관계형 데이터 모델 기반 군사용 데이터베이스의 XML 데이터베이스로의 변환

김창석* · 김응수**

목 차

1. 서론

웹이 활성화됨에 됨에 웹상에서의 문서 교환도 활발해 지게 되었다. 그런데 각각의 고유한 문서 편집기로 작성된 문서는 해당 문서

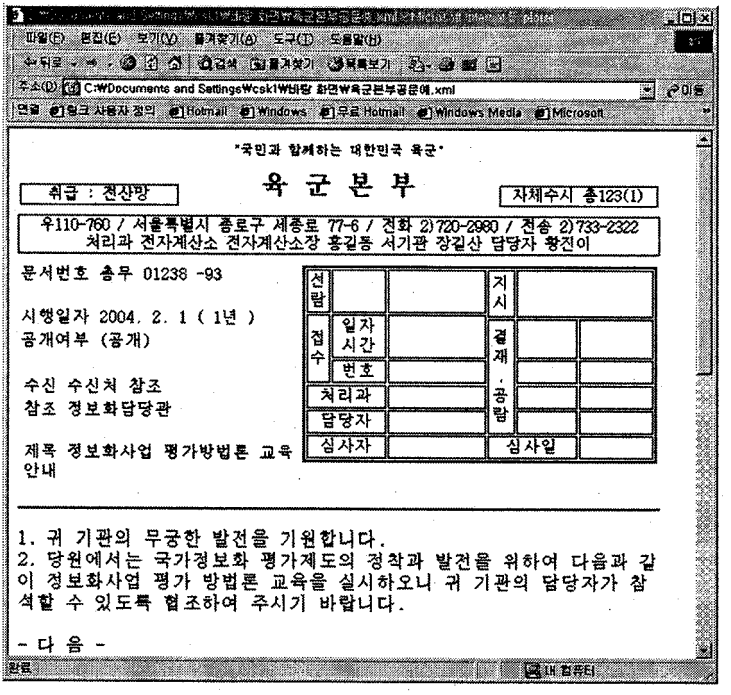
* 공주대학교 컴퓨터교육과 교수
** 대전대학교 컴퓨터전자공학과 교수

편집기가 있어야만 볼 수 있는 단점을 가진다. 그래서 웹상에서 데이터를 표현하고 교환하는 새로운 표준이 등장하게 되었다. 이러한 표준으로 XML(eXtensible Markup Language)[1]이 국제표준으로 자리잡고 가고 있다.

[그림 1.1]은 국제 공통 표맷인 군사 문서의 한 예를 나타낸 것이다. 이것은 작성된 편집기에 상관없이 XML로 변환되어 웹 브라우저를 통해 어디에서나 볼 수 있다. 저장된 문서나 데이터를 공통 포맷인 XML 문서로 변환하기 위해서는 먼저 XML 스키마인 DTD나 최근 표준으로 부상한 XML Schema를 먼저 작성해야 한다. 즉, 관계형 데이터베이스에 저장된 데이터를 XML 데이터로 변환하기 위해서는 그 스키마부터 변환해야 한다는 것이다.

XML Schema(W3C XML Schema Spec)는 DTD를 대체하기 위해 개발되었다. 이것은 강력하고 융통성 있는 언어로서 작성자가 문서에 대해 DTD보다 더 다양하게 활용할 수 있다. XML Schema는 풍부한 표현의 문서를 좀더 쉽게 처리할 수 있게 하는 데이터형을 제공한다. 그리고 XML Schema 자체가 XML로 되어 있어서 DTD의 EBNF 형태의 다른 구문을 배울 필요가 없고, DOM, XSLT 등 XML 도구들을 그대로 사용가능 하다.

그러나 XML Schema는 DTD에 비해 기능이 다양하고 복잡하여 설계가 쉽지 않다. 또한 설계를 위한 방법도 제안된 것이 많지 않다. 본 논문에서는 대부분의 군사 문서나 데이터가 관계형 데이터베이스에 저장된 것에 착안하여 관계형 데이터베이스 설계 시 기본적으로 사용되는 개체-관계(Entity-Relationship, 이하 E-R) 다이어그램으로 표현된 스키마를 XML Schema로 변환하는 방법을 제시하고 구현한다. 그래서 관계형 데이터베이스로 저장된 대용량의 데이터를 XML 데이터로 자동으로 변환하고자 한다.


"국민과 함께하는 대한민국 육군"

육 군 본 부

취급 : 전산망

자체수시 총123(1)

우110-760 / 서울특별시 종로구 세종로 77-6 / 전화 2)720-2980 / 전송 2)733-2322
처리과 전자계산소 전자계산소장 홍길동 서기관 장길산 담당자 황진이

문서번호 총무 01238 -93

시행일자 2004. 2. 1 (1년)
공개여부 (공개)

수신 수신처 참조
참조 정보화담당관

제목 정보화사업 평가방법론 교육 안내

선람			지시	
접수	일자 시간		결재		
	번호				
처리과			공람		
담당자					
심사자			심사일		

1. 귀 기관의 무궁한 발전을 기원합니다.
2. 당원에서는 국가정보화 평가제도의 정착과 발전을 위하여 다음과 같이 정보화사업 평가 방법론 교육을 실시하오니 귀 기관의 담당자가 참석할 수 있도록 협조하여 주시기 바랍니다.

- 다 음 -

[그림 1.1]은 국제 공통 표맷인 군사 문서의 한 예

2. 관련연구

2.1 스키마 변환에 관한 연구

관계형 데이터베이스를 DTD로 변환하는 연구는 캘리포니아 대학의 EXPRESS, 독일 응용과학대학의 DB2XML, 스탠포드 대학의 Lore 프로젝트 등으로 현재 활발히 진행되고 있다. 이 후 DTD를 대체하기 위해 개발된 XML Schema가 개발됨으로써 관계형 데이

터베이스를 XML Schema로 변환하는 연구가 진행되고 있으나 Elmasri의 연구 외에는 알려진 것이 없다[4]. Elmasri의 연구에는 E-R 다이어그램 구조가 엔티티들간의 마이그레이션으로 인해 XML Schema 문서가 간략해지는 장점이 있지만 참조해야할 뚜렷한 엘리먼트가 사라지므로, 하나의 엔티티에 중복된 관계가 있을 경우 같은 내용을 두 번 이상 정의해야 하는 경우가 발생하며 XML Schema의 큰 특징 중의 하나인 확장성 및 재사용성을 이용하는 데에 제약을 받게 된다. 본 논문에서는 기존 연구의 문제점을 보안하여 XML Schema의 전역 및 로컬 기능 및 재사용성을 활용하여 E-R 다이어그램에서 XML Schema 설계하는 방법을 제시한다.

2.2 XML

XML(Extensible Markup Language)이 등장한 것은 얼마 되지 않았지만 현재 사용되는 여러 애플리케이션들을 강력하게 지원하고 있어서 다른 기술들에 비해 급속히 성장하고 있다. 특히 데이터를 관리하고 조직화해야 하는 것은 물론이고, 보여주기까지 해야 하는 웹 사이트들이나 이러한 기능을 제공해야 하는 여러 애플리케이션에서 많이 사용되고 있다. XML은 디스플레이 언어인 XSL과 표준화된 문서 객체 모델인 DOM과 함께 웹에서 마크업 언어를 사용하기 위한 필수적인 기술이다.

XML은 플랫폼과 언어로부터 독립적이다. 이것은 어떤 컴퓨터가 XML을 사용해도 상관없다는 말이다. 예를 들면 마이크로소프트 운영체제에서 동작하는 비주얼베이직과 자바 코드를 가진 유닉스 운영체제 어느 쪽에서라도 다 사용할 수 있다. 실제로 컴퓨터 프로그

램이 다른 프로그램과 의사소통을 요구하는 어떤 때라도 XML은 데이터 교환의 형식으로 적합하다.

