XML Schema를 이용한 이질의 데이터베이스 스키마 통합

A Database Schema Integration Method Using XML Schema

박 우 창*
U-Chang Park

요 약

데이터베이스를 사용하여 분산된 환경에서 많은 응용이 구축되어감에 따라 지역 데이터베이스의 자치성은 유지하면서 데이터베이스들을 상호공유하는 데이터웨어하우징, 데이터마이닝 등의 응용이 많이 늘고 있다. 이러한 응용의 첫 번째 요구사항은 이질의 데이터베이스 스키마 통합이다. 기존의 데이터베이스 스키마 통합은 통합을 위한 스키마 공통 모델의 부재, 스키마 통합 프로그램의 복잡성으로 어려움을 겪어 왔다. 본 연구는 XML Schema를 데이터베이스 스키마 통합을 위한 공통된 데이터모델로 사용하여 데이터 모델을 표준화하고, 스키마 통합을 위한 연산들을 정의하여 이를 XSLT로 대응시킴으로써 스키마 통합 자동화 프로그래밍의 복잡성을 해결하며, 스키마 충돌 유형에 따라 통합하는 방법을 제시하였다. 제시된 방법은 기존의 통합 모델에 비하여 표준화, 확장성 등의 장점을 갖는다.

Abstract

In distributed computing environments, there are many database applications that should share data each other such as data warehousing and data mining with autonomy on local databases. The first step to such applications is the integration of heterogeneous database schema, but there is no accepted common data model for the integration and also are difficulties on the construction of integration program. In this paper, we use the XML Schema for the representation of common data model and exploit XSLT for reducing the programming difficulties. We define the schema integration operations and develop a methodology for the semi-automatic schema integration according to schema conflicts types. Our integration method has benefits on standardization, extendibility on schema integration process comparing to existing methodologies.

1. 서 론

분산된 환경에서 서로 다른 데이터베이스를 사용하여 많은 응용이 구축되고 있다. 이러한 데이터베이스의 이질성은 데이터 모델의 이질성, 데이터베이스 스키마의 이질성 등 다양한 형태를 가지고 있으며 이러한 데이터베이스들의 상호 작동은 지역 데이터베이스의 자치성 유지 여부, 데이터베이스 이질성 유지 여부, 데이터의 분산성 여부 등에 따라 통합 방법이 다양하게 결정된다. 그 예들로 간단한 게이트웨이를 통한 데이터 공유[1], 소프트웨어를 통한 뷰의 제공, 지역 데이터베이스의 자치성을 유지하면서 논리적인 스키마의 통합을 통한 데이터 연동을 하는 연합 데이터베이스 등 시스템의 요구에 따라 다양하게 상호작동의 필요성이 대두된다. 이러한 데이터베이스 상호 작동은 분산 데이터베이스, 연합 데이터베이스, 멀티 데이터베이스 등 여러 가지 개념으로 연구되어 왔고 이러한 연구의 공통 해결 요구사항은 이형, 이질 데이터베이스 스키마의 통합이다[3].

데이터베이스 스키마 통합을 위한 연구들은 주로 스키마 통합 규칙을 찾기와 스키마 통합을 위한 공통 모델의 개발에 치중해왔다[4]. 스키마의 통합에 있어서 통합 규칙은 종류가 다양할 뿐만 아니라, 규칙을 표현하는 표현형식이 대부분 수학적인 공식으로 이루어져 있어 통합하는 방식에

* 덕성여자대학교 자연과학대학 전산학과 부교수
ucpark@duksung.ac.kr

대한 이해가 어렵고, 복잡한 규칙을 기반으로 하기 때문에 통합을 위한 프로그램을 개발하기가 어렵다[2,3,4]. 또 통합 모델은 초기에는 개발자들이 임의로 정의하여 사용하였고, 93년에 만들어진 ODMG의 객체 지향 데이터베이스 모델의 표준화 후에는 ODMG 모델을 사용하는 연구가 많이 이루어졌다. ODMG 모델을 사용하는 방법은 스키마를 객체지향 데이터 모델로 변환하고 OQL을 사용하여 데이터베이스를 필터링하며 포맷팅한다[16].

Spaccapietra[4]가 제안한 GDM(Generic Data Model)은 데이터 모델링 개념의 집합이며, 일반화된 대응규칙과 통합규칙 정의의 기술이다. GDM의 정의는 속성의 튜플로 구성되는 객체(object)로 구성되며 속성은 값(value) 속성과 참조(reference) 속성으로 구성된다. 스키마 통합을 위한 변환은 링크(link)를 이용하여 스키마간의 단언(assertion)을 기술하여 통합될 스키마를 명시한다. GDM은 구조적으로는 객체지향이나 개념적으로 객체지향 모델과 다른 점이 있다. 객체지향 모델은 객체들 사이의 관계가 합성 객체에서 콤포넌트로 가는 단방향 관계를 가지게 되며, GDM은 객체들 사이의 관계가 양방향이며 속성에 대한 카디날리티(cardinalities)를 표현할 수 있다. GDM은 정형화된 스키마 통합 기술 방법으로 DBA가 단언(assertion)을 이용하여 원시 스키마와 통합 스키마간의 대응관계를 표시하는 도구로 사용할 수 있다.

Soon M. Chung[2]은 이질의 멀티 데이터베이스 시스템에서 하나의 인터페이스로써 다양한 데이터베이스를 접근하는데 있어서 스키마 충돌, 다른 데이터 모델의 이질성과 데이터 불일치 등과 같은 문제가 발생하는 것을 해결하기 위해 Unified Model을 제안하였다. 이 모델은 스키마 통합에 적절한 객체 모델과 질의 변화에 용이한 관계 모델의 각각에 대한 장점들의 조합이다. Unified Model은 클래스(class), 중첩클래스(nested class), 클래스계층과 상속(class hierarchy and inheritance), 메소드(method)로 구성된다. Unified Model은 관계 및 객체 지향 모델을 표현할 수 있고, 또 SQL을 통한 데이터 변환 방법을 제시하고 있다.

Lawrence[17]는 관계데이터베이스 스키마의 통합과 이에 따른 데이터 매핑 문제를 XML 기술을 이용하여 시도하였다. 관계 데이터베이스 스키마를 XML Schema 문법과 유사한 X-Spec을 사용하여 표현하여 표준화된 스키마 표현을 시도하였다. 스키마간의 충돌문제는 스키마를 구성하는 원소들 간에 사전(Dictionary)을 이용한 매핑으로 통합된 스키마와 소스 데이터간의 대응관계를 해결하였다. 질의처리기는 사전을 이용하여 전역질의를 지역질의로 바꾸는 기능을 한다.

본 연구는 현재까지 연구에 있어서 문제점인 표준 데이터 모델의 부재와 스키마 충동 해결 도구의 부족을 해결하고자 한다. 웹 표준양식으로 각광을 받고 있는 XML Schema를 이용하여 표준 모델 문제를 해결하고, 스키마 통합 규칙을 XSLT를 이용하여 표현하며, 이 XSLT를 이용하여 스키마를 통합하는 방법을 제시하고자 한다. 그 시험적인 시도로 관계 데이터베이스 스키마 통합을 대상으로 XML Schema 모델 사용을 제시하고, 스키마 충돌 유형은 Spaccapietra가 제안한 스키마간 대응규칙을 사용한다.

본 논문의 구성은 다음과 같다. 2장에서는 데이터베이스 스키마와 XML schema의 관계 및 변환을 설명하고, 3장에서는 DB 통합시스템 구조를 보이며, 4장에서는 스키마간 대응규칙의 표현 방식, 스키마 충돌의 유형과 통합 규칙에 따른 XSLT 생성을 설명하며, 5장은 XSLT을 이용한 통합 적용 예를 보인다. 마지막으로 6장에서는 결론 및 향후 연구과제를 설명한다.

