객체모델을 이용한 XML DTD의 OODB 스키마로의 변환

최 문 영[†]·주 경 수[††]

요 약

B2B 전자상거래와 같이 XML을 이용한 정보교환이 확산되고 있으며 이에 따라 상호 교환되는 정보에 대하여 체계적이며 안정적인 저장관리가 요구되고 있다. 이를 위해 XML 응용과 데이터베이스 연계를 위한 다양한 연구가 관계형 데이터베이스를 중심으로 수행되었다. 그러나 계층구조를 갖는 XML 데이터를 2차원 테이블의 집합인 관계형 정보로 표현하는 관계형 데이터베이스로의 저장에는 본질적인 한계가 있어, 계층구조를 지원하는 객체지향 데이터베이스로의 저장이 요망된다. 이에 따라 계층구조를 갖는 XML 데이터를 객체지향 데이터 베이스로 저장하기 위한 모델링 방안이 요구된다. 본 논문에서는 객체모델을 토대로 DTD를 객체지향 데이터베이스 스키마로 변환하기 위한 방법을 제안한다. 이를 위하여 먼저 DTD를 객체모델로 변환시키기 위한 객체변환 방안을 제시하고, 변환된 객체모델을 객체지향 데이터베이스 스키마로 변경시키기 위한 스키마 변환 방법을 제안했다.

Transformation from XML DTD to OODB Schema using Object Model

Mun-Young Choi[†] · Kyung-Soo Joo[††]

ABSTRACT

Information exchange on XML such as B2B electronic commerce is spreading. Therefore the systematic and stable management mechanism for storing the exchanged information is needed. For this goal there are many research activities for connection between XML application and relational database. But because XML data have hierarchical structures and relational database can store only flat-structured data, we need to store XML data in object-oriented database that support hierarchical structure. Accordingly the modeling methodology for storing XML data in object-oriented database is needed. In this paper, the transforming methods based on object model from XML DTD to object-oriented database schema is proposed. For this, we first introduce mapping methods that map XML DTD to object model and then we propose soem methods that transform from the object model to object-oriented database schema.

키워드 : 객체지향데이터베이스(object-oriented database), OODB, XML DTD, ODL

1. 서 론

웹의 대중화에 가장 기여한 것이 HTML이다. 하지만 제한된 태그와, 데이터베이스처럼 구조화된 데이터의 지원이 불가능하여 정보의 교환과 데이터의 조작에 많은 애로점이 있었다. 이러한 문제점 해결을 위해 생겨난 XML은, 기업간 서로 다른 운영체제와 상이한 언어기반 및 소프트웨어 응용프로그램들간의 이질성으로 인해 생기는 비용을 현저히 줄일 수 있어 정보의 교환과 응용프로그램의 통합에 효율적인 방안을 제공할 수 있게 한다[7].

한편 웹에서 정보교환을 위한 XML 메시지의 소스 데이터는 Legacy 데이터베이스에 저장되어 있기 때문에, 이에 따라 XML 응용과 데이터베이스의 원활한 연동이 필요하다[2]. 그러나 계층구조를 갖는 XML 데이터를 2차원 테이블의 집합인 관계형 정보로 표현하는 관계형 데이터베이스로의 저장에는 본질적인 한계가 있어, 계층구조를 지원하는 객체지향 데이터베이스로의 저장이 요망된다. 따라서 계층구조를 갖는 XML 데이터를 객체지향 데이터베이스로 저장하기 위한 모델링 방안이 요구된다.

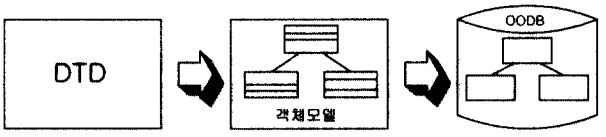
본 논문에서는 객체모델을 토대로 DTD를 객체지향 데이터베이스 스키마로 변환하기 위한 방법을 제안한다. 이를 위하여 먼저 DTD를 객체모델로 변환시키기 위한 객체변환 방안을 제시하고, 변환된 객체모델을 객체지향 데이터베이스 스키마로 변경시키기 위한 스키마 변환 방법을 제안한다. 본 논문의 2절에서는 객체 기반 변환 방법에 대하여 설명하고, 3절에서는 XML DTD를 객체로 변환하는 방법을 다루며, 4절에서는 객체를 OODB 스키마로 변환하는 방법을 다루고, 마지막으로 결론을 기술한다.

