EXPRESS 데이타를 XML 문서로 변환하는 번역기

(An EXPRESS-to-XML Translator)

이 기 호 † 김 혜 진 ††

(Kiho Lee) (Hye-Jin Kim)

요 약 EXPRESS는 공학 분야의 제품 정보를 기술하는 언어로서 서로 다른 시스템 사이의 데이타 교환이 가능하게 한다. 그러나 EXPRESS를 사용할 수 있는 소프트웨어의 종류는 제한적이며 이에 비용이 소요된다. 한편, XML은 웹 상에서 데이타의 관리 및 유지를 가능하게 한다. 웹은 사용 및 접근이 쉽고 저렴하기 때문에 XML로 쓰여진 데이타는 특정 애플리케이션이나 시스템에 의존하지 않아도 되므로 데이타 교환에도 유용하게 쓰일 수 있다. 따라서, EXPRESS로 기술된 정보를 XML로 표현할 경우, 누구나 사용하기 쉬운 웹 상에서 그 정보를 사용할 수 있으므로 제품 정보가 기존보다 더 폭넓고 손쉽게 사용되어 원활한 정보 교환이 이루어질 수 있다. 본 연구에서는 이를 위하여, EXPRESS 정보를 각각 그에 대응하는 XML DTD와 XML 스키마로 변환하는 방법을 제시한다. EXPRESS의 각 문법 요소들을 분류하고 이 요소들로 인해 발생할 수 있는 복합적인 경우를 고려하여, 이에 대응하는 XML DTD 요소와 XML 스키마 요소로 나타내는 번역 규칙을 제시한다. 또한, 이 번역 규칙에 의해 각각의 경우에 대응하는 XML DTD와 XML 스키마로 변환하는 번역기를 구현한다.

키워드 : XML, EXPRESS, 번역기

Abstract EXPRESS is product information description language. It is interpretable by human and software. Product data written in EXPRESS make it possible to exchange between heterogeneous systems. However, the number of software that can use EXPRESS is limited and it is expensive to use the software. XML makes it possible to update and manage data on the Web. Because the Web is easier to use and access than other tools comparatively, data represented by XML need not depend on specific applications or systems and it can be used for exchange of data. Therefore, if we represent EXPRESS-driven data in XML, there will be more active data exchange widely and easily. In this work, a method of translation EXPRESS document to XML DTD and XML Schema is proposed. By classification all of EXPRESS syntax element and consideration complex cases caused by this syntax element, a translation rule that represent XML DTD and XML Schema is suggested. Also, a translator which is corresponding to this rule is implemented.

Key words : XML, EXPRESS, translator

1. 서 론

EXPRESS는 공학 분야의 제품 정보 등의 복잡한 정보 구조를 문자로 기술하는 언어로서, 1994년에 ISO 10303-11로 발표되어 사용되어왔다. EXPRESS는 데이타의 엔티티(entity), 애트리뷰트(attribute), 관련성에 기초를 두고 있다. 또한 데이타의 일반화와 제약조건 사양의 개념을 가지며 객체 지향의 특징도 일부 포함하였고, 소프트웨어가 해석 및 처리할 수 있도록 되어 있다. EXPRESS는 텍스트 형태로 되어 있는 데이타이므로 인간이 읽을 수 있고, 서로 다른 CAD 시스템간의 데이타 교환이 가능하게 하게 하는데 그 목적이 있다. 그러나 EXPRESS를 사용할 수 있는 소프트웨어의 종류는 제한적이며 이를 사용하는 데에는 비용이 상당히 든다.

최근 몇 년 전부터 인터넷의 사용이 보편화되면서 웹

† 종신회원 : 이화여자대학교 컴퓨터학과 교수
khlee@ewha.ac.kr
†† 비 회 원 : 삼성전자 영상디스플레이사업부 연구원
hjkim74@dreamwiz.com
논문접수 : 2001년 1월 22일
심사완료 : 2002년 10월 1일

문서의 작성 및 사용이 활발해지고, 그로 인해 웹에서 발생하는 데이타의 양도 기하급수적으로 증가하게 되었다. 웹 문서에는 HTML이 표준 언어로서 사용되어 왔으나, HTML은 표현 방법 위주의 문법이라는 한계가 있었다. 이러한 한계를 극복하기 위해 XML이 등장하였다. HTML은 한정된 수의 태그의 집합만을 정의하는 반면, XML(Extensible Markup Language)은 사용자가 필요로 하는 태그를 정의하여 웹 상에서 구조화된 문서를 사용할 수 있도록 해준다. 즉, XML은 웹 상에서의 데이타의 관리 및 유지를 가능하게 한다. XML은 텍스트 형태로 되어 있으며, 웹에서의 정보를 표현하는 통일된 규격이 되어 가는 추세에 있으므로, 특정 애플리케이션이나 특정 시스템에 의존하지 않아도 된다. 따라서 데이타 교환에도 유용하게 쓰일 수 있다.

그러므로, EXPRESS로 기술된 정보를 XML로 표현할 경우, 누구나 사용하기 쉬운 웹 상에서 그 정보를 사용할 수 있으므로 제품 정보가 기존보다 더 폭넓고 손쉽게 사용되어 원활한 정보 교환이 이루어질 수 있다.

