객체모델을 이용한 XML DTD의 ORDB 스키마로의 변환

이상태*, 주경수**
*신성대학 컴퓨터응용계열
**순천향대학교 컴퓨터학부
*e-mail:stlee@shinsung.ac.kr
**e-mail:gsoojoo@asan.sch.ac.kr

Transformation from XML DTD to ORDB Schema using Object Model

Lee Sang-Tae*, Joo Kyung-Soo**
*Dept of Computer Science, Shinsung College
**Dept of Computer Science, SoonChunHyung University

요 약

XML은 웹에서 데이터 교환을 위한 마크업 언어로 반-구조화된 정보나 구조화된 정보를 교환하고 저장할 수 있으며, 표준언어로 채택되어 가고 있고, 강력한 표현력을 가지고 있기 때문에 다양한 애플리케이션을 가능하게 한다. 문서와 객체가 구조화 될 수 있는 방법을 제공하는 XML은 계층적 구조로 이루어진다. ORDB에서는 스키마 객체들이 계층적 구조로 구성되어 하나의 큰 객체를 이룬다. XML과 ORDB 사이에 단순한 XML 문서는 직접 변환이 가능하나 복잡한 XML 문서를 다룰 때는 객체 기반 변환 방법으로 처리되어야 한다. 따라서 본 논문에서는 XML DTD를 객체로 변환하고 변환된 객체를 ORDB 스키마로 변환하는 방법에 대하여 연구하였다.

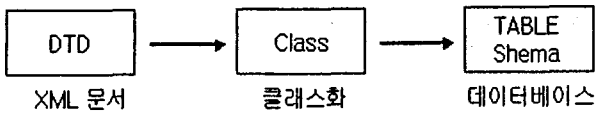
1. 서론

XML은 웹에서 데이터 교환을 위한 표준 마크업 언어이다. 마크업이란 HTML 문서에서 사용되던 "<"와 ">"로 둘러싸인 태그들을 의미한다. XML에서도 태그는 시작 태그와 끝 태그로 구성되어 있는데 HTML과 다른 점은 HTML의 태그들은 HTML 문서가 어떻게 화면에 보여질 것인가를 나타내는 반면에, XML의 태그들은 문서의 구조나 데이터의 의미를 나타내기 위해서 사용된다. 따라서 문서를 작성할 때 문서의 구조 혹은 데이터의 의미에 따라 사용자가 필요한 태그들을 자유롭게 정의할 수 있다. DTD는 XML 문서 내에서 사용 가능한 요소를 관리하고, 사용할 수 있는 각 요소형의 내용과 특성을 지정하는 규칙들을 담고 있다. 본 논문에서는 이 DTD에 정의한 요소와 특성들을 객체화하여 ORDB 스키마 객체모델로 변환하는 기법들을 다룬다. 본 논문의 제2절에서는 객체 기반 변환 방법에 대하여 설명하고, 제3절에서는 XML DTD에서 객체로 변환하는 방법을 다루며, 4절에서는 객체를 ORDB 스키마로 변환하는 방법을 다루고, 마지막으로 결론을 기술한다.

본 연구는 정보통신부의 ITRC 사업에 의해 수행된 것임

2. 객체 기반 변환

XML DTD를 ORDB 스키마로 변환하고자 할 때 단순한 XML 문서는 직접 변환이 가능하나 복잡한 XML 문서를 다룰 때는 객체 기반 변환 방법으로 처리되어야 한다. 따라서 테이블 기반 변환 방법은 복잡한 XML 문서를 변환하기가 불가능하므로 이 단점을 개선시키기 위해 중간에 객체를 이용하여 ORDB 스키마로 변환한다. [그림 1]