웹 기반의 애플리케이션들은 웹 서버의 부하를 줄이기 위해 XML을 사용할 수 있다. 이것은 가능한 모든 정보를 클라이언트에 저장함으로써 가능하다. 그리고 클라이언트는 이런 서버들에 하나의 커다란 XML 문서의 형태로 정보를 보내주게 된다.

W3C는 XML을 사용하여 그들의 규격을 기록하고 있다. 이런 XML 문서들은 디스플레이를 위해서 HTML로 변환(XLST에 의해)할 수 있다. 또는 여러 가지 다양한 다른 형식으로도 변환할 수 있다. 전통적으로 HTML을 사용해 왔던 몇몇 웹 사이트 들 또한 그들의 내용 형식을 전부 XML로 바꾸어 사용하기도 한다. 이 XML은 XSLT를 사용하여 HTML로 변환할 수 있고, CSS를 통해 브라우저에서 직접 보여 줄 수도 있다. 심지어 웹 서버들은 정보를 요구하는 브라우저가 어떤 종류의 브라우저인지까지도 동적으로 결정하여 어떤 작업을 할지 결정할 수도 있다. 이러한 작업을 통해 서버 부하는 감소된다.

또한 XML은 원격 프로시저 호출(Remote Procedure Calls, RPC)에도 사용할 수 있다. RPC는 하나의 컴퓨터에 있는 객체가 분산 컴퓨팅 작업을 하기 위해서 다른 컴퓨터에 있는 객체를 호출하는 것을 허용하는 프로토콜이다. RPC 호출을 위해 XML과 HTTP를 사용하는 SOAP(Simple Object Access Protocol)은 방화벽이 있어도 문제없이 사용할 수 있다.

XML은 전자상거래에 활용된다. 회사들은 전통적인 방법 대신에 인터넷을 이용하여 비용을 절감하고 일 처리를 빨리 할 수 있다. 또한 회사는 그들의 프로세서를 좀더 매끄럽게 할 수 있다. 한 회사가 다른 회사로 데이터를 전송하려고 할 때 데이터 교환 형식으로 XML을 사용하면 언제나 문제없이 원하는 작업을 할 수 있다.

XML Schema에 대하여 알아보자.

```
<?xml vE-Rsion="1.0"?>
<CustomE-R>
<FirstName>Raymond</FirstName>
<MiddleInitial>G<MiddleInitial>
<LastName>BayLiss</LastName>
</CustomE-R>
```

위와 같이 스키마를 따르는 문서를 인스턴스 문서라 한다. XML Schma 파일은 .xsd 라는 확장자로 저장한다.

```
<?xml vE-Rsion="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema ">
  <element name="CustomE-R">
    <complexType>
      <sequence>
        <element name="FirstName" type="string"/>
        <element name="MiddleInitial" type="string"/>
        <element name="Lastname" type="string"/>
      </sequence>
    </complexType>
  </element>
</schema>
```

스키마는 그 자체로 XML문서이다. 모든 XML Schema 문서의 루트요소는 스키마 요소가 되어야 한다. 여는 스키마 태그에는 XML Schema 권고안에 대한 네임 스페이스를 선언한다.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema ">
```

다음 줄은 가장 첫 요소인 CustomE-R요소를 선언하는 방법이

다.

```
<element name="CustomE-R">
....
</element>
```

element 속성을 사용하여 요소들을 선언하며 이 요소에 부여할 이름은 name 이라는 속성 값에 지정한다.

```
<sequence>
  <element name="FirstName" type="string"/>
  <element name="MiddleInitial" type="string"/>
  <element name="Lastname" type="string"/>
</sequence>
```

sequence 요소 안에 선언된 요소들은 이 스키마를 따르는 문서에서 정확하게 이와 동일한 순서로 나와야만 한다. sequence는 컴퍼지터(compositor) 라고 부르며 complexType요소의 내부에는 필수적으로 컴퍼지터를 지정해야 한다.

이 요소 선언은 그 값으로 string을 갖는 자료형태 속성을 지니고 있다.

▪ 단순형식(simple type)

속성 값으로 나타날 수 있는 텍스트, 또는 텍스트만을 가지는 요소 내용을 제한한다.

```
<element name="FirstName" type="string"/>
<element name="MiddleInitial" type="string"/>
<element name="Lastname" type="string"/>
```

▪ 복잡 형식(complex type)

한 요소가 지닐 수 있는 속성들과 한 요소가 포함할 수 있는 자식 요소들을 정의한다. 한 요소가 속성이나 자식 요소를 가질 수 있도록 하려면 반드시 복잡 형식을 정의해야 한다. 아래 선언은 복잡 형식이 될 필요가 있다. 여기서 선언된 CustomE-R요소의 내부에 중첩되어 있는 complexType요소를 사용하여 CustomE-R 요소를 복잡형식으로 만들어 준다. 요소 선언 안에 또 다른 요소 선언들은 중첩 시킬수 없다.

```
<element name="FirstName" type="string"/>
<element name="MiddleInitial" type="string"/>
<element name="Lastname" type="string"/>


<complexType>
  <sequence>
    <element name="FirstName" type="string"/>
    <element name="MiddleInitial" type="string"/>
    <element name="Lastname" type="string"/>
  </sequence>
</complexType>
```

위에서 정의한 복잡 형식은 익명 복잡 형식(anonymous complex type)이라고 부른다. 그 이유는 복잡 형식이 요소 선언(이 경우엔 CustomE-R) 내부에 중첩되어 있기 때문이다.

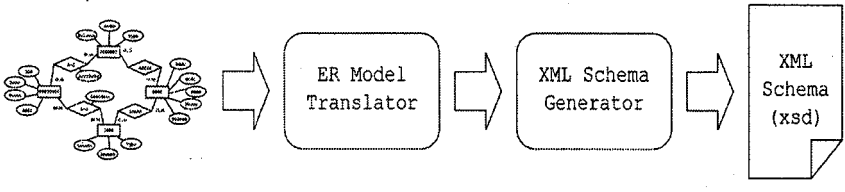
```
<?xml vE-Rsion="1.0"?>
<CustomE-R customE-RID="24332">
  <FirstName>Raymond</FirstName>
  <MiddleIntial>G</MiddleInitial>
  <LastName>Bayliss</LastName>
</CustomE-R>
```

속성을 추가하려면 complexType정의 안에 선언한다. 속성 선언은 sequence 컴퍼지터 태그를 닫은 다음 complexType 태그를 닫기 전에 해준다.