2. 객체 기반 변환

XML DTD를 데이터베이스 스키마로 변환하고자 할 때 단순한 XML 문서는 직접 테이블로 변환이 가능하나 복잡한

※ 이 논문은 1998년도 순천향대학교 교수 연구년제에 의하여 연구하였음.
† 준 회 원 : 순천향대학교 대학원 전산학과
†† 정 회 원 : 순천향대학교 정보기술공학부 교수
논문접수 : 2001년 9월 12일, 심사완료 : 2002년 1월 18일

XML 문서를 다룰 때는 객체 기반 변환 방법으로 처리되어야 한다. 따라서 테이블로 직접 변환하는 방법으로는 복잡한 XML 문서를 변환하기가 불가능하므로 이 단점을 개선시키기 위해 중간에 객체를 이용하여 (그림 1)과 같이 객체-관계형 데이터베이스 스키마로 변환한다[3].



(그림 1) 객체 기반 변환

3. XML DTD의 객체 변환

3.1 요소

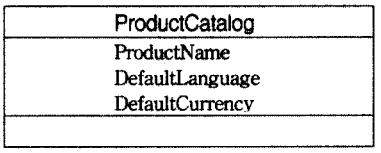
HTML은 태그가 바로 스타일 시트로 사용된다. 시작 태그는 성질을 열고, 끝 태그는 다시 닫는다. XML에서는 시작 태그와 끝 태그가 함께 하나의 요소를 이룬다. 요소는 하나의 루트 요소만 가져야 하며, 다른 태그는 모두 요소 안에 확실하게 중첩되어야 한다[8]. 이것은 한 요소가 다른 요소들을 포함할 경우, 그 요소들은 한 요소 안에 들어가야 한다는 뜻이다. 요소 타입은 두 개의 타입으로 분류하는데 PCDATA만 가진 요소의 타입을 단순 요소 타입이라고 하며, PCDATA을 제외한 모든 요소의 타입을 복합 요소 타입이라고 한다. 다시 말해 복합 요소 타입은 요소의 자식 요소, 혼합 요소, 요소의 특성들이다[10].

단순 요소 타입을 객체로 변환할 때는 클래스의 속성으로 변환된다. (그림 2)는 단순 요소 타입의 예를 보여준다.

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

(그림 2) 단순 요소 타입

(그림 2)는 클래스 속성으로 변환하면 (그림 3)과 같다.



(그림 3) 단순 요소 타입을 객체로 변환

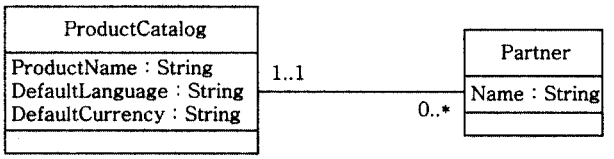
(그림 3)에서 보면 클래스 이름은 (그림 2)에 있는 요소가 포함하고 있는 부모 요소가 클래스 이름으로 지정하고, 데이터 형은 작성자가 요소의 의미에 따라 정수형, 문자형 등으로 지정한다.

복합 요소 타입은 단순 요소 타입을 제외한 모든 요소를 말하는데 (그림 4)와 같다.

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

(그림 4) 복합 요소 타입

(그림 4)에서는 루트 요소가 ProductCatalog이고 그에 자식 요소는 ProductName, DefaultLanguage, DefaultCurrency 이다. 따라서 객체로 변환하면 (그림 5)와 같다.



(그림 5) 복합 요소 타입의 객체로 변환

(그림 5)는 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.

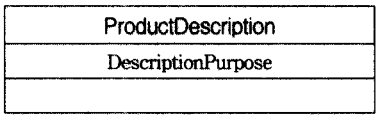
특성은 XML 파서가 애플리케이션에게 보내는 값들을 뜻하는데 요소의 내용과는 구별되는 값이므로 요소와 마찬가지고 특성도 단일 값을 가진 특성과 다중 값을 가진 특성으로 분류한다[6].

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

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

(그림 6) 단일 값을 가진 특성

(그림 6)에서 보면 클래스 이름이 ProductDescription이고 클래스의 속성은 DescriptionPurpose이다.