2. XML

2.1 XML(eXtensible Markup Language) Schema

XML[5]은 문서를 기술하는 표준으로 DTD (Document Type Definition)를 통하여 문서자체에

자신의 XML 문서가 허용되는 문서의 구조를 DTD 고유의 문법을 이용하여 기술한다. XML은 문서의 구조를 사용자가 원하는 대로 정의할 수 있으며 이러한 구조적 융통성은 모든 형태의 데이터가 XML로 기술될 수 있도록 해줌으로써 모든 데이터가 동일한 형태로 통합, 저장, 처리될 수 있는 기반을 제공한다.

2.1.1. XML Schema 정의 및 특징

XML Schema는 DTD만을 사용하여 XML 문서를 표현하는데 한계를 극복하기 위해 W3C에서 제안한 표준규약이다. XML Schema는 특히 데이터 이동이나 변환에 있어서의 단점을 보완하기 위하여 만들어졌으며 XML 원소에 허용되는 내용이나 속성의 값을 제한하는 기능이 있어서 XML 데이터에 대한 자세한 명세를 가능하게 한다.

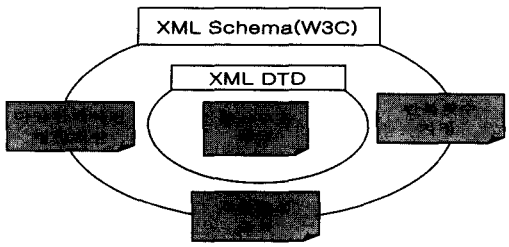
XML Schema의 특징은 아래와 같다.

- XML 규약에 의거하여 만들어졌으므로 일반 XML 문서와 같이 취급할 수 있다.
- 여러 가지 자료타입(int, float, boolean, date) 및 사용자 정의 타입을 제공한다.
- 관계 및 객체 데이터베이스의 데이터를 XML로 상호 변환할 수 있는 타입 시스템을 제공한다.
- 이름 영역을 표준으로 지원하므로 테이블이나 필드 이름끼리 충돌할 가능성을 미리 방지한다.
- 엘리먼트 타입 내용에 대해 제약을 가할 수 있다.

2.1.2 XML Schema와 XML DTD와의 비교

XML DTD와 XML Schema의 차이점을 구체적으로 살펴보면 다음과 같다. 그림 1은 기능적 측면에서 XML DTD와 XML Schema의 관계를 도식화하였다.

첫째, XML 문서의 구조를 정의하는데 있어서 XML DTD는 XML과 다른 문법을 사용한다.



(그림 1) 기능적 측면에서 본 XML DTD와 XML Schema의 관계

XML DTD는 간단하긴 하지만 XML 형식이 아니라 SGML의 DTD에서 일부분을 활용하는 형식이다. 반면에 XML Schema는 그 자체가 XML 문서이기도 하면서 특정 XML문서의 구조를 정의하는 규칙을 표현하기도 한다. 그러므로 문서구조를 정의하는 부분과 실제 사용하는 부분, 모든 부분들이 XML로 표현되게 되므로 XML Schema와 그에 맞게 작성되어진 XML 인스턴스 문서들은 서로 잘 연결이 된다.

둘째, XML DTD는 사용할 수 있는 데이터 형식이 제한적이다. 숫자와 알파벳 문자를 구분하여 지정할 수 없다. 그 뿐만 아니라 "나는 0부터 1200까지의 Integer형만을 쓴다"라고 사용자가 원하는 형태의 데이터 형식을 정의하여 사용할 수가 없다. 어떤 측면으로 보면 범용적이긴 하지만 좀더 데이터형식과 값에 대한 제한을 주고 유효성을 검증하고 싶은 사용자에게 필요한 기능을 제공하지 못한 것이다. 이러한 이유로 인해 데이터 형식에 대하여 좀더 강력하면서도 유연한, 그리고 재사용이 가능하도록 설계가 가능한 XML Schema가 필요하다. XML Schema에서는 총 45개의 기본적인 데이터 타입을 지원하고, 사용자 정의 데이터 타입을 선언할 수 있다.

셋째, XML DTD는 반복 횟수를 지정한다는 것보다 반복 여부에 대한 결정 기능만을 제공한다. 반면에, XML Schema는 정확하게 몇 번까지 반복하는지, 몇 번은 최소한 반복해야 하는지에 관한 정보를 지원할 수 있으므로 더욱 엄격하게 문서의 구조를 정의할 수 있다.

2.2 XSLT(eXtensible Stylesheet Language : Transformations)

XSLT는 XML 문서에서 원하는 정보를 추출하여 사용자가 원하는 문서의 형태로 변환해 주는 언어이다. 일반적으로 변환은 XML 문서를 입력으로 받아들여 다른 XML문서, HTML, WML 그리고 텍스트 문서 등으로 출력한다. XSLT는 XML 문서를 구조적으로 변환하거나, 포맷팅하는 기능이 주이며 본 논문에서는 XML Schema로 표현된 두 개의 데이터베이스 스키마를 원소나 속성의 이름을 변경하면서 구조를 변경하는 방법으로 새로운 XML Schema를 생성하는 작업을 수행한다. XML 문서를 변환하거나 처리하는 방법은 SAX나 DOM API를 이용하는 방법도 있지만 이들 언어를 이용한 방법이 절차적인데 반하여 XSLT는 선언적으로 언어 자체가 변환의 내용을 설명해주는 방법이 있기 때문이다.

XSLT의 주요한 특징은 다음과 같다.

- 다른 포맷으로 데이터를 보여주는 것이다. 여기서 변환은 순수 데이터로부터 포맷팅 된 정보로 즉, 주로 HTML 또는 XHTML 뿐만 아니라, SVG, PDF, 또는 Microsoft's RTF의 포맷으로 데이터를 변환한다.
- 다른 컴퓨터 시스템간에 데이터 상호교환시 유용하다. 예를 들어, 고객과 공급자사이에 상호 교환하는 전자 상거래 부분에서 많은 이용이 된다.
- 선택적으로 데이터를 추출하여, 재배열하고, 속성들을 요소로(또는 이와 반대로) 바꾸는 작업을 할 수 있으며, 단순히 데이터에 유효성을 제공하기 위해 사용할 수 있다.

2.3 데이터베이스 스키마와 XML Schema

데이터베이스 스키마 통합을 용이하게 하기 위한 기존의 연구들이 많이 있다. 기존의 연구들에서 스키마 충돌 해결을 쉽게 하기 위하여 중간 모델로 GDM(Generic Data Model)[4], Object-Oriented 개념의 기반으로 한 Unified Model[2]등을 사용하였다. 본 연구에서는 XML 표준에 의한 XML Schema를 데이터베이스 스키마를 표현하는 공통된 모델로 사용한다. 공통된 데이터 모델로 XML Schema를 사용하면 여러 가지 장점을 얻을 수 있다. 표준화된 모델을 사용함으로서 데이터베이스간의 통합 및 스키마간의 변환을 용이하게 하고, 변환 프로그램을 재활용하기 쉽다.

중간 모델로 XML Schema를 사용하기 위해서는 기존의 데이터베이스를 XML Schema로 변환하는 방법이 필요하다[9]. 스키마는 데이터베이스에 정의된 데이터 타입을 XML Schema의 타입으로 변환하며 이 과정은 해당 데이터베이스 시스템에 따라 변환기를 만들어 자동화 할 수 있다. 변환 알고리즘은 다음과 같으며 Duckett[9]에 자세히 기술되어 있다. 개략적인 변환 알고리즘은 다음과 같다.

(알고리즘 1) : 관계 데이터베이스 스키마를 XML Schema로 변환하는 알고리즘

(1) 스키마의 이름을 루트 엘리먼트(root element)로 정의한다.
(2) 데이터베이스 테이블 속성들의 데이터 타입은 simpleType 엘리먼트로 정의한다.
(3) 테이블 구조는 complexType 엘리먼트와 attribute 엘리먼트로 정의한다.
(4) 테이블간 참조는 KEY/KEYREF 엘리먼트 및 부모-자식 관계의 엘리먼트를 이용하여 정의한다.

