XML Schema 기반의 객체지향 데이터베이스 응용 시스템 개발을 위한 3단계 통합 설계방법론

순천향대학교 공과대학 컴퓨터학부
최문영 · 주경수*
griffin@hyejeon.ac.kr · gsoojoo@sch.ac.kr*

Unified Design Methodology in Three Phases for XML Schema based on Object Oriented Database Application System Development

Choi Mun-Young · Joo Kyung-Soo*
Dept. of Computer Science and Engineering, College of Engineering SoonChunHyang University, Asan 336-745, Korea

요 약

XML이 기업간 데이터 교환뿐만 아니라 차세대 기업 정보시스템의 핵심기술로 떠오르면서 XML 데이터를 원형 그대로 저장하는 데이터베이스 관리시스템이 차세대 DBMS (DataBases Management System) 플랫폼으로 부상하고 있다. DBMS 업계는 XML이 현재 전자상거래 등 기업간 정보유통에 주로 적용되고 있으나 웹서비스가 본격 구현되면 기업간시스템도 XML 기반으로 점차 전환될 것으로 보고 XML 데이터의 효율적인 저장·관리하는 데이터베이스 개발에 경쟁적으로 나서고 있다.

현재 주류를 이루는 관계형 데이터베이스는 XML 데이터를 지원하기 위해 다른 포맷의 데이터를 XML로 변환하거나 반대의 작업을 하는 툴을 이용하고 있으나, DBMS 처리속도가 떨어지는 등의 문제가 있어 XML 데이터를 원형 그대로 저장하거나 불러올 수 있는 DBMS 수요가 늘어날 것으로 전망된다. 그러므로 본 논문에서는 XML 데이터를 효율적으로 처리할 수 있는 3단계를 통한 데이터베이스 스키마와 XML Schema로의 변환을 위한 통합 설계 방법론을 제안한다.

1. 서론

XML 데이터를 처리할 수 있는 데이터베이스 개발에 있어서 XML 데이터들을 어떻게 불러오고 저장하는지에 대한 처리는 중요하다. 데이터베이스 스키마는 데이터베이스에 대한 청사진이며 데이터베이스의 내용을 기술하는 것이다. 그러므로 데이터베이스 스키마를 어떻게 모델링 하느냐에 따라 사용자와 DBMS사이의 상호작용에 많은 영향을 미칠 수 있다.

본 논문에서 제안한 3단계 중, 개념적 모델링 단계는 새로운 건물을 설계하기 위하여 서로간의 대화를 여는 것과 같고, 논리적 모델링 단계는 계획에 대한 청사진을 개발하고 다이어그램을 설계하는 것과 같다. 그리고 물리적 모델링 단계는 건물의 돌을 쌓는 방법과 유사하다.

본 논문에서는 그림 1처럼 3단계의 모델링 방법을 제안하고 있다. 모델링 방법은 개념적 모델링, 논리적 모델링, 물리적 모델링으로 이루어져 있고 마지막 단계인 물리적 모델링에서는 논리적 모델링에서 제안한 클래스 다이어그램을 이용하여 XML 모델링과 데이터 모델링을 제안한다.

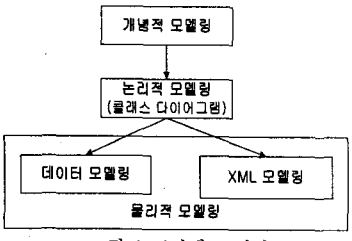

그림 1 3단계 모델링

2. 관련 연구

XML은 구조화된 정보를 포함하고 있는 문서들을 위한 마크업 언어이다. 구조화된 정보는 구체적인 내용과 그 내용이 수행해야 할 역할을 포함하고 있다[1]. XML Schema에 대한 연구는 W3C Recommendation으로 채택된 W3C XML Schema가 표준 XML Schema의 스펙이 될 가능성이 가장 높은 실정이다. W3C XML Schema는 XML DTD보다 다양한 데이터 타입을 정의할 수 있고 강력한 표현력을 이용하여 다양한 어플리케이션으로 사용하기에 편리한 장점을 가지고 있다. 또한 attribute, element, 사용자 정의 데이터 타입에 대해서 상속하는 메커니즘을 갖고 있다. 그러므로 XML 관련자들의 많은 관심을 끌고 있는 것은 사실이다[2,3].