EXPRESS로 기술된 제품 정보 개념에 대한 실제 데이타 교환 형식은 ISO 10303-21(Clear text encoding of the exchange structure)에서 정의되어 있으며, 이 형식을 사용한 제품 정보는 Part 21 데이타라고 한다. EXPRESS와 Part 21 데이타와의 관계는 XML DTD (Document Type Definition) 또는 XML 스키마와 XML 인스턴스의 관계와 같다.

따라서, 본 논문은 EXPRESS로 쓰여진 제품 정보 개념(이하 EXPRESS 스키마)이 웹을 사용하여 좀 더 폭넓고 손쉽게 관리되고 공유될 수 있도록 XML DTD 또는 XML 스키마로 나타낼 방법을 연구하고, 이 방법을 사용하여 번역기를 설계하고 구현하고자 한다.

본 논문에서는 EXPRESS 스키마를 각각 그에 대응하는 XML DTD와 XML 스키마로 표현하는 방법 및 변환하는 방법을 제시한다. 이를 위해 2장에서는 EXPRESS를 XML로 표현하려는 움직임 및 관련 기술 동향에 대해서 살펴보고, 3장에서는 EXPRESS의 각 문법 요소들에 따른 번역 규칙 및 이에 의해 EXPRESS 스키마를 XML DTD와 XML 스키마로 변환하는 번역기의 설계에 대해 기술하고, 4장에서는 이 번역기의 구현 및 실제 문서에 적용한 예를 보여준다. 이어서 5장에서 결론 및 향후 연구에 대해서 기술한다.

2. 관련 연구 및 기술 동향

2.1 EXPRESS와 XML

EXPRESS는 1994년에 ISO 10303-11로 국제 표준으로 지정되었으며, 제품 정보를 기술하는 언어이다.

원래는 제품모델 데이타의 교환을 위한 STEP (International STandard for the Exchange of Product model data : 공식 이름은 Industrial automation systems-Product data representation and exchange) 표준의 한 부분으로 개발되었으나, 다양한 대규모의 모델링 애플리케이션에도 쓰이며[1], 전자제품 데이타 교환(EDIF : 전자 설계 데이타 교환 형식)이나 석유화학산업의 모델링(POSC : 원유가스 탐사와 생산을 위한 표준)에서도 사용된다.

EXPRESS의 문법의 주요 요소는 크게 다음과 같이 나뉘어진다.

- 스키마(Schema) : 관련된 정보의 집합
- 타입(Type) : 값의 영역(domain)을 정의한다. 이미 정의되어 있는 엔티티를 이용하여 정의하거나 기본 타입을 이용하여 정의할 수 있다.
- 엔티티(Entity) : 실제 세계 개념에서의 객체의 정의이며, 사용자가 관심을 두는 개념을 표현한다.
- 애트리뷰트(Attribute) : 객체의 특성이며, 각 엔티티 정의에서 선언된 애트리뷰트는 기본 타입인 STRING, REAL, BOOLEAN 및 집단데이타 타입 등과 TYPE으로 정의된 타입들로 정의된다.
- 규칙(Rule) : 서로 다른 엔티티나 애트리뷰트의 값과 관계를 정의하며, WHERE, UNIQUE, INVERSE, DERIVE 등이 있다.

이 외에 CONSTANT, FUNCTION, ALGORITHM, PROCEDURE가 있다.

ISO 10303-21은 교환 구조 문법에 대한 명세를 기술하며, EXPRESS 스키마에서 이 문법으로의 대응을 포함한다. 이 표준의 목적은 파일 내의 데이타로 코드를 변환하기 위해서 표준 포맷을 정의하여 애플리케이션 간의 데이타 교환을 지원하는 데 있다.

여기서 정의한 교환 구조는 헤더 부분(header section)과 데이타 부분(data section)으로 나뉘어진다. 헤더 부분은 모든 교환 구조에 적용될 수 있는 파일 이름 등의 정보를 포함한다. 데이타 부분은 교환 구조에 의해 전달되는 제품 정보를 포함한다[2].

XML DTD와 XML 스키마는 둘 다 XML 인스턴스의 표현규칙을 나타내는 형식이다. XML DTD는 W3C에서 표준으로 지정되었고, 2000년 10월 1.0 버전의 두 번째 판이 발표되었으며, XML 스키마는 2000년 10월 현재 표준후보 상태이다.

XML DTD는 문서의 구조에 사용되는 규칙의 집합을 지정한다. XML DTD는 엘리먼트, 애트리뷰트, 표기

법, 문서에 포함된 엔티티의 목록뿐만 아니라 각 요소 사이의 관계도 알려준다. DTD는 이들을 사용하여 문서 안에 정확히 무엇이 포함될 수 있고, 무엇이 포함될 수 없는지를 정의한다.

XML DTD에서의 애트리뷰트가 가질 수 있는 데이타타입은 10가지로 매우 제한적이라 할 수 있다.