프로그램 1의 관계 데이터베이스 스키마를 알고리즘 1에 적용하여 변환한 예는 프로그램 2와 같다.

```
CREATE TABLE Expensive_car (
  modelname        varchar(50) PRIMARY KEY,
  manufacturer     varchar(50),
  maximumspeed     int,
  price            int
)
```

(프로그램 1) 데이터베이스 스키마(SQL)

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified">
  <xs:element name="Expensive_car" type="cartype">
  <xs:complexType name="cartype">
    <xs:attribute name="modelname"
          type="nametype"/>
    <xs:attribute name="manufacturer"
          type="nametype"/>
    <xs:attribute name="maximumspeed"
          type="valuetype"/>
    <xs:attribute name="price" type="valuetype"/>
  </xs:complexType>
  <xs:simpleType name="nametype">
    <xs:restriction base="xs:string">
      <xs:length value="50"/>
    </xs:restriction>
  <xs:simpleType name="valuetype">
    <xs:restriction base="xs:integer"/>
  </xs:simpleType>
</xs:schema>
```

(프로그램 2) XML Schema을 이용한 데이터베이스 스키마 표현

3. 데이터베이스 스키마 통합

3.1 데이터베이스 통합 시스템 구조

스키마 통합에 대한 이해를 돕기 위하여 여러 데이터베이스가 상호 작동하는 연합 데이터베이스를 모델로 스키마 통합을 위한 데이터 모델을 설명하고자 한다. 데이터베이스 스키마 통합이 사용되는 대표적인 모델로는 연합 데이터베이스(FDBS, Federated Database System)가 있으며 Sheth[13]에 의하면 연합 데이터베이스는 이질성(heterogeneity), 자치성(autonomy), 분배성(distribution)의 특징을 지원하기 위해 그림 2와 같이 5-Level의 스키마 구조를 가진다. 그림 2는 스키마 계층이고 처리기의 위치와 명칭은 그림 3과 같다. Sheth의 모델은 통합을 위한 기본 모델로 받아들여지고 있으며 Mermaid, DDTS 그리고 SIRIUS-DELTA 등에서 사용되고 있다[13].

Sheth의 5-Level의 스키마 구조 및 시스템구조는 다음과 같은 구성 요소를 가진다.

- 지역 스키마 : 지역 스키마는 각 지역 데이터베이스에서 사용하는 DB의 개념적 스키마이며, 데이터모델에 따라 표현되는 스키마 표현 방식이 다르다.
- 공통 스키마 : FDBS의 지역 스키마에서 CDM (common data model)을 통해서 변화되어 추론된 스키마이다. CDM은 하나의 공통된 표현을 사용함으로써, 분산된 지역 스키마를 하나로 표현하고, 지역 스키마에서 빠진 의미를 공통 스키마에서 추가시킬 수가 있다. 지역 스키마에서 공통 스키마로의 변환은 변환 프로세서(Transforming Processor)를 통해서 수행이 된다. 변환프로세서는 FDBS의 특징 중 이질성(heterogeneity)을 지원한다.
- Export 스키마 : Export 스키마는 FDBS가 이용할 수 있는 공통 스키마의 부분집합으로 표현될 수 있다. DBA가 필요한 스키마를 골라 구성한 스키마 통합 전 단계이다.
- 통합 스키마 : 몇 개의 Export 스키마의 통합이며, Export 스키마를 통합할 때 생성되는 데이터 분류 정보까지 포함된다. DBA의 판단에 따라 이루어지며 많은 연구에서 다양한 변수로 인하여 자동화하는 것이 불가능함을 지적하고 있다.
- 외부(External) 스키마 : 사용자와 어플리케이션에 따라서 정의된 스키마이다. 외부 스키마는 통합 스키마 전체를 사용하기에는 어렵고 복잡한 점을 감안하여 사용자의 요구에 맞게 정보를 특정화시키는 Customization, 통합제약의 추가, 접근제어를 제공하는데 의의를 두고 사용된다. 통합 스키마에서 외부 스키마로 변환은 외부 스키마에서 제안된 필터 프로세서를 이용한다.

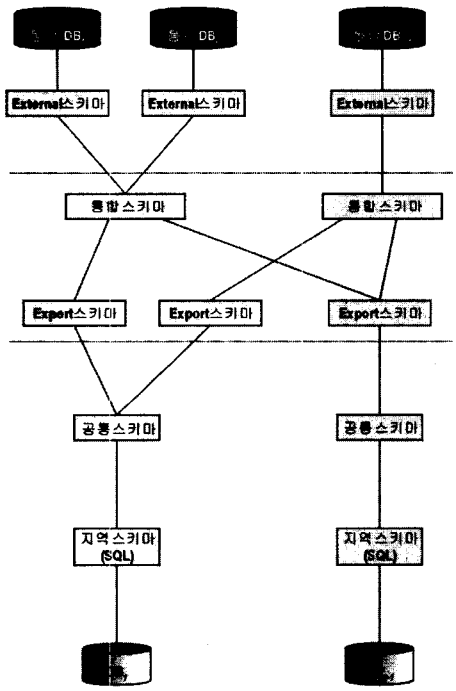
3.2.스키마 통합시스템 연구 범위

연구 대상 데이터베이스 모델은 먼저 관계 데이터베이스 모델을 기초로 한다. 기존의 구축된 데이터베이스의 90퍼센트 이상이 관계 데이터베이스 모델에 기반하고 있으며 또 객체 데이터베이스의 경우 본 연구에서 지역 스키마에서 Component 스키마

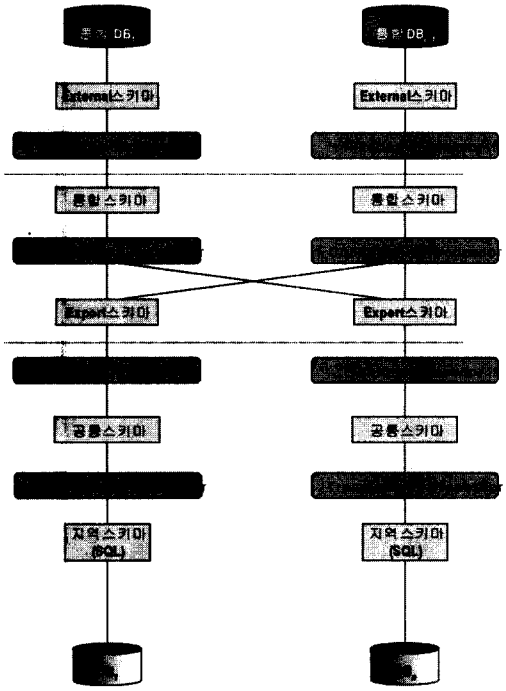
로의 변환만 되면 나머지 과정은 모델에 관계없이 가능하기 때문에 확장이 쉽게 이루어 질 수 있다.

본 연구에서의 스키마 통합은 3.1절에서 제시된 5-level 스키마 변환 단계의 구성처리기(Constructing Processor) 부분으로 스키마 변환, 대응(Correspondence Assertion), XSLT 생성, 통합(Integration), 역변환의 다섯 단계로 나누었다. 각 단계의 관계는 그림 4와 같다.

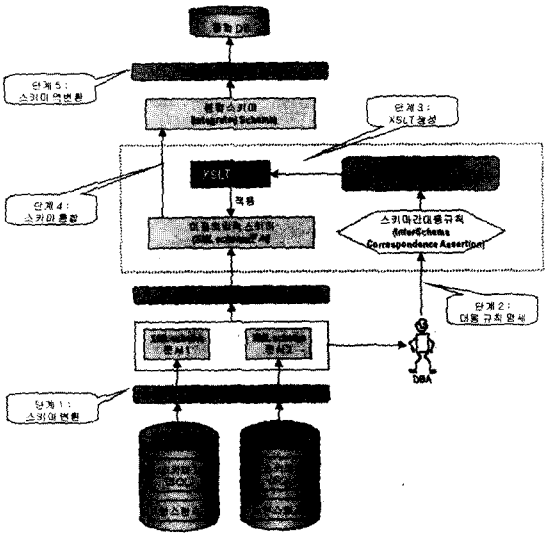
- 단계 1 : 스키마 변환 : SQL 언어로 표현된 데이터베이스 스키마를 XML 스키마 변환기를 통해서 XML Schema 형태로 바꾼다. 이것은 앞의 5-level 스키마에서 지역스키마에서 공통/외부스키마로 변환하는 과정이다. 이 과정은 Duckett[9]에서 기술된 것 처럼 기계적이며 표준화된 작업으로 볼 수 있다.
- 단계 2 : 대응 규칙 명세 : XML Schema로 표현된 두 이질 스키마 사이에 공유성과 의존성을 전역 DBA가 조사하여, 의미(semantic), 구조(structural), 기술(descriptive) 충돌 등을 포함한 스키마간 대응 규칙(inter-schema correspondence assertion)을 정하여 스키마 통합기(Integrator)에 전달한다. 스키마간 대응 규칙은 한 스키마에서 다른 스키마와 연관되어 있는 정보를 선언적 문장 형태로 나타내며 표현 방법은 4장에서 기술한다.
- 단계 3 : 스키마 통합 XSLT 생성 : XML Schema로 표현된 이질의 스키마를 통합 스키마로 변환하는 XSLT 생성하는 단계이다. 통합기(Integrator)는 DBA가 정의한 스키마간 대응 규칙을 전달받아 4장에서 정의된 통합규칙을 이용하여 통합된 스키마를 추론하고, 입력된 스키마와 통합된 스키마 사이에 매핑(Mapping)을 시킬 수 있는 XSLT 코드를 생성한다.
- 단계 4 : 스키마 통합 : 통합기에서 생성된 XSLT 코드를 단계 3의 이질의 입력스키마에 XSLT 코드를 적용시켜 통합된 스키마로 변환한다.



(그림 2) 5 level 스키마 통합 시스템 구조



(그림 3) 5 level 스키마 처리기 위치



(그림 4) XML Schema를 이용한 데이터베이스 스키마 통합시스템 구조

- 단계 5 : 스키마 역변환 : 통합된 스키마를 각 데이터베이스에 맞는 스키마 언어 형태로 변환한다. 이 과정은 단계 1과정의 역 과정이며 마찬가지로 기계적인 작업이다.

그림 4는 스키마 통합을 위한 데이터베이스 스키마 통합시스템 구조를 나타내며, 본 논문에서는 점선 안 부분, 즉 스키마 통합기 부분을 연구대상으로 한다. 즉 FDBS의 5단계로 구성처리기(Constructing Processor) 부분이며 그림 4의 단계 2, 3, 4이다. 단계 1이 끝나면 데이터베이스 스키마는 XML Schema로 바뀌게 된다. 단계 2에서의 대응 규칙 명세는 스키마 충돌 유형에 따라 본 연구에서 제안한 연산을 이용하여 기술한다. 단계 3은 단계 2에서 제안된 연산을 XSLT로 변환하는 템플릿에 따라 XSLT를 생성하는 과정이다. 단계 4는 단계 3에서 생성된 XSLT를 이용하여 통합된 XML Schema 생성하는 과정이다.