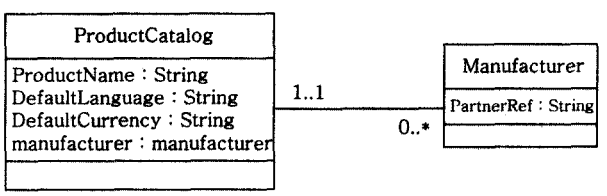
(그림 7) 단일 값을 가진 특성을 객체로 변환

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

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

(그림 8) 다중 값을 가진 특성

(그림 9)에서 보면 요소의 특성인 PartnerRef은 IDRFS로 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 IDREF 값으로 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)와 같다. 따라서 두 개의 클래스를 연결하려면 참조형으로 연결하면 된다.

3.3 복합 내용 모델 변환

3.3.1 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.

3.3.2 선택

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

3.3.3 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.

3.3.4 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.

요소의 생성규칙은 EBNF법을 따르고 있다. Extended Backus-Naur Form(EBNF) 표기법은 각각의 규칙으로 하나의 심볼을 정의한다[9].

① | : or 의 조건 즉 파이프라인을 중심으로 하나만 나타낼 때 쓰인다.
<!ELMENT XXX(A | B)>

② , : 명시된 순서대로 나타난다. 즉 A, B일 때 A가 나타난 다음에 B가 나타난다.
<!ELMENT XXX(A,B)>

③ ? : 해당 엘리먼트가 나타나지 않을 수도 또는 한번은 나타날수도 있다.
<!ELMENT XXX(A ?)>

④ (기호없음) : 반드시 나타난다. A을 명시했을 때 A는 반드시 나타나야 한다.
<!ELMENT XXX(A)>

⑤ * : 나타나지 않거나 여러번 나타날 수 있다. 1을 기준으로 수를 곱하는 형식으로 나타난다. 0번에서 ∞번까지 정의한다.
<!ELMENT XXX(A *)>

⑥ + : 한번은 반드시 나타나면서 여러번 반복해 나타날 수 있다. 1을 기준으로 수를 더하는 형식으로 나타난다. 1번에서 ∞번까지 정의한다.
<!ELMENT XXX(A +)>

⑦ () : 그룹으로 묶어서 처리, 우선적인 처리가 필요한 부분에 쓰인다. (A | C), B일 경우 A또는 C가 반드시 나온 뒤에 B가 나타난다.
<!ELMENT XXX((A | C),B)>

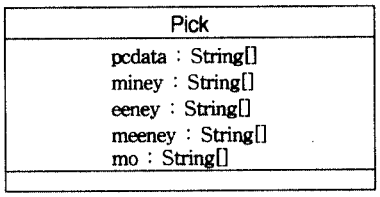
3.4 혼합 내용 모델 변환

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 태그사이에 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[9]. #PCDATA만 유일한