```
<?xml vE-Rsion = "1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema  ">
  <element name="CustomE-R">
    <complexType>
      <sequence>
        <element name="FirstName" type="string"/>
        <element name="MiddleInitial" type="string"/>
        <element name="Lastname" type="string"/>
      </sequence>
      <attribute name="customE-RID" Type="intE-RgE-R"/>
    </complexType>
  </element>
</schema>
```

3. E-R 모델에서 계층적 구조 모델로 변환

E-R 모델을 XML Schema 문서로 생성하기 위해서는 우선 계층적 구조 모델로 변환해야 한다. 계층형으로 변환된 E-R 모델을 최종 XML Schema 문서(xsd)로 생성해 내어야 한다. [그림 3.1]은 변환 과정을 나타낸 것이다.



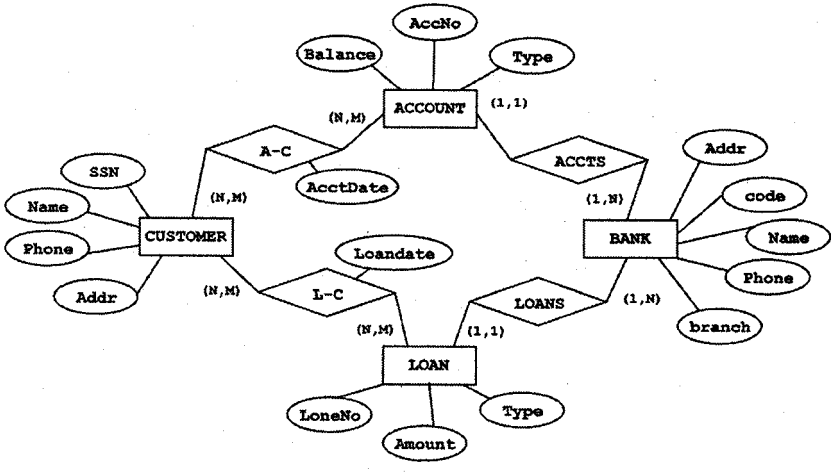
[그림 3.1] E-R 모델을 XML Schema 문서로 변환 과정

• E-R 모델 변환기 : 설계된 E-R 모델을 XML 스키마 생성을 용이하게 하기 위해서 의미 변화를 최소화 하는 범위 내에서 E-R 모델을 계층적 구조 모델로 변환하는 모듈.

• XML Schema 생성기 : 변환된 계층적 구조 모델을 이용하여 XML Schema로 생성해내는 모듈.

본 변환과정에서의 중요한 포인트는 E-R 모델을 그대로 XML Schema 로 생성하는 것이 아니라 중복된 참조가 있을 때 중복된 엔티티가 없게 XML Schema 의 중요한 특징 중인 하나인 재사용성을 사용하여 생성한다는 것이다. 이전에 발표되었던 생성 방법[2]은 엔티티간의 병합 및 마이그레이션을 통해 XML Schema 의 계층구조가 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 Schema 의 특성중의 하나인 재사용성을 사용하여 중복 참조 엔티티가 있을 경우에 지명모델그룹(model group)을 사용하여 XML Schema를 생성해 낸다.

일반적으로 관계형 데이터베이스 설계시 사용되는 E-R 모델은 그래프(graph) 구조로 되어 있다. 관계형 데이터베이스는 그 구조가 플랫한 구조이므로 E-R 모델을 관계형 데이터베이스 스키마로 설계할 때에는 매우 용이하다. 그러나 XML Schema는 계층구조로 되어 있어 E-R 모델과는 그 형태는 유사하지만 일치하지는 않기 때문에 XML Schema 생성 이전에 E-R 모델을 계층적 구조 모델로 변환하는 단계를 거쳐야 한다.



[그림 3.2] 입력용 E-R 모델

또한 계층 구조의 특성은 하나의 항목(요소:element)은 서로 관련 있는 것들을 내포(nest)한다. 다시 말해 부모/자식(parent/child)관계를 맺고 있다. 그러므로 E-R 모델에서 계층적 구조 모델을 생성해 내기 위하여 flat 한 E-R 모델에서 계층적 구조의 특성을 찾아내야 한다. 이 계층적 구조의 특성을 찾아내기 위해 각 엔티티 간의 대응 제약 조건을 이용한다.

E-R 모델을 XML Schema로 표현하기 위해 그래프 형태로 되어 있는 E-R 모델구조를 계층형 구조로의 변환이 필요하다. 이 계층형 구조로의 변환을 위해 미리 결정된 최초 탐색 엘리먼트로 부터 BFS(너비우선탐색 : Breath First Scan) 탐색방법과 와 대응 제약조건(mapping cardinality)을 이용한다. BFS 탐색방법을 사용하면 중복되어 있는 그래프의 연결고리를 끊어 낼 수 있고 중복 엔티티를 찾아 낼 수 있으며 엔티티의 탐색 순서를 결정할 수 있다. 그리고 대응제약조건을 이용하여 상위 엔티티와 하위 엔티티 간의 출현 회수를 결정할 수 있다. 다시 말해 XML Schema의 요소출현지시자

(minOccurs, maxOccurs)를 결정할 수 있다.

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

- E-R 모델에서의 엔티티는 지명복잡형식(named complexType)으로 전역으로 생성되며 E-R 모델에서의 애트리뷰트는 단순 형식(Simple Type)로 생성된다.
- E-R 다이어그램에서 릴레이션쉽은 각 엔티티 사이의 요소출현지시자를 결정하는데 사용되며 릴레이션쉽에 애트리뷰트가 있다면 엔티티 사이의 대응제약조건에 의해 상위 엔티티 혹은 하위 엔티티에 병합 된다.
- 중복된 엔티티는 중복을 피하기 위해 XML Schema 의 지명모델그룹(model group)을 이용하여 생성하여 관계가 있는 엔티티에서 참조하도록 한다.

3.1 루트 엔티티 선택

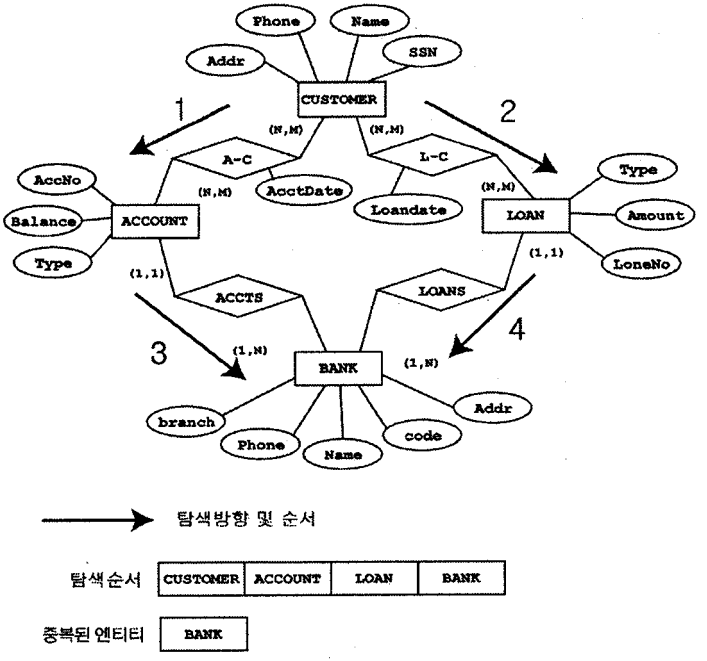
E-R 모델을 계층형 구조로 나타내기 이전에 먼저 E-R 모델의 엔티티들 중에서 어떤 엔티티를 중심으로 XML Schema를 표현할 것인가를 결정해야 한다. 이 루트 엔티티 선택은 사용자마다 관심있어 하는 분야가 다르기 때문에 사용자가 루트 엔티티를 선택하도록 한다.

이 예에서는 사용자가 루트 엔티티로서 CUSTOME-R 를 선택하였다고 가정한다.

3.2 BFS로 계층형 구조 표현

최상위 루트 엔티티가 결정되었다면 결정된 루트 엔티티를 중심으로 그래프 형태인 E-R 모델을 XML Schema의 특징인 계층형 구조로의 생성을 용이하게 하기 위해 계층형 구조로 변환을 해야 한다. 이를 위해 우선적으로 BFS(Breadth First Search) 탐색방법을 사용한다. BFS의 탐색 범위는 엔티티만을 탐색한다. 위에서 결정된 루트엔티티로부터 BFS 탐색방법을 사용하여 스캔은 시작되고 모든 엔티티(노드)를 스캔한다. 이때 만일 중복된 엔티티(노드)가 발생을 하면 BFS 실행 이전 지정해 놓은 어떠한 특정 큐에 중복 엔티티(노드)를 등록한다. 이로써 E-R 모델에서 중복참조된 엔티티를 알아 낼 수가 있다. 또한 BFS 탐색방법을 사용하여 E-R 모델을 스캔함으로써 루트 엔티티로부터 최하위 엔티티까지의 탐색 순서를 알아 낼 수 있다.

BFS 탐색을 위해 두개의 큐(bfsQue, duplicateQue)를 준비한다. bfsQue 는 탐색 순서 및 이전에 탐색했는지 여부를 알려주는 큐이고 duplicateQue는 만일 중복이 된 엔티티가 있을 경우 그 엔티티를 등록할 큐이다. 탐색 방법은 아래와 같다.



[그림 3.3] BFS 스캔

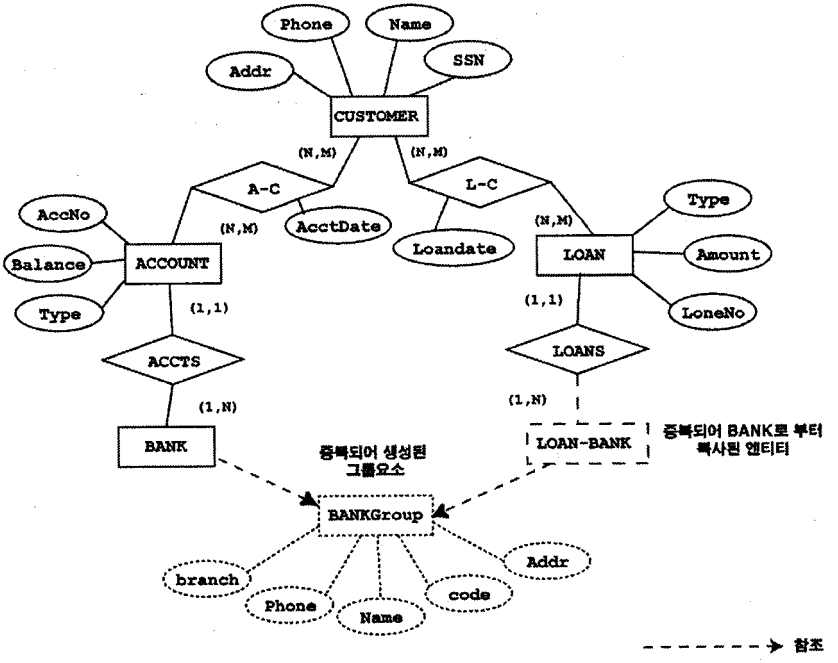
```
지정된 루트 엔티티로부터 탐색 시작;
do
{
지정된 엔티티로부터 탐색 시작;
        if (중복이 나타난다면){
                지정한 큐(duplicateQue)에 중복 엔티티 등록;