4. 스키마 통합

4장에서는 3장에서 설명한 구성처리기(Constructing Processor)의 단계 2, 3, 4에 대한 방법을 기술한다. 단계 2의 스키마간 충돌 해소를 위한 대응규칙의 표현 방법을 1절에서 설명하고, 스키마 충돌에 관한 내용을 2절에서, 스키마 충돌 유형에 따른 스키마 통합 연산의 적용에 관한 내용은 3절에서 설명한다. 스키마 충돌의 유형과 규칙이 많이 있지만 Spaccapietra의 연구[4]에 따라 스키마 충돌 유형 및 통합 규칙을 2절과 3절에서 정리하였다. 통합 유형에 따라 XSLT를 생성하는 통합 규칙이 각각 만들어진다.

4.1 스키마간 대응규칙(inter-schema correspondence assertion) 표현방법

문법적인 구조를 갖는 문서를 다른 문서로 변환하는 방법은 SDTS, AG, TT-grammar 등 여러 가지 기법이 있다[15]. 이러한 방법 중 TT-grammar 같은 트리 구조를 갖는 문서를 변환하는 방법을 본 연구의 XML Schema 파일을 변환하는 데 적용할 수 있다. XML 파일은 문서의 구조가 트리 구조이고 XSLT는 XML 파일을 다른 트리 구조로 변환하는 프로그램이다. 본 논문에서는 이러한 개념 아래 스키마 변환을 위하여 필요한 연산을 정의하고 여기에 데이터베이스 특성상 필요한 키 값 변경 연산 change를 따로 추가하였다.

XML 스키마 문서를 트리 구조로 볼 때 한 개의 트리를 다른 트리로 변환하는 작업은 몇 가지 기본적인 작업들이 합쳐져서 구성된다. 관계 데이터베이스 스키마를 XML 스키마로 표현하면 루트 노드는 데이터베이스 이름이 되고 자식 노드는 테이블이며 손자 노드는 속성의 이름인 높이가 3인 트리가 된다. 스키마의 통합은 2개의 트리를 1개의 트리로 만드는 과정이다.

스키마 통합은 DBA의 판단에 따라 이루어진다. DBA는 스키마를 통합하는 그래픽도구나 텍스트 방식의 명령어를 이용하여 스키마 통합에 관한 작업을 할 수 있다. 기존의 연구들에서는 이러한 대응 규칙을 수학적인 기호나 문장으로 표

시하였지만 체계적이고 기계적인 도구의 사용이 불가능함을 지적하고 있다. 본 연구에서는 이러한 부분을 조금이라도 해소하기 위하여 스키마 통합이 트리 구조로 된 XML Schema를 다른 트리 구조로 변형함에 착안하여 변형과정에 필요한 단위 작업을 연산(operation)으로 정의하고 DBA가 이 연산들을 사용하여 원하는 스키마 통합 작업을 할 수 있도록 하였다.

(정의) 기호 T :

데이터베이스 스키마에 대한 테이블 이름을 나타낸다. <schema>는 스키마 명, <tablename>은 테이블 이름을 나타낸다.

문법) T : <schema>.<tablename>

(정의) rename 연산, R :

테이블의 이름과 테이블에 속한 속성의 이름을 한꺼번에 바꾸는 연산이다. 명시되지 않은 이름들은 변하지 않는다. T1, T2는 테이블명, <attributename>은 속성명, <domainname>은 도메인 이름을 나타낸다고 하자.

문법) rename(T1 → T2,
{<attributename> → <attributename>}*,
{<domainname> → <domainname>}*)

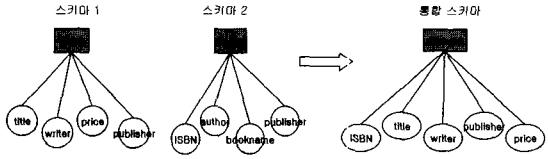
예) rename(S1.person → S1.people, age → myage, name → surname)

스키마 S1의 person 테이블의 이름은 people로 바꾸고, 그에 따르는 속성 age는 myage로 name은 surname으로 변경한다. 나머지는 그대로이다.

(정의) copy 연산 :

1개 이상의 인자를 가지며 원본 테이블을 통합 결과 테이블에 복사한다. 테이블 이름이 바뀌지 않는다. 2번째 인자는 원본테이블의 빠지는 속성을 말하며 결과테이블에는 이 속성을 제외한 모든 속성을 복사한다. 기본키에 대한 정보는 보존된다.

문법) copy(TR → T, <-{attributename}>*)



(그림 5) concatenate 연산의 예

예) copy(S1.book → S.books, -ISBN)

스키마 S1의 book 테이블을 결과 스키마인 S의 books 테이블에 복사하며, 복사할 때 스키마 S1의 book 테이블의 ISBN 속성을 제외하고 스키마 S의 books 테이블에 복사한다.

(정의) concatenate 연산 :

1개 이상의 인자를 갖고 두 개의 테이블을 합치는 연산이다. 속성의 이름이 같은 속성에 대해서는 합쳐서 출력하고 다른 속성들에 대해서는 추가하여 결과테이블에 포함시킨다. copy함수와 마찬가지로 제외되는 속성에 대해서는 두 번째 인자에서 정의한다. T가 릴레이션이고 S가 통합 스키마의 테이블 이름이라고 하자.

문법) concatenate(T, T, S, <-{attributename}>*)

예) rename(S2.books → S2.book, author → writer, bookname → title) → S.allbooks)
concatenate(S1.book + S2.book, S.allbooks)

스키마 S1의 book 테이블과 S2의 books 테이블을 book으로 이름을 변경하고 그에 속하는 속성들의 이름을 변경한 후, 두 개의 스키마에 존재하는 테이블의 공통부분은 하나로 합치며, 서로 다른 속성은 첨가하여 allbooks 테이블로 통합한다 (그림 5).

(정의) change 연산 :

통합된 테이블에 속성의 기본키 및 외래키에 대한 속성을 지정한다. PK 인자는 기본키를 변경하며, FK 인자는 외래키를 변경한다.

문법) change(FK or PK, T1.attributename, T2.attributename)

(표 1) 스키마 통합 연산

사용용도	함수명
이름 변경	rename(T → T, {<attributename>→<attributename>}*, {<domainname>→<domainname>}*)
테이블 복사	copy(T → T, <-{attributename}>*)
테이블 합치기	concatenate(T, T, S,<-{attributename}>*)
키 속성 변경	change(FK or PK, T.attributename, T.attributename)

예) change(FK, S.Car.ownerid, S.Person.pin) :

스키마 S의 Car 테이블 속성 ownerid를 Person 테이블 속성 pin을 참조하는 외래키 속성으로 변경한다.

change(PK, S.Car.ownerid, NULL) : 기존에 기본키로 정의된 제약을 제거한다.

스키마 연산함수들은 표 1과 같이 요약된다.

4.2 스키마 충돌의 유형

스키마 충돌은 여러 가지 유형이 있으나[2,4,14], Spaccapietra[4]에 따르면 주로 3가지 스키마 충돌 형태가 있다. 그리고 이러한 스키마 충돌은 실제 유형에 따라 복합적으로 나타나며, 4.3절에서 이를 해결하는 통합 규칙을 정한다. 3가지 스키마 충돌 유형은 다음과 같다.

(1) 의미 충돌(Semantic Conflicts)

스키마 디자이너들은 실객체(real world object)를 같은 집합으로 보지 않고, 중첩(overlapping)된, 포함(included)된 또는 교차(intersecting)하는 집합으로 인지한다. 예를 들어, 어떤 한 스키마에서 "Student" 객체로 나타나고, 또 다른 스키마에서는 더 제한적인 "CS-Student"(컴퓨터를 전공하는 학생 그룹) 객체로 나타난다. 의미 충돌을 해결하기 위해 일반화 개념을 사용한다[11]. 예를 들어 "CS-Student"객체는 "Stutdent" 객체에 속하는 형태로써 통합된다.

(2) 기술 충돌(Descriptive Conflicts)

같은 객체이면서 속성을 다르게 표현하는 경우이다. 예를 들어, Art 저널에 있는 광고의 내용과 Computer 저널에 있는 광고의 내용의 특성들은 서로가 다르다. 이 밖에도 동음이의어와 유사어에 해당하는 이름(naming) 충돌, 속성 도메인(attribute domain), 범위(scale), 제약(constraint) 충돌 등이 있다.

(3) 구조적 충돌(Structural Conflicts)

