EER 다이어그램을 이용한 XML 스키마 설계 방법

김형석[0] 허보진 김창석
공주대학교 대학원 멀티미디어학과
e-mail:{replay76, bobe04, csk}@kongju.ac.kr

Design XML Schemas Using EER Diagram

Hyeong-Seok kim[0], Bo-Jin heo, Chang Suk Kim
Dept. of Multimedia, kongju National University

요 약

DTD를 대체하기 위해 개발된 XML 스키마는 전역 및 로컬 기능, 확장성과 다양한 타입변화등 많은 특성을 가지고 있어 향후 DTD를 대신하여 XML 스키마의 사용이 날로 늘어날 것이다. 본 논문에서는 XML 스키마를 설계하는 방법으로 EER 다이어그램을 이용하여 XML 스키마를 설계하는 기존의 방법을 수정 보완한 새로운 설계방법을 제시한다.

1. 서론

인터넷 비즈니스에서 XML(eXtensible Markup Language)[1] 문서는 정보 통신망의 발달과 함께 문서 교환의 표준으로 자리잡고 있으며 인터넷상에서 데이터를 표현하고 교환하는 새로운 표준으로 등장하고 있다.

또한, XML 스키마(W3C XML Schema Spec)은 DTD를 대체하기 위해 개발되었다. 이것은 강력하고 융통성 있는 언어로서 작성자가 문서에 대해 DTD 보다 더 강력하게 혹은 더 약하게 제약할 수 있다. 물론 DTD와 마찬가지로 유효성 검사도 제공한다. XML 스키마는 풍부한 표현의 문서를 좀더 쉽게 처리할 수 있게 하는 데이터형 만들기를 제공한다. 그리고 XML 스키마 자체가 XML로 되어 있어서 DTD 의 EBNF 형태의 다른 구문을 배울 필요가 없고, XML Well-formed 문서 DOM, XSLT 등 XML 도구들을 그대로 사용가능 하다. 또한 XML 스키마에는 풍부한 표현의 문서를 좀더 쉽게 처리할 수 있게 하는 데이터형 만들기를 제공하고 있다[5].

지금까지 유용한 XML 스키마를 설계하는 방안은 몇가지 제시되었다. 본 논문에서는 관계형 데이터베이스의 전체적인 논리적 구조를 알아보거나 데이터베이스 설계 시 기본적으로 사용되는 EER 다이어그램을 이용하여 XML 스키마를 설계하는 방법을 제시하고자 한다. 또한 변환 시 XML 스키마의 특성[5]인 전역 및 로컬기능, 확장성과 다양한 타입변화 등을 최대한 활용하여 변환하는 방법을 제시한다.

2. 관련연구 및 선행연구의 문제점

본 논문에서는 다루지 않지만, 궁극적으로는 RDB 스키마를 EER 다이어그램으로 역 변환하여 변환된 결과인 EER 다이어그램을 이용하여 XML 스키마 혹은 DTD로의 변환을 주 목표로 하고 있다[2,3]. RDB를 DTD로 변환하는 연구는 UCLA 대학의 EXPRESS, Univ of Applied Sciences 의 DB2XML, stanford 대학의 Lore 프로젝트, SilkRoute[6], XPERANTO[7] 등으로 현재 활발히 진행되고 있다. 이 후 DTD를 대체하기 위해 개발된 XML 스키마가 개발됨으로써 RDB를 XML 스키마로 변환하는 연구가 진행되고 있다. 대표적으로 elmasri의 연구가 있다[4]. elmasri의 연구에는 EER 다이어그램 구조가 엔티티들간의 마이그레이션 으로 인해 XML 스키마 문서가 간략해지는 장점이 있지만 참조해야할 뚜렷한 엘리먼트가 사라지므로, 하나의 엔티티에 중복된 관계가 있을 경우 같은 내용을 두 번 이상 정

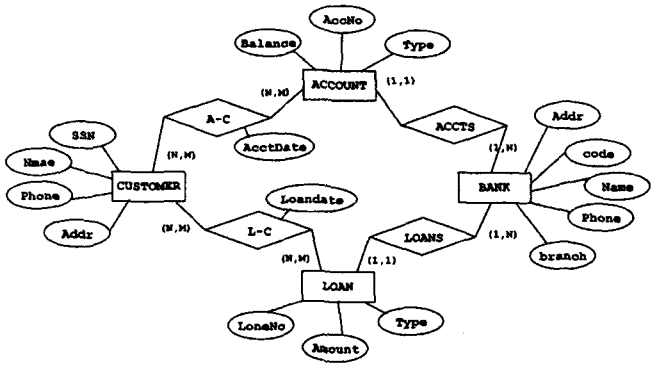

그림 1 EER 다이어그램(예제)

의해야 하는 경우가 발생하며 XML 스키마의 큰 특징 중의 하나인 확장성 및 재사용성을 사용는 데에 제약을 받게 된다. 본 논문에서는 elmasri 연구의 기존의 문제점을 보안하여 XML 스키마의 전역 및 로컬 기능 및 재사용성을 활용하여 EER 다이어그램에서 XML 스키마 설계하는 방법을 제시한다.