XML 스키마는 XML DTD처럼 문서의 형태를 정의해주지만, XML DTD에 비해 많은 이점을 가지고 있다. 우선 XML DTD는 10가지 데이타타입을 갖는 데 비해 XML 스키마는 41가지 이상의 데이타타입과 그 외에 사용자 정의 타입을 가질 수 있다. 또한 DTD는 XML 인스턴스와 다른 형태의 문법을 사용하지만 XML 인스턴스 문서와 같은 형태의 문법을 사용하므로 사용자가 쉽게 친숙해질 수 있다.

XML 스키마는 또한 네임스페이스를 사용하기 때문에 여러 XML 스키마로부터의 동일한 이름의 엘리먼트와 애트리뷰트를 구분해주며, 중복해서 엘리먼트나 애트리뷰트를 정의해줄 필요 없이 네임스페이스만으로 다른 사람이 정의했던 엘리먼트를 사용할 수 있도록 해준다.

2.2 EXPRESS로 기술된 데이타를 XML로 표현하려는 시도

EXPRESS를 XML로 표현하는 방법은 ISO에서 제안하였으며 이외에도 최근 몇몇 회사에서 각자의 표현방식 및 이를 사용한 애플리케이션을 연구 중이다.

ISO에서 제안한 것은 10303-28(XML representation of EXPRESS-driven data)이다[3]. 이 제안은 1999년 12월에 제출되어 2001년 1월 현재 정식 표준이 아닌 작업 초안 상태로서 아직까지 시험적인 단계를 벗어나지 못하고 있다. 또한 이 연구는 EXPRESS를 XML DTD로 표현하는 데에 집중되어 있으며, XML 스키마로 표현하는 데까지는 미치지 못했다.

PDML(Product Data Markup Language)은 미국 PDIT사에서 제안한 언어로서, XML로 제품 정보를 나타내어 인터넷 상에서 제품 정보를 보여주는 것이 가능하도록 하여 DoD 무기 시스템 지원 인사부의 사용에 그 목적을 두고 제안된 언어이다[4]. 이외에도 이를 사용한 프로토콜 및 툴킷을 제시하였다.

FirstStep EXML은 1999년에 미국 PDIT사에서 .exe 형식으로 배포하였으며[5], 주어진 EXPRESS 파일을 CEB 방식에 의해 XML DTD로 바꾼다. 그러나 윈도우즈 운영체제 하에서만 사용 가능하다는 점으로 인해 XML의 시스템 독립적인 특징을 잘 살리지 못했으며, EXPRESS의 엔티티 및 애트리뷰트 외에는 번역되는 부분이 없다는 단점이 있다.

현재 EXPRESS 스키마와 이에 기반한 Part 21 데이타를 XML로 나타내는 방법으로 제시된 것으로는 CEB (Containment Early Binding), STEB(Strongly Typed Early Binding), OSEB(Object Serialization Early Binding)의 세 가지가 있다[6]. 이 중 CEB 방식은 교환 파일인 ISO 10303-21 형식에 맞추어 이에 필요한 데이타만 사용하는 것으로, 변환된 문서의 길이가 짧고 이해하기 쉽다. 반면에, STEB방식과 OSEB 방식은 정보의 손실이 전혀 없는 인스턴스를 기술하려고 하므로 링크가 많아 복잡하고 그 크기가 매우 크다. STEB 방식은 인간이 이해할 수 있으나, XML로 변환하는 과정에서 애트리뷰트 이름이 중복될 수 있으며 원래의 Part 21 데이타와는 달리 서브타입도 고유의 id를 갖는 경우가 생길 수 있다. OSEB 방식은 인간보다는 애플리케이션에 의해 번역될 수 있는 구문을 사용하며 번역하는 과정에서 문법 상의 오류가 생길 수 있다. 따라서, 본 논문에서 XML DTD로 표현하는 부분은 가독성이 높고 데이타의 길이가 짧은 CEB 방식에 바탕을 두고 이를 변형하였다.

3. EXPRESS 데이타를 XML로 표현하는 변환기 설계

EXPRESS 스키마를 XML DTD로 바꾸는 방법으로는 크게 두 가지 접근 방법을 생각해볼 수 있다. 하나는 각 EXPRESS 스키마를 각각의 인스턴스로 간주하고, 단일한 커다란 XML DTD에 적용하는 방법이다. 또 다른 하나는 각 EXPRESS 스키마를 각각 하나의 XML DTD로 바꾸고 ISO 10303-21 형식의 교환 파일이 XML 인스턴스 역할을 하도록 하는 방법이다. 이들은 각각 ISO 10303-28에서는 Late Binding 방식과 Early Binding 방식으로 언급되어 있다.

실제로 사용되는 경우를 생각할 때, 하나의 EXPRESS 스키마를 바탕으로 ISO 10303-21형식의 여러 교환 데이타가 존재한다는 점과, 하나의 XML DTD (또는 XML 스키마)가 여러 XML 인스턴스를 둔다는 점이 동일하므로 Early Binding 방식이 유용할 것으로 생각된다. 따라서, 본 논문에서는 Early Binding 방식을 사용하여 EXPRESS 스키마를 XML DTD로 변환하고자 한다.

