAD와 같이 관계형 DBMS에 기반한 매핑만을 제공하므로, 객체관계형 DBMS에서 제공하는 참조 타입과 컬렉션 타입을 활용한 매핑을 지원하지 못하는 단점이 있다. 또한 XML 문서 구조에서 데이타베이스 스키마로의 매핑를 기술하기 위해 독자적인 매핑 언어를 사용하므로, 사용자에게 부담을 주는 단점이 있다.

2.3.3 Oracle

Oracle은 XML SQL Utility(XSU)를 통하여 XML 문서를 데이타베이스에 저장하는 기능을 제공하고 있다[16]. XSU는 IBM DB2나 Microsoft SQL Server와 달리 사용자가 매핑을 기술하는 기능을 제공하지 않는다. 따라서, XSU를 통하여 XML 문서를 데이타베이스에 저장하기 위해서는 먼저 XML 문서 구조와 동일한 구조를 가지는 데이타베이스 스키마를 생성하여야 한다. 이때 복합 타입의 엘리먼트는 데이타베이스 스키마 상에서 복합 객체(complex object)로 나타내고, 단순 타입의 엘리먼트와 애트리뷰트는 복합 객체의 컬럼으로 나타낸다. 이 방법은 가장 단순한 형태의 자동 매핑 방법으로서, 입력 XML 문서의 구조가 저장이나 검색에 효율적인 형태가 아니라면 저장 및 검색 성능이 떨어지는 단점을 가진다.

한편, Oracle에서는 XML 문서 전체를 하나의 LOB (Large Object)로 저장하는 방법도 제공하고 있다. 이 방법은 저장 시 여러 테이블에 나누는 부담이 줄게 되어 저장 성능은 좋으나, LOB로 저장되므로 검색이나 변경이 용이하지 않다는 단점을 가진다.

3. ODYSSEUS/XMLStore의 설계

본 절에서는 오디세우스 객체관계형 DBMS를 위한 XML 저장 시스템인 ODYSSEUS/XMLStore의 설계에 대해 설명한다. 제 3.1절에서는 오디세우스 객체관계형 DBMS에 대해 소개하고, 제 3.2절에서는 ODYSSEUS/XMLStore의 아키텍춰에 대해 설명한다. 제 3.3절에서는 매핑 파일의 설계에 대해 설명하고, 제 3.4절에서는 XML 문서 저장의 설계에 대해 설명한다.

3.1 오디세우스 객체관계형 DBMS의 소개

본 절에서는 본 논문에서 구현한 XML 저장 시스템의 기반 시스템인 오디세우스(Object-oriented Database sYStEm for the Unix System)에 대해 소개한다. 오디세우스는 한국과학기술원 전산학과 데이타베이스 및 멀티미디어 연구실에서 개발한 객체관계형 DBMS이다[8].

객체관계형 DBMS는 관계형 DBMS를 확장하여 타입, 객체, 참조, 메소드, 상속 등의 객체지향 개념을 추가한 시스템이다[17]. 객체관계형 DBMS는 기존의 관계형 DBMS에 비해 풍부한 시멘틱스를 제공하므로, XML, CAD, CAM 등과 같은 복잡한 응용을 구현하는데 적합한 시스템으로 알려져 있다.

오디세우스는 위와 같은 객체관계형 DBMS의 기능을 지원하기 위한 질의 언어로서 OOSQL(Odysseus

Object-oriented SQL)을 제공하고 있다. OOSQL은 기존 SQL의 구문과 의미를 가능한 그대로 유지하면서 객체지향 개념을 지원할 수 있도록 확장한 객체지향 질의 언어이다[18]. OOSQL은 객체 식별자(object identifier: OID)와 함께 객체에 대한 참조 타입을 제공하며, 집합(set), 멀티셋(multiset), 리스트(list) 타입과 같은 다양한 컬렉션 타입을 제공한다.

3.2 시스템 아키텍춰

ODYSSEUS/XMLStore는 크게 매핑 파일 저장 모듈, XML 문서 저장 모듈, 그리고 오디세우스 객체관계형 DBMS로 구성된다. 그림 4는 ODYSSEUS/XMLStore의 아키텍춰를 나타내고 있다. 그림에서 보듯이 매핑 파일의 저장은 매핑 파일 저장 모듈을 통해 이루어지고, XML 문서의 저장은 XML 문서 저장 모듈을 통해서 이루어진다. 그리고 저장된 XML 문서의 검색, 변경, 삭제는 OOSQL 질의를 통해서 이루어진다. XML 문서의 검색, 변경, 삭제는 향후 XML 질의 언어인 XQuery[19] 질의를 통해 지원되도록 확장할 예정이다.

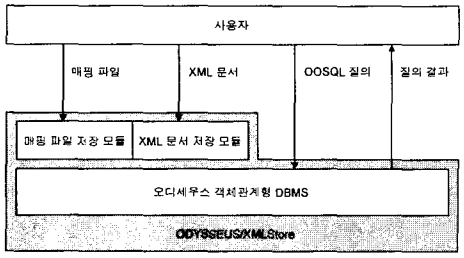


그림 4 ODYSSEUS/XMLStore의 아키텍춰

매핑 파일 저장 모듈은 매핑 파일을 파싱하여 매핑 파일에 기술된 XML 문서 구조에 대한 정보와 각 구조의 매핑에 대한 정보를 데이타베이스에 저장하는 기능을 담당하는 모듈이다. 매핑 파일에 기술된 정보들을 데이타베이스에 저장해 두면 매번 정보를 얻기 위해 매핑 파일을 파싱할 필요가 없고, 정보를 보다 효율적으로 검색하고 관리할 수 있는 장점이 있다. 데이타베이스에 저장된 정보는 XML 문서 저장 모듈과 향후 구현될 XQuery 질의 처리 모듈에서 사용된다.

XML 문서 저장 모듈은 XML 문서를 파싱하여 문서에 나타나는 엘리먼트와 애트리뷰트들의 값을 매핑에 따라 데이타베이스 테이블의 레코드로 변환하여 저장하는 기능을 담당하는 모듈이다. XML 문서 저장 모듈은 오디세우스의 OOSQL API를 통해 직접 데이타베이스에 저장하는 기능과 오디세우스의 벌크 로드 입력 포맷으로 변환하여 벌크 로딩하는 기능을 제공한다.