[그림 1] 객체 기반 매핑

3. XML DTD의 객체 기반 매핑

3.1 요소

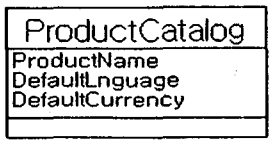
HTML은 태그가 바로 스타일 시트로 사용된다. 시작 태그는 성질을 열고, 끝 태그는 다시 닫는다. XML에서는 시작 태그와 끝 태그를 컨테이너로 사용한다. 시작 태그의 내용과 끝 태그가 함께 하나의 요소를 이룬다. 요소는 하나의 루트 요소만 가져야 하며, 다른 태그는 모두 요소 안에 확실하게 중첩되어야 한다. 이것은 한 요소가 다른 요소들을 포함할 경우, 그 요소들은 한 요소 안에 들어가야 한다는 뜻이다. 요소 타입은 두 개의 타입으로 분류하는데 PCDATA만 가진 요소의 타입을 단순 요소 타입이라고 하며, PCDATA을 제외한 모든 요소의 타입을 복합 요소 타입이라고 한다. 다시 말해 복합 요소 타입은 요소의 자식 요소, 혼합 요소, 요소의 특성들이다.

단순 요소 타입을 객체로 변환할 때는 클래스의 속성으로 변환된다. 다음 예제는 단순 요소 타입의 예를 보여준다.[그림 2]

```
<!ELEMENT ProductCatalog (ProductName,
          DefaultLanguage, DefaultCurrency)>
<!ELEMENT ProductName (#PCDATA)>
<!ELEMENT DefaultLanguage (#PCDATA)>
<!ELEMENT DefaultCurrency (#PCDATA)>
```

[그림 2] 단순 요소 타입

[그림 2]은 클래스 속성으로 변환하면 다음과 같다. [그림 3]



[그림 3] 단순 요소 타입을 객체로 변환

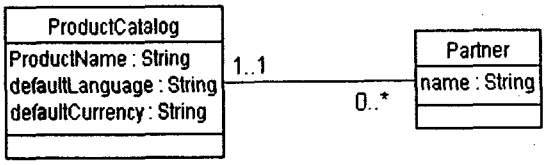
[그림 3]에서 보면 클래스 이름은 [그림 2]에 있는 요소가 포함하고 있는 부모 요소가 클래스 이름으로 지정하고, 데이터 형은 작성자가 요소의 의미에 따라 정수형, 문자형 등으로 지정한다.

복합 요소 타입은 단순 요소 타입을 제외한 모든 요소를 말하는데 다음과 같다.[그림 4]

```
<!ELEMENT ProductCatalog (ProductName,
          DefaultLanguage, DefaultCurrency,
                                Partner*)>
<!ELEMENT ProductName (#PCDATA)>
<!ELEMENT DefaultLanguage (#PCDATA)>
<!ELEMENT DefaultCurrency (#PCDATA)>
<!ELEMENT Partner (Name)>
```

[그림 4] 복합 요소 타입

[그림 4]에서는 루트 요소가 ProductCatalog이고 그에 자식 요소는 ProductName, DefaultLanguage, DefaultCurrency이다. 따라서 객체로 변환하면 다음과 같다.[그림 5]



[그림 5] 복합 요소 타입의 객체로 변환

[그림 5]은 객체로 변환된 클래스이다. 클래스 이름은 ProductCatalog이다. 여기서 데이터 형은 [그림 2]에서 요소의 의미가 문자형으로 사용되므로 객체로 변환할 때 String형으로 변환된다.

3.2 특성

모든 요소는 특성을 가질 수 있다. 특성은 이름/값의 형태를 가진다. 특성은 요소에 포함되며, 특성에 부여된 값을 통해서 그 요소에 어떠한 특징을 제공하게 된다.

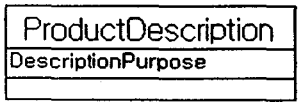
특성은 XML 파서가 애플리케이션에게 보내는 값들을 뜻하는데 요소의 내용과는 구별되는 값이므로 요소와 마찬가지고 특성도 단일 값을 가진 특성과 다중 값을 가진 특성으로 분류한다.