3. EER 다이어그램에서 계층적 구조 생성하기

일반적으로 관계형 데이터베이스 설계시 사용되는 EER 다이어그램은 그래프(graph) 형태로 되어 있다. 그러나 XML 스키마는 계층형(hierarchcal) 구조로 되어 있어 두 형태는 매우 유사하지만 일치하지는 않는다. 그러므로 우선 XML 스키마로 표현하기 위해 그래프 형태로 되어있는 EER 다이어그램 구조를 계층형 구조로의 변환이 필요하다. 이 계층형 구조로의 변환을 위해 BFS 탐색방법과 와 대응 제약조건(*mapping cardinality*)을 이용한다.

3.1 루트 엔티티 선택하기[2]

EER 다이어그램을 계층형 구조로 나타내기 이전에 먼저 EER 다이어그램의 엔티티들 중에서 어떤 엔티티를 중심으로 XML 스키마를 표현할 것인가를 결정해야 한다. 이 루트 엔티티는 최종사용자(enduser)마다 관심 있어 하는 분야가 다르기 때문에 최종사용자가 최 상위 루트엔티티를 선택하도록 한다.

이 예에서는 최종사용자가 최 상위 루트엔티티로써 CUSTOMER 를 선택하였다고 가정한다.

3.1 BFS(Breadth First Search)로 계층형 구조로 나타내기

최 상위 루트엔티티가 결정되었다면 그래프 형태인 EER 다이어그램을 계층형 구조로 나타내야 한다. 이를 위해 BFS(Breadth First Search)를 최 상위 루트엔티티로부터 스캔을 시작하여 어떤 하위노드도 중복이 없을 때 까지 모든 노드를 스캔한다. 만일, 스캔중에 중복 노드가 나타났을 경우에는 그 노드를 복사(duplicate) 하여 완전한 트리구조로 변환한다.

3.3 EER 다이어그램 Customize 하기

계층적 구조로 변환된 EER 다이어그램의 의미변화를 최소화하는 범위 내에서 Customize 한다. 이를 위해 대응 제약 조건을 사용하며, 엔티티를 중심으로 다음의 조건에 따라 변환한다. 여기서 상위 엔티티를 AE, 하위 엔티티를 E, 관계를 편의상 L 로 표현한다.

만일 L 이

1. M:N or 1:n 일때

 만일 M:N 관계라면

  의미의 변화가 없는 1:N 관계로 변환한다.

 만일 EER 다이어그램에서 Relationship 의 Attribute 가 있다면 E 에 병합된다.

2. N:1 or 1:1 관계라면

 만일 N:1 관계라면

  의미의 변화가 없는 1:1 관계로 변환한다.

 만일 EER 다이어그램에서 Relationship 의 Attribute 가 있다면 AE 에 병합된다.

만일 여기서 엔티티의 애트리뷰트를 마이그레이션 하게 되면, XML 스키마의 고유한 특성인 전역 및 로컬기능을 최대한 살릴 수 없고 만약에 중복된 엔티티가 있을 경우 같은 내용을 두 번 이상 정의를 할 가능성이 있기 때문에 엔티티에서의 마이그레이

선은 하지 않는다.

위의 조건으로 그림1의 EER 다이어그램을 변환하면 아래와 같다.

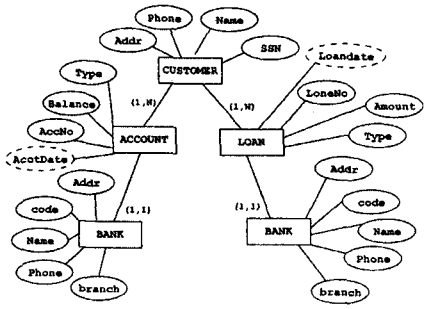

그림 2 변환된 계층형 구조

4. Customized 된 계층형 구조를 XML 스키마로 표현하기[2]

XML 스키마로 표현하기 위해 처음으로 필요한 요소는 XML 스키마의 최 상위 루트 엘리먼트를 정의하는 것이다. 최상위 루트 엘리먼트는 편의상 "CustomerDoc"라 한다.

