ER2XML : 개체-관계 모델을 기반으로한 XML Schema 생성기의 구현

김 창 석[†] · 손 동 철[††]

요 약

XML이 웹 상에서 문서 교환의 표준으로 자리잡고 있으며 그 수요가 나날이 증가하고 있다. 그에 따라 XML 데이터나 문서 구조를 모델링하는 XML Schema(W3C XML Schema Spec) 또한 수요가 증가하고 있다. 그러나 XML Schema는 다양한 자료형과 풍부한 표현력을 제공하지만 그 복잡성으로 인해 모델링하기가 어려운 단점이 있다. 본 논문에서는 관계형 데이터베이스 설계의 기본적인 도구인 개체-관계 모델을 이용하여 XML Schema를 간단하게 생성하는 방법을 제시한다. 개체-관계 모델과 변환될 XML Schema의 구조는 서로 일대일로 매핑되지 않아 직접 변환할 수는 없다. 그래서 몇 가지 알고리즘을 이용하여 개체 관계 모델을 계층적 구조모델로 변환을 한다. 이렇게 변환된 계층적 구조 모델을 이용하여 최종적으로 XML Schema를 생성한다. 기존의 XML Schema 생성 방법은 개체 간의 속성이 상위 혹은 하위로 이동하면서 최초 설계시의 개체가 사라지므로 XML Schema의 중요한 특성인 재사용성을 이용할 수 없다는 단점을 가진다. 여기서 제시한 알고리즘은 XML Schema의 중요한 특성들인 재사용성, 전역 및 지역 기능 등을 가진 문서를 생성한다는 것이다.

ER2XML : An Implementation of XML Schema Generator based on the Entity-Relationship Model

Chang Suk Kim[†] · Dong-Cheul Son[††]

ABSTRACT

The XML is emerging as standard language for data exchange on the Web. Therefore a demand of XML Schema(W3C XML Schema Spec.) that verifies XML document becomes increasing. However, XML Schema has a weak point for design because of its complication despiteof various data and abundant expressiveness. This paper shows a simple way of design for XML Schema using a fundamental means for database design, the Entity-Relationship model. The conversion from the Entity-Relationship model to XML Schema can not be directly on account of discordance between the two models. So we present some algorithms to generate XML Schema from the Entity-Relationship model. The algorithms produce XML Schema codes using a hierarchical view representation. An important objective of this automatic generation is to preserve XML Schema's characteristics such as reusability, global and local ability, ability of expansion and various type changes.

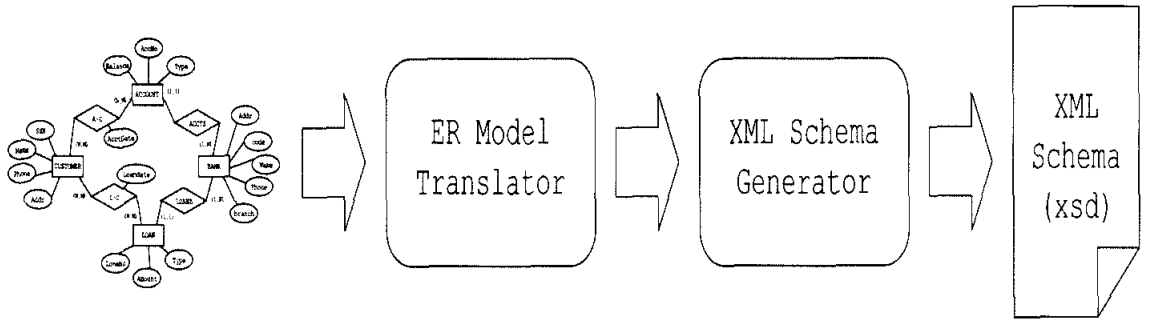
키워드 : XML Schema(XML Schema), 개체-관계 모델(Entity-Relationship model), XML 문서(XML document), 재사용성(Reusability)

1. 서 론

XML(eXtensible Markup Language) 문서는 인터넷상에서 데이터를 표현하고 교환하는 새로운 표준으로 등장하고 있다. XML로 문서를 작성할 때 문서 구조를 기술하는 모델 언어 또는 스키마라 할 수 있는 DTD(Document Type Definition)를 이용하여 문서나 데이터를 모델링한다. 그러나 DTD는 XML 문서와는 그 문법이 다르고 XML 네임스페이스를 제대로 지원하지 못하며, 데이터 형식에 대한 지원이 미흡하고 내용 모델을 기술하는데 한계를 가진다. 그래서 최근에는 DTD 보다 강력하고 융통성 있는 스키마 언어인 XML Schema(W3C XML Schema Spec)를 사용하는 추세이며, 표준으로도 확정되어가고 있다.

XML Schema는 풍부한 표현의 문서를 좀더 쉽게 처리할 수 있도록 여러 가지 자료형을 제공한다. XML Schema 자체가 XML로 되어 있어서 DTD의 EBNF 형태의 다른 구문을 배울 필요가 없고, DOM, XSLT등 XML 도구들을 그대로 사용할 수 있다. 또한 XML Schema 네임스페이스 권고안을 완전히 지원하고, 복잡한 내용모델과 재사용 가능한 내용모델을 쉽게 만들 수 있으며 객체 상속 및 형식 대체와 같은 개념들을 모델링 할 수 있게 해주는 장점들을

※ 본 연구는 한국과학재단 목적기초연구 R01-2002-000-00068-0와 정보통신부지원 정보통신기초기술연구지원사업 04 기초 018 지원을 받았음.
† 정 회 원 : 공주대학교 컴퓨터교육과
†† 정 회 원 : 천안대학교 정보통신공학부
논문접수 : 2004년 7월 30일, 심사완료 : 2004년 12월 28일



(그림 1) 개체-관계 모델에서 XML Schema로의 변환 과정

가진다[1, 2].

그러나 XML Schema는 다양한 자료형과 풍부한 표현력을 제공하지만 그 복잡성으로 인해 설계하기가 어렵고 복잡하다. 즉 어떤 복잡한 데이터나 문서를 XML Schema의 특성인 전역 및 지역 기능, 확장성과 다양한 자료형 등을 활용하여 설계하기는 쉽지 않다. 지금까지 DTD를 설계하는 방법은 제안된 것이 있으나[3], XML Schema를 체계적으로 설계하는 방법은 제시된 것이 거의 없다[4]. 본 연구의 동기는 대부분의 사람들이 쉽게 접근할 수 있는 개체-관계(Entity-Relationship model) 모델을 이용하여 XML Schema로 표현하고자 하는 문서나 데이터 모델을 설계하면 자동으로 복잡한 XML Schema를 생성해 내도록 하는 것이다.

개체-관계 모델과 변환될 XML Schema의 구조는 서로 일대일로 매핑되지 않아 직접 변환할 수는 없다. 개체-관계 모델에서 XML Schema 문서를 자동 생성하기 위해서 개체-관계 모델을 XML 문서의 특징인 계층형으로 변환해야 한다. 개체 관계 모델을 XML Schema 생성을 용이하게 하기 위해서 의미 변화를 최소화 하는 범위 내에서 계층적 구조 모델로 변환한다. 계층형으로 변환된 개체-관계 모델을 변환 규칙과 제약사항을 이용하여 XML Schema 문서(xsd)로 생성한다. 본 논문에서는 개체-관계 모델에서 XML Schema로 XML 문서나 데이터를 쉽게 모델링할 수 있는 알고리즘을 제안하고 구현한다(그림 1). 그리고 기존의 방법과 비교하여 본 논문에서 제시한 알고리즘은 XML Schema의 중요한 특성들인 재사용성, 전역 및 지역 기능 등을 가진 문서를 생성한다는 것을 보인다.