UML를 이용한 XML 모델링에 대한 연구는 UML를 XML Schema로 변환할 때 UML 확장 메커니즘을 사용하여 XML 구조를 표현하였다[4,5]. 또한 UML 클래스 다이어그램을 XML DTD로 자동 변환하고 XML DTD를 XML Schema로 자동 변환하는 도구(tool)의 개발은 모델을 설계하는 자가 XML Schema 구문에 익숙하지 못해도 XML 모델링을 신속하게 하고 개별적으로 서로 다른 언어 또는 환경에서 부분적 모델 어휘들을 재 사용할 수 있게 하였다[6].

UML을 기반으로 하여 XML 형식으로 자동 변환하는 부분은 XMI(XML Metadata Interchange)라는 이름으로 표준화를 진행중이며 UML의 각종 다이어그램들이 XMI에서 정한 규정에 따라 결국은 XML로 표현되게 된다[7,8].

3. 개념적 모델링

개념적 모델링 그래프는 개념과 도메인의 관계를 나타내는 것으로 사용된다. 개념적 모델링을 하는 이유 중에 하나는 정확한

도메인 정보를 추출하여 데이터베이스를 개발하는데 사용하기 위해 있다. 개념과 서로간의 관계성은 그래프를 세분화하여 도메인의 근본적인 성질을 충분히 나타낸다.

개념과 관계성은 그래프의 노드로 표현되고 개념과 관계성의 특징은 모서리로 표현한다.

개념적 모델 그래프는 개념과 도메인의 관계성이 그래프로 표현되는 것이다. 도메인을 그래프로 표현하는 방법은 다음과 같다.

① 그래프는 노드, 모서리, 그리고 카디널리티(cardinality:계량수)의 집합으로 이루어져 있다.

② 두 개의 개념과 개념의 특성 그리고 도메인의 이진 관계는 모서리로 연결한다.

③ 카디널리티 한 쌍은 완전한 정수 모서리로 연결된다. 카디널리티는 정수대신에 조건 "Many(다수)"로 표현한다. 다수 카디널리티는 0 또는 더 많은 출현을 언급하는 것이다. 그리고 카디널리티 "one-or-more"는 "+"에 의해서 나타내어진다. 만약 카디널리티가 없으면 "1"로 가정한다.

3.1 개념적 모델링 그래프

그림 2처럼 7단계 처리를 통한 개념적 그래프를 모델링하는 알고리즘을 제안한다. 이 처리에서 하나의 이점은 데이터베이스 시스템에 대한 지식이 없이도 실행될 수 있다는 것이다.

① 도메인 개념과 복잡한 관계성을 리스트로 만든다. 개념은 실사회의 객체, 관계, 또는 사건이다.
② 도메인에 관계성을 기술한 단문을 생성하는 것으로 도메인 개념을 연결한다. 그 단문은 "사원은 이름을 가지고 있다", "단문은 글귤을 가지고 있다", 또는 "관리자는 사원을 관리한다"와 같은 도메인 개념의 특성을 기술하거나 관련 단어 또는 어구와 두 개의 도메인 개념을 연결한다. 관련 어구는 두 개의 도메인 개념에 대한 설명이다. 만약 관계성이 두 개념보다 더 많이 요구되면, 단계 ①의 리스트에 관계성을 추가할 수 있다.
③ 리스트로부터 도메인의 중요한 개념을 선택한다.
④ 개념 스키마에서 중요한 개념을 노드로 표현한다.
⑤ 그래프에서 모서리로 간단한 문장을 표현한다. 도메인 개념의 특성을 그리는 2가지 방법이 있다. 가장 완전한 방법은 타원형에 새로운 개념을 표현하는 것으로써 특성을 그리는 것이다. 그리고 지시된 모서리의 특성에 도메인 개념을 연결하고 특성에 대한 이름으로 모서리에 표현한다. 다음 방법으로 특성은 도메인이 개념적 모델링을 위해 명백한 특성을 표현하는 타원형을 생략하는 것이다. 두 개의 개념 화살표사이에 모서리를 그릴 수 있다. 예를 들면, 모서리에 "관리자 관리 사원"은 "관리자"에서 "사원"까지의 "관리"를 표현한다. 그 개념이 단계 ①의 리스트에 모두를 표현하기 위하여 모서리를 그릴 때 그래프에 추가적인 개념을 추가하는 것이 필요하다.
⑥ 모서리에 카디널리티 상태를 추가하고 도메인 그래프에 카디널리티 상태를 추가한다. 논리적 설계를 처리하는 동안에 카디널리티 상태를 추가 할 수도 있다. 모서리와 개념사이의 연결에서 단순한 관계에 일어날 수 있는 반복 횟수에 대한을 표현을 숫자로 표현한다. 카디널리티 "다수"는 "*"로 표현한다. 그리고 카디널리티 "1"은 생략할 수 있다. 만약 카디널리티 "zero-or- one"를 필요로 하면, 기호 "0"으로 나타낸다.
⑦ 도메인을 조사하여 스키마를 수정한다. 이 처리는 데이터베이스의 구현에 대한 근거로 개념 스키마를 설명하기 위한 예비 스키마가 될 것이다. 단계 ①의 리스트에 모두를 표현하기 위하여 모서리를 그릴 때 그래프에 추가적인 개념을 추가하는 것이 필요하다.