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

위의 BFS 탐색 과정을 거치면서 duplicateQue를 통해 어떤 엔티티(노드)가 중복되었는지 알 수 있고 bfsQue 큐를 통해서 E-R 모델의 탐색 순서를 알 수가 있다. 위의 예제에서는 BFS 탐색 과정을 통해 BANK 엔티티가 중복참조 되었음을 알 수가 있고 탐색순서가 CUSTOME-R, ACCOUNT, LOAN, BANK 라는 것을 알 수가 있다. 여기서 탐색순서가 CUSTOME-R, LOAN, ACCOUNT, BANK 로 되어도 E-R 모델의 의미 변화는 없으므로 ACCOUNT를 먼저 탐색해도 되고 LOAN을 먼저 탐색해도 된다.

위의 과정을 통해 엔티티(노드)의 탐색 순서가 정해졌으므로 정해진 순서에 의해 모든 엔티티(노드)의 개수만큼 반복(loop)를 통하여 중복 참조된 엔티티의 그룹요소 생성 및 복사를 한다. 만일 중복 참조된 엔티티가 loop를 통하여 최초로 발견이 된다면 그룹요소를 [엔티티명Group] 의 이름으로 생성을 하게 되고 현재의 엔티티의 애트리뷰트들은 삭제되며 삭제된 애트리 뷰트들은 생성된 그룹요소의 애트리뷰트가 되고 현재의 탐색중인 엔티티는 생성된 그룹요소를 참조하게 된다. 또한 중복 참조된 엔티티가 loop를 통하여 두 번째 이상 발견되면 그 중복참조된 엔티티를 [부모엔티티명-엔티티명]의 이름으로 복사를 하고 그 복사된 엔티티는 이전에 생성된 그룹엔티티를 참조하게 한다. 여기서 현재의 부모 엔티티와 복사된 엔티티의 관계는 이전에 설정되었던 현재 부모엔티티와 자식 엔티티의 관계로 설정한다.

아래는 탐색 방법을 간략히 기술한 것이다.



[그림 3.4] 계층적 구조 모델로의 변환(1)

for(엔티티의 총 개수){
  if(현재 엔티티의 자식 엔티티가가 중복된 엔티티이면){
    if(처음 중복이면){
      새로운 group 요소 생성;
        //group 이름은 "노드명Group" 로 지정한다.
      현재 엔티티의 자식 엔티티의 애트리뷰트를 생성된 group 요소로 이동;
      자식엔티티는 생성된 group 요소 참조;
    }else{
      중복된 엔티티 복사;
        //이름은 "부모엔티티-자식엔티티"로 한다.
      중복된 엔티티는 이전에 생성된 group 요소 참조;

```
                }
            }
        }
```

위의 탐색과정을 거치면 위의 그림과 같이 된다. 위의 예제에서는 현재 엔티티가 ACCOUNT 일때 처음으로 중복 엔티티인 BANK 엔티티를 스캔하게 된다. 그러면 BANK 엔티티는 위의 탐색과정중의 조건에 따라서 새로운 그룹요소가 "BANKGroup"의 이름으로 생성이 된다. 그리고 BANK 엔티티의 애트리뷰트인 branch, phone, name, code, addr 애트리뷰트는 생성된 BANKGroup 의 애트리 뷰트로 이동하게 되고 BANK 엔티티는 생성된 BANKGroup 그룹요소를 참조하게 된다.

계속 스캔이 진행되면서 현재의 엔티티(노드)가 LOAN 일때 다시 한번 중복노드인 BANK를 스캔하게 된다. BANK 엔티티는 ACCOUNT 일때 최초로 스캔되었으므로 위에 탐색과정에 따라 "LOAN-BANK"의 이름으로 기존의 BANK 노드를 복사 하게 된다. 그리고 복사된 "LOAN-BANK"엔티티는 새로 생성된 BANKGroup를 참조하게 된다. LOAN 엔티티와 BANK 엔티티를 복사한 LOAN-BANK 와의 관계는 최초 관계 설정했을때의 LOAN 엔티티와 BANK 엔티티의 관계를 따른다. 또한 생성된 LOAN-BANK 엔티티는 처음에 설정해 놓은 큐(bfsQue)에 등록되게 되어 엔티티의 총 개수는 4개에서 5개로 늘어나게 된다.

3.3 릴레이션쉽 관계 정리

계층적 구조로 변환된 E-R 모델의 엔티티변화를 최소화하는 범

위 내에서 릴레이션쉽 관계를 정리 한다. 각 엔티티 사이의 릴레이션쉽의 대응제약조건을 따라 실시하며 두 가지로 분류하여 실시한다.

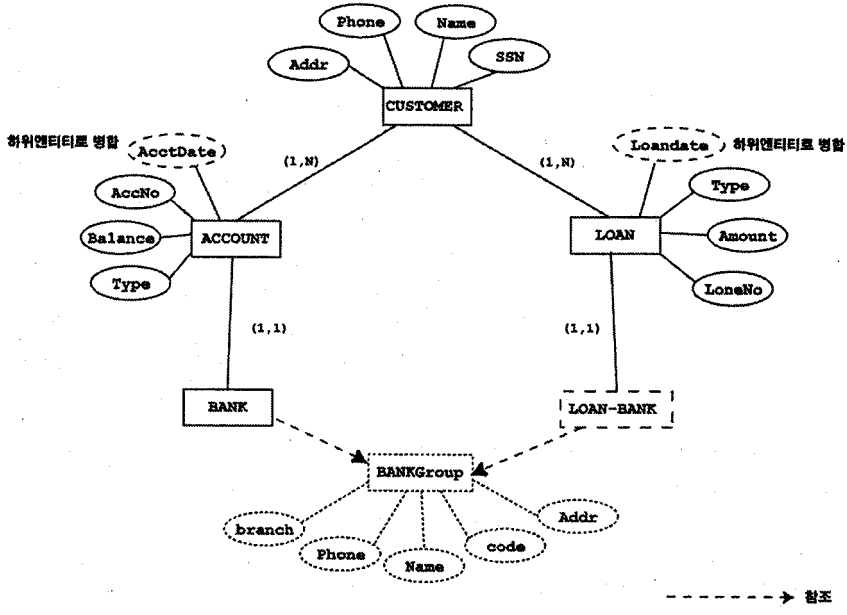
만일 상위 엔티티와 하위 엔티티 사이의 대응제약조건이 M:N 혹은 1:N 인 경우는 1:N으로 변환되며 만일 릴레이션쉽에 애트리뷰트가 있을 경우에는 하위엔티티에 병합된다. 그리고 만일 상위 엔티티와 하위 엔티티 사이의 대응제약 조건이 N:1 혹은 1:1 이라면 1:1로 변환되며 만일 릴레이션쉽에 애트리뷰트가 있을 경우에는 상위 엔티티에 병합된다.