3.3 매핑 파일의 설계

본 절에서는 ODYSSEUS/XMLStore에서 사용되는 매핑 파일에 대해 설명한다. 매핑 파일은 XML 문서 구조에서 데이타베이스 스키마로의 매핑을 기술하는 파일이다. 제 3.3.1절에서는 XML 문서 구조에서 데이타베이스 스키마로의 매핑에 대해 분석한다. 제 3.3.2절에서는 분석에 기반하여 매핑 파일에서 관계형 데이타베이스 스키마로의 매핑을 기술하는 방법에 대해 설명하고, 제 3.3.3절에서는 객체관계형 데이타베이스 스키마로의 매핑을 기술하는 방법에 대해 설명한다.

3.3.1 XML 문서 구조에서 데이타베이스 스키마로의 매핑 분석

XML 문서 구조에서 관계형 및 객체관계형 데이타베이스 스키마로의 가능한 매핑은 그림 5와 같다. 그림에서 왼쪽 사각형은 XML 문서 구조의 구성 요소를 나타내고 오른쪽 사각형은 데이타베이스 구조의 구성 요소를 나타낸다. 그리고 화살표는 두 구조 사이에 가능한 매핑을 나타낸다. 화살표 중에서 점선으로 표시된 화살표는 두 구조 사이에 가능한 매핑이지만 ODYSSEUS/XMLStore에서는 허용하지 않는 매핑을 나타낸다. 그림에서 보듯이 XML 문서의 각 엘리먼트나 애트리뷰트는 데이타베이스의 테이블이나 컬럼으로 매핑될 수 있다(화살표 1~4). 그러나, XML 문서의 애트리뷰트를 데이타베이스의 테이블로 매핑하면, 질의 처리 시에 조인 연산이 불필요하게 발생되므로 본 논문에서는 이를 허용하지 않도록 매핑 파일을 설계하였다(화살표 2).

엘리먼트와 애트리뷰트 사이의 릴레이션쉽은 테이블과 컬럼 사이의 릴레이션쉽이나 테이블 사이의 렐레이션쉽으로 매핑될 수 있다(화살표 5, 7). 그러나, ODYSSEUS/XMLStore에서 애트리뷰트는 컬럼으로만 매핑되므로 엘리먼트와 애트리뷰트 사이의 릴레이션쉽은 테이블과 컬럼 사이의 릴레이션쉽으로만 매핑된다(화살표

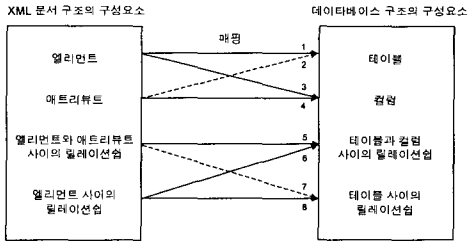


그림 5 XML 문서 구조에서 데이타베이스 스키마로의 매핑

5). 그리고 엘리먼트 사이의 릴레이션쉽은 테이블과 컬럼 사이의 릴레이션쉽이나 테이블 사이의 렐레이션쉽으로 매핑된다(화살표 6, 8).

그런데 릴레이션쉽의 매핑(화살표 5, 6, 8) 중에서 테이블과 컬럼 사이의 릴레이션쉽으로의 매핑(화살표 5, 6)은 기술할 필요가 없고, 테이블 사이의 릴레이션쉽으로의 매핑(화살표 8)만 기술하면 된다. 왜냐하면 테이블과 컬럼 사이의 릴레이션쉽은 데이타베이스 스키마에서 이미 유지하고 있으므로, 엘리먼트가 어떤 테이블로 매핑되고 애트리뷰트가 어떤 컬럼으로 매핑되는지를 기술하면, 엘리먼트와 애트리뷰트 사이의 릴레이션쉽이 어떤 테이블과 컬럼 사이의 릴레이션쉽으로 매핑(화살표 5)되는지를 알 수 있기 때문이다. 마찬가지로 엘리먼트들이 각각 어떤 테이블과 어떤 컬럼으로 매핑되는지를 기술하면, 엘리먼트 사이의 릴레이션쉽이 어떤 테이블과 컬럼 사이의 릴레이션쉽으로 매핑(화살표 6)되는지를 알 수 있다. 하지만 테이블 사이의 릴레이션쉽을 나타내기 위해서는 두 테이블을 연결하는 방법에 대한 정보가 추가적으로 필요하므로, 엘리먼트 사이의 릴레이션쉽에서 테이블 사이의 릴레이션쉽으로의 매핑(화살표 8)을 기술하기 위해서는 두 테이블을 연결하는 방법에 대한 정보를 추가적으로 기술하여야 한다.

이와 같은 매핑(화살표 1, 3, 4, 8에 해당)을 기술하기 위해서는 1) XML 문서 구조에 대한 정보, 2) 데이타베이스 스키마에 대한 정보, 3) XML 문서 구조에서 데이타베이스 스키마로의 매핑에 대한 정보가 필요하다. 매핑 파일에서 XML 문서 구조에 대한 정보는 XML Schema를 사용하여 기술하고, 데이타베이스 스키마에 대한 정보 및 XML 문서 구조에서 데이타베이스 스키마로의 매핑에 대한 정보는 XML Schema의 *appinfo* 엘리먼트 내에 이러한 정보들을 기술하는 새로운 엘리먼트들을 추가하여 기술한다. 이렇게 함으로써 사용자는 표준 XML Schema 언어의 기능만을 사용하여 매핑 정보를 기술할 수 있다.