그림 2 개념적 모델링 알고리즘

예를 들어, 회사(Company)에 대한 도메인을 만들고 그래프를 그려보면 다음과 같다.

1. 도메인 개념과 복잡한 관계성의 리스트

- 회사 (Company)
- 부서 (Department)
- 부서 이름 (Department name)
- 사무실 (Office)
- 사무실 주소 (Office address)
- 사원 (Employee)
- 사원 이름 (Employee name)
- 사원 ID (EmployeeID)
- 사원 직위 (Employee title)
- 사원 연락정보 (Contact Information)
- 사원 연락정보 주소 (Contact Information address)
- 개인 기록 (Personal Record)
- 주민번호 (JuminID)

2. 도메인에 관계성을 기술한 단문을 생성하는 것으로 도메인 개념을 표현한다. 문장은 제한적인 단문 형식으로 도메인을 표현하기 때문에 문장 표현이 잘못될 수도 있다.

- 회사는 부서를 가지고 있다.
- 회사는 사무실을 가지고 있다.
- 회사는 ID를 가지고 있다.
- 부서는 이름을 가지고 있다.
- 사무실은 주소를 가지고 있다.
- 사무실은 ID를 가지고 있다.
- 부서는 부서원이 있다.
- 부서는 ID를 가지고 있다.
- 사원은 부서와 부서원을 관리하는 관리자가 있다.
- 사원은 이름을 가지고 있다.

- 사원은 ID를 가지고 있다.
- 사원은 직위를 가지고 있다.
- 사원 연락정보는 주소를 가지고 있다.
- 개인 기록은 주민번호를 가지고 있다.

3. 리스트로부터 도메인에 주요한 개념을 선택한다.

- 회사 (Company), 부서 (Department), 사무실 (Office), 사원 (Employee), 연락정보 (Contact Information), 개인기록 (Personal Record)

4. 그림 3처럼 개념 스키마의 노드로써 추가 되는 개념들을 그린다.
5. 그림 4처럼 그래프의 모서리로 단문을 표현한다. 각각의 문장은 하나의 모서리가 된다.
6. 그림 5처럼 모서리에 카디널리티 상태를 추가한다.

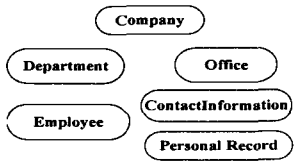

그림 3 회사 개념 그래프

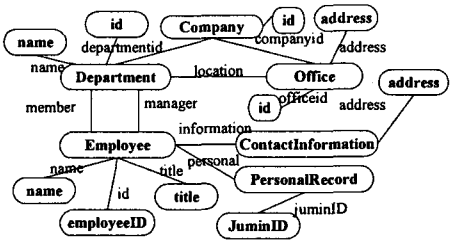

그림 4 회사 개념 스키마 그래프

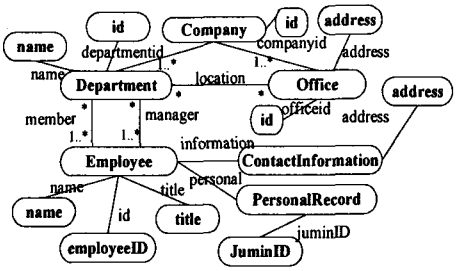

그림 5 카디널리티 상태를 가진 회사 스키마

7. 도메인을 조사하여 스키마를 수정한다. 추가적인 정보는 각각의 도메인에 추가할 수 있다.