본 변환과정에서의 중요한 사항은 개체-관계 모델을 그대로 XML Schema 로 생성하는 것이 아니라 중복된 참조가 있을 때 중복된 개체가 없도록 XML Schema의 중요한 특징 중의 하나인 재사용성을 사용하여 생성한다는 것이다. 이전에 발표되었던 생성방법은 개체간의 병합 및 속성이동(migration)을 하므로 XML Schema의 계층 구조가 단순해지는 이점은 있지만, 중복된 관계가 있을 경우 하나의 XML Schema 문서 내에서 같은 내용을 두 번 이상 정의해야 하는 경우가 생길 수 있다[5]. 그러나 본 논문에서 제시하는 생성과정은 XML Schema의 특성 중의 하나인 재사용성을 이용하여 중복 참조 개체가 있을 경우에 지명 모델 그룹을 사용하여 XML Schema를 생성해 낸다. 그래서 XML Schema의 중요한 특성들인 재사용성, 전역 및 지역 기능 등을 가진 문서를 생성한다.

2. 관련 연구

XML 연구분야 중에서 XML 스키마인 DTD나 XML Schema(W3C XML Schema Spec)를 생성하는 연구는 이 분야에서 중요한 비중을 차지하고 있다. 관계형 데이터베이스 스키마를 DTD로 변환하는 연구는 UCLA 대학의 EXPRESS, 독일 응용과학대학(Univ of Applied Sciences)의 DB2XML, Stanford 대학의 Lore 프로젝트, 그 외에 SilkRoute, XTRACT 등 현재 활발히 진행되고 있다[6, 7, 8, 9, 10]. 최근에는DTD를 대체하기 위해 개발된 XML Schema가 표준화되어감에 따라 관계형 데이터베이스를 XML Schema로 변환하는 연구도 진행되고 있다[5, 11, 12, 13].

복잡한 XML Schema를 어떻게 하면 쉽게 생성할 수 있을까 하는 연구는 텍사스대학(알링턴 소재)의 Elmasri가 시작하였다[5]. Emasri의 연구는 개체-관계 모델 구조에서 XML Schema를 생성하는 연구를 하였으나 다음과 같은 제약사항을 가진다. 그래프 형태의 개체 관계 다이어그램을 계층구조 형태의 트리로 만들면서 개체들 간의 속성이동으로 인해 최초 설계시의 개체 형태가 사라지게 된다. 즉, 처음의 개체-관계 다이어그램이 훼손된 다른 형태의 계층구조 형태의 트리가 만들어 진다는 것이다(5.2절 참조). 개체들 간의 속성이동으로 인해 개체가 사라지면 XML Schema의 큰 특징 중의 하나인 객체지향 개념이 사라진다. 즉, XML Schema의 전역 및 지역 기능, 재사용성 등을 이용할 수 없게 된다.

본 논문의 차별성은 XML Schema의 전역 및 지역 기능, 재사용성을 보존하면서 개체-관계 모델에서 XML Schema 생성하는 방법을 제시하는 것이다. 그리고 제안된 알고리즘을 자바로 구현하여 생성된 XML Schema 코드와 Elmasri 방법으로 생성된 코드를 비교하여, 제안된 방법이 전역 기능 및 지역 기능과 재사용성이 있음을 보인다.

3. 개체-관계 모델에서 계층적 구조 모델로 변환

3.1 전체 구성도

개체-관계 모델에서 XML Schema 코드를 생성하는 ER2XML의 전체 구성도는 (그림 2)와 같이 크게 계층 구조 변환부와 XML Schema 코드 생성부로 이루어진다.

■ 계층 구조 변환부

설계된 개체-관계 모델을 XML Schema 생성을 용이하게 하기 위해서 의미 변화를 최소화 하는 범위 내에서 개체-관계 모델을 계층적 구조 모델로 변환한다. BFSScan 모듈은 그래프 형태인 개체 관계 모델을 계층형으로 변환하기 위한 탐색 순서를 구한다. 계층구조변환 모듈은 앞에서 구한 탐색순서를 이용하여 그래프 형태를 트리 형태로 변환한다. 대응제약조건 변환모듈은 관계(relationship)의 대응제약조건을 분석하여 개체만 존재하는 트리 구조를 만든다. 간략화 모듈은 불필요한 개체를 제거한다.

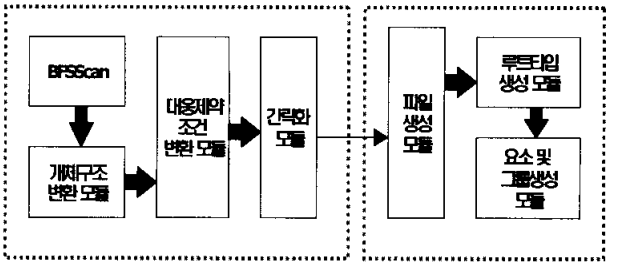
■ XML Schema 코드 생성부

변환된 계층적 구조 모델을 이용하여 XML Schema코드를 생성한다. 파일생성 모듈은 코드를 생성할 파일과 스키마 선언부를 생성한다. 루트타입 생성 모듈은 최상위 루트와 각 루트 타입을 생성한다. 요소 및 그룹생성 모듈은 각 개체의 개체를 검사하여 개수만큼 요소 및 그룹요소를 생성한다.

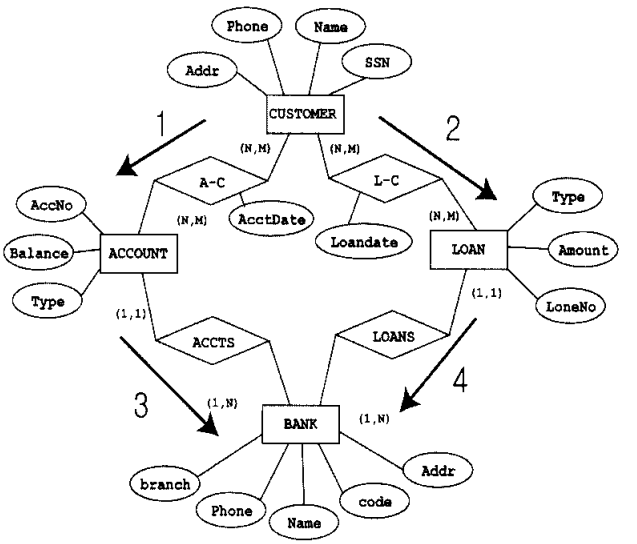
3.2 변환의 개요

본 변환과정에서의 중요한 사항은 개체-관계 모델을 그대로 XML Schema 로 생성하는 것이 아니라 중복된 참조가 있을 때 중복된 개체가 없도록 XML Schema의 중요한 특징 중의 하나인 재사용성을 사용하여 생성한다는 것이다. 이전에 발표되었던 생성방법[4]은 개체간의 병합 및 속성이동을 통해 XML Schema의 계층 구조가 단순해지는 이점은 있지만 중복된 관계가 있을 경우 하나의 XML Schema 문서 내에서 같은 내용을 두 번 이상 정의해야 하는 경우가 생길 수 있다. 본 생성과정은 XML Schema의 특성 중의 하나인 재사용성을 사용하여 중복 참조 개체가 있을 경우에 지명 모델 그룹을 사용하여 XML Schema를 생성해 낸다.