같은 데이터 모델을 이용하면서 실객체의 표현을 다른 구조의 형태로 표현하는 경우 발생하는 것이다. 예를 들어 객체모델에서 객체 형태로 선언되어 있는 것이 다른 스키마에서는 속성으로 선언되어 있는 경우이다.

4.3 스키마 통합 규칙

4.2절에 기술된 충돌의 유형은 실제 데이터베이스에서 단독으로 혹은 복합적으로 나타난다. 이 절에서는 Spaccapietra[4]에서 제시한 도메인 충돌 유형 1가지와 스키마 충돌 5가지 유형에 대하여 설명하고 본 논문에서 제시한 연산을 이용하여 스키마 대응 연산을 만들어 통합을 표현하는 방법을 설명한다. DBA는 지역스키마로부터 데이터베이스의 의미를 분석하여 충돌에 따라 앞에서 정의한 스키마 통합 연산을 적용하여 스키마 통합에 관한 내용을 기술한다.

4.3.1 도메인 속성 충돌 통합

스키마 S1과 S2에 속하는 테이블을 X1, X2라 하고, 각각 속성 A1, A2를 갖고 있다고 하자. 속성 A1과 A2의 도메인의 통합은 다음과 같은 규칙을 적용하여 통합 속성 A로 정의된다.

- 도메인 속성의 통합은 다음과 같은 형태로 정의한다.
  - A1 = A2 또는 A1 ⊇ A2 이면 ⇒ 통합 도메인은 domain(A1)으로 한다.
  - A1 ∩ A2 또는 A1 ≠ A2 이면 ⇒ 통합 도메인은 domain(A1) ∪ domain(A2)

스키마 대응 연산은 해당 스키마의 속성의 도메인의 이름을 바꾸는 rename( )을 적용한다. 즉 X1의 도메인 A1을 통합된 스키마에서 A로 바꾼다.

▸ 스키마 대응 연산(Correspondence Assertion)
rename(S1.X1 → S.X1, A1 → A)

4.3.2 독립 테이블 통합

스키마 S1의 테이블 X1이 통합된 스키마 S의 X로 복사하는 것이다. 여기서 테이블 X의 속성은 테이블 X1과 동일하다. 기본키는 유지되나 참조키는 버리게 된다. 스키마 S에서의 새로운 참조키는 change 연산으로 새로 정의해준다. 복사가 필요없는 속성은 -A 인자를 사용하여 명시한다.

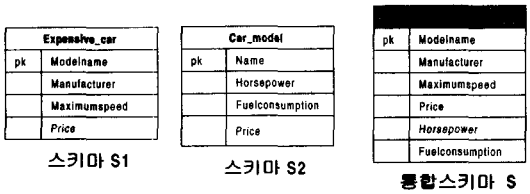
▸ 스키마 대응 연산(Correspondence Assertion)
copy(S1.X1 → S.X, -A)

4.3.3 두 개의 스키마에서 테이블과 속성 통합

스키마 S1의 테이블이 X1(A1, B1)이며, 스키마 S2의 테이블이 X2(A2, C2)를 가지는 경우이다. 두 스키마에서 같은 도메인의 속성으로 나타난 A1과 A2의 통합된 이름은 스키마 S의 A를 선택하면 된다. 각 테이블의 독립된 속성 X1.B1과 X2.C2에 대해서는 연합된 구조로 구성하면 된다. 즉, 통합된 스키마 S는 X(A, B, C)의 형태로 나타낸다.

▸ 스키마 대응 연산(Correspondence Assertion)
① rename(S1.X1 → S1.X, A1 → A),
② rename(S2.X2 → S2.X, A2 → A)
③ concatenate(S1.X, S2.X, S.X),

예) S1의 Expensive_car와 S2의 Car_model을 통합하여 Expensive_car를 만든다(그림 6).



(그림 6) 두 개의 테이블의 통합

▸ 그림 6의 스키마 대응 연산(Correspondence Assertion)
① rename(S1.Expensive_car → S1.Expensive_car)
② rename(S2.Car_model → S.Expensive_car, Name → Modelname)
③ concatenate(S1.Expensive_car, S2.Expensive_car, S.Expensive_car)

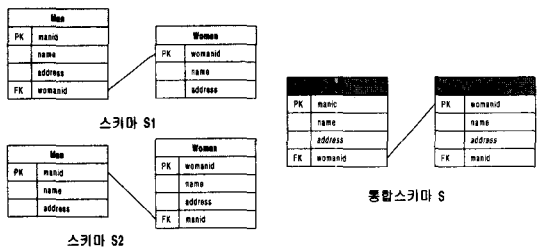
스키마 대응연산 1, 2번째 규칙에서 스키마 S1, S2의 테이블명과 해당 속성명을 변경한 후, concatenate 함수를 이용하여 통합된 스키마 구조를 나타내게 된다.

4.3.4 관계통합

스키마 S1의 두 테이블간의 관계 A1 - B1과 스키마 S2의 두 테이블간의 관계 A2 - B2가 성립한다고 하자. 관계통합은 통합된 스키마의 테이블 A - B에서 B의 기본키를 참조하는 A의 속성을 참조키(reference key)로 테이블 A에 첨가시키거나 A의 기본키를 참조하는 B의 속성을 참조키로 테이블 B에 첨가시키면 된다.

▸스키마 대응 연산(Correspondence Assertion)
① concatenate(S1.X1, S2.X2, S.X)
② concatenate(S1.Y1, S2.Y2, S.Y)
③ change(FK, S.X.A, S.Y.B)

예) 관계 통합의 예(그림 7)



(그림 7) 관계 통합

▸ 그림 7의 스키마 대응 연산(Correspondence Assertion)
① concatenate(S1.Man, S2.Man, S.Man)

② concatenate(S1.Women, S2.Women, S.Women)
③ change(FK, S.Man.womanid, S.Woman.womanid)

4.3.5 테이블과 관계의 통합

스키마 S1의 두 테이블 A1, B1이 A1 - B1로 서로 관계를 맺고 있으며, 스키마 S2의 세 테이블 A2, X, B2가 A2-X, X-B2의 관계를 맺고 있을 때 테이블 뿐 아니라 릴레이션까지 통합을 하는 경우이다. 이 경우는 스키마 통합 연산 중 copy, change 등을 DBA가 적용하여야 한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.0.1 (http://www.xmlspy.com) by mijin,kim (DS) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">

   <xs:element name="S1">
      <xs:complexType>
         <xs:sequence>
          <xs:element name="Expensive_car" maxOccurs="unbounded">
              <xs:complexType>
                  <xs:attribute name="modelname" type="nametype"/>
                  <xs:attribute name="manufacturer" type="nametype"/>
                  <xs:attribute name="maximumspeed" type="valuetype"/>
                  <xs:attribute name="price" type="valuetype"/>
              </xs:complexType>
          </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>

   <xs:element name="S2">
      <xs:complexType>
         <xs:sequence>
          <xs:element name="Car_model" maxOccurs="unbounded">
              <xs:complexType>
                  <xs:attribute name="name" type="nametype"/>
                  <xs:attribute name="horsepower" type="valuetype"/>
                  <xs:attribute name="fuelconsumption" type="valuetype"/>
                  <xs:attribute name="price" type="valuetype"/>
              </xs:complexType>
          </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>

<xs:simpleType name="nametype">
      <xs:restriction base="xs:string">
         <xs:length value="50"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="valuetype">
      <xs:restriction base="xs:integer"/>
   </xs:simpleType>
</xs:schema>
```