1. XML 스키마 문서를 위한 루트엘리먼트를 정의하며 이것은 전체 스키마 문서에 이름을 주는데 사용한다. 이 엘리먼트의 타입은 "rootType" 로 하고 complexType 으로 정의한다. 이 예에서 우리는 루트엘리먼트의 이름으로 CustomerDoc를 사용하였다.
2. complexType을 하나 생성하고 그 안에 엘리먼트를 최 상위 엔티티인 'E1'이름으로 하나 생성하고 타입으로 'E1Type'으로 한다.
3. 재귀적 프로시져인 "Generate_XML_Schema"를 호출하고, argument로써 계층의 루트엔티티를 사용한다.[2]

```
Generate_XML_Schema(EntityTypeName E1)
{
  E1과 그 타입을 위해 complexType의 'E1 타입'을
  생성한다. E1 타입 아래에는
  DO{
    FOR EACH EER attribute A of E1
    {
      IF E1이 스캔되지 않았다면
      {
        A에 상응하는 complexType e를 하나 생
        성하고,
        e 안에 simpleContent를 정의하고
        e 의 애트리 뷰트로써 FIELD_NAME,
        FIELD_TYPE, FIELD_LEN,
        NULLABLE 등을 정의한다.
      }
    }
    FOR EACH 계층구조에서 E1의 자식요소 E2
    { Generate_XML_Schema(E2) }
  }
}
```

그림 2로 예를 들면 XML 스키마 문서를 위한 루트 엘리먼트를 정의한다. 그리고 이 스키마 표현을 위해 "CustomerDoc" 라는 이름을 사용하였다. 계층적 구조의 루트로 CUSTOMER을 선택했으므로, CUSTOMER 은 변형된 재귀적 함수 Gemerate_XML_Schema 로 들어가는 최초의 현재 엔티티타입(E1)이 될 것이다. Gemerate_XML_Schema에서 우리는 CUSTOMER 의 엔티티 타입 아래에 EER 애트리뷰트로 Addr, Phone, Name, SSN가 있으므로 XML 스키마 문서에서 콤플렉스 타입의 엘리먼트를 각각 생성한다. 그리고 각각의 엘리먼트는 데이터베이스 시스템으로부터의 그들의 속성을 설명해 주는 네 개의 XML 애트리뷰트를 가진다. 그리고 하위 엔티티인 ACCONT 와 LOAN 이 있으므로, 재귀적 방식으로 각각 엔티티를 적용한다. ACCONT 엔티티가 적용될 때는 하위 엔티티로 BANK 엔티티가 있으므로 재귀적으로 BANK 엔티티를 적용한다. LOAN 엔티티가 적용될때는 하위 엔티티로 BANK 가 존재하지만 ACCONT 엔티티 적용시 BANK 엔티티가 스캔되어 적용되었으므로 재귀적으로 호출하지 않는다. 최종적으로 재귀적 함수를 통하여 나온 XML 스키마는 그림 3과 같다.

5. 결론 및 향후 연구 방향

본 논문에서는 EER 다이어그램에서 XML 스키마로의 변환 방법에 대해 제시하였다. 기존에 제시된 변환방법에 XML 스키마의 큰 특징중의 하나인 확장성 및 재사용성에 초점을 두었다. 본 논문에서는 EER 다이어그램에서 XML 스키마로 변환 시에 대응제약조건을 사용하여 의미를 보존하면서 Customerize 하였고 재사용성을 위해 엔티티의 마이그레이션 없이 변환하였다. 향후 연구과제로는 RDB의 스키마를 역변환한 EER 다이어그램을 이용

하여 XML 스키마로 변환하고 그 스키마 구조에 맞는 XML 인스턴스 문서를 생성해 내는 알고리즘을 구현하는 것이다. 또한 향후 사용자의 취향에 맞게 RDB 스키마를 로드하여 XML 스키마로 변환하고 또는 XML 스키마를 로드하여 RDB 스키마로 변환하는 시스템을 개발하는 것이다.

6. 감사의 글

본 연구는 한국과학재단 목적기초연구 (R01-2002-000-00068-0)의 지원으로 수행되었음

참 고 문 헌

[1] Bray, t., Paoli, j., Sperberg-McQueen, c., Maler, E.: Extensible Markup Language(XML)1.0 W3C Recommendation, October 2000.

[2] Dongwon Lee, "Schema Conversion Methods between XML and Relational Models", Knowledge Transformation for the semantic Web, 2003

[3] 허보진, 김형석, 김창석 ,"XML 스키마 변환방법에 관한 비교론적 고찰" 한국정보처리학회 춘계학술발표대회, 2003년 5월

[4] Ramez Elmasri, "Conceptual Modeling for Customized XML Schema", Proceedings of the 21st International Conference on Conceptual Modeling 2002, page 429-443

[5] Jon Duckett 외 8인 공저, "Professional XML Schemas", wrox

[6] Fernandez, M.F., Tan. W.C., Suciu, D.:"SilkRoute:Trading between Realations and XML". In: International Word Wide Web Conference, Amsterdam, Netherlands(2000)