Early Binding 방식으로 변환하는 경우 염두에 두어야할 것은 교환할 파일 형태가 어떤 모습을 갖느냐 이다. 이 교환할 파일 형태는 EXPRESS는 ISO 10303-21에서 지정한 파일 형식을 가지며 XML에서는 XML 인스턴스가 그 역할을 하게 되므로, EXPRESS 스키마를

변환한 결과인 XML DTD 및 XML 스키마의 모습은 XML 인스턴스가 ISO 10303-21에서 지정한 형태를 가지고 교환될 수 있는 토대가 되도록 한다. 따라서, 우선 EXPRESS 스키마와 그에 따른 ISO 10303-21의 교환 구조 중 데이타 부분의 간단한 예를 알아보면 다음과 같다.

다음은 자동차에 관한 간단한 EXPRESS 스키마이다.

```
SCHEMA car_specification;
ENTITY car;
   make : STRING;
   model : STRING;
   year : INTEGER;
   owner : person;
END_ENTITY;
ENTITY person;
  first_name : STRING;
  last_name : STRING;
END_ENTITY;
END_SCHEMA;
```

그림 1 자동차에 관한 간단한 EXPRESS 스키마

이것은 car라는 엔티티가 make, model, year, owner라는 애트리뷰트를 가지며, person이라는 엔티티가 first_name, last_name이라는 애트리뷰트를 갖는다는 것을 의미한다. 각각의 애트리뷰트들은 애트리뷰트 이름 옆에 정의된 데이타타입을 갖게 된다. car 엔티티 애트리뷰트 중 owner라는 애트리뷰트는 person이라는 엔티티로 나타내어진다. 이 EXPRESS 스키마의 ISO 10303-21 형식 파일 중 데이타 부분은 다음과 같다.

이 데이타 모델의 인스턴스가 1999년형 Kia사 제품의 모델명은 Carnival이고 소유주가 young Kim이라면[7],

```
#1 = CAR('Kia', 'Carnival', 1989, #2);
#2 = PERSON('young', 'Kim');
```

와 같이 표현한다. 위에서 보는 것과 같이, 각 인스턴스마다 고유의 번호가 있으며, 해당하는 엔티티 이름에 각 애트리뷰트의 데이타타입에 맞는 인스턴스가 들어가게 된다. 여기서, car 엔티티 인스턴스의 owner애트리뷰트는 person이라는 엔티티를 애트리뷰트로 가지므로 person 엔티티 인스턴스의 고유번호를 갖게 된다.

따라서, EXPRESS를 XML로 변환할 때에는 각 엔티티가 고유의 ID를 기본 애트리뷰트로 갖도록 하고, 원래 갖고 있던 애트리뷰트를 모두 표현해주되, ISO 10303-21 교환 파일에서는 구체적인 고려 대상이 되지 않는 where, unique 등의 제약조건은 함축적인 애트리뷰트로 표현해준다.

아직까지는 XML 스키마가 W3C의 표준이 아니므로 EXPRESS를 XML 스키마로 변환하려는 노력은 별로 보이지 않고 있다. 따라서 기존의 제시된 방법에서는 EXPRESS를 XML DTD로 변환하는 것만 보여주고 있으나, XML 스키마가 XML DTD에 비해 훨씬 장점이 많고, 곧 표준이 될 가능성이 높다는 점에서, EXPRESS를 XML 스키마로 변환하는 방법에 대해서도 고려하였다.

3.1 EXPRESS를 XML로 변환하는 번역기의 구조

EXPRESS 스키마를 XML로 변환하는 번역기의 전반적인 구조는 그림 2와 같다.

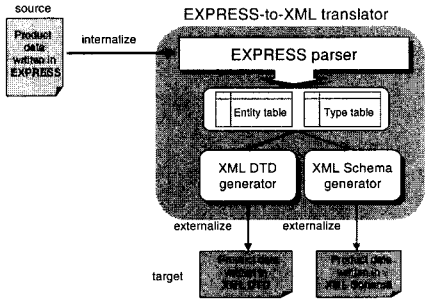

그림 2 EXPRESS 스키마를 XML로 변환하는 번역기의 구조

입력된 EXPRESS 문서는 우선 EXPRESS 파서를 거치면서 파스 트리를 생성하게 된다. 이 과정에서 파스 트리를 이용하여 번역에 필요한 토큰들을 따로 테이블에 저장하게 되고, 파싱이 성공적으로 끝나면 엔티티 테이블과 타입 테이블이 생성된다. 그러면, XML DTD 생성기와 XML 스키마 생성기는 엔티티 테이블과 타입 테이블의 각 요소들에 변환 규칙을 적용하여 각각 XML DTD 문서와 XML 스키마 문서를 생성해낸다.

3.2 EXPRESS 파서의 설계

EXPRESS 파서는 입력이 되는 EXPRESS 스키마의 문법 오류 검사 및 파스 트리 구축을 목적으로 한다. 이것은 파서 생성 도구를 사용하여 생성하며, 파서 생성도구를 사용할 때에는 EXPRESS 문법 규칙들을 모아놓은 명세를 입력으로 하여 파서를 생성한다. 파싱 과정에서 생성되는 파스 트리를 바탕으로 엔티티 테이블과 타입 테이블을 생성한다.