릴레이션쉽 관계정리의 방법은 아래와 같다.

```
for(엔티티의 총 개수){
    if(현재 엔티티와 하위 엔티티의 관계가 M:N 혹은 1:N 이라면){
      if(M:N 관계라면){
          의미의 변화가 없는 1:N 관계로 변환한다.
              }
    만일 E-R 모델에서 Relationship 의 Attribute 가 있다면 하위 엔티
  티에 병합된다.
      }else{                          //        N:1 or 1:1 관계라면
          if(N:1 관계라면){
             의미의 변화가 없는 1:1 관계로 변환한다.
               }
            만일 E-R 모델에서 Relationship 의 Attribute 가 있다면
            현재 엔티티에 병합된다.
      }
}
```

예제를 살펴보면 루트 엔티티인 CUSTOME-R 엔티티로부터 릴레이션쉽 관계정리는 시작된다. 현재 엔티티(노드)가

CUSTOME-R 일때 자식엔티티는 ACCOUNT, LOAN 두개가 존재한다. 먼저 위에서 정의한 순서에 따라 CUSTOME-R 엔티티에서 ACCOUNT 엔티티 사이의 관계를 먼저 정리하게 된다.



[그림 3.5] 계층적 관계 모델로의 변환(2)

여기서 CUSTOME-R 와 ACCOUNT 사이에는 A-C 릴레이션쉽이 존재하며 두 엔티티 사이의 관계는 N:M 이다. 그리고 A-C 릴레이션쉽에는 AcctDate 라는 애트리뷰트가 존재한다. 이 두 엔티티 사이의 관계를 위의 프로시져의 동작에 따라 관계를 정리하게 되면 CUSTOME-R 엔티티와 ACCOUNT 엔티티 사이의 관계는 M:N 이므로 1:N 으로 변화되고, A-C 릴레이션쉽의 AcctDate 애트리뷰

트는 하위 엔티티인 ACCOUNT 엔티티에 병합되게 된다. 현재 엔티티가 CUSTOME-R 엔티티 일때 또 다른 자식 엔티티인 LOAN 엔티티와 관계를 정리하게 되면 두 엔티티 사이에는 L-C 릴레이션쉽이 존재하며 L-C 릴레이션 쉽에는 LoanDate 라는 애트리 뷰트가 존재한다. 이 두 엔티티 사이의 관계에서는 두 엔티티 사이의 관계가 M:N 이므로 1:N 으로 변환되고 L-C의 애트리 뷰트인 LoanDate 애트리뷰트는 하위 엔티티인 LOAN엔티티에 병합되게 된다.

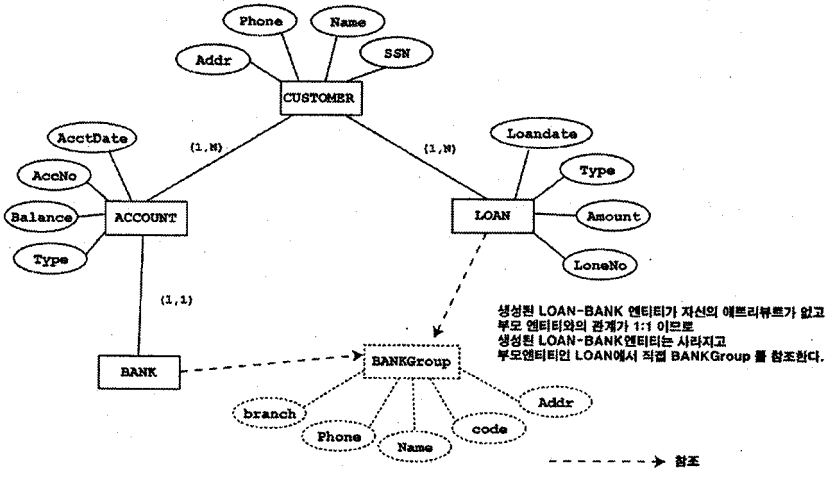
위의 loop 문을 실행하면서 현재 엔티티가 위에서 정의한 순서에 따라 ACCOUNT 가 되게 된다. ACCOUNT의 자식 엔티티는 BANK 가 존재하며 ACCTS 라는 릴레이션 쉽이 있다. 두 엔티티 사이의 관계는 1:1 이므로 관계의 변화는 없고 ACCTS 릴레이션쉽에 애트리 뷰트가 없으므로 변환은 없다. 위의 loop 문을 실행하면서 현재 엔티티가 위에서 정의한 순서에 따라 LOAN 엔티티가 되면 엔티티의 자식 엔티티는 LOAN-BANK 이다. 두 엔티티 사이에는 LOANS 라는 릴레이션쉽에 존재하고 릴레이션쉽에 애트리뷰트는 존재하지 않으며 그 사이의 관계는 1:1 이다. 두 엔티티 사이의 관계를 정리하게 되면 관계가 1:1 이므로 관계에 대한 변환은 없으며 릴레이션쉽에 애트리뷰트가 존재하지 않으므로 병합되는 애트리뷰트는 없다. 위의 loop 과정을 거치면서 생성되는 계층적 구조 모델은 위의 [그림 3.5]와 같다.

만일 여기서 엔티티의 애트리뷰트를 마이그레이션 하게 되면, XML Schema의 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 마이그레이션은 하지 않는다.

3.4 불필요한 엔티티 제거

위의 변환과정들을 거치며 생성된 계층형 구조에서 생성된 엔티티들 중에 단순히 자신의 고유한 애트리뷰트들 없이 그룹요소만을 참조만 하는 엔티티가 존재할 수 있다. 만일 자신의 애트리뷰트들이 없고 상위 엔티티와의 관계가 1:1 인 경우에는 생성된 해당 엔티티를 제거하고 해당 엔티티가 참조하였던 그룹요소는 상위 엔티티에서 직접 참조하게 한다.

위의 조건으로 E-R 모델을 변환하면 위의 [그림 3.6]과 같다.



[그림 3.6] 계층적 구조 모델로의 변환

4. 계층형 구조 모델에서 XML Schema 생성

위에서 E-R 모델을 XML Schema로의 생성을 용이하게 하기 위하여 E-R 모델을 계층형 구조모델로 의미 변환 없게 변환하였다. 또한 XML Schema의 특성중의 하나인 재사용성을 표현하기 위해서 중복된 엔티티를 찾아내어 변환을 하였다. E-R 모델에서 계층적 구조 모델로 변환된 모델을 이용하여 XML Schema를 생성해 낸다. 위에서 기술한 것과 같이 XML Schema 는 같은 데이터적 요소를 표현하는 데에도 여러 가지로 표현해 낼 수 있기 때문에 XML Schema로의 생성을 위해 아래와 같은 몇가지 가이드 라인을 제시한다.

■ E-R 모델에서 변환된 계층적 구조 모델에서 엔티티는 지명 복잡형식(named complexType Type)으로 전역으로 생성한다. 이때 타입은 "엔티티명Type" 으로 정의한다.