(프로그램 3) 입력된 이질 XML Schema 프로그램

▸ 스키마 대응 연산(Correspondence Assertion)

① concatenate(S1.A1, S2.A2, S.A)

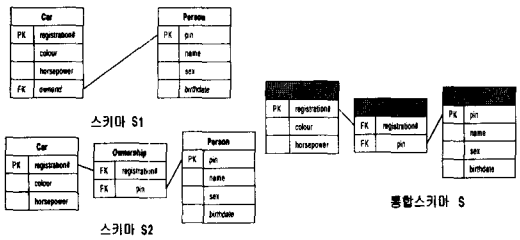
② concatenate(S1.B1, S2.B2, S.B)

③ copy(S2.X → S.X)

④ change(FK, S.X.a, S.A.a)

⑤ change(FK, S.X.b, S.B.b)

예) 테이블과 관계 통합의 예(그림 8 참조)



(그림 8) 테이블과 관계 통합

▸ 그림 8의 스키마 대응 연산(Correspondence Assertion)

① concatenate(S1.Car, S2.Car, S.Car, -ownerid)

② concatenate(S1.Person, S2.Person, S.Person)

③ copy(S1.Ownership, S.Ownership)

④ change(FK, S.Ownership.registration#, S.Car.registration#)

⑤ change(FK, S.Ownership.pin, S.Person.pin)

4.3.6 스키마 테이블과 복잡한 속성값 통합

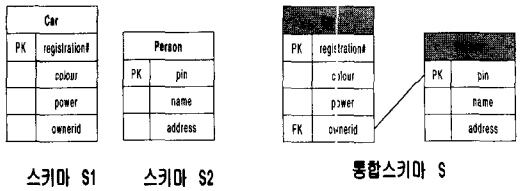
스키마 S1의 테이블 X1과 스키마 S2의 테이블 X2의 속성 A2와 통합하는 경우이다. S1.X1 - S2.X2.A2와 같은 통합은 X1과 X2를 참조할 수 있는 테이블 E를 생성한다. 즉, X1-E- X2 형태로 통합 스키마를 만든다.

▸ 스키마 대응 연산(Correspondence Assertion)

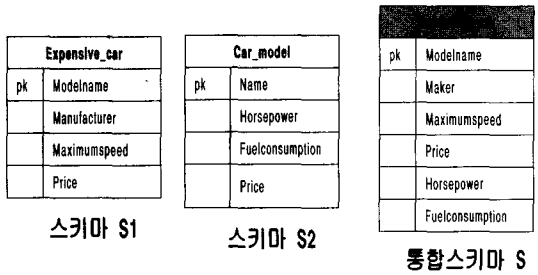
① copy(S1.X1, S.X)

② copy(S2.X1, S.Y)

③ change(FK, S.X.A, S.Y.B)



(그림 9) 스키마 테이블과 복잡한 속성값 통합



(그림 10) 두 개의 테이블 Expensive_car, Car_model 통합

예) 스키마 테이블과 복잡한 속성값 통합 (그림 9 참조)

▸ 그림 9의 스키마 대응 연산(Correspondence Assertion)

① copy(S1.Car, S.Car)

② copy(S2.Person, S.Person)

③ change(FK, S.Car.ownerid, S.Person.pin)

5. 통합사례 및 기존 통합 방법의 비교

5.1 예제 스키마

5.1.1 예제 데이터베이스 스키마

스키마 통합 유형 중 4.3.3의 두 개의 스키마에서 테이블과 속성 통합 방법에 대하여 XSLT 생성과 통합 예를 보기로 한다. 통합규칙의 연산들은 스키마 통합기에 의하여 XSLT 프로그램을 생성한다. 생성된 XSLT 프로그램은 XML Schema를 입력받아 통합된 XML Schema로 바꾼다. 본절에 예로 아래의 그림 10은 입력된 데이터베이스 스키마 S1, S2과 통합 후 데이터베이스 스키마 S를 보여준다.

5.1.2 XML 스키마

프로그램 3은 입력된 각 이질의 스키마의 XML Schema "Expensive_car", "Car_model"을 통합을 위해서 하나로 나타낸 XML Schema 문서이다.