단일 값을 가진 특성은 문자열 특성(CDATA), 토큰 특성(ID, IDREF, NMTOKEN, ENTITY, NOTATION), 열거형 특성이며, 객체로 변환될 때 클래스의 속성으로 변환한다.[그림 6]

```
<!ELEMENT ProductDescription>
<!ATTLIST ProductDescription
   DescriptionPurpose CDATA #IMPLIED>
```

[그림 6] 단일 값을 가진 특성

[그림 6]에서 보면 클래스 이름이 ProductDescription이고 클래스의 속성은 DescriptionPurpose이다.[그림 7]



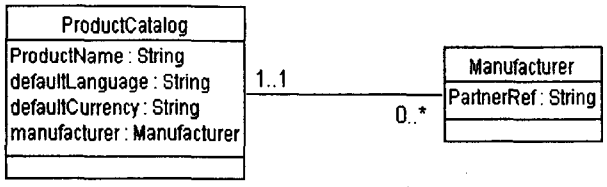
[그림 7] 단일 값을 가진 특성을 객체로 변환

다중 값을 가진 특성은 특성에 IDREFS, NMTOKENS, ENTITIES으로 선언된 것이다. 이것들을 객체로 변환할 때 값이 하나 이상으로 들어가기 때문에 별도의 객체로 만들 수 있다. 다음 그림은 특성을 포함한 DTD를 보여준다.[그림 8]

```
<!ELEMENT ProductCatalog (ProductName,
     DefaultLanguage,DefaultCurrency,
                          Manufacturer)>
<!ELEMENT ProductName (#PCDATA)>
<!ELEMENT DefaultLanguage (#PCDATA)>
<!ELEMENT DefaultCurrency (#PCDATA)>
<!ELEMENT Manufacturer >
<!ATTLIST Manufacturer
        PartnerRef IDREFS #IMPLIED>
```

[그림 8] 다중 값을 가진 특성

[그림 9]에서 보면 요소의 특성인 PartnerRef은 IDREFS로 선언되어 있다. IDREF는 단일 값을 가지고 있는 토큰 특성 ID와 의미는 같으나 ID는 요소에 대한 ID로 같은 문서에서 같은 ID 특성 값을 가진 두 개의 요소를 가질 수 없으며, IDREF는 ID를 가리키는 포인터이고, IDREFS는 하나 이상의 IDREF값으로 이루어진다. 따라서 객체로 만들 수 있다.



[그림 9] 복합 요소 타입의 객체로 변환

[그림 9]에서는 두 개의 클래스로 이루어지며, 한 클래스는 부모 클래스이고 다른 하나는 자식 클래스로 구분된다. [그림 8]에서 부모 요소는 ProductCatalog이고 자식 요소는 Manufacturer이므로 객체로 변환하면 [그림 9]과 같다. 따라서 두 개의 클래스를 연결하려면 참조형으로 연결하면 된다.

3.3 복합 내용 모델 변환

(1) 순차

DTD에 있는 요소와 요소에 속하는 자식 요소들을 순차적으로 객체로 변환하면, 자식 클래스와 부모 클래스간의 연결을 참조형으로 변환할 수 있다. 테이블 스키마로 변환할 때 그 테이블이 속하는 컬럼들은 반드시 NOT NULL 제약조건이 따른다.

(2) 선택

DTD 요소 안에 자식 요소들이 있는데 그 중에 하나만 선택할 경우에도 객체로 변환될 수 있고, 변환된 객체는 다시 ORDB 스키마 객체 모델로 변환된다.