4. 논리적 모델링

논리적 모델링 단계는 데이터베이스 관리 시스템의 데이터 모델을 위해 개념 스키마를 맵핑하는 과정이다. 데이터베이스 설계자가 개념적 모델링에서의 도메인을 파악하고 나면, 데이터베이스 관리 시스템의 데이터 모델은 유일한 모델링 언어로 사용될 것이다. 논리적 데이터 모델은 개념적 모델보다 기능성과 이상에 더 많이 제한적이다.

논리 스키마 형식의 작성은 개념 스키마 그래프로부터 객체 모델로 표현할 수 있다.

4.1 객체 모델

개념적 스키마 그래프에서 객체 모델까지 변환키 위한 알고리즘은 그림 6에 있다.

① 카디널리티의 특성에 의하여 개념적인 모델을 수정한다. 만약 특성을 알고 있으면 숫자의 범위 또는 세트를 사용하고 0 또는 다수의 상태를 구별한다.("*" 다수를 표현). 특성들은 "0", "0..1", "?", "opt", 또는 dashed-edge 선으로 표시될 수 있다. 만약 '이런' 일이 발생하면 특성의 여 러 가지 구성들을 설명하는 선을 다이어그램에 추가한다.
② 특성을 가지고 있는 각각의 개념은 클래스 다이어그램에서 클래스가 된다.
③ 카디널리티가 1보다 더 많거나 또는 추가적인 도메인 정보에 대한 연관을 생성하는 것을 제안한다면 개념에 대한 각각의 특성이 연관된다. 만약 그렇지 않으면, 카디널리티가 "1" 또는 "opt"이면 특성은 속성이 된다.
④ 특성을 가지고 있지 않은 개념은 속성 또는 연관된다. 정리안된 개념은 새로운 클래스, 내장 클래스, 또는 부분형이 될 수도 있다. 예를 들면, "Name"은 "String" 또는 문자열의 부분형으로 될 수 있다.
⑤ 다양한 구성의 개념에 대하여 일반화는 유용하다.

그림 6 논리적 모델링의 객체 모델링 알고리즘

개념 스키마 그래프를 객체 모델로 변환하기 위하여 중간 다이어그램을 생성하는 것은 클래스 다이어그램을 만드는데 매우 도움이 될 수 있다. 카디널리티를 추가한 그래프는 도메인 객체 다이어그램과 집합 트리를 만드는데 두 가지의 유용한 다이어그램이다. 도메인 객체 다이어그램의 중심적인 개념은 클래스 다이어그램에서의 클래스로써 그려진다.

집합 트리는 도메인 객체 다이어그램으로부터 생성된다. 하나의 도메인 객체가 존재를 위하여 또 다른 하나에 의존할 때, 그 관계성은 집합 다이어그램에서 변환된다. 집합은 도메인 객체와 다른 객체사이의 관계성이다. 객체의 설명은 다른 객체에 바탕을 둔다. 집합 트리는 개념적 다이어그램에서의 속성 역할을 하는 도메인 객체를 기술한다. 그러나 도메인 객체는 자신의 속성을 가지고 있기 때문에 각각 객체로서 모델링 되어야한다. 예를 들면, 부서에 관리자와 사원이 있으면 집합으로 변환시킨다. 만약 도메인 문장에 관리자와 사원이 없이 부서만 가지고 있는 것은 가능하다. 연관에서 집합을 구별하는 방법은 도메인 개념의 정의를 기록하는 것이다. 개념을 정의하는 것이 필요한 조건은 집합 관계성에 있다. 그리고 다른 관계성은 연관이다.

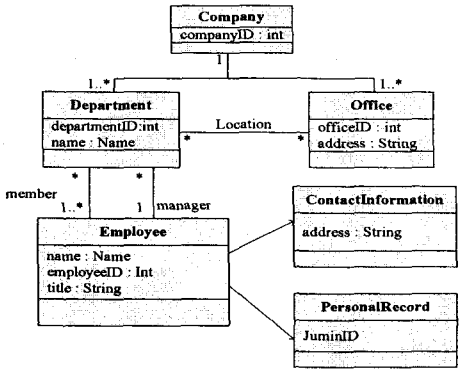

그림 7 회사 UML 클래스 다이어그램

객체 모델를 만드는 것은 개념적 다이어그램, 카디널리티 정보, 도메인 객체 다이어그램, 그리고 집합 트리를 필요로 한다. 객체 모델은 UML(Unified Modeling Language) 같은 객체를 위한 모델링 언어로 표현한다. 회사 UML 클래스 다이어그램은 그림 7에 주어진다.