3.3 ENTITY 테이블과 TYPE 테이블의 설계

엔티티 테이블은 엔티티 이름(ENTITY name)과 그 엔티티가 갖는 애트리뷰트의 이름(ATTRIBUTE name), 애트리뷰트의 타입(ATTRIBUTE type), 집단데

이타타입인 경우 개별 값의 기본 타입(aggregation data type base_type), optional 여부(OPTIONAL flag), 집단 데이타타입의 최저개수(LOW), 집단데이타타입의 최대개수(HIGH)와 같은 총 7개의 항목으로 이루어진다.

타입 테이블은 사용자가 정의한 타입을 알아내기 위해 타입 이름(TYPE name)과 그 데이타 타입(TYPE type), 집단 데이타타입의 기본 타입(aggregation data type base_type) 등 3개의 항목으로 이루어진다.

EXPRESS의 구문을 분석하여 파스 트리가 생성되어 가는 과정에서 엔티티 테이블과 타입 테이블이 생성되므로, 특정 문법 생성 규칙에서의 특정 토큰이 발생할 때마다 이를 테이블에 넣는 의미수행코드가 필요하다.

3.4 EXPRESS-to-XML DTD 생성기의 설계

EXPRESS 스키마를 XML DTD로 변환할 경우, XML DTD는 그 데이타타입의 종류가 제한되어 있기 때문에 EXPRESS의 많은 데이타타입과 사용자 정의 타입을 그대로 표현해주지는 못한다. 또한 EXPRESS에 있는 문법 요소 중 ALGORITHM과 PROCEDURE, USE, REFERENCE는 ISO 10303-21 교환 파일 형식에 지정되어 있지 않으며 DTD에서는 표현할 방법이 없기 때문에 변환 대상에서 제외하였다.

파싱이 성공적으로 끝난 EXPRESS 스키마는 파싱 중에 만들어진 두 테이블을 이용하여 XML DTD로 변환된다. 이 때의 변환 규칙은 그림 3과 같이 나타낼 수 있으며 그 설명은 다음과 같다.

```
규칙1.SCHEMA -> root element
규칙2.ENTITY -> element 및 기본적으로 e-id가짐
규칙3.ATTRIBUTE
   (1)normal data type -> CDATA  or enumeration type
   (2)constructed type -> type table 검색
   (3)Entity name -> entity table 검색
   (4)Derived type -> 해당 항목을 명시만 함
규칙4.Local rule -> 해당 항목을 DTD에서는 명시만 함
규칙5.aggregation data type->subelement 및 그 item으로
규칙6.상속: 자식 entity는 부모 entity의 애트리뷰트 받음
규칙7.함수 및 프로시져는 변환 대상에서 제외
```

그림 3 EXPRESS 데이타를 XML DTD로 변환하는 규칙

규칙 1. EXPRESS에서의 각 SCHEMA는 XML DTD에서의 최상위(root) 엘리먼트가 되고, 이 SCHEMA 내의 모든 엔티티는 이 엘리먼트의 서브엘리먼트가 된다.

규칙 2. EXPRESS에서의 각 엔티티는 '<!ELEMENT 엔티티이름 .../>'로 변환되며, 기본적으로 e-id 애트리뷰트를 갖는다. 따라서, 'e-id CDATA #REQUIRED'는 모든 엘리먼트가 갖는다.

규칙 3. 각 애트리뷰트는 '<!ATTLIST 엘리먼트이름 ..' 으로 시작하고 각 애트리뷰트는 그 데이타타입에 따라서 다음과 같이 변환한다. 또한 각 애트리뷰트는 그 선언에서 'OPTIONAL'이 포함된 경우 '#IMPLIED'로 변환한다.

EXPRESS에서의 각 애트리뷰트는 우선 '애트리뷰트 이름'을 나타낸 다음, 애트리 뷰트를 EXPRESS에서 정의된 데이타타입 별로 다음과 같이 변환한다.

(1) REAL, INTEGER, STRING은 'CDATA'로, BOOLEAN은 '(true | false)'로, LOGICAL은 '(true | false | unknown)'으로 변환한다.

(2) TYPE 선언으로 만들어지는 Constructed 데이타타입은 TYPE선언에서 정의된 형태가 어떤 것인지 찾아보고 ①을 적용한다. 단, TYPE 선언에서의 타입이 집단 타입일 때에는 방법 5를 적용한다.

(3) 애트리뷰트가 엔티티이면, 그 애트리뷰트에 해당 엔티티이름을 넣고, 'CDATA'로 선언한다.

(4) DERIVE 애트리뷰트는 애트리뷰트 이름 앞에 'derived:' 를 붙이고, 이것이 참조하는 애트리뷰트의 데이타타입에 의해 결정한다.

(5) SELECT와 ENUMERATION는 나열된 값들의 타입에 따라 (1),(2),(3),(4)에 의해 표시한다.

규칙 4. Local rule의 경우는 뒤에 '(true | false | unknown) #IMPLIED'를 붙이고, inverse, unique, where의 순으로 각각 'inverse:'와 'unique:', 'where:'로 변환한다.