5.1.3 스키마 대응 연산

입력된 스키마 S1, S2에서 전역 DBA가 정의한 스키마간 대응 규칙을 스키마 통합 연산을 이용하여 다음과 같이 기술한다. 이 연산은 통합기에 의하여 XSLT 프로그램으로 변환된다. 스키마 통합 단위 연산은 템플릿을 이용하여 XSLT 프로그램으로 생성된다.

```
/* 스키마 대응 연산(Correspondence Assertion) */
rename(S1.Expensive_car→S1.Expensive_car,
        manufacturer→maker)
rename(S2.Car_model→S.Expensive_car,
        name→modelname)
concatenate(S1, S2, S.Expensive_car)
```

(프로그램 4) 스키마 통합 연산 프로그램

위의 스키마 대응 연산에 대하여 스키마 통합기가 생성한 코드는 다음과 같다. rename, concatenate 연산은 템플릿 형태로 연산의 인자만 변화하고 나머지는 같다. 코드 생성은 JDOM을 사용하였다. 먼저 첫 번째 rename 연산은 프로그램 5의 XSLT로 생성된다. 두 번째 rename 연산은 첫 번째 연산과 비슷하기 때문에 생략하였다. 프로그램 6은 concatenate 연산에 대한 XSLT 프로그램이다.

5.2 통합 시스템 모형과 결과

5.1절의 예제 스키마의 처리는 그림 11의 스키마 통합 처리기에서 다음과 같이 진행이 된다. 두 개의 이질 스키마는 XML Schema로 바뀌어 처리를 위하여 한 개의 파일 프로그램 4로 통합된다. 스키마간의 구분을 위하여 element name 값에 "S1"과 "S2"를 지정하였다. DBA가 기술한 스키마 통합 연산은 프로그램 4에 기술되어 있다. 프로그램 4는 스키마 통합기에 의하여 해석되어 프로그

```
<xsl:template match="/xs:schema/xs:element[@name='S1']/xs:complexType
                /xs:sequence/xs:element/xs:complexType">
 <xsl:for-each select="/xs:schema/xs:element[@name='S1']/
    xs:complexType/xs:sequence/xs:element/xs:complexType/xs:attribute">
  <xsl:choose>
   <xsl:when test="@name='manufacturer'">
    <xsl:element name="xs:attribute">
     <xsl:attribute name="name">maker</xsl:attribute>
     <xsl:attribute name="type">
      <xsl:value-of select="@type" /></xsl:attribute>
    </xsl:element>
   </xsl:when>
   <xsl:otherwise>
    <xsl:element name="xs:attribute">
     <xsl:attribute name="name">
     <xsl:value-of select="@name" /></xsl:attribute>
     <xsl:attribute name="type"><xsl:value-of select="@type" /></xsl:attribute>
    </xsl:element>
   </xsl:otherwise>
  </xsl:choose>
 </xsl:for-each>
</xsl:template>
```

(프로그램 5) rename(S1.Expensive car→S1.Expensive car, manufacturer→maker)

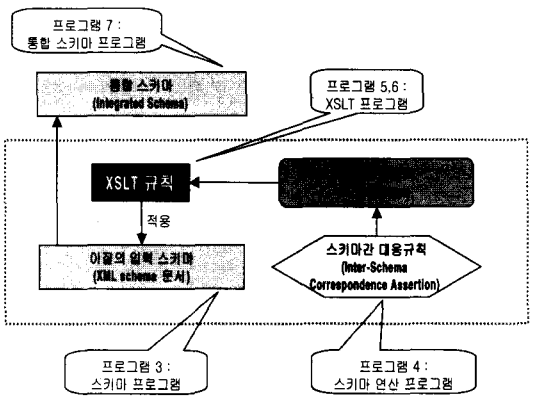
```
<xsl:template match="/">
 <xs:schema>
  <xs:element name="S">
   <xs:complexType>
    <xs:sequence>
     <xsl:element name="xs:element">
      <xsl:attribute name="name">
       <xsl:value-of select="/xs:schema/xs:element[@name='S1']/xs:complexType
                    /xs:sequence/xs:element/@name" /></xsl:attribute>
      <xsl:attribute name="maxOccurs">
       <xsl:value-of select="/xs:schema/xs:element[@name='S1']/xs:complexType
                    /xs:sequence/xs:element/@maxOccurs" /></xsl:attribute>
      <xs:complexType>
       <xsl:apply-templates select="/xs:schema/xs:element[@name='S1']/
            xs:complexType/xs:sequence/xs:element/xs:complexType" />
       <xsl:for-each select="/xs:schema/xs:element[@name='S1']/
            xs:complexType/xs:sequence/xs:element/xs:complexType">
        <xsl:for-each select="/xs:schema/xs:element[@name='S2']/
             xs:complexType/xs:sequence/xs:element/xs:complexType/xs:attribute">
         <xsl:choose>
          <xsl:when test="/xs:schema/xs:element[@name='S1']/
               xs:complexType/xs:sequence/xs:element/xs:complexType/
               xs:attribute/@name = self::xs:attribute/@name">
          </xsl:when>
          <xsl:otherwise>
           <xsl:element name="xs:attribute">
            <xsl:attribute name="name">
             <xsl:value-of select="self::xs:attribute/@name" />
            </xsl:attribute>
            <xsl:attribute name="type">
             <xsl:value-of select="self::xs:attribute/@type" />
            </xsl:attribute>
           </xsl:element>
          </xsl:otherwise>
         </xsl:choose>
        </xsl:for-each>
       </xsl:for-each>
      </xs:complexType>
     </xsl:element>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xsl:copy-of select="/xs:schema/xs:simpleType" />
 </xs:schema>
</xsl:template>

<xsl:template match="/xs:schema/xs:element[@name='S1']/xs:complexType
          /xs:sequence/xs:element/xs:complexType">
 <xsl:copy-of select="*[(self::xs:attribute/@*)]" />
</xsl:template>
```