■ E-R 모델에서 변환된 계층적 구조 모델에서 애트리뷰트는 단순 형식(Simple Type)로 생성된다. 만일 현재 엔티티에 자식 엔티티가 존재한다면 전역으로 선언될 하위 엔티티를 포함시킨다. 만일 참조할 그룹요소가 있다면 참조 그룹요소를 참조한다.

■ 엔티티간의 대응제약조건은 각 생성될 엘리먼트의 요소출현 지시자를 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 가이드 라인을 이용하여 변환된 계층적 구조 모델에 적용하여 XML 스키마를 아래와 같이 생성해 낼 것이다.

4.1 문서 정의

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

모든 XML Schema문서의 루트 요소는 schema 요소가 선언되어야 하므로 처음 여는 schema 태그에는 XML Schema 권고안에 대한 네임 스페이스를 아래와 같이 선언해 준다.

아래는 위에서 변환한 예제를 XML Schema로 생성한 일부이다.

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

4.2 최상위 루트타입 생성

E-R 모델에서 계층적 구조 모델로 변환된 모델을 이용하여 XML Schema로 생성해 내기 위해 처음으로 필요한 요소는 XML Schema의 최 상위 루트 엘리먼트를 정의하는 것이다. 최상위 루트 엘리먼트는 편의상 엔티티 이름에 "Doc"를 붙여 "CUSTOME-RDoc"라 칭하기로 한다. type 은 "rootType"으로 하여 앞으로 정의될 "rootType"을 포함한다.

아래는 위에서 변환한 예제를 XML Schema로 생성한 일부이다.

<xs:element name="CUSTOME-RDoc" type="rootType"/>

4.3 루트타입 생성

XML Schema 문서를 위한 루트엘리먼트를 정의하며 이것은 전체 스키마 문서에 이름을 주는데 사용한다. 이 엘리먼트의 타입은 "rootType" 로 하고 complexType 으로 정의한다. 컴퍼지터는 <sequence>로 정의하고 엘리먼트를 루트 엔티티인 CUSTOME-R의 이름으로 정의한다. 그리고 아래과정에서 전역으로 선언될 루트 엔티티인 "CUSTOME-RType" 을 포함하게 되며 출현회수는 minOccurs = 0 maxOccurs = "unbound" 로 선언한다.

아래는 위에서 변환한 예제를 XML Schema로 생성한 일부이다.

```
.....
<xs:complexType name="rootType">
                <xs:sequence>
                        <xs:element                    name="CUSTOME-R"
type="CUSTOME-RType" minOccurs="0"          maxOccurs="unbounded">
                </xs:sequence>
        </xs:complexType>
.....
```

4.4 엔티티 생성

E-R 모델에서 계층적 구조모델로 변환된 모델에서 엔티티의 개수만큼 위에서 정의한 순서대로 loop 문을 돌면서 각각의 엔티티를

XML Schema의 엘리먼트로 선언하며 엔티티는 지명복잡형식(named complexType Type)으로 전역으로 선언한다. 이때 타입은 "엔티티명Type" 으로 정의한다. 컴퍼지터는 <sequence>로 선언한다.

계층적 구조모델에서의 애트리뷰트는 단순형식으로 생성한다. name은 애트리뷰트의 이름으로 정의하고 타입은 "xs:string" 으로 정의한다. 계층적 구조모델에서 만일 하위 엔티티가 존재한다면 엔티티 이름으로 요소를 생성하고 type 은 "엔티티명Type"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 정의한다.

아래는 위에서 변환한 예제를 XML Schema로 생성한 일부이다.

```
<xs:complexType name="CUSTOME-RType">
        <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"
minOccurs="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"
minOccurs="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"/>
                </xs:sequence>
        </xs:complexType>

        <xs:complexType name="BANKType">
                <xs:sequence>
                    <xs:group ref="BANKGroup"/>
                </xs:sequence>
        </xs:complexType>
```

4.5 그룹요소 생성

만일 계층적 구조모델에서 그룹요소가 존재하면 그룹요소의 개수만큼 그룹요소를 생성하고 이름은 "그룹요소명Group" 로 정의한다. 컴퍼지터는 <sequence> 로 선언한다. 그룹요소의 애트리뷰트는 단순형식으로 생성한다. name은 애트리뷰트의 이름으로 정의하고 타입은 "xs:string" 으로 정의한다.

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>
```

E-R 모델에서 계층적 구조모델로 변환된 모델을 이용하여 위에서 제시한 방법으로 XML Schema를 생성해 낼 수 있다. 아래는 위에서 제시한 방법으로 위에서의 예제를 모두 생성한 것이다.

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

        <xs:element name="CUSTOME-RDoc" type="rootType"/>

        <xs:complexType name="rootType">
                <xs:sequence>
                        <xs:element                    name="CUSTOME-R"
type="CUSTOME-RType" minOccurs="0" maxOccurs="unbounded">
                </xs:sequence>
        </xs:complexType>

        <xs:complexType name="CUSTOME-RType">
                <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"
minOccurs="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"
minOccurs="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"/>
                </xs:sequence>
        </xs:complexType>

        <xs:complexType name="BANKType">
                <xs:sequence>
                        <xs:group ref="BANKGroup"/>
                </xs:sequence>
        </xs:complexType>

        <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>

</xs:schema>
```

5. 구현

3장과 4장에서 서술된 E-R 모델을 계층형 구조 모델로 변환과 계층형 구조 모델을 XML Schema로 생성의 구현은 자바언어로 구현하였다. 자바는 버전 JDK 1.4.2를 사용하였고 구형 및 테스트 환경은 아래와 같다.

구분	사양 및 기종
OS	windows 2000 propessional
RAM	512MB
CPU	1.5GHz

5.1 개요

구현은 GenE-RateXmlSchema 클래스를 사용한다.

GenE-RateXmlSchema 클래스는 엔티티를 표현해 주는 MakeE 클래스, 릴에이션쉽을 표현하는 MakE-R 클래스를 import 하여 사용한다.

먼저 E-R 모델 설계후 E-R 모델을 입력을 받는다. 입력은 엔티티와 엔티티의 애트리뷰트, 릴레이션쉽과 릴에이션쉽의 애트리뷰트를 받고 입력 받은 엔티티와 엔티티의 애트리뷰트는 클래스로 표현하며 릴레이션쉽과 그 애트리뷰트도 클래스로 표현한다. 엔티티 및 엔티티의 애트리뷰트 그리고 릴레이션쉽과 릴레이션쉽의 애트리뷰트가 입력 될 때 마다 인스턴스를 생성하여 E-R 모델에서 설계한

엔티티와 릴레이션쉽을 표현하였다.

엔티티를 표현해주는 클래스는 MakeE, 릴레이션쉽을 표현해주는 클래스는 MakE-R로 정의하였고 group 요소를 표현하기 위해서 그 형태가 유사한 클래스 MakeE를 사용하였다.

위의 과정에서 인스턴스로 생성된 엔티티들의 집합은 자바에서의 백터로서 표현하였다. 백터는 배열과는 달리 데이터 구조를 쉽게 만들 수 있고 데이터 찾기, 삽입, 삭제등 데이터를 관리하기 편하게 제작된 클래스이다. 위에서 설명한 엔티티는 중복된 엔티티의 생성 및 복사, 삭제에 따라 그 개수가 증가하기도 하고 삭제되기도 하기 때문에 크기가 고정되어 있는 배열을 사용하기 보다는 크기가 탄력적인 백터를 사용하여 엔티티집합을 표현하였다. 마찬가지로 릴레이션쉽의 집합, XML Schema에서 그룹요소로 생성될 요소 또한 백터로 표현하였다.