3.3.2 관계형 데이타베이스 스키마로의 매핑 기술 방법

관계형 데이타베이스 스키마로의 매핑을 기술하는 매핑 파일에서 데이타베이스 스키마에 대한 정보는 데이타베이스의 테이블을 나타내는 *Table* 엘리먼트와 컬럼을 나타내는 *Column* 엘리먼트를 사용하여 기술한다. *Table* 엘리먼트는 테이블의 이름을 나타내는 *name* 애트리뷰트와 컬럼을 나타내는 *Column* 자식 엘리먼트를 가진다. 그리고 *Column* 엘리먼트는 컬럼의 이름과 타입을 나타내는 *name*과 *type* 애트리뷰트를 가진다.

XML 문서 구조에서 데이타베이스 스키마로의 매핑에 대한 정보에서 1) 엘리먼트와 애트리뷰트의 매핑(화살표 1, 3, 4)에 대한 정보는 해당 엘리먼트와 애트리뷰트를 선언하는 *element*와 *attribute* 엘리먼트의 자식 엘리먼트로 *Table*과 *Column* 엘리먼트를 사용하여 기술하며, 2) 엘리먼트 사이의 릴레이션쉽에서 테이블 사이의 릴레이션쉽으로의 매핑(화살표 8)에 대한 정보는 *Relationship* 엘리먼트를 사용하여 기술한다. *Relationship* 엘리먼트는 *parent*, *child*, *cardinality*, *isOrdered* 애트리뷰트를 가진다. *parent* 애트리뷰트는 부모 엘리먼트가 매핑된 테이블의 주 키를 나타내고, *child* 애트리뷰트는 자식 엘리먼트가 매핑된 테이블의 외래 키를 나타낸다. *cardinality* 애트리뷰트는 릴레이션쉽의 카디널리티(cardinality)를 나타내며, *isOrdered* 애트리뷰트는 릴레이션쉽을 맺을 때에 순서 정보를 유지할 지의 여부를 나타낸다.

예를 들어 그림 1과 같은 XML 문서 구조를 그림 2와 같은 관계형 데이타베이스 스키마로 매핑하기 위해 사용되는 매핑 파일은 그림 6과 같다. 이 매핑 파일은 그림 1의 XML Schema에 *appinfo* 엘리먼트를 추가하는 방식으로 작성되었다. 매핑 파일에서 데이타베이스 스키마에 대한 정보는 *Table*과 *Column* 엘리먼트를 사용하여 *schema* 엘리먼트의 *appinfo* 자식 엘리먼트 내에 기술되고 있다. 그리고 엘리먼트와 애트리뷰트의 매핑에 대한 정보는 각 *element* 엘리먼트와 *attribute* 엘리먼트의 *appinfo* 자식 엘리먼트 내에 *Table*과 *Column* 엘리먼트를 사용하여 기술되고 있다. 마지막으로 엘리먼트 사이의 릴레이션쉽에서 테이블 사이의 릴레이션쉽으로의 매핑에 대한 정보는 *schema* 엘리먼트의 *appinfo* 자식 엘리먼트 내에 *Relationship* 엘리먼트를 사용하여 기술되고 있다.

3.3.3 객체관계형 데이타베이스 스키마로의 매핑 기술 방법

객체관계형 데이타베이스 스키마로의 매핑을 기술하는 방법은 관계형 데이타베이스 스키마로의 매핑을 기술하는 방법과 유사하나, 객체관계형 DBMS에서 제공하는 참조 타입과 컬렉션 타입을 활용하여 매핑을 기술할 수 있다는 점이 다르다.

객체관계형 데이타베이스 스키마로의 매핑에서는 엘리먼트 사이의 릴레이션쉽에서 테이블 사이의 릴레이션쉽으로의 매핑(화살표 8)을 참조 타입과 컬렉션 타입을 조합하여 표현할 수 있다. 릴레이션쉽의 종류에는 카디널리티에 따라 참조 타입만으로 나타내는 일대일(one-to-one) 릴레이션쉽, 참조 타입의 컬렉션 타입으

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
    <xsd:appinfo>
      <Table name="book">                                          ┐
        <Column name="book.id" type="integer"/>                    │
        <Column name="book.title" type="varchar(100)"/>            │
      </Table>                                                     │ 데이타베이스
      <Table name="author">                                        │ 스키마에 대한 정보
        <Column name="author.bookId" type="integer"/>              │
        <Column name="author.name" type="varchar(100)"/>           │
        <Column name="author.email" type="varchar(100)"/>          │
      </Table>                                                     ┘
      <Relationship parent="book.id" child="author.bookId"         ┐ book과 author
                    cardinality="onetoMany" isOrdered="no"/>       ┘ 엘리먼트 사이의
    </xsd:appinfo>                                                   릴레이션쉽의 매핑 정보
  </xsd:annotation>

  <xsd:element name="book">
    <xsd:annotation>
      <xsd:appinfo>                                                ┐ book
        <Table name="book"/>                                       ┘ 엘리먼트의
      </xsd:appinfo>                                                 매핑 정보
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="title" type="xsd:string">
          <xsd:annotation>
            <xsd:appinfo>                                          ┐ title
              <Column name="book.title"/>                          ┘ 엘리먼트의
            </xsd:appinfo>                                           매핑 정보
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="author" minOccurs="0" maxOccurs="unbounded">
          <xsd:annotation>
            <xsd:appinfo>                                          ┐ author
              <Table name="author"/>                               ┘ 엘리먼트의
            </xsd:appinfo>                                           매핑 정보
          </xsd:annotation>
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="name" type="xsd:string">
                <xsd:annotation>
                  <xsd:appinfo>                                    ┐ name
                    <Column name="author.name"/>                   ┘ 엘리먼트의
                  </xsd:appinfo>                                     매핑 정보
                </xsd:annotation>
              </xsd:element>
              <xsd:element name="email" type="xsd:string">
                <xsd:annotation>
                  <xsd:appinfo>                                    ┐ email
                    <Column name="author.email"/>                  ┘ 엘리먼트의
                  </xsd:appinfo>                                     매핑 정보
                </xsd:annotation>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:integer">
        <xsd:annotation>
          <xsd:appinfo>                                            ┐ id
            <Column name="book.id"/>                               ┘ 애트리뷰트의
          </xsd:appinfo>                                             매핑 정보
        </xsd:annotation>
      </xsd:attribute>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

그림 6 관계형 데이타베이스 스키마로의 매핑을 기술한 매핑 파일의 예

로 나타내는 일대다(*one-to-many*) 릴레이션쉽이 있다. 이때, 순서 정보를 유지할 경우에는 컬렉션 타입으로 리 경우에는 집합(set) 타입을 사용한다. 순서 정보에 관해서는 제 3.4절에서 자세히 설명한다.