5. 물리적 모델링

5.1 XML 모델링

UML 클래스를 사용하여 XML Schema로 변환하기 위한 규칙을 제안한다.

① empty elements 또는 empty attributes를 인정 안함. 그 대신 그들이 어떤 값을 갖는지 또는 값이 존재하지 않는지 둘 중의 하나를 분명히 함. 만일 element가 필수적이면 minOccurs="1", attribute가 필수적이면 use="required"로 설정함. 또한 그 값이 미심쩍을 때는 empty 문자열을 인정함. 그 때 element인 경우 minOccurs="0", attribute인 경우 use="optional"로 설정함.
② whitespace는 instance document의 크기를 크게 할 수 있음. 그 경우 파일 크기가 제한적임을 분명히 하기 위해 XML instance document를 수신하는 어플리케이션이 필요함. 또한 일반적으로 프로세싱 어플리케이션은 데이터구조의 크기에 관하여 약간의 제한을 가짐. 그렇기 때문에 개별적인 필드들은 길이를 제한함. 특히 keys일 경우 대부분의 데이터베이스들은 필드들의 일정한 형식에 입각하여 크기를 제한함.
③ 문자 집합과 필드크기를 제한할 경우 base types(string, decimal, integer)를 이용하여 속성 값의 데이터 형식을 지정함.
④ element 또는 attribute값의 유일함을 나타내기 위해 unique element를 사용.
⑤ document에서 2개의 location 연관시키기 위해 key/keyref element를 사용.
⑥ 만일 Schema가 클 경우 그룹 참조 방법을 사용하여 별개의 파일로 sections을 이동시킴. 이때 추상적 원리를 균일하게 적용하고 별개의 파일로 이동한 sections의 Schema 파일은 전역적 element를 갖지 않음. simple type정의, complex type정의, 그룹정의를 모두 포함하고 있는 파일을 가질 수 있고, 그 들을 관련된 block으로 분할 가능함.
⑦ namespace를 어디서 변경해도 instance document에서는 어떤 element 라도 사용 가능함. 그리고 그 location에서 허락한 자식 elements의 number와 namespace를 일일이 지정함. 새로운 namespace에서 element의 이름을 분명히 하기 위해 element ref=". . ." element를 상술함.
⑧ 만일 element 또는 attribute가 특정한 조건으로 존재하지 않으면, 어플리케이션은 기본값(default value)이 존재함을 표시 가능함. 그것은 element 또는 attribute선언에서 default attribute사용으로 지정 가능함.
⑨ instance document에서 유도형(derived type)이 치환되는 것을 방지하기 위해 Schema element에 blockDefault="#all" attribute를 포함시킴.
⑩ 현재 Schema에서 선언된 것으로부터 어떤 새로운 형식이 유도되는 것을 방지하기 위해 Schema element에 finalDefault= "#all" attribute를 포함시킴. 이것은 blockDefault보다 엄격한 제한 조건임.

그림 8 UML 기반의 XML 모델링을 위한 변환규칙

5.1.1 XML 모델링의 예

그림 7에서 "Company" 객체와 "Department", "Office" 객체는 관계성에서 집합 관계이며 이 의미는 "Company" 객체가 "Department", "Office"를 갖고 있어야 하며 따라서 "Department", "Office"쪽이 다중성 값이 1..*이라는 것을 보여주고 있다 반면 "Company" 객체의 다중성 값이 1..1인 것은 "Company"의 존재에 따라서 "Department", "Office"가 존재할 수 있음을 의미한다. 이를 XM모델링을 하기 위해서는 그림 8의 UML 기반의 XML 모델링을 위한 변환규칙 ①과 ④를 적용하여 XML Schema로 모델링 하면 그림 9와 같다

```
<element name="Company" type="Company">
 <complexType name="Company">
  <sequence>
    <element name="CompanyID" type="Int"/>
    <element name="department" minOccurs="1"/>
    <element name="Office" minOccurs="1"/>
    <complexType>
       <element REF="Department"/>
       <element REF="Office"/>
    </complexType> </sequence>
 </complexType></element>
<element name="Department" type="Department">
 <complexType name="Department">
  <element name="DepartmentID"type="Int" use= "required"/>
  <element name="name" type="Name" use="required"/>
 </complexType> </element>
<element name="Office" type="Office">
 <complexType name="Office">
  <element name="OfficeID" type="Int" use="required"/>
  <element name="Address" type="String" use= "required"/>
 </complexType> </element>
```