[7] Carey, M., Florescu, D., Ives, Z., Lu, Y., Shanmugasundaram, J., Shekita, E., Subramanian, S.:"XPERANTO: Publishing Object-Relational Data as XML". In: International Workshop on the Web and Databases, Dallas, TX(2000)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
          <xs:element name="customerDoc" type="rootType"/>

          <xs:complexType name="rootType">
                    <xs:sequence>
                    <xs:element name="CUSTOMER" type="CUSTOMERType"
                    minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
          </xs:complexType>
          <xs:complexType name="CUSTOMERType">
                    <xs:sequence>
                    <xs:element name="Addr" type="xs:string"/>
                    <xs:element name="Phone" type="xs:string"/>
                    <xs:element name="Name" type="xs:string"/>
                    <xs:element name="SSN" type="xs:string"/>
                    <xs:element name="ACCOUNT" type="ACCOUNTType"
                    minOccurs="0" maxOccurs="unbounded"/>
          <xs:element name="LOAN" type="LOANType" minOccurs="0"
                    maxOccurs="unbounded"/>
                    </xs:sequence>
          </xs:complexType>
          <xs:complexType name="ACCOUNTType">
                    <xs:sequence>
                    <xs:element name="Type" type="xs:string"/>
                    <xs:element name="Balance" type="xs:string"/>
                    <xs:element name="AccNo" type="xs:string"/>
                    <xs:element name="AcctDate" type="xs:string"/>
                    <xs:element name="BANK" type="BANKType"
                              minOccurs="1" maxOccurs="1"/>
                    </xs:sequence>
          </xs:complexType>
          <xs:complexType name="BANKType">
                    <xs:sequence>
                    <xs:element name="Addr" type="xs:string"/>
                    <xs:element name="code" type="xs:string"/>
                    <xs:element name="Name" type="xs:string"/>
                    <xs:element name="Phone" type="xs:string"/>
                    <xs:element name="branch" type="xs:string"/>
                    </xs:sequence>
          </xs:complexType>
          <xs:complexType name="LOANType">
                    <xs:sequence>
                    <xs:element name="Loandate" type="xs:string"/>
                    <xs:element name="Amount" type="xs:string"/>
                    <xs:element name="LoneNo" type="xs:string"/>
                    <xs:element name="Type" type="xs:string"/>
                    <xs:element name="BANK" type="BANKType"
                    minOccurs="1" maxOccurs="1"/>
                    </xs:sequence>
          </xs:complexType>
</xs:schema>
```

그림 3 본 논문에서 제시한 방법으로 변환결과(세부변환과정생략)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
          <xs:element name="customerDoc" type="rootType"/>

          <xs:complexType name="rootType">
                    <xs:sequence>
                    <xs:element name="CUSTOMER" type="CUSTOMERType"
                    minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
          </xs:complexType>
          <xs:complexType name="CUSTOMERType">
                    <xs:sequence>
                    <xs:element name="Addr" type="xs:string"/>
                    <xs:element name="Phone" type="xs:string"/>
                    <xs:element name="Name" type="xs:string"/>
                    <xs:element name="SSN" type="xs:string"/>
                    <xs:element name="ACCOUNT" type="ACCOUNTType"
                    minOccurs="0" maxOccurs="unbounded"/>
                    <xs:element name="LOAN" type="LOANType"
                    minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
          </xs:complexType>
          <xs:complexType name="ACCOUNTType">
                    <xs:sequence>
                    <xs:element name="Type" type="xs:string"/>
                    <xs:element name="Balance" type="xs:string"/>
                    <xs:element name="AccNo" type="xs:string"/>
                    <xs:element name="AcctDate" type="xs:string"/>
                    <xs:element name="BankAddr" type="xs:string"/>
        <!--엘리먼트가 중복됨-->
                    <xs:element name="Bankcode" type="xs:string"/>
                    <xs:element name="BankName" type="xs:string"/>
                    <xs:element name="BankPhone" type="xs:string"/>
          <xs:element name="Bankbranch" type="xs:string"/>
                    </xs:sequence>
          </xs:complexType>
          <xs:complexType name="LOANType">
                    <xs:sequence>
                    <xs:element name="Loandate" type="xs:string"/>
                    <xs:element name="Amount" type="xs:string"/>
                    <xs:element name="LoneNo" type="xs:string"/>
                    <xs:element name="Type" type="xs:string"/>
                    <xs:element name="BankAddr" type="xs:string"/>
        <!--엘리먼트가 중복됨-->
                    <xs:element name="Bankcode" type="xs:string"/>
                    <xs:element name="BankName" type="xs:string"/>
                    <xs:element name="BankPhone" type="xs:string"/>
          <xs:element name="Bankbranch" type="xs:string"/>
                    </xs:sequence>
          </xs:complexType>
</xs:schema>
```

그림 4 elmasri 방식으로 변환(세부변환과정생략)