옵션이며, "*"는 반드시 괄호를 닫은 바로 뒤에 와야 한다. (그림 10)과 같이 DTD 예를 들어보자.

```
<!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



(그림 11) 혼합 내용 모델 DTD를 객체화로 변환

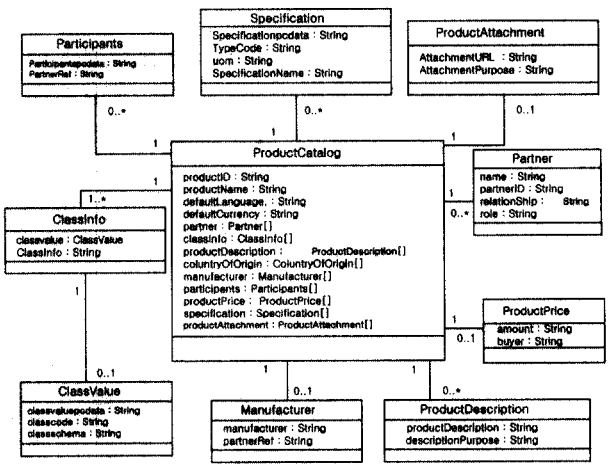
```
<?xml version = "1.0" encoding = "EUC-KR"?>
<!ELEMENT ProductCatalog (ProductName, DefaultLanguage,
   DefaultCurrency, Partner *, ClassInfo +, ProductDescription *,
ColuntryOfOrigin?. Manufacturer?. Participants *.
ProductPrice *, Specification *,
ProductAttachment?)>
<!ATTLIST ProductCatalog ProductID ID #REQUIRED >
<!ELEMENT ProductName (#PCDATA)>
<!ELEMENT DefaultLanguage (#PCDATA)>
<!ELEMENT DefaultCurrency (#PCDATA)>
<!ELEMENT Partner (#PCDATA)>
<!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
            ClassSchema 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, Attachment-
Purpose?)>
<!ELEMENT AttachmentURL (#PCDATA)>
<!ELEMENT AttachmentPurpose (#PCDATA)>
```

(그림 12) XML DTD

위에 있는 DTD는 혼합 내용 모델이다. pick요소는 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 (그림 11)과 같다.

DTD에서 "*" 연산자는 요소나 요소그룹이 생략하거나 한 번 이상 나타날 수 있으므로 설계하는 작성자는 몇 개가 나타나는지 알 수 없기 때문에 String[] 형으로 선언한다.

· 앞에 제시한 객체변환 방법에 따라 (그림 12)의 XML DTD가 (그림 13)의 객체모델로 변환된다.



(그림 13) 객체 모델

4. OODB 스키마 변환

앞에서 제시한 XML DTD의 객체변환에 따라 변환된 객체들을 4.1의 변환방법에 의해 OODB 스키마로 변환시킨다 [5, 4].

4.1 변환 방법

① "지속적인" 클래스는 지속적인 OODB 클래스가 된다.

② 인터페이스가 "지속적인" 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 가지고 있지 않는다.

③ type 분류자는 enum 또는 typedef로 표시된다.

④ 클래스의 attribute는 적당한 type 변환과 제한으로 OODB 클래스의 attribute가 된다.

⑤ 만약 nullable 태그가 나타나면 nullable 상수형을(nullable _short) 사용하고 바인딩을 지원하는 널값을(ODL) 사용한다. 만약 그렇지 않으면, 그것을(C++, JAVA) 무시한다.

⑥ 만약 attribute가 initializer를 가지고 있으면, 생성자에 초

기화 코드를 추가한다. 생성자 메소드의 어느 한쪽의 부분으로서 또는 C++ 멤버에 초기화는 목록에 나열한다.

⑦ 서브 클래스는 클래스 선언에 슈퍼클래스 사양을 포함한다.

⑧ 연관 클래스는 연관 클래스의 attribute와 클래스를 생성한다.

⑨ 명백한 객체 일치 {oid} 또는 후보 키 {alternate oid}가 만약 바인딩을 지원하면 키의 선언을 명시한다. 만약 지원하지 않으면, 객체의 집합에 억제를 점검하는 적당한 메소드를 공급한다.

⑩ 각 명백한 억제에 대한 적당한 클래스에 메소드를 추가하고 안전하게 한다. 시스템은 메소드를 시스템이 그 억제가 만족할 만한 것을 요구할 때마다 호출한다.

⑪ 적당한 객체 또는 연관된 Multiplicity로부터 유래한 컬렉션 타입에 대해서는 연관 클래스를 가지고 있지 않은 각 이진 연관을 위해 관계성을 생성한다. 명백한 화살표들이 연관되어 있지 않다면 역 관계성들을 사용한다.

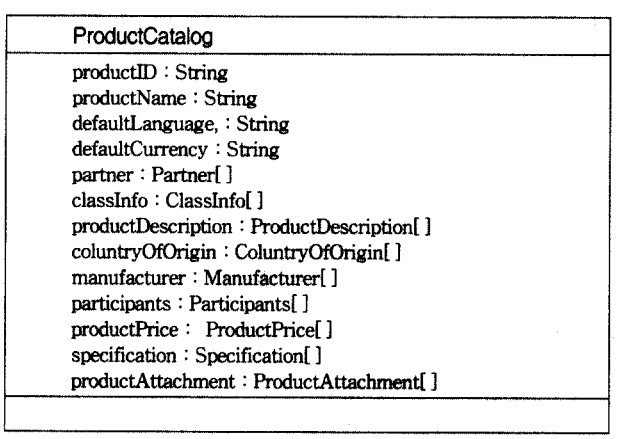
⑫ 다른 클래스의 각 역할 링크를 위한 연관 클래스에 관계성을 생성한다.

⑬ 코드를 생성하거나 또는 특별한 OODB의 요구처럼 어떠한 복합 집단화 연관을 위해 삭제를 전파하기 위하여 OODB 특징들을 사용한다.

⑭ 3진으로 이루어진 연결을 위해 연관 클래스를 생성한다. 그리고 연관 클래스에서 Multiplicity를 주었던 적당한 자료 type의 연합한 클래스까지 관계성을 생성한다.

4.1 변환 예

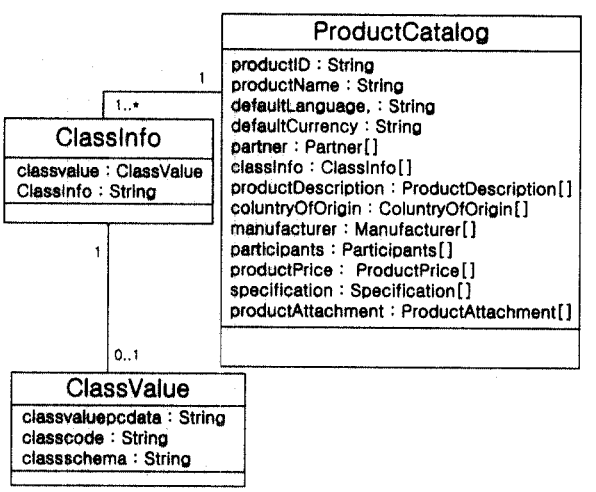
1. 첫 번째로 변환방법 ④에 의해서 클래스를 정의하면 (그림 14)와 같다.



(그림 14) ProductCatalog 객체