일반적으로 관계형 데이터베이스 설계시 사용되는 개체 관계 모델은 (그림 3)와 같이 그래프 구조로 되어 있다. 관계형 데이터베이스는 그 구조가 평면적(flat) 구조이므로 개체 관계 모델을 관계형 데이터베이스 스키마로 설계할 때에는 매우 용이하다. 그러나 XML Schema는 계층 구조로 되어 있어 개체-관계 모델과는 그 형태는 유사하지만 일치하지는 않기 때문에 XML Schema 생성 이전에 개체-관계 모델을 계층적 구조 모델로 변환하는 단계를 거쳐야 한다.



(그림 2) ER2XML 전체구성도



(그림 3) BFS 스캔

또한 계층 구조의 특성은 하나의 항목은 서로 관련 있는 것들을 내포한다. 다시 말해 부모/자식 관계를 맺고 있다. 그러므로 개체-관계 모델에서 계층적 구조 모델을 생성해 내기 위하여 평면구조의 개체-관계 모델에서 계층적 구조의 특성을 찾아내야 한다. 이 계층적 구조의 특성을 찾아내기 위해 각 개체 간의 대응 제약조건을 이용한다.

개체-관계 모델을 XML Schema로 표현하기 위해 그래프 형태로 되어있는 개체-관계 모델구조를 계층형 구조로의 변환이 필요하다. 이 계층형 구조로의 변환을 위해 미리 결정된 최초 탐색 요소로부터 너비우선탐색(BFS)과 대응 제약 조건을 이용한다. 너비우선탐색방법을 사용하면 중복되어 있는 그래프의 연결고리를 끊어 낼 수 있고 중복 개체를 찾아 낼 수 있으며 개체의 탐색 순서를 결정할 수 있다. 그리고 대응제약조건을 이용하여 상위 개체와 하

위 개체 간의 출현회수를 결정할 수 있다. 다시 말해 XML Schema의 요소출현 지시자(minOccurs, maxOccurs)를 결정할 수 있다.

XML Schema는 같은 내용을 표현하더라도 여러 가지 표현방법이 존재한다. 다시 말해서 XML 문서에 데이터를 넣을 때 요소(element)로 넣을 수도 있고 속성으로 넣을 수도 있다. 그 이외에도 같은 내용이지만 많은 데이터의 표현을 할 수가 있다. 그러므로 본 변환방법에서 사용된 몇 가지 일반적인 제약조건을 제시한다.

- 개체-관계 모델에서의 개체는 지명 복잡 형식으로 전역으로 생성되며 개체-관계 모델에서의 속성은 단순 형식로 생성된다.
- 개체-관계 모델에서 관계는 각 개체 사이의 요소출현지시자를 결정하는데 사용되며 관계(relationship)에 속성이 있다면 개체 사이의 대응제약조건에 의해 상위 개체 혹은 하위 개체에 병합된다.
- 중복된 개체는 중복을 피하기 위해 XML Schema의 지명모델그룹을 이용하여 생성하여 관계가 있는 개체에서 참조하도록 한다.

3.3 BFS를 이용한 탐색

최상위 루트개체가 결정되었다면 결정된 루트 개체를 중심으로 그래프 형태인 개체-관계 모델을 XML Schema의 특징인 계층형 구조로의 생성을 용이하게 하기 위해 계층형 구조로 변환해야 한다. 이를 위해 우선적으로 너비우선탐색 방법을 사용한다(그림 4).

BFS의 탐색 범위는 개체만을 탐색한다. 위에서 결정된 루트 개체로부터 BFS 탐색방법을 사용하여 스캔은 시작되고 모든 개체를 스캔한다. 이때 만일 중복된 개체가 발생을 하면 BFS 실행 이전 지정해 놓은 어떠한 특정 큐에 중복 개체를 등록한다. 이로써 개체-관계 모델에서 중복 참조된 개체를 알아 낼 수가 있다.

BFS 탐색을 위해 두 개의 큐(bfsQue, duplicateQue)를 준비한다. bfsQue는 탐색 순서 및 이전에 탐색했는지 여부를 알려주는 큐이고 duplicateQue는 만일 중복이 된 개체가 있을 경우 그 개체를 등록할 큐이다. 탐색과정은 (그림 3)와 같다.

위의 BFS 탐색 과정을 거치면서 duplicateQue를 통해 어떤 개체가 중복되었는지 알 수 있고 bfsQue 큐를 통해서 개체-관계 모델의 탐색 순서를 알 수가 있다.

(그림 3)에서는 BFS 탐색 과정을 통해 BANK 개체가 중복 참조되었음을 알 수가 있고 탐색 순서가 CUSTOMER, ACCOUNT, LOAN, BANK라는 것을 알 수가 있다. 여기서 탐색 순서가 CUSTOMER, LOAN, ACCOUNT, BANK로 되어도 개체-관계 모델의 의미 변화는 없으므로 ACCOUNT를 먼저 탐색해도 되고 LOAN을 먼저 탐색해도 된다.

```
Algorithm BFSScan
지정된 루트 개체로부터 탐색 시작;
do
{
지정된 개체로부터 탐색 시작;
        if (중복이 나타난다면){
                지정한 큐(duplicateQue)에 중복 개체 등록;

        }else{
                큐(bfsQue)에 현재 개체 등록;
        }
}while(탐색중이면);
```

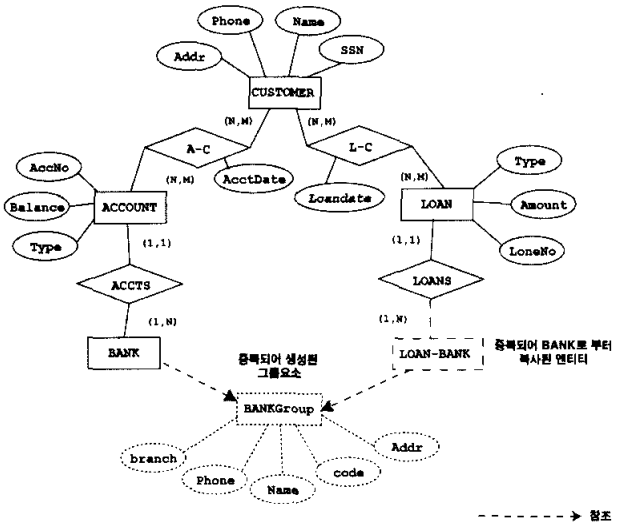
(그림 4) Algorithm BFSScan

3.4 계층 구조로 변환

(그림 3) 과정을 통해 개체의 탐색 순서가 정해졌으므로 정해진 순서에 의해 모든 개체의 개수만큼 반복(loop)을 통하여 중복 참조된 개체의 그룹요소를 생성하고 복사한다. 만일 중복 참조된 개체가 반복을 통하여 최초로 발견이 된다면 그룹요소를 '개체명Group'의 이름으로 생성을 하게 되고 현재 개체의 속성들은 삭제되며, 삭제된 속성들은 생성된 그룹요소의 속성이 되고 현재의 탐색 중인 개체는 생성된 그룹요소를 참조하게 된다. 또한 중복 참조된 개체가 반복를 통하여 두 번째 이상 발견되면 그 중복 참조된 개체를 '부모 개체명-개체명'의 이름으로 복사를 하고 그 복사된 개체는 이전에 생성된 그룹개체를 참조하게 한다. 여기서 현재의 부모 개체와 복사된 개체의 관계는 이전에 설정되었던 현재 부모 개체와 자식 개체의 관계로 설정한다.