(3) 반복

요소 안에 자식 요소가 중복된 경우 객체화로 변환할 때 요소 안에 중복된 같은 요소들이 클래스 속성으로 변환하는 경우 배열 형태로 변환된다. 이렇게 변환된 클래스 속성은 별도의 테이블로 변환된다. 하나 이상의 요소들이 존재한다는 의미로 클래스 속성의 데이터형을 String[]으로 변환한다. 즉 작성자가 알 수 없는 만큼 요소가 존재한다는 의미로 별도의 테이블로 변환한다.

(4) 서브그룹

<!ELEMENT A (B, (C | D))> 와 같은 경우를 말하며, 이 요소는 <!ELEMENT A (B, C))> 와 <!ELEMENT A (B, D))> 요소로 분류된다. 따라서 두 개의 객체로 변환할 수 있다.

3.4 혼합 내용 모델 변환

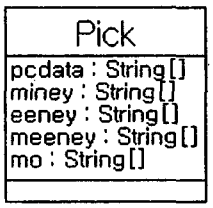
텍스트나 요소, 그 둘을 모두 포함할 수 있는 요소들을 혼합 내용 모델이라고 하며, XML 프로세서는 공백, 탭 등의 PCDATA와 요소 내용간의 구분이 어렵다. 왜냐하면 끝 태그와 다음의 시작 태그 사이에 공백이 있으면 불분명하게 된다. 혼합 내용

모델에서 선택이 가능한 그룹은 하나이고, #PCDATA로 시작하며, 그 뒤에는 혼합 내용의 연산자수를 나타내는 타입 순서로 되며, 각각은 한 번만 선언된다. #PCDATA만 유일한 옵션이며, "*"는 반드시 괄호를 닫은 바로 뒤에 와야 한다.
다음 DTD 예를 들어보자.[그림 10]

```
<!ELEMENT pick (#PCDATA | eeney | meeney
                | miney | mo)* )>
<!ELEMENT eeney (#PCDATA)>
<!ELEMENT meeney(#PCDATA)>
<!ELEMENT miney(#PCDATA)>
<!ELEMENT mo(#PCDATA)>
```

[그림 10] 혼합 내용 모델 DTD

위에 있는 DTD는 혼합 내용 모델이다. pick 요소는 루트 요소이고 자식 요소들은 반복적으로 이루어진다. 따라서 루트 요소는 클래스 이름으로 변환하고 자식 요소는 클래스의 속성으로 변환이 된다. 그리고 반복의 의미가 있어 데이터 형은 배열로 선언이 된다. 이 DTD를 객체로 변환하면 다음과 같다.[그림 11]