규칙 5. 애트리뷰트가 set, bag, list, array와 같은 집단 데이타타입인 경우, 이 애트리뷰트는 '엘리먼트이름.서브엘리먼트'가 된다. 이 서브엘리먼트는 그 특성에 따라 애트리 뷰트로서 boolean 타입인 unique, order 및 최소개수, 최대개수를 갖는다. 또한 이 서브엘리먼트는 인덱스와 해당하는 각 값을 애트리뷰트로 갖는 서브엘리먼트를 갖게 된다. 각각 다음에 기초하여 'unique (true| false) #FIXED "지정된 값"', 'order(true | false) #FIXED "지정된 값"'을 붙여준다.

SET : unique - true, order - false,
LIST : unique - false, order - true
BAG : unique - false, order - false
ARRAY : unique - false, order - true

규칙 6. EXPRESS에서의 엔티티가 SUBTYPE OF를 갖고 있다면, ( )안에 있는 엔티티가 갖는 애트리뷰트를 모두 계승해서 이 엔티티 목록에 넣는다. SUPERTYPE OF는 고려하지 않는다.

규칙 7. 기타 ALGORITHM, FUNCTION, USE, REFERENCE 등은 실제 교환 파일에서는 쓰이지 않으므

로 변환 대상이 아니다.

이 규칙을 바탕으로 하는 흐름도를 개략적으로 나타내면 다음과 같이 나타낼 수 있다.

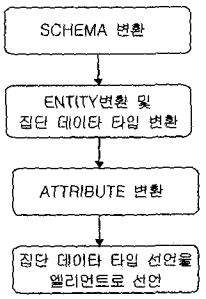

그림 4 EXPRESS 데이타를 XML로 변환하는 흐름도

3.5 EXPRESS-to-XML 스키마 생성기의 설계

XML 스키마는 XML DTD에 비해 다양한 데이타 타입 및 사용자 정의 데이타타입을 가지고 있으므로 EXPRESS의 각 데이타 타입을 될 수 있는 한 그대로 옮겼으며, 정의 가능한 타입은 따로 정의하였다. 또한 네임스페이스를 사용하여 유일성 확보 및 재사용이 가능하도록 하였다. 이 번역 부분을 실행하였을 경우, XML 스키마 파일인 .xsd 파일이 생성된다.

EXPRESS를 XML 스키마로 변환하는 규칙은 그림 5와 같이 나타낼 수 있으며 그 설명은 다음과 같다.

```
규칙 1.서두 부분 첨가
            Target namespace 및 e-id type와 logical type
정의
규칙 2.SCHEMA -> 최상위 엘리먼트로 선언
규칙 3.ENTITY -> element 및 e-id 애트리뷰트 선언
규칙 4.ATTRIBUTE
    (1) normal data type -> 각각에 해당하는 데이타타입으로 변환
    (2) constructed type -> type table 검색
    (3) Entity name -> entity table 검색
    (4) Derived type -> 해당 항목을 명시만 함
규칙 5.Local rule -> 해당 항목을 명시만 함
규칙 6.aggregation data type->subelement 및 그 item으로 list
      type
규칙 7.상속: 자식 entity는 부모 entity의 애트리뷰트 받음
```

그림 5 EXPRESS 데이타를 XML 스키마로 변환하는 규칙

규칙 1. 서두(Prologue) 부분을 쓴다. 서두는 XML의 버전 및 XML 스키마의 네임스페이스 정의를 포함한다. 네임스페이스는 각 구성요소들이 참조되었음을 나타내며, target 네임스페이스는 이 XML 스키마에 정의된 요소들이 이 네임스페이스에 있는 걸로 정해진다는 뜻으로, 여기서는 'http://www.hyejin.pe.kr/namespaces/EXPRESS'로 한다. 또, 자주 쓰이는 데이타 타입인 logical과 e-id를 정의한다. logical은 true, false, unknown의 값을 가지며, e-id는 엔티티를 지칭할 때 쓰인다.

규칙 2. EXPRESS에서의 SCHEMA는 XML 스키마에서는 최상위 엘리먼트로서 '<element name="스키마이름"../>'가 된다. EXPRESS에서의 각 엔티티는 이 엘리먼트의 부엘리먼트가 되어 <complexType>와 </complexType>의 사이에 '<element ref="엔티티 이름" minOccurs="0" maxOccurs="unbounded">'로 나열하여 표현한다.

규칙 3. EXPRESS의 각 엔티티는 엘리먼트가 되어 '<element name="엘리먼트 이름" ../>'으로 표현하고, e-id를 갖는다.

규칙 4. EXPRESS에서의 각 엔티티가 갖는 애트리뷰트는 다음을 따른다.

각 해당 엘리먼트에 대한 attribute는 <attribute Group name=" ">로 변환한다. 이 때 attribute의 데이타 타입은 다음과 같이 변환한다.

(1) normal type : OPTIONAL로 선언되면 use="optional">을 붙이고, 아닌 한 use="required">를 붙인다. EXPRESS에서의 애트리뷰트 타입이 REAL이면 XML 스키마에서는 type="double"로, INTEGER, STRING, BOOLEAN인 것은 XML 스키마에 있으므로 타입을 그대로 선언하여 사용한다. LOGICAL은 정의된 사용자 정의 타입을 사용한다. SELECT는 <choose>를 사용하여 정의한다.