(그림 6)의 탐색과정을 거치면 (그림 5)와 같이 된다. (그림 6)에서 현재 개체가 ACCOUNT일 때 처음으로 중복 개체인 BANK 개체를 스캔하게 된다. 그러면 BANK 개체는 위의 탐색과정 중의 조건에 따라서 새로운 그룹요소가 BANKGroup의 이름으로 생성이 된다. 그리고 BANK 개체의 속성인 branch, phone, name, code, addr 속성은 생성된 BANKGroup의 속성으로 이동하게 되고 BANK 개체는 생성된 BANKGroup 그룹요소를 참조하게 된다.

계속 스캔이 진행되면서 현재의 개체가 LOAN일 때 다시 한번 중복노드인 BANK를 스캔하게 된다. BANK 개체는 ACCOUNT일 때 최초로 스캔되었으므로 위에 탐색과정에 따라 LOAN-BANK의 이름으로 기존의 BANK 노드를 복사하게 된다. 그리고 복사된 LOAN-BANK개체는 새로 생성된 BANKGroup를 참조하게 된다. LOAN 개체와 BANK 개체를 복사한 LOAN-BANK와의 관계는 최초 관계를 설정했을 때의 LOAN 개체와 BANK 개체의 관계를 따른다. 또한 생성된 LOAN-BANK 개체는 처음에 설정해 놓은 큐(bfsQue)에 등록되게 되어 개체의 총 개수는 4개에서 5개로 늘어나게 된다.



(그림 5) 계층적 구조 모델로의 변환(1)

```
algorithm GenerateHierView
{
 for(개체의 총 개수){
   if(현재 개체의 자식 개체가 중복 개체이면){
     if(처음 중복이면){
        새로운 group 요소 생성;
        현재 개체의 자식 개체의 속성을
          생성된 group 요소로 이동;
        자식개체는 생성된 group 요소 참조;
     }else{
        중복된 개체 복사;
        중복된 개체는 이전에 생성된 group 요소
            참조;
     }
   }
 }
```

(그림 6) Algorithm GenerateHierView

3.5 개체-관계 모델 관계 정리

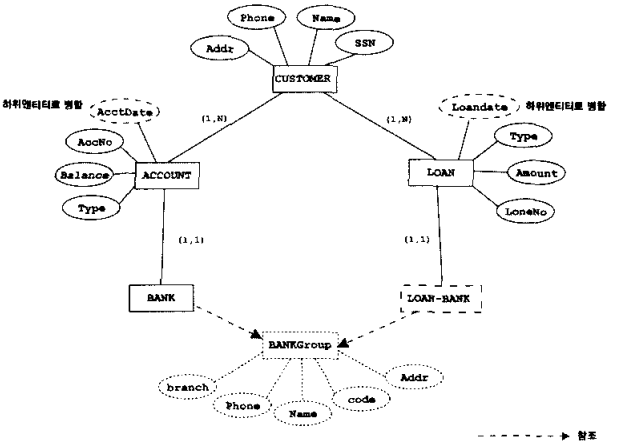
계층적 구조로 변환된 개체-관계 모델의 개체 변화를 최소화하는 범위 내에서 릴레이션쉽 관계를 정리한다. 각 개체 사이의 릴레이션쉽의 대응제약조건을 따라 실시하며 두 가지로 분류하여 실시한다.

만일 상위 개체와 하위 개체 사이의 대응제약조건이 M:N 혹은 1:N일 때 만일 M:N이라면 1:N으로 변환되며 만일 릴레이션쉽에 속성이 있을 경우에는 하위개체에 병합된다. 그리고 만일 상위 개체와 하위 개체 사이의 대응제약 조건이 N:1 혹은 1:1일 때 만일 N:1이라면 1:1로 변환되며 만일 릴레이션쉽에 속성이 있을 경우에는 상위 개체에 병합된다.

(그림 7)을 살펴보면 루트 개체인 CUSTOMER 개체로부터 릴레이션쉽 관계정리는 시작된다. 현재 개체가 CUSTOMER일 때 자식개체는 ACCOUNT, LOAN 두 개가 존재한다. 먼저 위에서 정의한 순서에 따라 CUSTOMER 개체에서 ACCOUNT 개체 사이의 관계를 먼저 정리하게 된다.

여기서 CUSTOMER 와 ACCOUNT 사이에는 A-C 릴레이션쉽이 존재하며 두 개체 사이의 관계는 N:M이다. 그리고 A-C 릴레이션쉽에는 AcctDate라는 속성이 존재한다. 이 두개체 사이의 관계를 위의 프로시져의 동작에 따라 관계를 정리하게 되면 CUSTOMER 개체와 ACCOUNT 개체 사이의 관계는 M:N이므로 1:N으로 변화되고, A-C 릴레이션쉽의 AcctDate 속성은 하위 개체인 ACCOUNT 개체에 병합되게 된다.

현재 개체가 CUSTOMER 개체일 때 또 다른 자식 개체인 LOAN 개체와 관계를 정리하게 되면 두 개체 사이에는 L-C 릴레이션쉽이 존재하며 L-C 릴레이션 쉽에는 LoanDate라는 속성이 존재한다. 이 두 개체 사이의 관계에서는 두 개체 사이의 관계가 M:N이므로 1:N으로 변환되고 L-C의 애트리 뷰트인 LoanDate 속성은 하위 개체인 LOAN 개체에 병합되게 된다.