(프로그램 6) concatenate(S1, S2, S.Expensive_car)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="IS">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Expensive_car" maxOccurs="unbounded"/>
          <xs:complexType>
            <xs:attribute name="modelname" type="nametype"/>
            <xs:attribute name="manufacturer" type="nametype"/>
            <xs:attribute name="maximumspeed" type="valuetype"/>
            <xs:attribute name="horsepower" type="valuetype"/>
            <xs:attribute name="fuelconsumption" type="valuetype"/>
            <xs:attribute name="price" type="valuetype"/>
          </xs:complexType>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="nametype">
    <xs:restriction base="xs:string">
      <xs:length value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="valuetype">
    <xs:restriction base="xs:integer"/>
  </xs:simpleType>
</xs:schema>
```

(프로그램 7) 생성된 통합 스키마



(그림 11) 통합 처리기에서 예제의 처리 과정

램 5, 6의 XSLT 프로그램을 생성한다. 프로그램 5, 6은 XML 처리기에 의하여 프로그램 3을 입력 데이터로 받아서 통합 스키마 프로그램 7을 생성한다.

스키마 통합에 해당되는 두 개의 스키마는 한 개의 XML Schema 파일로 통합되어 생성된 XSLT 변환 프로그램에 입력이 된다. XSLT에 적용하여 생성된 결과 XML Schema 문서는 다음과 같다.

5.3 기존 방법과의 비교

데이터베이스 스키마 통합은 오랜 연구 대상이었지만 그 복잡성으로 인하여 자동화된 혹은 반자동화된 도구를 만들기 어렵다[4]. 기존의 스키마 통합 방법은 대부분 다음과 같은 단계를 거친다[18].

(1) 통합 전 단계(Preintegration) : 스키마를 공통 데이터 모델로 바꾼다.
(2) 스키마 비교(Comparison) : 스키마 간 대응 관계를 조사한다.
(3) 통합(Integration) : 스키마 간 대응 관계와 통합 규칙을 이용하여 전역 스키마를 생성한다.
(4) 스키마 변환(Transformation) : 소스 스키마로 변환한다.

모델	공통 모델	통합 규칙 표현	알고리즘 자동화	발표년도	특징
Larson[3]	ECR model (Entity Cartegory Relationship) - Extended Entity Relationship	매핑 (mapping)	manual	1989	속성과 통합 전략의 형식적 정의
Spaccapietra[4]	GDM (Generic Data Model) - object, value, reference	단언 (assertions)	semi-automatic	1992	충돌 유형에 따른 단언
Soon M. Chung[2]	Unified Model (Unified Relational and Object-Oriented Model) - class, method, inheritance	매핑 (mapping)	manual	1995	관계, 객체, 계층 등 다양한 모델 수용
Lawrence[7]	X-spec XML-based specification document	매핑 (mapping)	semi-automatic	2001	XML 사용
본 연구	XML Schema	연산 (operation)	semi-automatic	2002	XML Schema 스키마 연산

본 연구는 단계 1의 공통 데이터 모델을 정의하였으며 단계 3의 통합을 위한 연산을 제공하였고 데이터베이스와 스키마간 상호 변환하는 프로그램을 개발하였다. 또 위 스키마 통합 과정을 수행하는 반 자동화된 도구를 사용자 인터페이스와 함께 제공하였다. 기존의 방법 중 몇 가지를 비교하면 다음의 표와 같다.

기존 연구에서 스키마 통합의 핵심은 공통 데이터 모델의 제시이다. 공통 데이터 모델은 주로 각 연구에서 제안되어 왔으나 본 연구에서는 표준화된 모델로 XML Schema를 처음으로 시도하였다. 알고리즘 자동화에서 semi-automatic 방법들은 수동적인 스키마 통합 기술과 자동화된 알고리즘을 사용하는 방법이다. 본 연구의 스키마 연산을 이용하는 방법은 스키마 자동화의 전 단계이며 단위 연산들의 구현을 통하여 스키마 통합 중간 과정의 검증이 쉬운 장점이 있다. 단위 연산은 또 스키마 뿐 아니라 인스턴스의 통합 과정을 단계적으로 진행할 수 있는 방법이 될 수 있다.

6. 결론 및 향후과제

본 논문은 XML의 표준 중 데이터베이스 스키마 표현에 적합한 XML Schema를 처음으로 시도하여 이질의 데이터베이스 스키마를 통합하는 방법을 제시하였다. 또 스키마 통합시 일어날 수 있는 스키마 충돌 유형에 따라 충돌을 기술할 수 있는 스키마 연산을 정의하여 DBA가 통합을 기술할 수 있는 방법을 제공하였다. 스키마 연산은 XSLT 템플릿에 적용되어 XSLT를 생성하는 방법으로 사용되며 생성된 XSLT를 스키마에 적용하여 통합된 스키마를 생성하였다.

본 연구의 방법은 다음과 같은 여러 가지 장점이 있다. 표준으로 인정된 XML을 공통 데이터 모델로 사용함으로써 스키마 통합을 표준화 할 수 있고, 스키마의 통합뿐 아니라 데이터의 통합에도 같은 모델을 사용하여 데이터의 변환의 표준화와 확장성을 얻을 수 있으며, 또 일관된 데이터베이스 스키마를 표현함으로써 서로 다른 데이터모델의 스키마 통합 뿐 아니라 XML 스키마로 기술된 XML Schema 파일의 통합에도 유용하게 사용될 수 있다. 본 연구에서 제시한 스키마 통합 연산은 기존의 스키마 통합 기술 방법이 주로 자연어에 의존한 반면 알고리즘으로 표현 가능한 연산의 형태로 표현하였다. 이 방법은 DBA가 통합 내용을 기술하기 쉽고 검증하기 쉬우며 프로그램화하기 쉬운 장점이 있다.

향후 연구는 제안된 방법을 프로그램을 구현

하여 스키마 통합 뿐 아니라 스키마에 해당되는 지역 데이터베이스의 인스턴스를 통합된 스키마에 매핑(Mapping)될 수 있는 XSLT를 구현하는 것이다.

Acknowledgement

본 연구는 2001학년도 덕성여자대학교 자연과학연구소 연구비 지원으로 이루어졌음

참 고 문 헌

[1] Litwin W., Mark L., Roussopoulos N., "Inter operability of Multiple Autonomous Databases", ACM Computing Survey 22, 3, Sept. 1990, pp. 267~193.

[2] Soon M. Chung and Pyeong S. Mah, "Schema Integration for Multidatabases Using the Unified Relational and Object-Oriented Model", ACM, 1995, pp. 208~215

[3] J.A. Larson, S.B. Navathe, and R. Elmasri. "A Theory of Attribute Equivalence in Database with Application to Schema Integration", IEEE Transactions on Software Engineering, pp. 449~463, 1989.

[4] Spaccapietra S., Parent C., Dupont Y., "Model-Independent Assertions for Integration of Heterogeneous Schemas", Very Large Data Bases Journal, Vol.1, No.1, July 1992.

[5] W3C XML Spec. http://www.w3.org/TR/1998/REC-xml-19980210.html.

[6] W3C XML Stylesheet Language, http://www.3.org/Style/XSL/.

[7] W3C XML Stylesheet Language Spec, http://w.w3.org/TR/1999/REC-xslt-19991116.html.

[8] Wolfgang Klas at. al., "Database Integration Using the Open Object-Oriented Database System VODAK", in Object-Oriented Multidatabase Systems, Prentice Hall, 1996, pp. 473~532.

[9] Jon Duckett at. al., "Professional XML Schemas", Wrox press, 2001, pp. 440~457.

[10] Wrox Author Team, "Professional XML Database", Wrox press,2000, pp. 47~109.

[11] M. Mannino, W. Effelsberg, "Matching Techniques in Global Schema Design", IEEE International Conference on Data Engineering, Los Angeles, April 24-27, 1984, pp. 418~425.

[12] C.Batini, M. Lenzerini, "A Methodology for Data Schema Integration in the Entity-Relationship Model", IEEE Transactions On Software Engineering, Vol. SE-10, n ° 6, November 1984, pp. 650~664.

[13] A. P. Sheth and J. A. Larson. "Federated Databases Systems and Managing Distributed, Heterogeneous, and Autonomous Databases", ACM Computing Surveys, 22(3), September 1990.

[14] Christine Parent and Stefano Spaccapietra, Issues and Approaches of Database Integration, Communications of the ACM, 41(5), May 1998.

[15] Martin Endig, Thomas Herstel, Eike Schallehn, Zoltan Sera, "An Approach for Semi-Automatic Derivation of XSLT Information Based on DTD Descriptions", Proceeding of the Fifth East-European Conference on Advances in Databases and Information Systems (ADBIS'2001), 2001, Vilnius, Lithuania, September pp. 25~28, 2001.

[16] M. Roantree, J. B. Kennedy, P. Barclay, "Interoperable Services for Federations of Database Systems", Oasis Technical Report No. OAS-10, May 2001.

[17] Ramon Lawrence and Ken Baker, "Integrating Relational Database Schemas using a Standardized Dictionary", Proceedings of the 16th ACM SAC 2001, Las Vegas, USA, pp. 225~230.

[18] Michel Bonjour, Gilles Falquet, "Concept Bases : A Support to Information Systems Integration", Proceedings of CAiSE*94 Conference, Utrecht, University of Geneva, 1994.

◐ 저 자 소 개 ◑

박 우 창

1982년 서울대학교 자연과학대학 계산통계학과 졸업(이학사)
1985년 서울대학교 대학원 계산통계학과 계산학전공 졸업(이학석사)
1993년 서울대학교 대학원 계산통계학과 전산과학전공 졸업(이학박사)
1985년 2월~1988년 8월 울산대학교 공과대학 전자계산학과 전임강사
1996년 1월~1996년 12월 미국 LOS ALAMOS 국립연구소 방문연구원
1988년~현재 : 덕성여자대학교 자연과학대학 전산학과 부교수
관심분야 : 데이터베이스, 질의어처리, 알고리즘
E-mail : ucpark@duksung.ac.kr