릴레이션쉽은 다시 릴레이션쉽의 방향에 따라 양방향 릴레이션쉽과 단방향 릴레이션쉽으로 구분될 수 있다. 양방향 릴레이션쉽에는 부모 엘리먼트에서 자식 엘리먼트으로 가는 릴레이션쉽과 자식 엘리먼트에서 부모 엘리먼트로 가는 릴레이션쉽이 있으며, 단방향 릴레이션쉽에는 부모 엘리먼트에서 자식 엘리먼트로 가는 릴레이

선쉽만 있다. 단방향 릴레이션쉽일 때는 매핑 파일에서 *Relationship* 엘리먼트의 *child* 애트리뷰트가 생략된다.

그림 7은 객체관계형 데이타베이스 스키마로의 매핑을 기술한 매핑 파일의 예를 나타내고 있다. 그림에서 앞에서 설명한 매핑 파일의 예와 동일한 부분은 생략되어 있다. 그림의 매핑 파일에서 *book* 엘리먼트와 *author* 엘리먼트 사이의 릴레이션쉽은 book 테이블과 author 테이블 사이의 릴레이션쉽으로 매핑되며, 릴레이션쉽이 일대다 릴레이션쉽이므로 참조 타입의 컬렉션 타입인 list(ref(author))으로 릴레이션쉽이 표현되어 있다. 이때 컬렉션 타입으로 리스트 타입을 사용하여 릴레이션쉽을 맺을 때에 순서 정보가 유지되도록 하고 있다. 또한, *book* 엘리먼트에서 *author* 엘리먼트로 가는 릴레이션쉽과, *author* 엘리먼트에서 *book* 엘리먼트로 가는 릴레이션쉽이 있으므로 릴레이션쉽이 양방향으로 정의되어 있다.

3.4 XML 문서 저장의 설계

본 절에서는 XML 문서 저장 시에 고려해야 할 사항들에 대해 설명하고, 각 사항들에 대한 설계 방안에 대해 설명한다. 고려해야 할 사항들 중에서 주 키 값의 자동 생성, 동일한 타입의 자식 엘리먼트들 사이의 순서 정보 유지, 컬렉션으로의 저장 등은 상용 DBMS에서는 제공되지 않지만 XML 문서를 저장할 때에 필요한 사항들이다.

첫째, XML 문서를 관계형 데이타베이스에 저장할 때에 주 키의 값을 자동적으로 생성해 주는 기능이 필요하다. 왜냐하면 XML 문서에 데이타베이스 테이블의 주 키로 매핑되는 엘리먼트나 애트리뷰트가 없는 경우에도 데이타베이스 테이블 사이의 주 키-외래 키 릴레이션쉽을 맺어 주어야 하기 때문이다. 이를 위해 ODYSSEUS/XMLStore에서는 매핑 파일에 SystemID 타입으로 명시된 컬럼에 대해서는 유일한 값을 할당해 주는 기능을 제공한다. 단, XML 문서를 객체관계형 데이타베이스에 저장할 때에는 각 객체에 고유하게 부여되는 OID를 사용하므로 별도의 주 키 값을 생성할 필요가 없다.

둘째, XML 문서를 저장할 때에 동일한 타입의 자식 엘리먼트들 사이의 순서 정보를 유지할 수 있어야 한다. 왜냐하면 XML 문서에서 동일한 타입의 자식 엘리먼트들 사이의 순서가 의미를 가지며, 이러한 순서 정보에 기반하여 질의식을 구성할 수 있기 때문이다. 예를 들어 그림 8의 XML 문서에서 *book* 엘리먼트는 두 개의 *author* 자식 엘리먼트로 가지는데, 이때 *author* 자식 엘리먼트들 사이의 순서가 의미를 가지며, *book* 엘리먼트의 두 번째 *author* 자식 엘리먼트를 검색하는 질의식을 구성할 수 있다. 동일한 타입의 자식 엘리먼트들 사이의 순서 정보는 부모 엘리먼트가 동일한 타입의 자식 엘리먼트들을 문서 상에서의 순서대로 참조하도록 저장함으로써 유지할 수 있다. 즉, 자식 엘리먼트를 저장하는 객체의 OID 값들을 문서 상에서의 순서대로 리스트에 저장하는 것이다.

이러한 순서 정보의 유지는 XML 문서를 객체관계형 데이타베이스에 저장할 때에 지원되며, 관계형 데이타베이스에 저장할 때에는 지원되지 않는다. 관계형 데이타베이스에 저장 시에 순서 정보를 유지하기 위해서는 각 자식 엘리먼트에 순서를 나타내는 번호를 부여하여 저장해야 하는데, 번호를 부여하여 저장하면 업데이트 시에

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
    <xsd:appinfo>
      <Table name="book">                                          ┐
        <Column name="book.authors" type="list(ref(author))"/>     │
        <Column name="book.id" type="integer"/>                    │
        <Column name="book.title" type="varchar(100)"/>            │  데이타베이스
      </Table>                                                     │  스키마에 대한 정보
      <Table name="author">                                        │
        <Column name="author.book" type="ref(book)"/>              │
        <Column name="author.name" type="varchar(100)"/>           │
        <Column name="author.email" type="varchar(100)"/>          │
      </Table>                                                     ┘
      <Relationship parent="book.authors" child="author.book"      ┐  book과 author
                    cardinality="onetoMany" isOrdered="yes"/>      ┘  엘리먼트 사이의
    </xsd:appinfo>                                                   릴레이션쉽의 매핑 정보
  </xsd:annotation>
  ...
</xsd:schema>
```

그림 7 객체관계형 데이타베이스 스키마로의 매핑을 기술한 매핑 파일의 예