```
class ProductCatalog (extent ProductCatalogs) {
    attribute String productID ;
    attribute String cproductName ;
    attribute String defaultLanguage ;
    attribute String defaultCurrency ;
    relationship set<Partner> ;
    relationship set<ClassInfo> ;
    relationship set<ProductDescription> ;
    relationship ColuntryOfOrigin ;
    relationship Manufacturer ;
    relationship set<Participants> ;
    relationship ProductPrice ;
    relationship set<Specification> ;
    relationship ProductAttachment ;
} ;
```

2. (그림 15)에서 ClassInfo 객체는 변환방법 ⑦, ⑪에 따라 서브 클래스를 가지고 있다. ProductCatalog 객체와 ClassInfo 객체는 one-to-many 관계이며 ClassInfo 객체와 ClassValue 객체는 one-to-zero or one관계이다. (그림 13)에서는 모두 연관관계를 의미하며 one-to-many는 ProductCatalog 객체에서 연관된 객체들이 여러 개가 존재하고 one-to-zero or one은 ProductCatalog 객체에서 연관된 객체들이 하나이상 존재하지 않는다는 의미를 갖는다[1].

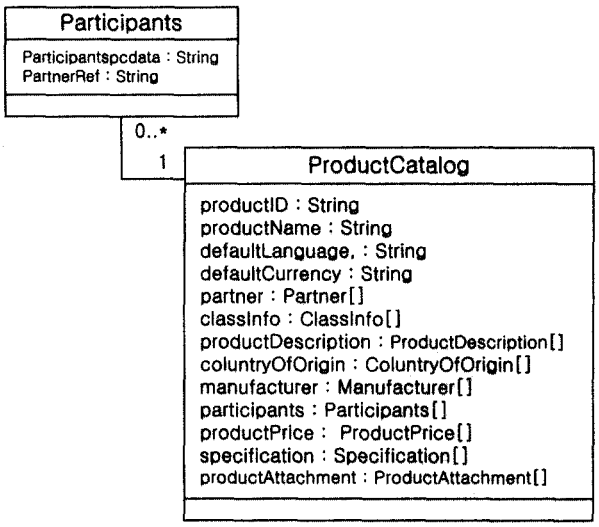


(그림 15) ClassInfo 객체, ClassValue 객체