(그림 7) 계층적 구조모델로의 변환(2)

```
Algorithm RelationshipMerge
 for(개체의 총 개수){
    if(현재 개체와 하위 개체의 관계가 M:N 혹은 1:N이라
면){
       if(M:N 관계라면){
      의미의 변화가 없는 1:N 관계로 변환한다.
       }
    만일 개체-관계 모델에서 관계의 속성이 있으면 하위 개체
에 병합된다.
    }else{          // N:1 or 1:1 관계라면
       if(N:1 관계라면){
    의미의 변화가 없는 1:1 관계로 변환한다.
       }
        만일 개체-관계 모델에서 관계에 속성이 있으면 현재
개체에 병합된다.
    }
}
```

(그림 8) Algorithm RelationshipMerge

위의 loop 문을 실행하면서 현재 개체가 위에서 정의한 순서에 따라 ACCOUNT가 되게 된다. ACCOUNT의 자식 개체는 BANK가 존재하며 ACCTS라는 릴레이션 쉽이 있다. 두 개체 사이의 관계는 1:1이므로 관계의 변화는 없고

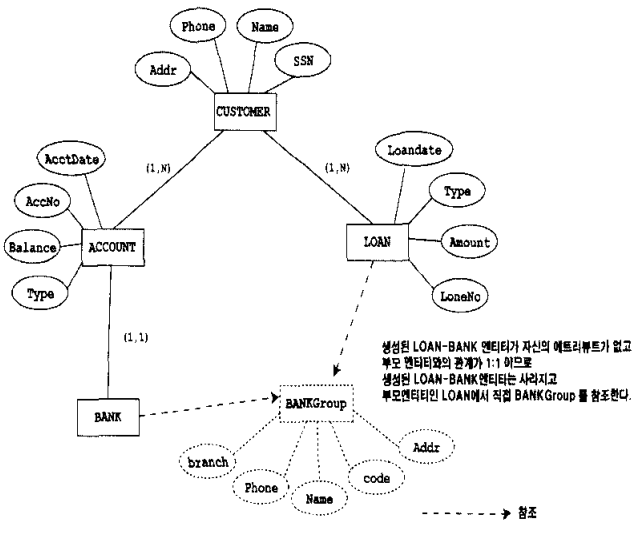
ACCTS 릴레이션쉽에 속성이 없으므로 변환은 없다.

위의 loop 문을 실행하면서 현재 개체가 위에서 정의한 순서에 따라 LOAN 개체가 되면 개체의 자식 개체는 LOAN-BANK이다. 두 개체 사이에는 LOANS라는 릴레이션쉽에 존재하고 릴레이션쉽에 속성은 존재하지 않으며 그 사이의 관계는 1:1이다. 두 개체 사이의 관계를 정리하게 되면 관계가 1:1이므로 관계에 대한 변환은 없으며 릴레이션쉽에 속성은 존재하지 않으므로 병합되는 속성은 없다. 위의 loop 과정을 거치면서 생성되는 계층적 구조 모델은 (그림 7)과 같다.

만일 여기서 개체의 속성을 마이그레이션 하게 되면, XML Schema의 고유한 특성인 전역 및 지역기능을 최대한 살릴 수 없고 만약에 중복된 개체가 있을 경우 같은 내용을 두 번 이상 정의를 할 가능성이 있기 때문에 개체에서의 마이그레이션은 하지 않는다.

■ 불필요한 개체 제거

위의 변환과정들을 거치며 생성된 계층형 구조에서 생성된 개체들 중에 단순히 자신의 고유한 속성들 없이 그룹요소만을 참조만 하는 개체가 존재할 수 있다. 만일 자신의 속성들이 없고 상위 개체와의 관계가 1:1인 경우에는 생성된 해당 개체를 제거하고 해당 개체가 참조하였던 그룹요소는 상위 개체에서 직접 참조하게 한다. 최종적으로 변환된 과정은 (그림 9)와 같다.



(그림 9) 계층적 구조 모델로의 변환(3)

4. 계층형 구조 모델을 XML Schema로 변환

본 장에서는 계층적 구조 모델에서 XML Schema를 변환하는 방법을 기술한다. 동일한 데이터 요소로 여러 가지 형태의 XML Schema를 생성할 수 있기 때문에 다음과 같은 제약사항을 가정한다.

- 개체-관계 모델에서 변환된 계층적 구조 모델에서 개체는 지명복잡형식으로 전역으로 생성한다. 이때 타입은 '개체명Type'으로 정의한다.
- 개체-관계 모델에서 변환된 계층적 구조 모델에서 속성은 단순 형식로 생성된다. 만일 현재 개체에 자식 개체가 존재한다면 전역으로 선언될 하위 개체를 포함시킨다.
  만일 참조할 그룹요소가 있다면 참조 그룹요소를 참조한다.
- 개체간의 대응제약조건은 각 생성될 요소의 요소출현지시자를 결정하는데 사용된다. 만일 1:1 관계이면 minOccurs="1" maxOccurs="1"로 생성되고 1:N 관계이면 minOccurs="1" maxOccurs="unbounded"으로 생성한다.
- 그룹요소가 존재한다면 그룹요소를 전역으로 선언한다.

위의 제약사항을 계층적 구조 모델에 적용하여 XML 스키마를 아래와 같이 생성한다.

4.1 XML Schema 파일 생성 및 Schema 선언

XML Schema를 생성할 때 다른 URI와 결합된 동일한 이름의 다른 요소와 속성과는 구별될 수 있게 하기 위하여 접두사를 사용한다. 앞으로 생성될 XML Schema 에서는 모든 요소에 접두사를 사용하며 본 예제에서는 'xs:'라는 접두사를 사용한다.

모든 XML Schema 문서의 루트 요소는 schema 요소가 선언되어야 하므로 처음 여는 schema 태그에는 XML Schema 권고안에 대한 네임스페이스를 아래와 같이 선언해 준다.

개략적인 알고리즘은 다음과 같다.

```
Algorithm createXmlSchemaDoc
{
 XML Schema 파일생성;
 write("<?xml                          version="1.0"
encording="UTF-8"?>");
 write("<xs:schema xmlns:xs= http://www.w3.org
                          /2001/XMLSchema">");
}
```

아래는 위의 알고리즘에 의해 XML Schema로 생성한 일부 코드이다.

```
<?xml version "1.0" encording="UTF-8"?>
<xs:schema xmlns:xs=
                    "http://www.w3.org/2001/XMLSchema">
.....
</xs:schema>
```

4.2 최상위 루트타입 생성

개체-관계 모델에서 계층적 구조 모델로 변환된 모델을

이용하여 XML Schema로 생성해 내기 위해 처음으로 필요한 요소는 XML Schema의 최상위 루트 요소를 정의하는 것이다. 최상위 요소의 이름요소는 '선택한 개체명Doc'라 한다. type은 'rootType'으로 하여 앞으로 정의될 'rootType'을 포함한다.

위의 예제에서는 최상위 루트 요소는 편의상 개체 이름에 "Doc"를 붙여 "CUSTOMERDoc"라 칭하기로 한다.

개략적인 알고리즘은 다음과 같다.

```
algorithm docCteateRootType
{
 write("<xs:element name="[선택한 루트개체명Doc]"
                          type="rootType"/>");
}
```

아래는 위의 알고리즘에 의해 XML Schema로 생성한 일부 코드이다.

```
.....
<xs:element name="CUSTOMERDoc" type="rootType"/>
.....
```

4.3 루트타입 생성

XML Schema 문서를 위한 루트 요소를 정의하며 이것은 전체 스키마 문서에 이름을 설정하는 것에 대한 내용을 설명한다. XML Schema 문서를 위한 루트 요소를 생성할 때에는 지명복잡형식으로 생성하며 생성할 지명 복잡 형식의 요소의 타입은 'rootType'로 하고 complexType으로 정의한다. 컴퍼지터는 <sequence>로 정의하고 요소를 루트 개체명으로 정의한다. 아래 과정에서 전역으로 선언된 루트 개체인 '루트개체명Type'을 포함하게 되며 출현회수는 minOccurs="1" maxOccurs="unbound"로 선언한다.