```
<book>
  <title>XML Databases</title>
  <author>
    <name>Kevin Williams</name>
    <email>williams@wrox.com</email>
  </author>
  <author>
    <name>Johnny Papa</name>
    <email>papa@wrox.com</email>
    <email>papa@hotmail.com</email>
  </author>
</book>
```

그림 8 XML 문서의 예

번호 유지가 어려운 단점이 있다. 또한 관계형 DBMS 기반의 상용 시스템에서도 이를 지원하지 않고 있다.

셋째, 반복해서 나타나는 단순 타입의 엘리먼트들을 하나의 컬렉션으로 저장하는 기능이 필요하다. 이와 같이 하나의 컬렉션으로 저장하면 질의 처리 시에 비용이 많이 드는 조인(join) 연산의 횟수를 줄일 수 있는 장점이 있다. 예를 들어, 그림 8의 XML 문서에서 반복해서 나타나는 단순 타입의 *email* 엘리먼트들을 *author* 엘리먼트가 매핑된 테이블에 컬렉션으로 저장하면, *author* 엘리먼트의 *email* 자식 엘리먼트들을 검색하는 질의를 처리할 때 테이블 사이의 조인이 필요 없기 때문에 보다 효율적인 검색이 가능하다.

넷째, 메모리에 로딩할 수 없는 대용량의 XML 문서도 저장할 수 있어야 한다. 이를 위해 본 논문에서는 적은 양의 메모리만을 사용하여 XML 문서를 파싱할 수 있는 SAX(Simple API for XML)[20] API를 사용하여 XML 문서를 저장한다. SAX API는 XML 문서 전체를 메모리에 로딩하여 파싱하는 DOM(Document Object Model)[21] API와 달리 XML 문서를 점진적으로 파싱하는 특징을 가진다. 따라서 SAX API를 사용하면 대용량의 XML 문서도 저장할 수 있다.

4. ODYSSEUS/XMLStore의 구현

본 절에서는 제 3절의 설계를 바탕으로 ODYSSEUS/XMLStore의 구현에 대해 설명한다. 제 4.1절에서는 매핑 파일에 기술된 정보를 데이타베이스에 저장하는 저장 구조에 대해 설명하고, 제 4.2절에서는 XML 문서를 데이타베이스에 저장하는 방법에 대해 설명한다.

4.1 매핑 파일의 저장

본 절에서는 매핑 파일에 기술된 정보들을 데이타베이스에 저장하는 저장 구조에 대해 설명한다. 매핑 파일은 1) XML 문서 구조에 대한 정보, 2) 데이타베이스 스키마에 대한 정보, 3) XML 문서 구조에서 데이타베이스 스키마로의 매핑에 대한 정보로 구성된다. 이 중에서 데이타베이스 스키마에 대한 정보는 데이타베이스 시스템 카탈로그에 이미 저장되어 있기 때문에 XML 문서 구조에 대한 정보와 XML 문서 구조에서 데이타베이스 스키마로의 매핑에 대한 정보를 데이타베이스에 저장하면 된다.

본 논문에서는 XML 문서 구조에 대한 정보와 각 구조의 매핑에 대한 정보를 별도의 테이블에 저장하지 않고 하나의 테이블에 저장한다. 이와 같이 하나의 테이블에 저장하면 조인 연산의 횟수를 줄일 수 있는 장점이 있다. XML 문서 구조에 대한 정보와 각 구조에 대한 매핑 정보를 저장하는 테이블에는 1) 엘리먼트에 대한 정보와 엘리먼트의 매핑 정보를 저장하는 xmlSysElements 테이블, 2) 애트리뷰트에 대한 정보와 애트리뷰트의 매핑 정보를 저장하는 xmlSysAttributes 테이블, 3) 엘리먼트 사이의 릴레이션쉽에 대한 정보와 엘리먼트 사이의 릴레이션쉽의 매핑 정보를 저장하는 xmlSysRelationships 테이블이 있으며, 각 테이블의 구조는 그림 9와 같다.

xmlSysElements 테이블에서 엘리먼트에 대한 정보를 저장하는 컬럼으로는 elementId, elementName 컬럼이 있다. elementId 컬럼은 각 엘리먼트에 고유하게 부여되는 식별자를 저장하는 컬럼이고, elementName 컬럼은 엘리먼트의 이름을 저장하는 컬럼이다. xmlSysElements 테이블에서 엘리먼트의 매핑 정보를 저장하는 컬럼으로는 flag, tableId, columnNo 컬럼이 있다. flag 컬럼은 엘리먼트가 테이블로 매핑되는지 아니면 컬럼으로 매핑되는지에 대한 정보를 저장하는 컬럼이다. tableId 컬럼은 엘리먼트가 매핑된 테이블의 테이블 식별자를 저장하는 컬럼이고, columnNo 컬럼은 엘리먼트가 매핑된 컬럼의 컬럼 번호를 저장하는 컬럼이다. 테이블 식별자와 컬럼 번호는 시스템 카탈로그에 저장된 데이타베이스 스키마 정보를 접근하기 위해서 사용된다.

xmlSysAttributes 테이블에서 애트리뷰트에 대한 정보를 저장하는 컬럼으로는 elementId, attributeNo, attributeName 컬럼이 있다. elementId 컬럼은 애트리뷰트가 속한 엘리먼트의 식별자를 저장하는 컬럼이고, attributeNo 컬럼은 엘리먼트의 각 애트리뷰트에 고유하게 부여되는 번호를 저장하는 컬럼이다. attributeName 컬럼은 애트리뷰트의 이름을 저장하는 컬럼이다. xmlSysAttributes 테이블에서 애트리뷰트의 매핑 정보를 저장하는 컬럼으로는 tableId, columnNo 컬럼이 있다. tableId 컬럼은 애트리뷰트가 매핑된 테이블의 테이블 식별자를