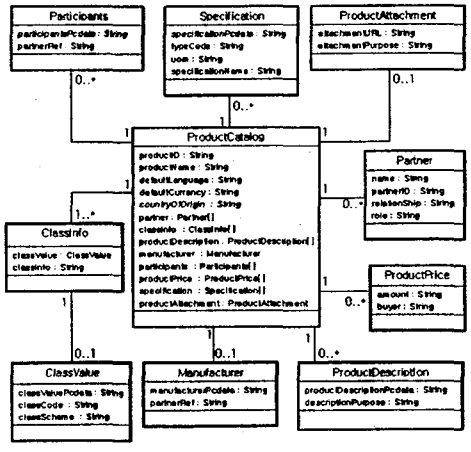


[그림 11] 혼합 내용 모델 DTD를 객체화로 변환

DTD에서 "*" 연산자는 요소나 요소 그룹이 생략하거나 한 번 이상 나타날 수 있으므로 설계하는 작성자는 몇 개가 나타나는지 알 수 없기 때문에 String[] 형으로 선언한다.
위에 설명된 내용에 따라 아래의 XML DTD[그림 12]는 객체모델[그림 13]로 변환될 수 있다.

```
<?xml version="1.0" encoding="EUC-KR"?>
<!ELEMENT ProductCatalog (ProductName,
        DefaultLanguage, DefaultCurrency, Partner*,
      ClassInfo+, ProductDescription*,
CountryOfOrigin?,          Manufacturer?, Participants*,
ProductPrice*,              Specification*,
ProductAttachment?)>
<!ATTLIST ProductCatalog  ProductID ID #REQUIRED >
<!ELEMENT ProductName (#PCDATA)>
<!ELEMENT DefaultLanguage (#PCDATA)>
<!ELEMENT DefaultCurrency (#PCDATA)>
<!ELEMENT Partner (Name)>
<!ATTLIST Partner
        PartnerID ID #IMPLIED
        RelationShip (Seller | Manufacturer |
                      Imtermediary) #IMPLIED
        Role CDATA #IMPLIED >
<!ELEMENT Name (#PCDATA)>
<!ELEMENT ClassInfo (ClassValue, ClassInfo?)>
<!ELEMENT ClassValue (#PCDATA)>
<!ATTLIST ClassValue
        ClassCode CDATA #IMPLIED
        ClassScheme CDATA #IMPLIED >
<!ELEMENT ProductDescription (#PCDATA)>
<!ATTLIST ProductDescription
        DescriptionPurpose CDATA #IMPLIED >
<!ELEMENT CountryOfOrigin (#PCDATA)>
<!ELEMENT Manufacturer (#PCDATA)>
<!ATTLIST Manufacturer
        PartnerRef IDREF #IMPLIED >
<!ELEMENT Participants (#PCDATA)>
<!ATTLIST Participants
        PartnerRef IDREF #IMPLIED >
<!ELEMENT ProductPrice (Amount, Buyer?)>
<!ELEMENT Amount (#PCDATA)>
<!ELEMENT Buyer (#PCDATA)>
<!ELEMENT Specification (#PCDATA)>
<!ATTLIST Specification
        TypeCode (Size | Weight | Ingredient | Color |
                  Shape | Packing | Performance |
                  Content | Others) #IMPLIED
        UOM CDATA #IMPLIED
        SpecificationName CDATA #REQUIRED >
<!ELEMENT ProductAttachment (AttachmentURL,
AttachmentPurpose?)>
<!ELEMENT AttachmentURL (#PCDATA)>
<!ELEMENT AttachmentPurpose (#PCDATA)>
```

[그림 12] XML DTD



[그림 13] 객체모델

4. 객체로부터 ORDB 스키마로 변환

4.1 변환

XML DTD의 요소들을 객체클래스를 이용하여 ORDB 스키마로 변환한다. 먼저 XML DTD의 요소들을 객체클래스로 변환하여 모델링하면 [그림 13]에서 나타나는 것과 같이 객체(Object), 관계성(Relationships), 멀티플리시티(Multiplicity), 방향성(Direction)이 만들어지며, 이들은 ORDB 스키마 객체, 즉 객체 타입(Object Type), 객체 테이블(Object Table), 참조 대 포함(Referenced vs Embedded), 컬렉션(Collections)으로 변환된다.

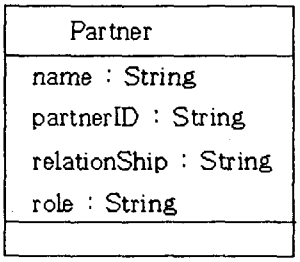
4.2 변환 과정

다음은 XML DTD로부터 변환된 객체클래스가 ORDB 스키마로 변환하는 과정은 아래와 같다.