본 예제에서는 루트타입의 이름을 CUSTOMER 로 정의한다.

개략적인 알고리즘은 다음과 같다.

```
algorithm createRootType
{
 wtite(이름요소의 값이 rootType'으로 된 지명복잡형식
       으로 생성);
 write(전역으로 선언될 루트요소 요소선언);
}
```

아래는 위에서 변환한 예제를 XML Schema로 생성한 일부이다.

```
.....
<xs:complexType name="rootType">
        <xs:sequence>
                <xs:element name="CUSTOMER" type=
                 "CUSTOMERType" minOccurs="1"
                 maxOccurs="unbounded"/>
        </xs:sequence>
</xs:complexType>
.....
```

4.4 개체를 지명 복잡 형식으로 생성

개체-관계 모델에서 계층적 구조모델로 변환된 모델에서 개체의 개수만큼 위에서 정의한 순서대로 loop 문을 돌면서 각각의 개체를 XML Schema의 요소로 선언하며 개체는 지명복잡형식으로 전역으로 선언한다. 이때 타입은 '개체명Type'으로 정의한다(그림 10).

```
algorithm createEntity
for(개체의 총 개수){
        write(현재의 개체를 지명복잡형식으로 생성);
        write(현재 개체의 속성들을 단순형식 생성);
        if(하위 개체가 존재하면){
                write(하위 개체 이름으로 요소 생성);
        }
        if(참조할 그룹이 있다면){
                write(참조할 그룹요소Group' 의 이름
                      으로 그룹 요소 생성 );
   }
}
```

(그림 10) Algorithm createEntity

- 컴퍼지터는 <sequence>로 선언한다.
- 계층적 구조모델에서의 속성은 단순형식으로 생성한다. name은 속성의 이름으로 정의하고 타입은 "xs:string"으로 정의한다.
- 계층적 구조모델에서 만일 하위 개체가 존재한다면 개체 이름으로 요소를 생성하고 type은 "개체명Type"으로 하며 요소 출현회수는 위의 변환과정에서 정의한 것에 따라 만일 1:1 관계이면 minOccurs="1" maxOccurs="1"로 생성되고 1:N 관계이면 minOccurs="1" maxOccurs="unbounded"으로 생성한다.
- 계층적 구조모델에서 만일 참조할 그룹요소가 있다면 그룹요소를 생성하고 ref는 '참조할 그룹요소Group'로 정의하고 생성한다.

이것을 알고리즘으로 나타내면 다음과 같다.

```
algorithm createRootType
{
 wtite(이름요소의 값이 'rootType' 으로 된 지명복잡형
       식으로 생성);
 write(전역으로 선언될 루트요소 요소선언);
}
```

4.5 그룹요소 생성

만일 계층적 구조모델에서 그룹요소가 존재하면 그룹요소의 개수만큼 그룹요소를 생성하고 이름은 '그룹요소명Group'로 정의한다. 컴퍼지터는 <sequence>로 선언한다. 그룹요소의 속성은 단순 형식으로 생성한다. name은 속성의 이름으로 정의하고 타입은 "xs:string"으로 정의한다. 개략적인 알고리즘은 아래와 같다.

```
algorithm createGroup
for(그룹의 총 개수){
        write('그룹요소명Group'의 이름값으로 그룹요소생성);
}
```

아래는 위에서 변환한 예제를 XML Schema로 생성한 일부이다.

```
<xs:group name="BANKGroup">
    <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:group>
```

4.6 코드 생성

개체-관계 모델에서 계층적 구조모델로 변환된 모델을 이용하여 위에서 제시한 방법으로 XML Schema를 생성해 낼 수 있다. 아래는 위에서 제시한 방법으로 위에서의 예제를 모두 생성한 결과물이다(그림 11).

```
<?xml version="1.0" encording-"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="un-
bounded">
    </xs:sequence>                    <!-- 지역 요소 선언
  </xs:complexType>

  <xs:complexType name-"CUSTOMERType">
    <xs:sequence>
      <xs:element name="Ssn" type-"xs:string"/>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Phone" type="xs:string"/>
      <xs:element name="Addr" type="xs:string"/>
      <xs:element name="ACCOUNT" type="ACCOUNTType"
        minOccurs="1" maxOccurs-"unbounded"/>
      <xs:element name="LOAN" type="LOANType" min-
        Occurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ACCOUNTType">
    <xs:sequence>
      <xs:element name="Balance" type="xs:string"/>
      <xs:element name="AccNo" type-"xs:string"/>
      <xs:element name="Type" type="xs:string"/>
      <xs:element name="AcctDate" type="xs:string"/>
      <xs:element name="BANK" type="BANKType" min-
        Occurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="LOANType">
    <xs:sequence>
      <xs:element name="LoanNo" type="xs:string"/>
      <xs:element name="Amount" type="xs:string"/>
      <xs:element name="Type" type-"xs:string"/>
      <xs:element name="LoanDate" type="xs:string"/>
      <xs:group ref="BANKGroup"/> <!-- 전역 그룹 참조
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="BANKType">
    <xs:sequence>
      <xs:group ref="BANKGroup"/> <!-- 전역 그룹 참조
    </xs:sequence>
  </xs:complexType>

  <xs:group name="BANKGroup">       <!-- 전역 그룹 선언으
                                       로 재사용 가능 부분
    <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:group>
</xs:schema>
```

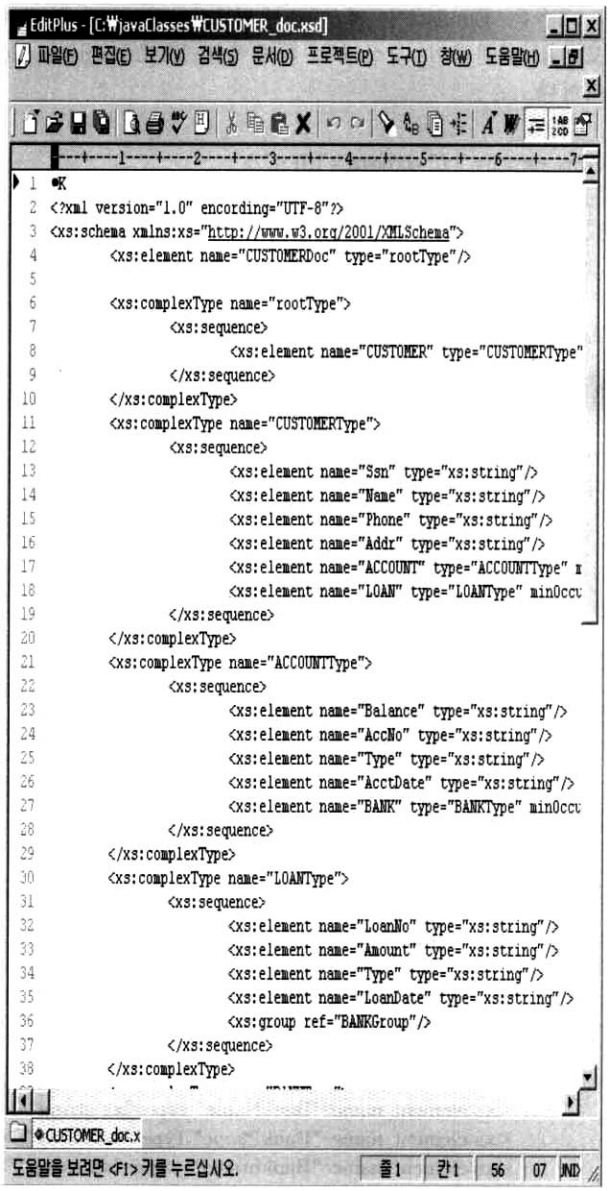
(그림 11) 제안된 알고리즘으로 생성된 XML Schema 코드

5. 구현 및 평가

5.1 구현 결과

본 논문에서 제안한 개체-관계 모델을 계층형 구조 모델로 변환과 계층형 구조 모델을 XML Schema로 생성하는 알고리즘은 자바 언어로 구현하였다. 자바는 버전 JDK 1.4.2를 사용하였고 구현 및 테스트 환경은 아래와 같다[14, 15, 16].