xmlSysElements 테이블

이름	elementId	elementName	flag	tableId	columnNo
타입	integer	varchar	char	integer	integer
설명	엘리먼트 식별자	엘리먼트 이름		테이블 식별자	컬럼 번호

xmlSysAttributes 테이블

이름	elementId	attributeNo	attributeName	tableId	columnNo
타입	integer	integer	varchar	integer	integer
설명	엘리먼트 식별자	애트리뷰트 번호	애트리뷰트 이름	테이블 식별자	컬럼 번호

xmlSysRelationships 테이블

이름	parentId	childId	cardinality	flag	isOrdered	parentTableId	parentColumnNo	childTableId	childColumnNo
타입	integer	integer	char	char	char	integer	integer	integer	integer
설명	엘리먼트 식별자	엘리먼트 식별자	릴레이션쉽의 카디널리티		순서 정보의 유지 여부	테이블 식별자	컬럼 번호	테이블 식별자	컬럼 번호

그림 9 XML 문서 구조에 대한 정보와 각 구조에 대한 매핑 정보를 저장하는 테이블

저장하는 컬럼이고, columnNo 컬럼은 애트리뷰트가 매핑된 컬럼의 컬럼 번호를 저장하는 컬럼이다.

xmlSysRelationships 테이블에서 엘리먼트 사이의 릴레이션쉽에 대한 정보를 저장하는 컬럼으로는 parentId, childId, cardinality 컬럼이 있다. parentId 컬럼은 부모 엘리먼트의 식별자를 저장하는 컬럼이고, childId 컬럼은 자식 엘리먼트의 식별자를 저장하는 컬럼이다. 그리고 cardinality 컬럼은 릴레이션쉽의 카디널리티를 저장하는 컬럼이다. xmlSysRelationships 테이블에서 엘리먼트 사이의 릴레이션쉽의 매핑 정보를 저장하는 컬럼으로는 flag, isOrdered, parentTableId, parentColumnNo, childTableId, childColumnNo 컬럼이 있다. flag 컬럼은 자식 엘리먼트가 부모 엘리먼트가 매핑된 테이블의 컬럼으로 매핑되는지, 아니면 자식 엘리먼트와 부모 엘리먼트가 각각 테이블로 매핑되는지에 대한 정보를 저장하는 컬럼이다. isOrdered 컬럼은 순서 정보가 유지되는지에 대한 정보를 저장하는 컬럼이다. parentTableId와 parentColumnNo 컬럼은 각각 부모 엘리먼트가 매핑된 테이블의 테이블 식별자와 주 키인 컬럼의 컬럼 번호를 저장하는 컬럼이다. childTableId와 childColumnNo 컬럼은 각각 자식 엘리먼트가 매핑된 테이블의 테이블 식별자와 외래 키인 컬럼의 컬럼 번호를 저장하는 컬럼이다.

4.2 XML 문서의 저장

본 절에서는 XML 문서를 데이타베이스에 저장하는 방법에 대해 설명한다. 제 4.2.1절에서는 XML 문서를 관계형 데이타베이스에 저장하는 방법에 대해 설명하고, 제 4.2.2절에서는 XML 문서를 객체관계형 데이타베이스에 저장하는 방법에 대해 설명한다.

4.2.1 관계형 데이타베이스로의 XML 문서 저장

XML 문서를 관계형 데이타베이스에 저장하는 알고리즘 StoreXML_RDB는 그림 10과 같다. 알고리즘 StoreXML_RDB는 XML 문서의 각 엘리먼트를 하나씩 읽어 가며 이를 데이타베이스에 저장한다. 라인 3 ~ 10에서는 엘리먼트 E가 테이블로 매핑된 경우, E가 매핑된 테이블의 객체를 하나 생성하고, 부모 엘리먼트를 저장하는 객체와 주 키-외래 키 릴레이션쉽을 맺는다. 라인 11 ~ 12에서는 엘리먼트 E가 컬럼으로 매핑된 경우 E의 값을 매핑되는 객체의 컬럼에 저장한다. 라인 13 ~ 14에서는 엘리먼트 E에 속한 각 애트리뷰트의 값을 매핑되는 객체의 컬럼에 저장한다.

알고리즘 StoreXML_RDB는 제 3.4절에서 설계 시에 고려한 사항들 중에서 주 키 값의 자동 생성을 반영하고 있다. 즉, 라인 6 ~ 7에서 주 키로 매핑되는 엘리먼트나 애트리뷰트가 없으면 주 키 값을 생성하도록 하고 있다. ODYSSEUS/XMLStore에서는 주 키 값을 생성할 필요가 있는 컬럼에 대해 별도의 카운터(counter)를 유지하도록 하여 유일한 값을 생성할 수 있도록 한다.

4.2.2 객체관계형 데이타베이스로의 XML 문서 저장

XML 문서를 객체관계형 데이타베이스에 저장하는 알고리즘 StoreXML_ORDB는 그림 11과 같다. 알고리즘 StoreXML_ORDB는 XML 문서의 각 엘리먼트를 하나씩 읽어가며 이를 데이타베이스에 저장한다. 라인 3 ~ 15에서는 엘리먼트 E가 테이블로 매핑된 경우 E가 매핑된