(2) constructed type은 타입 테이블을 검색하여 해당하는 타입을 적용한다.

(3) 엔티티 이름인 경우 e-id 타입을 가지며, 해당하는 엔티티 이름을 애트리뷰트에 넣는다.

(4) derived type의 경우 해당 항목을 명시하고, derive한 애트리뷰트의 타입을 감안하여 선언한다.

규칙 5. local rule의 경우 XML DTD의 경우와 같고, 데이타타입은 logical을 사용한다.

규칙 6. SET, LIST, BAG, ARRAY와 같은 집단 데이타 타입은 XML 스키마에서의 list 선언에 바탕하여 다음의 애트리뷰트를 갖는다. 각각의 경우는 XML DTD에서의 경우와 같다. 단, SET에 minOccurs, maxOccurs가 설정된다는 점이 다르다.

규칙 7. 기타 ALGORITHM, FUNCTION, USE, REFERENCE 등은 실제 교환 파일에서는 쓰이지 않으므로 변환 대상이 아니다.

이 규칙들은 그 표현 양식은 XML DTD와 유사하며, 다만 사용자 데이타 타입인 e-id와 logical이 정의되었

으므로, unique, inverse, where 등이 logical로 선언된다는 점이 다르다. 따라서 그 기본적인 알고리즘은 그림 4와 같다.

4. EXPRESS에서 XML로 변환시키는 번역기의 구현

4.1 번역기 구현 환경

EXPRESS를 XML로 변환하는 번역기는 Windows NT workstation 4.0 운영체제 환경에서 JDK 1.1.8을 사용하여 구현하였다. 변환의 대상이 되는 EXPRESS 스키마 파일은 현재 쓰이고 있는 ISO 10303 시리즈 중 몇몇 파트에서 프로토콜로 사용되는 EXPRESS 문서들을 사용하였다. EXPRESS 문서를 파싱하기 위한 부분은 자바 파서 자동생성도구인 JavaCC를 사용하여 생성하였다. JavaCC를 개발한 환경은 JDK1.2 이상에서는 지원하지 않으므로[8] 비교적 낮은 버전의 자바환경에서 구현되었다.

4.2 번역기 각 부분의 기능

본 연구에서 구현된 EXPRESS-to-XML 번역기는 자바 언어를 사용하여 개발되었다. 자바는 플랫폼에 구애받지 않는다는 점에서 XML의 애플리케이션 및 시스템에 대해 독립적인 성질과 공통점을 가지며, 따라서 자바로 구현하면 XML의 장점을 잘 살릴 수 있다는 이점이 있다.

이 번역기는 파싱하는 부분, EXPRESS-to-XML DTD 번역 부분, EXPRESS-to-XML 스키마 번역 부분의 세 부분으로 나뉘어진다.

우선 파싱하는 부분은 JavaCC를 사용하여 구현하였다. JavaCC를 사용하여 EXPRESS를 파싱하는 부분을 만들어내기 위해서는 EBNF로 기술된 EXPRESS 문법 규칙을 바탕으로 하는 *.jj 파일이 필요하다. JavaCC는 이 *.jj 파일을 입력으로 받아 여러 .java 파일을 생성해 내며 이들을 사용하여 클래스 파일이 생성된다. 그림 6에서의 PARSER_BEGIN(Translator)과 PARSER_END(Translator) 사이에는 다음과 같이 클래스 Translator의 정의가 들어가며 여러 메소드를 제어할 명령을 수행할 main 메소드가 들어간다. main method 내에서는 syntax()라는 메소드를 호출하여 EXPRESS의 파싱이 이루어지도록 하였다. 여기까지는 기본적인 파서 생성에 필요한 부분이며, 다음 그림 6에서 보는 것과 같이 사용자 메소드를 삽입하여 파싱 외의 기능을 추가할 수 있다.

본 논문에서는 이 과정 중에 사용자가 정의한 의미수행코드가 실행되도록 하여 엔티티 테이블과 타입 테이블을 생성하였다. 이어서 XML DTD generator인

```
PARSER_BEGIN(Translator)
public class Translator{
. . . . . .
        public static void main(String args[]){
                Translator parser;
                ........
                parser.syntax(); //입력받은 문서를 파싱
                parser.translator(pw); //XML DTD 생성
                parser.stranslator(spw);//XML스키마생성
                ........
        }
        public void translate(PrintWriter pw)
        {       ........
        }
        public void stranslate(PrintWriter spw)
        {       ........
        }
}
PARSER_END(Translator)
```

그림 6 .jj 파일의 부분

translator() 메소드와 XML 스키마 generator인 stranslator() 메소드를 main 메소드 뒤에 정의하고, main 메소드에서 호출하여, .dtd 파일과 .xsd 파일이 생성되도록 하였다. 각 문법 규칙은 JavaCC에서 정의한 입력 형식으로 표현한다. EBNF 형식의 각 규칙은 각각 하나의 함수가 되며 말단 심볼을 제외한 비말단 심볼은 또다른 함수가 된다.