구현은 GenerateXmlSchema 클래스를 이용한다. GenerateXmlSchema 클래스는 개체를 표현해 주는 MakeE 클래스, 관계를 표현하는 MakeR 클래스를 포함한다.



(그림 12) 생성된 CUSTOMER_doc.xsd 파일의 내용

〈표 1〉 테스트 환경

구 분	사양 및 기종
OS	windows 2000 professional
RAM	512MB
CPU	1.5GHz
Programming Environment	JDK 1.4.2

먼저 개체-관계 모델 설계 후 개체-관계 모델을 입력을 받는다. 입력은 개체와 개체의 속성, 릴레이션쉽과 릴레이션쉽의 속성을 받고 입력받은 개체와 개체의 속성은 클래스로 표현하며 릴레이션쉽과 그 속성도 클래스로 표현한다. 개체 및 개체의 속성 그리고 릴레이션쉽과 릴레이션쉽의 속성이 입력될 때 마다 인스턴스를 생성하여 개체-관계 모델에서 설계한 개체와 릴레이션쉽을 표현하였다.

개체를 표현해주는 클래스는 MakeE, 릴레이션쉽을 표현해주는 클래스는 MakeR로 정의하였고 group 요소를 표현하기 위해서 그 형태가 유사한 클래스 MakeE를 사용하였다.

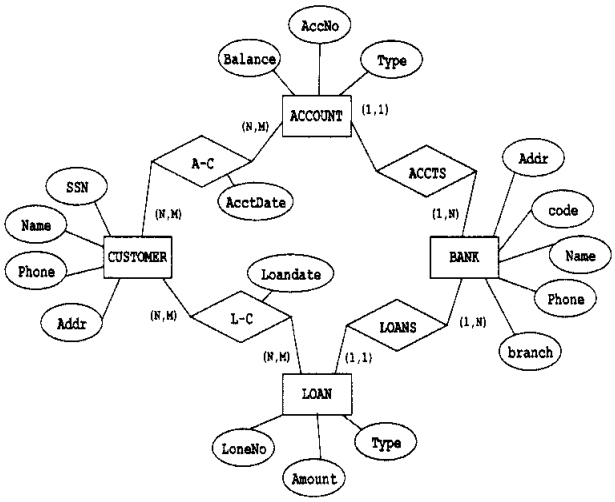
위의 과정에서 인스턴스로 생성된 개체들의 집합은 자바에서의 벡터로서 표현하였다. 벡터는 배열과는 달리 데이터 구조를 쉽게 만들 수 있고 데이터 찾기, 삽입, 삭제 등 데이터를 관리하기 편하게 제작된 클래스이다. 위에서 설명한 개체는 중복된 개체의 생성 및 복사, 삭제에 따라 그 개수가 증가하기도 하고 삭제되기도 하기 때문에 크기가 고정되어 있는 배열을 사용하기 보다는 크기가 탄력적인 벡터를 사용하여 개체집합을 표현하였다. 마찬가지로 릴레이션쉽의 집합, XML Schema에서 그룹요소로 생성된 요소 또한 벡터로 표현하였다.

구현에 대한 자세한 사항은 [14]에 자세하게 기술하였다. (그림 12)은 최종적으로 생성된 CUSTOMER_doc.xsd의 파일의 내용을 나타낸 것이다.

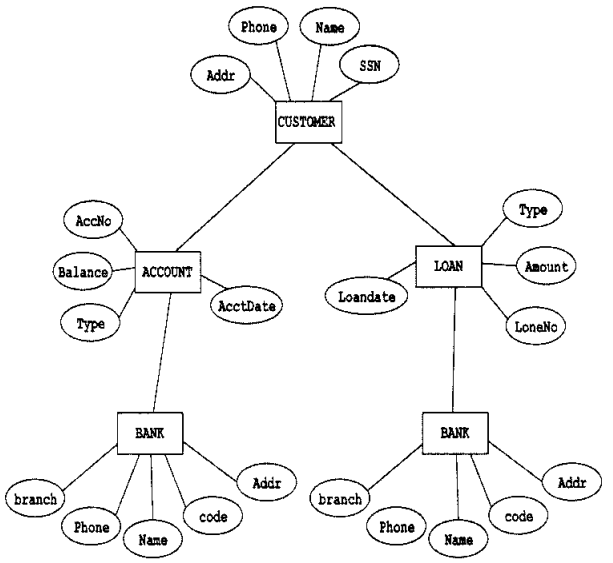
5.2 평 가

여기서는 본 논문에서 제시한 변환방법과 기존에 발표되었던 Elmasri[4]가 제시한 개체-관계 모델에서 XML Schema로의 변환방법에 대해 비교하고 본 논문의 독창성을 보인다. Elmasri의 연구에서 가장 주목할 부분은 개체-관계 모델에서 XML Schema로의 변환시 개체 사이의 대응제약조건을 이용하여 그 조건에 따른 속성의 병합 및 이동이다. 본 논문에서 제시한 개체 관계 모델을 Elmasri의 변환방법에 따라 변환하면 다음과 같다.