5.2 클래스 makeE

클래스 MakeE 의 구성은 멤버변수로서 엔티티의 이름을 담는 String형태의 name 멤버변수, 상위 엔티티의 이름을 담는 String 형태의 parentName 멤버변수, 하위 엔티티의 이름을 가지는 벡터 형태의 childDirList 멤버변수, group요소로 생성될 엔티티를 참조하는 경우가 있을 때 그 이름값이 저장되는 백터 형태의 childRefList 멤버변수 등으로 구성되며 메소드로서의 구성은 애트리뷰트들을 입력하는 insE-RtAttr() 메소드, 상위 엔티티를 입력하는 myStatus() 메소드, 하위 엔티티들을 추가하는 insDirect(), 하위 엔티티를 삭제하는 removeDirect(), 생성될 group요소를 참조할 엔티티명을 추가하거나 삭제하는 insRefE-Rence(), removE-RefE-Rence() 메소드

등으로 구성된다.

MakeE 클래스는 엔티티의 생성과 group 요소의 생성을 구분하기 위해 생성자를 구분하여 오버로딩하였고 각각의 변수를 초기화하였다.

클래스 MakeE의 소스코드는 아래와 같다.

```
MakeE.java
import java.util.*;

class MakeE{

        public String name;
        public String myStatus;
        public int id;
        public int step;
        public String parentName;
        public int parentId;
        public String group;

        public Vector attr;
        public Vector childDirList;
        public Vector childDirListRelate;
        public Vector childRefList;

        public MakeE(String name,int id){
                attr = new Vector();
                childDirList = new Vector();
                childRefList = new Vector();
                childDirListRelate = new Vector();

                this.name = name;
                this.id = id;
                myStatus=null;
        }
        public MakeE(String name){
                attr = new Vector();
                childDirList = new Vector();
                childRefList = new Vector();

                this.name = name;
                myStatus=null;
```

```
            this.group="Y";
        }
        public void insE-RtAttr(String name){
            attr.addElement(name);
        }
        public void myStatus(String parentName, int parentId){
            this.parentName = parentName;
            this.parentId = parentId;
        }
        public void myStep(int num){
            this.step = num;
        }
        public void group(){
            this.parentName = null;
            this.parentId = 0;
            this.group="Y";
        }
        public void insDirect(String name){
            childDirList.addElement(name);
        }
        public void insRefE-Rence(String name){
            childRefList.addElement(name);
        }
        public void removeDirect(String name){
            childDirList.removeElement(name);
        }
        public void removE-RefE-Rence(String name){
            childRefList.removeElement(name);
        }
        public void dummy(){
            myStatus="Y";
        }
}
```

5.3 class MakE-R

클래스 MakE-R 의 구성은 두 엔티티 사이의 하나의 관계는 반드시 쌍으로 존재하므로 하나의 관계는 두개의 클래스로 구성된다. 주된 구성은 릴레이션쉽의 이름을 가지는 String 형태의 name 멤

버변수, String 형태의 from 멤버변수, String 형태의 to 멤버변수, 대응관계를 나타내어 주는 String 형태의 relate 멤버변수, 애트리뷰트들의 이름을 가지는 백터형태의 attr 멤버변수가 대표적이다. 메소드로는 애트리뷰트를 입력하는 insE-RtAttr() 이 있다. 클래스 MakE-R 의 소스코드는 아래와 같다.

```
makE-R.java
import java.util.*;

class MakE-R{

        public String name;
        public int id;

        public String from;
        public String to;
        public String relate;

        public Vector attr;

        public MakE-R(String name, int id, String from, String to, String
relate){

                attr = new Vector();
                this.name = name;
                this.id = id;

                this.from = from;
                this.to = to;
                this.relate = relate;
        }

        public void insE-RtAttr(String name){
                attr.addElement(name);
        }
}
```

5.4 class GenE-RateXmlSchema

GenE-RateXmlSchema 클래스는 위에서 정의된 MakeE, MakE-R을 이용하여 E-R 모델에서 설계된 엔티티, 애트리뷰트, 릴에이션쉽을 생성하고 클래스 내에 정의된 메서드를 실행하면서 결과적으로 XML Schema 문서인 xsd 문서를 생성해내는 클래스이다.

클래스 구성은 멤버변수로는 E-R 모델에서 설계된 엔티티를 생성한 인스턴스들들 담는 Vector 형태의 insMakeE 변수, 릴레이션쉽을 담고 있는 배열 형태의 insMakE-R 변수, 루트 엔티티의 이름값을 가지고 있는 String 형태의 h 변수, 중복 참조되는 엔티티의 이름값을 가지고 있는 백터형태의 duplicate 변수 등이 대표적이다.

클래스 구성으로 메소드로는 크게 두가지로 나누어 볼 수 있는데 하나는 E-R 모델을 계층형 구조 모델로 변환할 때 쓰이는 메소드와 변환된 계층형 구조 모델을 이용하여 XML Schema 문서로 생성해내는 메소드로 구분된다.

먼저 E-R 모델을 계층형 구조 모델로 변환시 사용되는 메소드는 BFS 스캔을 통하여 중복된 엔티티를 찾아내고 스캔 순서를 정하는 BFSScan() 메소드, 찾아낸 중복 엔티티와 정해진 순서를통해 E-R 모델을 계층적 구조 모델로 변환하는 genE-RateHiE-RView() 메소드, 릴레이션쉽에 애트리뷰트가 있을때 상위 엔티티로의 병합 또는 하위 엔티티로의 병합을 처리하는 relateshipME-Rge() 메소드, 의미없는 엔티티를 정리하는 emptyEntityRemove() 메소드, XML Schema 문서 생성시 각 엘리먼트의 이름을 정해주는 naming()메소드 로 구성되어 있다.

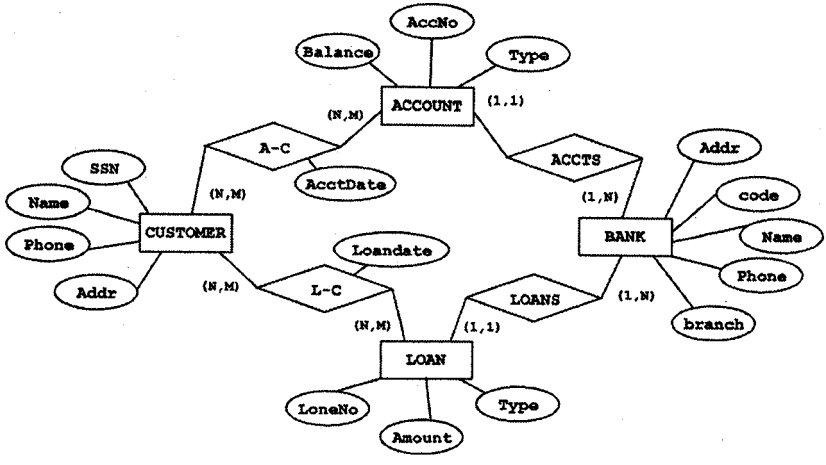
E-R 모델에서 위의 변환과정을 통하여 계층적 구조 모델로 변환된 모델을 이용하여 XML Schema를 생성하는데 쓰이는 메소드는

createXmlSchema() 메소드 이다.

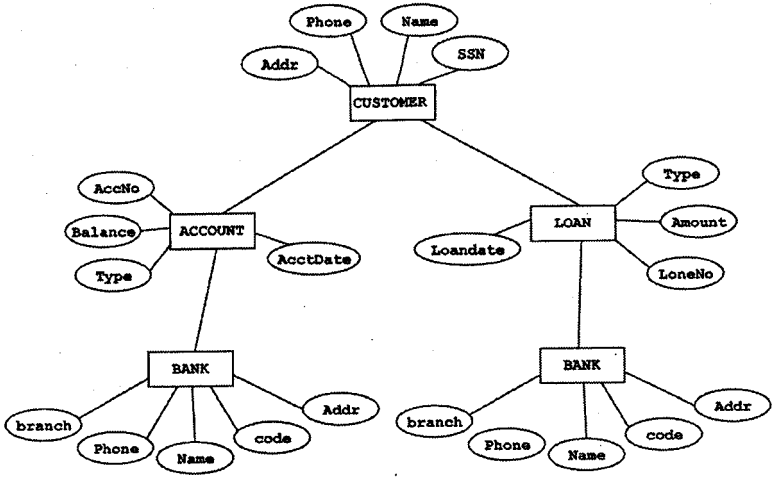
6. 비교 및 고찰

본 장에서는 본 논문에서 제시한 변환방법과 기존에 발표되었던 Elmasri[5]가 제시한 개체-관계 모델에서 XML Schema로의 변환방법에 대해 비교한다.