```
Algorithm StoreXML_RDB
Input:
  XML 문서

 1: for (XML 문서에 속한 엘리먼트 E)
 2: {
 3:   if (E가 테이블로 매핑됨)
 4:   {
 5:     E가 매핑된 테이블의 객체 O를 생성;
 6:     if (객체 O의 주 키가 매핑되지 않음)
 7:       주 키 값을 생성하여 객체 O의 주 키 컬럼에 저장;
 8:     if (부모 엘리먼트를 저장하는 객체 O_p와 릴레이션쉽 R이 존재함)
 9:       객체 O_p의 주 키 값을 객체 O의 외래 키 컬럼에 저장;
10:   }
11:   else if (E가 컬럼으로 매핑됨)
12:     E의 값을 매핑되는 객체 O의 컬럼에 저장;
13:   for (E에 속한 애트리뷰트 A)
14:     A의 값을 매핑되는 객체 O의 컬럼에 저장;
15: }
```

그림 10 XML 문서를 관계형 데이타베이스에 저장하는 알고리즘

테이블의 객체를 하나 생성하고, 부모 엘리먼트를 저장하는 객체 O_p와 릴레이션쉽을 맺는다. 이때, 릴레이션쉽이 일대일 릴레이션쉽인 경우에는 자신의 OID를 자신을 참조하는 객체 O_p의 컬럼에 저장하고(라인 9), 일대다 릴레이션쉽인 경우에는 자신의 OID를 자신을 참조하는 객체 O_p의 컬럼의 원소로 저장한다(라인 11). 그리고 양방향 릴레이션쉽인 경우에는 객체 O_p의 OID를 객체 O_p를 참조하는 자신의 컬럼에 저장한다(라인 13). 라인 16 ~ 22에서는 엘리먼트 E가 컬럼으로 매핑되는 경우 E의 값을 매핑되는 객체의 컬럼에 저장한다. 라인 23 ~ 24에서는 엘리먼트 E에 속한 각 애트리뷰트의 값을 매핑되는 객체의 컬럼에 저장한다.

알고리즘 StoreXML_ORDB는 제 3.4절에서 설계 시에 고려한 사항들 중에서 동일한 타입의 자식 엘리먼트들 사이의 순서 정보 유지 및 컬렉션으로의 저장을 반영하고 있다. 첫째, 동일한 타입의 자식 엘리먼트들 사이의 순서 정보 유지를 위해, XML 문서를 순차적으로 파싱하면서 자식 엘리먼트를 저장하는 객체 O의 OID를 객체 O를 참조하는 객체 O_p의 컬럼의 원소로 저장한다(라인 11). 둘째, 컬렉션으로의 저장을 위해, 단순 타입의 컬렉션으로 매핑되는 엘리먼트인 경우 엘리먼트의 값을 컬렉션의 원소로 저장한다(라인 21).

5. 결 론

본 논문에서는 오디세우스 객체관계형 DBMS를 이용하여 XML 문서를 데이타베이스에 효율적으로 저장하고 관리하는 XML 저장 시스템인 ODYSSEUS/XML Store를 설계하고 구현하였다. ODYSSEUS/XMLStore는 표준에 기반한 매핑 기술 방법을 제공하며, XML 문서를 관계형 및 객체관계형 데이타베이스에 효과적으로 저장하는 방법을 제공한다.