그림 9 Company 객체의 XML Schema모델링

그림 7에서 "Employee" 객체와 "ContactInfo- rmation", "PersonalRecord" 객체는 관계성에서 집합 관계이며 이 의미는 "Employee" 객체가 "Contact-Information", "Personal Record" 객체를 갖고 있어야 하며 "Employee" 객체의 존재 여부에 따라

"ContactInformation", "Personal Record" 객체가 존재할 수 있다. 이를 XML 모델링을 하기 위해서는 그림 8의 UML 기반의 XML 모델링을 위한 변환규칙 ①과 ④를 적용하여 XML Schema로 모델링 하면 그림 10과 같다.

```
<element name="Employee" type="Employee">
 <complexType name="Employee">
  <sequence>
   <element name="name" type="Name" minOccurs="1"/>
   <element name="employeeID" type="int" minOccurs="1"/>
   <element name="title" type="String" minOccurs="1"/>
   <complexType>
    <element REF="ContactInformation"/>
    <element REF="PersonalRecord"/>
   </complexType> </sequence>
 </complexType> </element>
<element name="ContactInformation">
 <complexType>
  <element name="address" type="string" use="required"/>
 </complexType> </element>
<element name="PersonalRecord">
 <complexType>
  <element name="JuminID" type="string" use="required"/>
 </complexType> </element>
```

그림 10 Employee객체의 XML Schema 모델링

그림 10에서 "Employee" 객체는 3개의 자식 객체와 2개의 참조 객체를 가지고 있다. 또한 "Employee" 객체에서 참조하는 "ContactInforma- tion", "PersonalRecord" 객체는 자식 객체를 갖고 있지 않으면서 "Employee" 객체에서 반드시 기술해야 하므로 use="required"로 설정하기 위해서 attribute로 정의하였다.

5.2 데이터 모델링

5.2.1 UML 클래스에서 객체지향 데이터베이스 스키마로의 변환규칙

객체지향 데이터베이스 스키마 도출을 위한 데이터 모델링의 변환 방법은 그림 11과 같은 방법을 사용한다.

① "영속(persistent)" 클래스는 영속 OODB 클래스임.
② 인터페이스가 "영속" 클래스 OODB 인터페이스로 실행되기 위한 인터페이스를 지원하는 언어(JAVA, ODL)나 또는 추상적인 베이스 클래스를 위해 오직 순수한 가상멤버 그리고 no data 멤버의 그 언어는 인터페이스를 갖지 않음.
③ type 분류자는 enum 또는 typedef로 표시함.
④ 클래스의 attribute는 적당한 type 변환과 제한으로 OODB 클래스의 attribute임.
⑤ 만약 nullable 태그가 나타나면 nullable 상수형을(nullable_short) 사용하고 바인딩을 지원하는 널값을(ODL) 사용함. 만약 그렇지 않으면, 그것을(C++, JAVA) 무시함.
⑥ 만약 attribute가 initializer를 가지면, 생성자에 초기화 코드를 추가함. 생성자 메소드의 어느 한쪽의 부분으로서 또는 C++ 멤버에 초기화는 목록에 나열함.
⑦ 서브 클래스는 클래스 선언에 슈퍼클래스 사양을 포함.
⑧ 연관 클래스는 연관 클래스의 attribute와 클래스를 생성함.
⑨ 명백한 객체 일치 (oid) 또는 후보 키 (alternate oid)가 만약 바인딩을 지원하 면 키의 선언을 명시함. 만약 지원하지 않으면, 객체의 집합에 억제를 점검하는 적당한 메소드를 공급함.
⑩ 각 명백한 억제에 대한 적당한 클래스에 메소드를 추가하고 안전함. 시스템은 메소드를 시스템이 그 억제가 만족할 만한 것을 요구할 때마다 호출함.
⑪ 적당한 객체 또는 연관된 Multiplicity로부터 유래한 컬렉션 타입에 대해서는 연관 클래스를 가지고 있지 않은 각 이진 연관을 위해 관계성을 생성함. 명백한 화살표들이 연관되어 있지 않다면 역 관계성들을 사용함.
⑫ 다른 클래스의 각 역할 링크를 위한 연관 클래스에 관계성을 생성함.
⑬ 코드를 생성하거나 또는 특별한 OODB의 요구처럼 어떠한 복합 집단화 연관을 위해 삭제를 전파하기 위하여 OODB 특징들을 사용함.
⑭ 3진으로 이루어진 연결을 위해 연관 클래스를 생성함. 그리고 연관 클래스에서 Multiplicity를 주었던 적당한 자료 type의 연합한 클래스까지 관계성을 생성함.