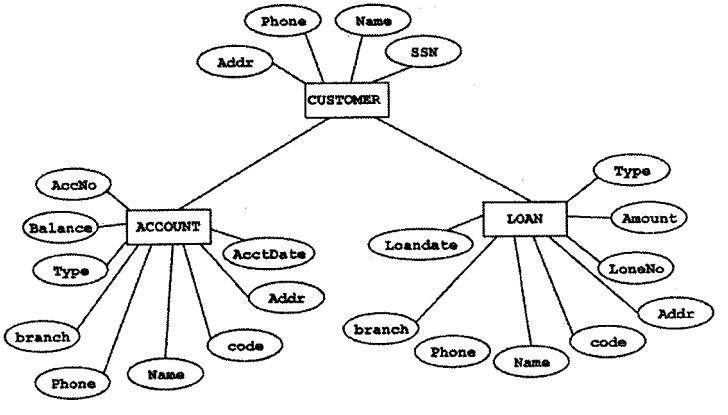
Elmasri의 연구에서 가장 주목할 부분은 개체-관계 모델에서 XML Schema로의 변환시 엔티티 사이의 대응제약조건을 이용하여 그 조건에 따른 애트리뷰트의 병합 및 이동이다. 본 논문에서 제시한 개체-관계 모델을 Elmasri의 변환방법에 따라 변환하면 다음과 같다.



[그림 6.1] 변환할 개체-관계 모델



[그림 6.2] Elmasri의 계층적 모델구조로의 변환과정



[그림 6.3] Elmasri 의 최종 계층적 모델구조 변환

Elmasri의 연구에서 제안한 방법에 따르면 설계된 개체-관계 모델에서 너비우선탬색을 통하여 개체-관계 모델을 탐색하면서 중복된 엔티티가 탐색된다면 [그림 6.2]와 같이 엔티티를 복사한다. 그리고 각 엔티티 사이의 대응제약관계의 조건에 따라 [그림 6.3]과 같이 엔티티의 애트리뷰트가 상위 엔티티로 이동되기도 한다.

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"/>
<!--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>
```

Elmasri의 연구에서 제시한 변환방법은 개체-관계 모델을 XML Schema 로 변환하고 생성할 때에 변환의 조건에 따라 엔티티의 수가 줄어들 수 있어 본 논문에서 제시한 변환방법 보다 XML Schema 문서가 간결해지고 그 구조가 단순해지는 장점을 가진다.

그러나 Elmasri가 제시한 방법은 엔티티간의 애트리뷰트가 상위 혹은 하위 엔티티로 이동하면서 최초 설계시의 엔티티 형태가 사라지게 되어 XML Schema의 특징인 재사용을 사용 할 수 없는 계층

적 구조모델이 된다. 또한 탐색 중에 중복참조 된 엔티티가 나타나게 되면 복사되기 때문에 하나의 XML Schema 문서에서 같은 내용으로 구성된 엔티티이지만 중복되는 엔티티를 두 번 이상 정의하는 경우가 발생 할 수가 있다.

그러므로 Elmasri 가 제시한 변환방법은 XML Schema 문서의 가장 큰 특징중의 하나인 DTD와 구별되는 가장 기본적인 특징인 전역 및 로컬 기능 및 재사용성을 전혀 고려하지 않고 XML Schema 문서의 단순화에만 치중하여 제시한 변환방법이라고 볼 수 있다.

만일 변환될 개체-관계 모델의 관계가 위에서 [그림 6.1] 보다 더 많은 엔티티가 존재하고 그에 따른 중복 참조되는 더 많이 존재하는 개체-관계 모델을 Elmasri가 제안한 방법으로 변환한다면 중복 참조에 따라 복사되는 엔티티의 수도 그 만큼 많이 존재하게 되어 결과적으로는 더욱 더 복잡해지는 XML Schema 문서가 될 수 있다.

7. 결론

지금까지 본 논문에서는 군사용 문서나 데이터가 대부분 관계형 데이터베이스 모델을 기반으로 작성된 것에 착안하여, 대량의 문서나 데이터를 XML 데이터베이스로 변환하기 위한 스키마의 변환 방법을 제안하고 구현하였다. 본 논문에서 달성한 연구 결과는 다음과 같다.

첫째, XML Schema 설계시 기존에 관계형 데이터베이스 설계 방법으로 널리 알려진 E-R 모델을 활용함으로서 XML Schema 설계

및 구현을 좀 더 용이하게 할 수 있는 방법을 제시하였다.

둘째, XML Schema 의 가장 큰 특징 중의 하나인 재사용성을 활용하여 XML Schema 문서를 생성한 것이다. 재사용성을 사용하였으므로 C++, java 등 요즘 널리 사용되고 있는 프로그램에서 사용되어지는 객체지향성이라는 구조와 흡사하고 현 프로그래밍 기법과도 일치한다. 또한 DTD와 구분되어지는 XML Schema 문서의 특징인 재사용성을 잘 표출하였다고 할 수 있다.

향후 연구과제로는 현재 구현된 시스템의 신뢰성을 향상시킬 조치가 필요하며 E-R 모델 입력을 GUI 환경에서 구축하여 사용자가 좀더 편리하게 본 논문에서 구현된 XML Schema 변환과정을 활용할 수 있는 프로그램 개발이 필요하다. 더 나아가 현재 제시한 방법을 사용하여 이전에 구축되어 있는 관계형 데이터베이스를 E-R 모델로 역컴파일[8]하여 E-R 모델을 생성한 후 본 논문에서 제시한 방법으로 XML Schema를 생성해 내는 것이다. 아울러 XML Schema 문서를 로드하여 관계형 데이터베이스 스키마로의 변환, 혹은 관계형 데이터베이스 스키마를 로드하여 XML Schema 로의 변환을 용이하게 하는 응용프로그램을 제작하여 궁극적으로 XML 문서의 고유 목적인 데이터 교환에 부합하는 응용프로그램 제작하는 것이다.

< 참고문헌 >

[1] Bray, t., Paoli, j., SpE-RbE-Rg-McQueen, c., MalE-R, E.: Extensible Markup Language(XML)1.0 W3C Recommendation, OctobE-R 2000.

[2] Dongwon Lee, "Schema ConvE-Rsion 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 IntE-Rnational ConfE-Rence on Conceptual Modeling 2002, page 429-443

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

[6] FE-Rnandez, M.F., Tan. W.C., Suciu, D.:"SilkRoute:Trading between Realations and XML". In: IntE-Rnational Word Wide Web ConfE-Rence, AmstE-Rdam, NethE-Rlands(2000)

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

[8] RogE-R H.L. Chian, TE-Rence M. Barron, Veda C. Storey, "RevE-Rse engineE-Ring of relational database:Extraction of an EE-R model from a relational database", DATA&KNOWLEDGE ENGINEE-RING, 1994

A Transformation Military Databases based on the Relational Data model into XML Databases

Kim Chang-Suk · Kim Eung-Soo

AS Extensible Markup Language(XML) is emerging as the data format of the Internet era, there are increasing needs to efficiently transform between database and XML documents. In this paper, we propose a schema transformation method from relational database to XML database. To transform the schema, we represent input schema as Entity-Relationship diagram. Entity-Relationship model translator scans the input Entity-Relationship diagram using BFS(breadth First Search) and translates the diagram into hierarchical structure model. The XML Schema generator produces XML Scema code using the transformed hierarchical structure model. The proposed method has a merit that having reusability facility of XML Schema property in comparison with existing researches.

Keywords : Extensible Markup Language, schema transformation method, relational database, XML database