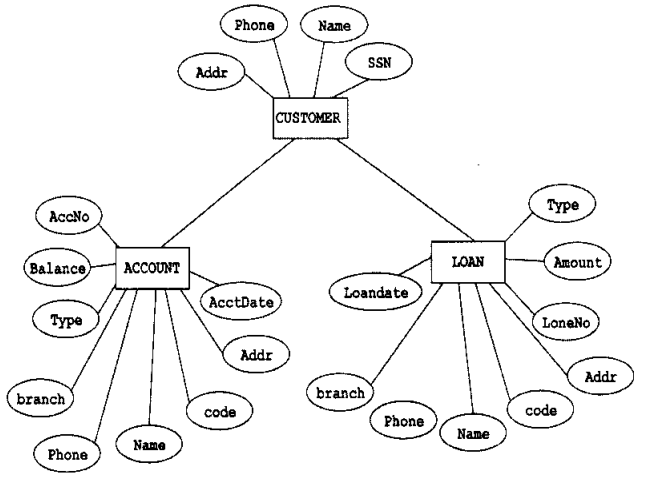
(그림 15)의 Elmasri의 최종 계층적 트리 구조를 보면 BANK 개체가 사라지고 그 속성들은 ACCOUNT와 LOAN에 마이그레이션된 것을 알 수 있다. 이와 같이 개체 간의 속성이 상위 혹은 하위 개체로 이동하면서 최초 설계시의 개체 형태가 사라지게 되면 XML Schema의 특징인 전역기능 및 지역기능을 정의할 수 없어 객체지향개념에서 중요한 재사용성을 활용할 수 없게 된다. 또한 탐색 중에 중복참조 된 개체가 나타나게 되면 복사되기 때문에 하나의 XML Schema 문서에서 같은 내용으로 구성된 개체이지만 중복되는 개체를 두 번 이상 정의하는 경우가 발생한다. 즉, Elmasri가 제시한 변환방법은 XML Schema 문서가 DTD와 구별되는 가장 기본적인 특징인 전역 및 지역 기능 및 재사용성을 전혀 고려하지 않고 XML Schema 코드 생성에만 치중한 것을 볼 수 있다. <표 2>는 Elmasri와 본 논문의 방법 비교하여 정리한 것이다.



(그림 13) 변환할 개체-관계 모델



(그림 14) Elmasri의 계층적 트리 구조로의 변환 과정



(그림 15) Elmasri 의 최종 계층적 트리 구조

다음은 Elmasri가 제시한 방법으로 위의 개체-관계 모델을 변환하여 생성되는 최종적인 XML Schema 문서이다.

```
<?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"/>
      <!--요소가 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"/>
       <!  요소가 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 16) Elmasri 알고리즘으로 생성된 XML Schema 코드

〈표 2〉 Elmasri와 본 논문의 방법 비교

	Elmasri 방법	본 논문의 방법
재사용성	없음	있음. 그룹 요소로 정의한 후, 다른 요소가 참조 함
전역 및 지역 기능	구분 없음	구분 있음
개체의 수	중복되는 개체는 상위 개체에 합병되므로 중복되는 개수만큼 줄어듦	중복되는 개체는 전역으로 사용되므로 중복되는 개수만큼 늘어남

6. 결 론

지금까지 본 논문에서는 XML Schema의 여러 장점에도 불구하고 그 문법이 복잡하고 표현방식이 다양하여 설계 시 어려움이 있었던 XML Schema 설계에 대한 문제를 기존의 데이터베이스 설계의 기본 도구인 개체-관계 모델을 이용하여 설계하는 방법을 제시하였다. 그리고 복잡하고 많은 표현양식을 가지고 있는 XML Schema 문법에 대하여 몇 가지 제약사항을 제시하여 XML Schema 문서를 생성해 내었고 XML Schema 의 장점인 재사용성을 활용하고 전역 및 지역 기능을 활용하여 설계하는 방법을 제안하고 서술하였다.

또한 설계한 개체-관계 모델을 입력받아 자동으로 조건에 따라 XML Schema 문서를 생성해 내는 변환 시스템을 자바로 구현하였다. 마지막으로 본 논문에서 제시한 방법과 기존의 Elmasri가 제시한 방법을 이용한 XML Schema 출력 결과를 비교하여, 여기서 제시한 방법이 전역 및 지역 기능, 재사용성 등의 특성을 포함한 XML Schema 문서를 생성함을 보였다.

참 고 문 헌

[1] Jon Duckett 외 8인 공저, "Professional XML Schemas", Wrox, 2001.

[2] Kevin Williams외 9인, "Professional XML Databases", Wrox, 2001.

[3] Garsten Kleiner and Udo Lipeck, "Automatic Generation of XML DTDs from Conceptual Database Schemas", *Informatik 2001-Wirtschaft und Wissenschaft in der Network Economy*-Visionen und Wirklichkeit, pp.396-405, 2001.

[4] Antonio Badia, "Automatic Generation of XML DTD from Relation : The Nested Relation Approach", *ER 2003 Workshops on Conceptual Modeling for Novel Application Domains*, pp.330-341, 2003.

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

[6] V. Turau, "Making Legacy Data Accessible for XML Applications", http://www.informatik.fh-wiesbaden.de/~turau/DB2XML, 1999.

[7] Mary F. Fernandez, Wang Chiew Tan, Dan Suciu, "SilkRoute : trading between relations and XML", Computer Networks, Vol.33, No.1-6, pp.723-745, 2000.

[8] Minos Garofalakis, Aristides Gionis, Rajeev Rastogi, S. Seshadri and Kyuseok Shim, "XTRACT : A System for Extracting Document Type Descriptors from XML Documents", *Proceedings of SIGMOD*, pp.165-176, ACM Press, 2000.

[9] Dongwon Lee, "Net & CoT : Translating Relational Schemas to XML Schema using Semantic Constraints", *Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management*, p. 282-291, 2002.

[10] Dongwon Lee, "Schema Conversion Methods between XML and Relational Models", *Knowledge Transformation for the Semantic Web*, p. 1-17, 2003.

[11] 김형석, 허보진, 김창석, "EER 다이어그램을 이용한 XML 스키마 설계방법", *한국정보처리학회 추계학술발표대회*, pp.1509-1512, 2003.

[12] Mark Vermeer and Peter Apers, "Reverse Engineering of Relational Database Applications", *Proceedings of 14th International Conference on Object-Oriented and Entity-Relationship Modelling*, pp.89-100, ACM Press, 1995.

[13] Chang Suk Kim, Dae Su Kim and Kwang-Baik Kim, "A Generation of XML Schema from Entity-Relationship Models", *Proceedings of SCIS and ISIS 2004*, Japan Society for Fuzzy Theory and Intelligent Informatics (SOFT), p. 139-143, 2004.

[14] 김형석, 김창석, "개체 관계 모델을 이용한 XML Schema의 생성", TD 2004 03, p.81, 2004.

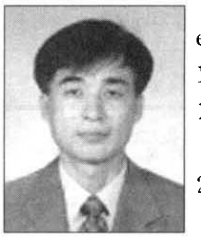
[15] Walter Savitch, "Java", Prentice Hall. 2000.

[16] Ellis Horowitz, Sartaj Sahni and Susan Anderson Freed, *Fundamentals of Data Structures in C*, Computer Science Press, 1993.

김 창 석

e-mail : csk@kongju.ac.kr
1983년 경북대학교 전자공학과(학사)
1990년 경북대학교 대학원 전자공학과(석사)
1994년 경북대학교 대학원 컴퓨터공학과(박사)
2000년~2001년 캘리포니아대학(샌디에고) 전산학과(박사후 연수)
1983년~1994년 한국전자통신연구원 선임연구원
1998년~현재 공주대학교 컴퓨터교육과 부교수
관심분야 : XML 데이터베이스, 인터넷 정보통합시스템, 지능형 데이터베이스 등

손 동 철



e-mail : dcson@cheonan.ac.k
1983년 경북대학교 전자공학과(학사)
1985년 경북대학교 대학원 전자공학과 (석사)
2001년 충북대학교 대학원 정보통신공학과 (박사)
2002년~현재 천안대학교 정보통신공학부 부교수
관심분야 : 모바일 통신, XML 데이터베이스, 인터넷 정보시스템