그림 11 UML 클래스에서 객체지향 데이터베이스 스키마로의 변환규칙

5.2.2 데이터 모델링의 예

그림 7의 "Company" 객체는 "Department", "Office" 객체와 one-to-many 관계이다. 하나의 회사 안에 부서와 사무실이 여러 개가 존재할 수 있다는 것을 의미한다. 그러므로, 그림 11의 변환 방법 ④, ⑦, ⑪에 의해서 그림 12와 "Company" 테이블로 변환된다.

```
Class Company(extent company) {
 attribute int        companyID;
 relationship   set<Department>;
 relationship   set<Office>; };
Class Department(exetent Departments) {
 attribute int        departmentID;
 attribute name       name;
 reationship          Company; };
Class Office(exetent Offices) {
 attribute int        officeID;
 attribute String     address;
 reationship          Company; };
```

그림 12 Company 테이블

그림 7의 "Employee" 객체는 "ContactInformati- on", "Personal Record" 객체와 집합 관계이며 그림 11의 변환 방법 ⑧, ⑪에 의해서 그림 13의 "Employee" 테이블로 변환된다.

```
Class Employee(extent Employee) {
 attribute int        EmployeeID;
 attribute name       name;
 attribute String     title;
 relationship   set<ContactInformation>;
 relationship   set<PersonalRecord>; };
```

```
Class ContactInformation(exetent ContactInformations) {
 attribute String     address;
 reationship          Employee; };
Class PersonalRecord(exetent PersonalRecords) {
 attribute int        JuminID;
 reationship          Employee; };
```

그림 13 Employee 테이블

6. 결 론

웹 상의 XML 데이터를 처리하기 위해서 본 논문에서는 객체지향 데이터베이스와 XML Schema를 설계하는 개념적, 논리적, 물리적 모델링 방법을 제안했다. 본 논문에서의 변환은 기본적인 자료 유형을 정의하는 다른 애플리케이션 영역에 쉽게 적용시킬 수 있다.

향후에 XML 개발은 스키마 선언과 기본적인 자료 유형의 전송에 대하여 개선될 것이다. UML의 직관적이고 시각적인 형식은 XML의 한정된 선언을 확장한다. 그리고 UML과 XML의 조합은 애플리케이션 사이의 자료 교환을 단순화할 것이다.

참고문헌

[1] What is XML?,http://www.xml.com/pub/a/98/10/guide1.html#AEN58.

[2] Migrating from XML DTD to XML-Schema using UML, http://www.rational.com/produ- cts/ whitepapers/412.jsp.

[3] Cheng, J., Xu, J.,: XML and DB2, In Sixteenth International Conference on Data Engineering (ICDE'00), 28 February-3 March 2000, San Diego, IEEE Computer Society Press, Los Alamitos, CA, 2000, 569-576.

[4] W3C Recommendation, http://www.w3.org/TR /2001/REC-xmlsc-hema-0-20010502/,05/02/2001.

[5] UML for XML Schema Mapping Specification, http://www.Rational.com/media/uml/resources/media/uml_xml-schema33.doc,12/08/99.

[6] Modeling XML vocabularies with UML, http://www.xml.com/p-pub/a/2001/09/19/uml.html, 09/19/2001.

[7] Bray, T., J. Paoli, and C.M. Sperberg McQ ueen, eds. Extensible Markup Language (XML) 1.0. REC-xml 19980210. W3C Recommendation 1998. http://www. w3.org /TR/REC-xml.

[8] Iyengar, S. and S.A. Brodsky, eds. XML Metadata Interchange (XMI). Proposal to the OMG Object Analysis & Design Task Force RFP 3: Stream-based Model Int erchange Format (SMIF) 1998, Object Management Group. http://www.omg.org.