```
Algorithm StoreXML_ORDB
Input:
  XML 문서

 1: for (XML 문서에 속한 엘리먼트 E)
 2: {
 3:   if (E가 테이블로 매핑됨)
 4:   {
 5:     E가 매핑된 테이블의 객체 O를 생성;
 6:     if (부모 엘리먼트를 저장하는 객체 O_p와 릴레이션쉽 R이 존재함)
 7:     {
 8:       if (R이 일대일 릴레이션쉽임)
 9:         객체 O의 OID를 객체 O를 참조하는 객체 O_p의 컬럼에 저장;
10:       else if (R이 일대다 릴레이션쉽임)
11:         객체 O의 OID를 객체 O를 참조하는 객체 O_p의 컬럼의 원소로 저장;
12:       if (R이 양방향 릴레이션쉽임)
13:         객체 O_p의 OID를 객체 O_p를 참조하는 객체 O의 컬럼에 저장;
14:     }
15:   }
16:   else if (E가 컬럼 C로 매핑됨)
17:   {
18:     if (C의 타입이 단순 타입임)
19:       E의 값을 매핑되는 객체 O의 컬럼에 저장;
20:     else if (C의 타입이 단순 타입의 컬렉션임)
21:       E의 값을 매핑되는 객체 O의 컬럼의 원소로 저장;
22:   }
23:   for (E에 속한 애트리뷰트 A)
24:     A의 값을 매핑되는 객체 O의 컬럼에 저장;
25: }
```

그림 11 XML 문서를 객체관계형 데이타베이스에 저장하는 알고리즘

본 논문의 공헌은 다음과 같다. 첫째, XML 문서 구조에서 데이타베이스 스키마로의 매핑에 대해 분석하였다. 이를 위해 XML 문서 구조의 각 구성 요소들이 데이타베이스 구조의 각 구성 요소들에 어떻게 매핑되는지를 분석하였다. 둘째, 분석된 매핑을 기술하는 방법을 표준 언어인 XML Schema를 활용하여 제안하였다. 본 논문에서 제안한 매핑 기술 방법은 XML 문서 구조에서 관계형 및 객체관계형 데이타베이스 스키마로의 매핑을 기술할 수 있는 특징을 가진다. 셋째, 사용자가 명시한 매핑 정보를 데이타베이스에 저장하는 저장 구조를 제안하였다. 이를 위해 매핑 정보를 저장할 수 있는 데이타베이스 테이블 구조를 제안하였다. 넷째, 사용자가 정의한 매핑 정보에 의거하여 XML 문서를 관계형

및 객체관계형 데이타베이스에 저장하는 방법을 제안하였다. 이를 위해 XML 문서를 데이타베이스에 저장할 때에 고려해야 되는 이슈들을 분석하였으며, XML 문서를 저장하는 세부 알고리즘들을 제안하였다.

참 고 문 헌

[1] Simon, H., *Strategic Analysis of XML for Web Application Development,* Computer Research Corp., 2000.

[2] Florescu, D. and Kossmann, D., A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database, Technical Report RR-3680, INRIA, May 1999.

[3] Florescu, D. and Kossmann, D., "Storing and Querying XML Data using an RDBMS," *IEEE Data Engineering Bulletin,* Vol. 22, No. 3, pp. 27-34, Sept. 1999.

[4] Shanmugasundaram, J. et al., "Relational Databases for Querying XML Documents: Limitations and Opportunities," In *Proc. the 25th Int'l Conf. on Very Large Data Bases,* pp. 302-314, Edinburgh, Scotland, UK, Sept. 1999.

[5] Cheng, J. and Xu J., "XML and DB2," In *Proc. the 16th Int'l Conf. on Data Engineering,* pp. 569-573, San Diego, California, USA, 2000.

[6] Microsoft Corp., Microsoft SQL Server 2000, 2000 (available from http://www.microsoft.com/sql/default.asp).

[7] Shanmugasundaram, J. et al., "A General Technique for Querying XML Documents Using a Relational Database System," *ACM SIGMOD RECORD,* Vol. 30, No. 3, pp. 20-26, Sept. 2001.

[8] 한욱신, 이민재, 이재길, 박상영, 황규영, "오디세우스 객체관계형 멀티미디어 DBMS의 아키텍처," 한국정보과학회 추계학술발표 논문집, pp. 45-47, 2000년 10월.

[9] Deutsch, A., Fernandez, M., and Suciu, D., "Storing Semistructured Data with STORED," In *Proc. Int'l Conf. on Management of Data,* ACM SIGMOD, pp. 431-442, Philadelphia, PA, May 1999.

[10] Christophides, V., Abiteboul, S., Cluet, S., and Scholl, M., "From Structured Documents to Novel Query Facilities," In *Proc. Int'l Conf. on Management of Data,* ACM SIGMOD, pp. 313-324, Minneapolis, Minnesota, May 1994.

[11] McHugh, J. et al., "Lore: A Database Management System for Semistructured Data," *ACM SIGMOD RECORD,* Vol. 26, No. 3, pp. 54-66, Sept. 1997.

[12] Shimura, T., Yoshikawa, M., and Uemura, S., "Storage and Retrieval of XML Documents Using Object-Relational Databases," In *Proc. the 10th Int'l Conf. on Database and Expert Systems Applications,* pp. 206-217, Florence, Italy, Sept. 1999.

[13] Biron, P. and Malhotra, A., XML Schema Part 2: Datatypes, May 2001 (available from http://www.w3.org/TR/xmlschema-2).

[14] Fallside, D., XML Schema Part 0: Primer, May 2001 (available from http://www.w3.org/TR/xmlschema-0).

[15] Thompson, H. et al., XML Schema Part 1: Structures, May 2001 (available from http://www.w3.org/TR/xmlschema-1).

[16] Chang, B. et al., *Oracle XML Handbook,* Oracle Press, 2000.

[17] Stonebraker, M. and Moore, D., *Object- Relational DBMSs: The Next Great Wave,* Morgan Kaufmann, 1999.

[18] 우준호, ODYSSEUS 객체지향 데이타베이스 시스템을 위한 질의 처리기의 설계 및 구현, 석사 학위 논문, KAIST 전산학과, 1995.

[19] Chamberlin, D. et al., XQuery 1.0: An XML Query Language, June 2001 (available from http://www.w3.org/TR/xquery).

[20] Megginson, D., Simple API for XML(SAX), May 2000 (available from http://sax.sourceforge.net).

[21] Apparao, V. et al., Document Object Model Level 1, Oct. 1998 (available from http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001).

이 기 훈

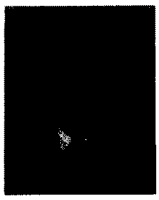
2000년 2월 한국과학기술원 전자전산학과 전산학전공 학사. 2002년 8월 한국과학기술원 전자전산학과 전산학전공 석사. 2002년 9월~현재 한국과학기술원 전자전산학과 전산학전공 박사 과정. 관심분야는 객체관계형 데이타베이스 시스템, XML 데이타베이스, 질의 최적화

한 욱 신

2003년 3월~현재 전임강사, 컴퓨터공학과, 경북대학교. 2002년 9월~2003년 2월 연구교수, 첨단정보기술연구센터, KAIST. 2001년 9월~2002년 8월 포스트닥, 첨단정보기술연구센터, KAIST. 1996년 3월~2001년 8월 Ph.D, 전산학과, KAIST. 1994년 3월~1996년 2월 MS, 전산학과, KAIST. 1990년 3월~1994년 2월 BS, 컴퓨터공학과, 경북대학교. 2002년 7월 방문연구원, 미국 HP Labs. 2001년 7월~2001년 8월 Visiting Scholar, 미국 HP Labs

김 민 수

1998년 2월 한국과학기술원 전자전산학과 전산학전공 학사. 2000년 2월 한국과학기술원 전자전산학과 전산학전공 석사 2000년 3월~현재 한국과학기술원 전자전산학과 전산학전공 박사 과정. 관심분야는 데이타베이스 시스템, XML



이 종 학

1982년 경북대학교 전자공학과(전자계산전공) 졸업(학사). 1984년 한국과학기술원 전산학과 졸업(공학석사). 1997년 한국과학기술원 전산학과 졸업(공학박사) 1991년 정보처리기술사. 1984년~1987년 금성통신(주) 부설연구소 주임연구원 1987년~1998년 한국통신 연구개발본부 선임연구원. 1998년~현재 대구가톨릭대학교 컴퓨터정보통신공학부 교수. 관심분야는 데이타베이스 시스템, 객체 데이타베이스, 트랜잭션 프로세싱, 지리정보 시스템 등

황 규 영

1973년 서울대학교 전자공학과 졸업(B.S). 1975년 한국과학기술원 전기 및 전자학과 졸업(M.S.). 1982년 Stanford University(M.S.). 1983년 Stanford University(Ph.D.). 1975년~1978년 국방과학연구소(ADD), 선임연구원. 1983년~1990년 IBM T.J. Watson Research Center, Research Staff Member. 1992년~1994년 한국정보과학회 데이타베이스 연구회(SIGDB) 운영위원장. 1995년 한국정보과학회 이사 겸 논문지 편집위원장. 1999년~2000년 한국정보과학회 부회장. Editior: the VLDB Journal, 1990년~현재 Editior: Distributed and Parallel Databases: An International Journal, 1991년~1995년 Editor: International Journal of Geographical Information Systems 1994년~현재 Associate Editor: IEEE Data Engineering Bulletin, 1990년~1993년 IEEE Transactions on Knowledge and Data Engineering, 2002년~현재. 1998년~2004년 Trustee, The VLDB Endowment. 1999년~2005년 Steering Committee Member, DASFAA. 1999년~현재 한국과학기술원 전자전산학과 전산학전공 교수 1990년~현재 첨단정보기술연구센터(과학재단 우수연구센터) 소장. 관심분야는 데이타베이스 시스템, 멀티미디어 GIS