① 타입, 테이블, 보조적인 객체를 생성하기 위해서는 ORDBMS로 확장되어 사용되며, 그러한 예로는 오라클8, 인포믹스, DB2가 있다.
② 객체 클래스는 구조화된 객체 타입을 생성하며, 테이블과 연결하여 타입을 하나 또는 여러 개의 테이블에 객체의 원소가 되도록 한다.
③ 객체 타입은 사용자 정의 타입이거나, 속성을 가진 객체타입이다.
④ 클래스에서의 객체 속성은 객체 타입에서의 속성이다.
⑤ 타입이 테이블이나 객체 또는 특정 타입으로 변환할 때 클래스에서의 객체 속성 타입은 객체 타입에서의 속성 타입이다.
⑥ 객체 속성이 NULL 제약조건을 가지면 NOT NULL 제약조건도 가진다.
⑦ 객체 속성이 초기값을 가지면 컬럼에 DEFAULT 값을 더한다.
⑧ 독립적인 클래스, 함축적인 성질을 가진 클래들을 위해 기본키로 정수를 생성한다. {oid}를 위해 태그된 컬럼에 기본키 제약조건에 따라 {oid}를 더한다.
⑨ 부 클래스를 위해 기본키 제약조건과 외래키 제약조건에 따라 각각의 부모 클래스의 키를 더한다.
⑩ 클래스들의 결합을 위해 객체 타입을 생성하고 기본키 제약조건과 외래키 제약조건에 따른 역할 테이블로부터 기본키를 추가한다.
⑪ 교환 oid가 <n>인 경우 UNIQUUE 제약조건에 따른 컬럼을 추가한다.
⑫ 각각의 확실한 제약조건을 위해 CHECK를 실시한다.
⑬ 결합하는데 있어서 각각의 0..1, 1..1 역할을 위한 참조 테이블에서 외래키 컬럼을 생성한다. 교대로 단일 객체들이나 컬럭션을 위한 그 자신 안에서 속성으로 객체를 선언하기 위하여 객체 타입에 참조를 사용하거나 또는 다중관계 객체를 위한 참조의 배열로 객체를 선언하기 위하여 객체 타입에 참조를 사용한다.
⑭ 집합 테이블에 외래키를 가진 다중 집합을 위해 기본키를 생성하고, 기본키를 위한 컬럼을 추가하며, 테이블 안에서 그 집합을 저장하기 위한 객체 타입을 사용한다. 또한 단일 객체를 위한 객체 속성 또는 컬렉션을 통하여 배열이나 네스티드 테이블을 사용한다.
⑮ 너무 많이 이동하게 되는 이진 결합 클래스들을 최적화 한다.
⑯ 외래키를 사용하여 결합이 안된 클래스와 함께 다대다 결합의 관계 테이블을 생성한다.
⑰ 다대다 결합에서 역할 테이블의 키로부터 기본키, 외래키의 제약조건을 생성한다.
⑱ 객체 클래스의 전달 작용을 위한 객체 타입에 메소드를 생성한다.

XML DTD[그림 12]로부터 변환된 객체모델[그림 13]은 위에 설명된 내용에 따라 아래의 ORDB 스키마로 변환되어 테이블이 생성된다.

(1) Partner 객체는 ProductCatalog 테이블에서 위에 열거된 ⑭번의 성질에 따라 네스티드 테이블로 변환된다.[그림 14]



[그림 14] Partner 객체

```
SQL> CREATE TYPE Partner_t AS OBJECT
(
```

```
  name          String,
  partnerID     String,
  relationShip  String,
  role          String
);

SQL> CREATE TYPE Partner_list AS TABLE OF
                                    Partner_t;
```

(2) ClassInfo 객체는 ProductCatalog 테이블에서 위에 열거된 ⑨번의 성질에 따라 부 클래스를 가지고 있으며, 또한 네스티드 테이블로 변환된다.[그림 15]

ClassInfo
classValue : ClassValue
classInfo : String

[그림 15] ClassInfo 객체

```
SQL> CREATE TYPE ClassInfo_t AS OBJECT
(
  classValue    ClassValue_t,
  classinfo     String
);

SQL> CREATE TYPE ClassInfo_list AS TABLE OF
                                    ClassInfo_t;
```

아래의 변환은 부 클래스로서 ClassValue 객체가 생성되고 위에 설명된 ClassInfo 테이블의 속성이 된다.