```
class ClassInfo (extent ClassInfos) {
    relationship ClassValue ;
    attribute String classinfo ;
    relationship ProductCatalog ;
} ;

class ClassValue (extent ClassValues) {
    attribute String classvaluepcdata ;
    attribute String classcode ;
    attribute String classschema ;
    relationship ClassInfo ;
} ;
```

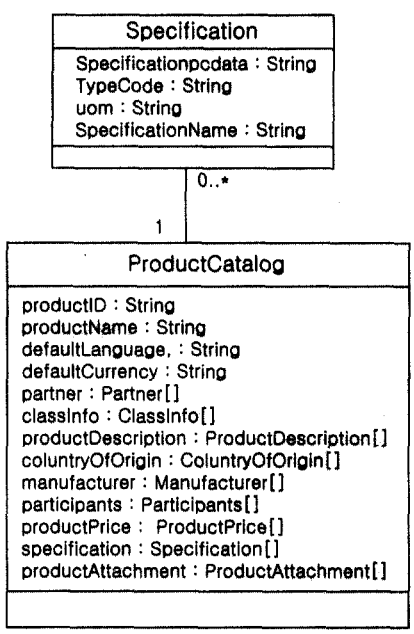
3. (그림 16)에서 Participants 객체는 변환방법 ⑧, ⑪에 따라 attribute를 작성한다. ProductCatalog 객체와 Participants 객체는 Multiplicity의 one-to-many관계이다.



(그림 16) Participants 객체

```
class Participants (extent Participants) {
    attribute String Participantspcdata ;
    attribute String PartnerRef ;
    relationship ProductCatalog ;
} ;
```

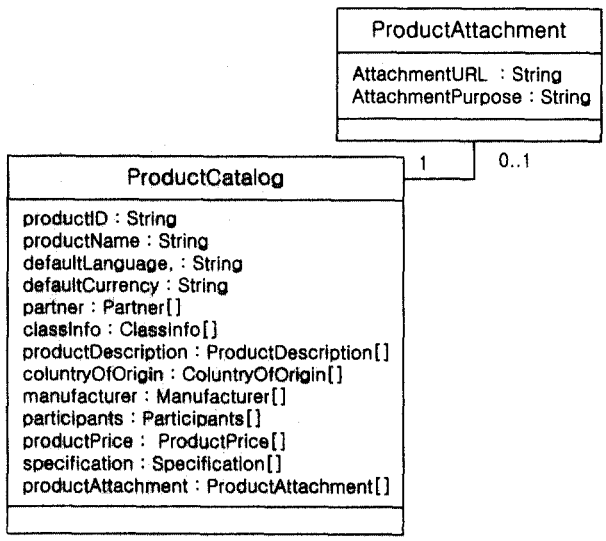
4. (그림 17)에서 Specification 객체는 변환방법 ⑧, ⑪에 따라 attribute를 작성한다. ProductCatalog 객체와 Specification 객체는 one-to-many관계이다.



(그림 17) Specification 객체

```
class Specification (extent Specifications) {
    attribute String Specificationpcdata ;
    attribute String TypeCode ;
    attribute String uom ;
    attribute String SpecificationName ;
    relationship ProductCatalog ;
} ;
```

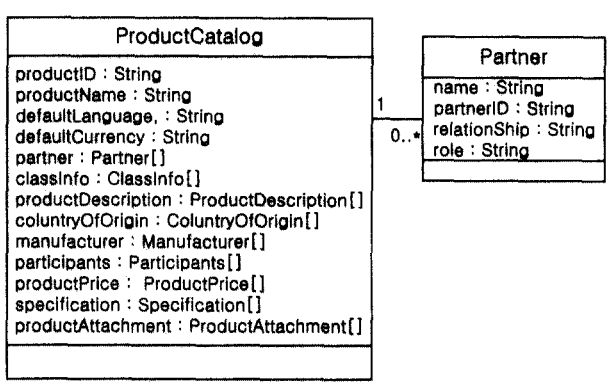
5. (그림 18)에서는 변환방법 ⑧, ⑪에 따라 attribute를 작성하고 관계는 one-to-zero or one관계이다.



(그림 18) ProductAttachment 객체