ClassValue
classValuePcdata : String
classCode : String
classScheme : String

[그림 16] ClassValue 객체

```
SQL> CREATE TYPE ClassValue_t AS OBJECT
(
  classValuePcdata    String,
  classCode           String,
  classScheme         String
);

SQL> CREATE TABLE ClassValue OF
                                    ClassValue_t;
```

(3) ProductDescrition 객체는 ProductCatalog 테이블에서 위에 열거된 ⑭번의 성질에 따라 네스티드 테이블로 변환된다.[그림 17]

ProductDescription
productDescriptionPcdata : String
descriptionPurpose : String

[그림 17] ProductDescription 객체

```
SQL> CREATE TYPE ProductDescrition_t AS
                                    OBJECT
(
  productDescritionPcdata    String,
  descritionPurpose          String
);

SQL> CREATE TYPE ProductDescrition_list AS
                        TABLE OF ProductDescrition_t;
```

(4) Manufacturer 객체는 ProductCatalog 테이블에서 위에 열거된 ⑤번의 성질에 따라 테이블 속성에 포함되어 변환된다.[그림 18]

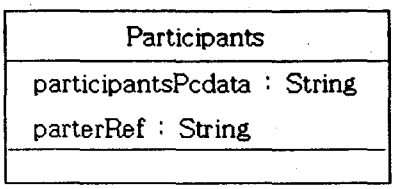
Manufacture
manufacturePcdata : String
parterRef : String

[그림 18] Manufacturer 객체

```
SQL> CREATE TYPE Manufacturer_t AS OBJECT
(
  manufacturerPcdata    String,
  partnerRef            String
);
```

```
SQL> CREATE TABLE Manufacturer OF
                                    Manufacturer_t;
```

(5) Participants 객체는 ProductCatalog 테이블에서 위에 열거된 ⑭번의 성질에 따라 네스티드 테이블로 변환된다.[그림 19]

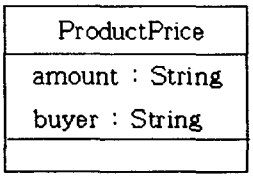


[그림 19] Participants 객체

```
SQL> CREATE TYPE Participants_t AS OBJECT
(
  participantsPcdata     String,
  partnerRef     String
);

SQL> CREATE TYPE Participants_list AS TABLE OF
                                    Participants_t;
```

(6) ProductPrice 객체는 ProductCatalog 테이블에서 위에 열거된 ⑭번의 성질에 따라 네스티드 테이블로 변환된다.[그림 20]

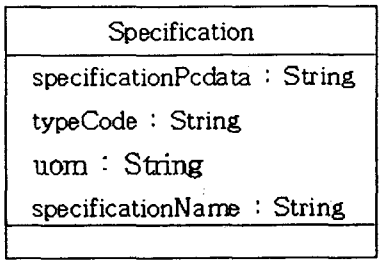


[그림 20] ProductPrice 객체

```
SQL> CREATE TYPE ProductPrice_t AS OBJECT
(
  amount          String,
  buyer           String
);

SQL> CREATE TYPE ProductPrice_list AS TABLE
                                    OF ProductPrice_t;
```

(7) Specification 객체는 ProductCatalog 테이블에서 위에 열거된 ⑭번의 성질에 따라 네스티드 테이블로 변환된다.[그림 21]

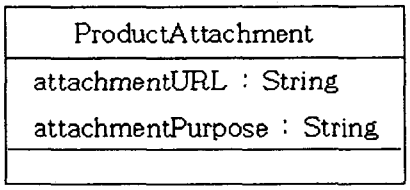


[그림 21] Specification 객체

```
SQL> CREATE TYPE Specification_t AS OBJECT
(
  specificationPcdata     String,
  typeCode                String,
  uom                     String,
  specificationName       String
);

SQL> CREATE TYPE Specification_list AS TABLE
                                    OF Specification_t,
```

(8) ProductAttachment 객체는 ProductCatalog 테이블에서 위에 열거된 ⑤번의 성질에 따라 테이블 속성에 포함되어 변환된다.[그림 21]



[그림 21] ProductAttachment 객체

```
SQL> CREATE TYPE ProductAttachment_t AS
                                    OBJECT
(
  attachmentURL          String,
  attachmentPurpose      String
);

SQL> CREATE TABLE ProductAttachment OF
                                    ProductAttachment_t;
```

위에서 변환된 각각의 객체들은 기본 테이블 스키마로 아래와 같이 구성되어 테이블이 생성된다.

```
SQL> CREATE TYPE ProductCatalog_t AS OBJECT
(
  productID           String,
  productName         String,
  defaultLanguage     String,
  defaultCurrency     String,
  countryOfOrigin     String,
  partner             Partner_list,
  classInfo           ClassInfo_list,
  productDescrition   ProductDescrition_list,
  manufacturer        Manufacturer_t,
  participants        Participants_list,
  productPrice        ProductPrice_list,
  specification       Specification_list,
  productAttachment   ProductAttachment_t
);

SQL> CREATE TABLE ProductCatalog OF
                                ProductCatalog_t
     NESTED TABLE partner STORE AS Partner_tbl;
     NESTED TABLE classInfo STORE AS ClassInfo_tbl;
     NESTED TABLE productDescrition STORE AS
                                ProductDescrition_tbl;
     NESTED TABLE participants STORE AS
                                Participants_tbl;
     NESTED TABLE productPrice STORE AS
                                ProductPrice_tbl;
     NESTED TABLE specification STORE AS
                                Specification_tbl;
```

5. 결론

XML은 웹에서 데이터 교환을 위한 표준 마크업 언어이며, DTD는 XML 문서 내에서 사용 가능한 요소를 관리하고, 사용할 수 있는 각 요소형의 내용과 특성을 지정하는 규칙들을 담고 있다. 본 논문에서 XML DTD를 ORDB 스키마 객체 모델로 변환하는 방법을 다루었으며, 객체 기반 변환 방법을 통하여 복잡한 구성의 객체도 ORDB 스키마 객체 모델로 변환이 가능함을 보여주었다. 이로 인하여 XML 문서를 객체로 변환하고 변환된 객체를 ORDB 스키마 객체 모델로 변환함으로서 웹 환경에서 데이터베이스를 효율적으로 구성할 수 있다.

참고문헌

[1] World Wide Wed Consortium. The XML Data Model. http://www.w3.org/XML/Datamodel.html, Jan 2000.

[2] Michel Goosens and Janne Saarela. A Practical Introduction to SGML. In TUGboat. volume 16(3),1995

[4] World Wide Web Consortium. XML Linking Language(XLink). TechnicalReportWD-xlink-2000021,W3C, February 2000.

[5] Andrew Davidson, Matthew Fuchs, Mette Hedin, Mudia Jain, JariKoistinen, Chris Lloyd, Murray Maloney, and Kelly Schwarzhof. Schema for Object-Oriented XML 2.0.July 1999. See http://www.w3.org/TR/NOTE-SOX

[6] Namespaces in XML, Tim Bray, David Hollander, Andrew Layman. See http://www.w3.org/TR/REC-xml-names/

[7] 김명주. "XML and Java", 이한출판사

[8] Frank Boumphrey외 11인 저 / 류광 역, "Professionnal XML Applications", 정보문화사

[9] Alexander & Tom 저 유진희, 박성준 역, "Professional Java XML Programming", "정보문화사"

[10] 이상태, 주경수. "제약조건 유지를 위한 XML DTD의 관계 스키마로 변환 방법", 「한국인터넷정보학회」 제1권 2호, 2000, pp.189-196

[11] Sandra E. Eddy & John E. Schnyder, "Teach Yourself XML", IDG BOOKS

[12] Simon North 저 노정운 역, "초보자를 위한 XML 21일 완성", 인포북.