```
class ProductAttachment
(extent ProductAttachments) {
    attribute String AttachmentURL ;
    attribute String AttachmentPurpose ;
    relationship ProductCatalog ;
} ;
```

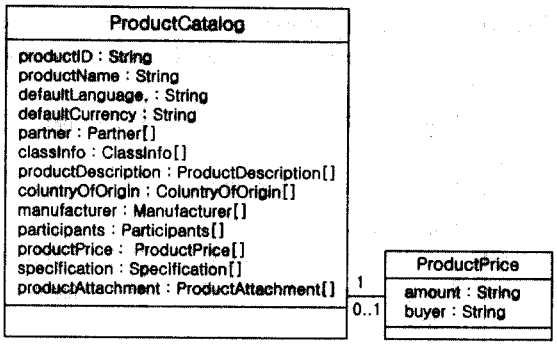
6. (그림 19)에서는 변환방법 ⑧, ⑪에 따라 attribute를 작성하고 관계는 one-to-many관계이다.



(그림 19) Partner 객체

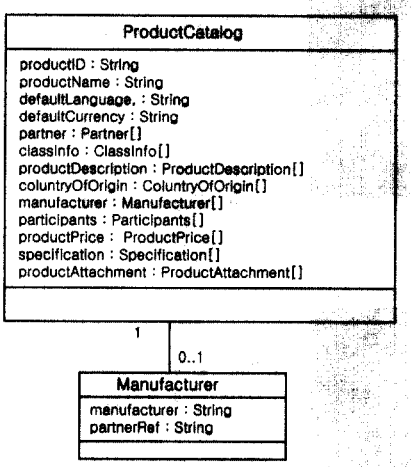
```
class Partner (extent partners) {
    attribute String name ;
    attribute String partnerID ;
    attribute String relationShip ;
    attribute String role ;
    relationship ProductCatalog ;
  ) ;
```

7. (그림 20)과 (그림 21)에서는 변환방법 ⑧, ⑪에 따라 attribute를 작성하고 관계는 one-to-zero or one관계이다.



(그림 20) ProductPrice 객체

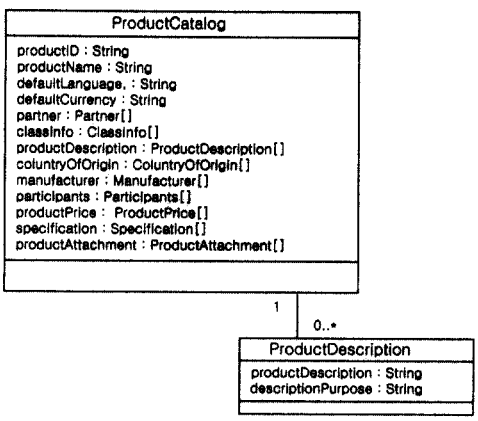
```
class ProductPrice (extent ProductPrices) {
    attribute String amount ;
    attribute String buyer ;
    relationship ProductCatalog ;
} ;
```



(그림 21) Manufacturer 객체

```
class Manufacturer (extent Manufacturers) {
    attribute String manufacturer ;
    attribute String partnerRef ;
    relationship ProductCatalog ;
} ;
```

8. (그림 22)에서는 변환방법 ⑧, ⑪에 따라 attribute를 작성하고 관계는 one-to-many관계이다.



(그림 22) ProductDescription 객체

```
class ProductDescription
(extent ProductDescriptionvs) {
    attribute String productDescription ;
    attribute String descriptionPurpose ;
    relationship ProductCatalog ;
} ;
```

(그림 13)의 객체모델을 4.1에서 제시한 변환방법에 따라 ODL을 이용하여 OODB 스키마로 변환하면 (그림 23)과 같다.

```
class ProductCatalog (extent ProductCatalogs){
    attribute String productID ;
    attribute String cproductName ;
    attribute String defaultLanguage ;
    attribute String defaultCurrency ;
    relationship set<Partner> ;
    relationship set<ClassInfo> ;
    relationship set<ProductDescription> ;
    relationship ColumtryOfOrigin ;
    relationship Manufacturer ;
    relationship set<Participants> ;
    relationship ProductPrice ;
    relationship set<Specification> ;
    relationship ProductAttachment ;
} ;
class ClassInfo (extent ClassInfos) {
    relationship ClassValue ;
    attribute String classinfo ;
    relationship ProductCatalog ;
} ;
class ClassValue (extent ClassValues) {
    attribute String classvaluepcdata ;
    attribute String classcode ;
    attribute String classschema ;
    relationship ClassInfo ;
} ;
class Participants (extent Participants) {
    attribute String Participantspcdata ;
    attribute String PartnerRef ;
    relationship ProductCatalog ;
} ;
class Specification (extent Specifications) {
    attribute String Specificationpcdata ;
    attribute String TypeCode ;
    attribute String uom ;
    attribute String SpecificationName ;
    relationship ProductCatalog ;
} ;
class ProductAttachment
(extent ProductAttachments) {
    attribute String AttachmentURL ;
    attribute String AttachmentPurpose ;
    relationship ProductCatalog ;
} ;
class Partner (extent partners) {
    attribute String name ;
    attribute String partnerID ;
    attribute String relationShip ;
    attribute String role ;
    relationship ProductCatalog ;
} ;
class ProductPrice (extent ProductPrices) {
    attribute String amount ;
    attribute String buyer ;
    relationship ProductCatalog ;
} ;
class Manufacturer (extent Manufacturers) {
    attribute String manufacturer ;
    attribute String partnerRef ;
    relationship ProductCatalog ;
} ;
class ProductDescription
(extent ProductDescriptionvs) {
    attribute String productDescription ;
    attribute String descriptionPurpose ;
    relationship ProductCatalog ;
} ;
```

(그림 23) OODB 스키마 구현

5. 결 론

XML은 웹에서 구조화된 정보나 반-구조화된 정보를 교환하기 위한 표준 마크업 언어로 채택되어 가고 있다. 한편 웹에서 정보교환을 위한 XML 메시지의 소스 데이터는 Legacy 데이터베이스에 저장되어 있기 때문에, 이에 따라 XML 응용과 데이터베이스 시스템과의 원활한 연계가 요구되어진다.

XML 응용과 데이터베이스 시스템 사이의 원활한 연계를 위해서 XML DTD를 관계형 데이터베이스 스키마로 변환하는 방법에 대해서는 많은 연구가 진행되었으나, XML DTD를 객체지향 데이터베이스 스키마로 변환키 위한 연구는 미미한 실정이다. 멀티미디어 응용 등을 위하여 객체지향 데이터베이스의 보급이 확대되고 있는 현실에서XML DTD를 객체지향 데이터베이스 스키마로 변환키 위한 연구는 조속히 요구되고 있다.

본 논문에서는 XML DTD 문서를 객체로 변환하고 변환된 객체를 ODMG 2.0의 ODL에 따른 OODB 스키마 객체 모델로 변환하는 방법을 다룸으로써, 웹 환경에서 객체지향 데이터베이스를 토대로 한 XML 응용시스템을 보다 효율적으로 구성할 수 있도록 하였다.

참 고 문 헌

[1] James Rumbaugh, Ivar Jacobson, and Grady Booch, The Unified Modeling Language Reference Manual, Addison Wesley, 1999.

[2] James Rumbaugh, Object-oriented modeling and design, Englewood Cliffs, N. J, Prentice Hall, 1991.

[3] Joo Kyung-soo, "A Design of Middleware Components for the Connection between XML and RDB," 2001 IEEE International Symposium on Industrial Electronics Proceedings, Pusan, Korea, June, 2001.

[4] R. G. G. Cattell, Douglas K. Barry, The Object Database Standard : ODMG 2.0, Morgan Kaufmann Publishers, Inc, 1997.

[5] Robert J. Muller, Database Design for Smarties : Using UML for Data Modeling, Morgan Kaufmann Publishers, Inc, 1999.

[6] Frank Boumphrey외 11인 저, 류광 역, Professional XML Applications, 정보문화사.

[7] Alexander & Tom 저, 유진희, 박성준 역, Professional Java XML Programming, 정보문화사.

[8] 이상태, 주경수, "객체모델을 이용한 XML DTD의 ORDB 스키마로의 변환", 한국데이터베이스학회 춘계 Conference, pp. 303-310, 2001.

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

[10] 김채미, 최학열, 김심석 공저, 전문가와 함께 가는 XML Camp, 마이트 Press, 2001.

최 문 영

e-mail : griffin@hyejeon.ac.kr

1998년 청운대학교 전자계산학과 졸업(학사)

2000년 청운대학교 정보산업대학원 전산전자정보학과 졸업(석사)

2000년~현재 순천향대학교 전산학과 재학중(박사과정)

관심분야 : Database Systems, Object-oriented, XML

주 경 수

e-mail : gsoojoo@sch.ac.kr

1980년 고려대학교 이과대학 수학과 졸업(학사)

1982년 고려대학교 일반대학원 전산학과 졸업(석사)

1986년 고려대학교 일반대학원 전산학과 졸업(박사)

1998년 University of North Carolina

1999년 Visiting Professor

1986년~현재 순천향대학교 정보기술공학부 교수

관심분야 : Database Systems, System Integration, Object-oriented Systems