이 부분을 JavaCC에서의 입력 파일인 .jj 파일에서의 형식으로 나타내면 다음과 같은 표현이 된다. JavaCC는 recursive descent 파서이기 때문에 각 문법규칙이 프로시져, 즉 자바에서는 메소드가 된다. 이 문법규칙에 필요한 다른 비단말 심볼 또한 메소드이므로 메소드 호출 방식의 표현이 사용되고, 단말 심볼은 .jj 파일에서의 TOKEN : {...}에 정의해놓은 토큰 중 이 문법 규칙에 필요한 토큰으로 표현한다. 즉, .jj 파일에서의 <PLUS>는 TOKEN 정의 부분에서는 < PLUS: "+" > 로 정의되어 있다. .jj 파일에서의 EXPRESS 문법의 모든 명세는 이와 같은 방식으로 표현하였다. 이 중 의미수행코드는 {와 }사이에 삽입하며 이 부분은 .java 파일에서의 진술문으로 그대로 옮겨진다.

본 연구에서 필요한 EXPRESS 문법 중 일부의 문법 규칙을 BNF로 표현하면 그림 7과 같으며 이를 JavaCC에서의 입력 파일로 나타낸 것은 그림 8과 같다.

```
<actual_parameter_list>::= '('<parameter> (','<parameter>)* ')'
<add_like_op>::= '+' | '-' | 'OR' | 'XOR'
<aggregate_initializer>::='[' (<element> (','<element>)*)?  ']'
<aggregate_source>::= <simple_expression>
```

그림 7 BNF로 표현된 EXPRESS 문법 규칙의 일부

```
/********** PRODUCTION 157 **********/
void actual_parameter_list() : {}
{
    <LPAREN> parameter() (<COMMA> parameter())* <RPAREN>
}
/********** PRODUCTION 158 **********/
void add_like_op() : {}
{
    <PLUS> | <MINUS> | <OR> | <XOR>
}
/********** PRODUCTION 159 **********/
void aggregate_initializer() : {}
{
    <LBRACKET> (element() (<COMMA> element())*)?
    <RBRACKET>
}
```

그림 8 JavaCC의 입력 파일에서의 문법 규칙의 일부

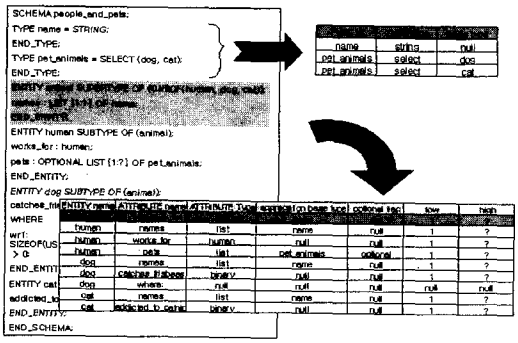

그림 9 EXPRESS 문서로부터 생성된 엔티티 테이블과 타입 테이블

그림 8은 그림 7의 문법 규칙을 나타낸 JavaCC 입력 파일이며, 그림 9는 왼쪽에 보이는 EXPRESS 문서인 people.exp로부터 그림 8의 각 함수가 포함하는 의미수행코드에 의해 테이블들이 생성되는 것을 보여준다.

첫 번째 TYPE 선언에서는 TYPE 이름인 name이 TYPE id가 되고, 이것의 애트리뷰트 타입인 STRING이 TYPE type이 되었다. STRING은 집단 데이타타입이 아니므로 base_type은 null이 된다. 다음, ENTITY animal의 선언을 보면, 이것이 가지고 있는 name이라는 애트리뷰트는 기본타입이 names인 LIST 형태의 데이타타입을 가지고 있음을 알 수 있다.

또한 human, dog, cat이라는 엔티티의 supertype이기도 하지만 이것은 교환 형식에서는 영향을 미치지 않으므로 무시한다는 변환규칙에 따라 특별한 행동을 취하지 않는다. 따라서, ENTITY name에는 animal이, ATTRIBUTE name에는 names가 들어가고, ATTRIBUTE type에는 LIST가, list의 기본 타입인 names가 aggregation base type에 들어가고, optional이 없으므로 optional flag는 null 상태가 된다. 또한 name이라는 애트리뷰트가 list type이고, 최소개수와 최대개수가 각각 1과 ?이므로 LOW에는 1이, HIGH에는 ?가 들어가게 된다.

translate 함수와 stanslate 메소드는 각각 EXPRESS 스키마를 입력으로 받아들여 파싱 중에 생성된 위의 두 테이블을 바탕으로 .dtd 파일과 .xsd 파일을 만들어낸다.

4.3 실험 자료 및 결과

변환의 대상이 될 파일은 미국 NIST의 SOLIS에 저장되어있는 part22의 dict.exp 파일로서 그 내용은 다음과 같다[9].

```
SCHEMA SDAI_dictionary_schema;
TYPE base_type = SELECT
  (simple_type,
  aggregation_type,
  named_type);
END_TYPE;
TYPE constructed_type = SELECT
  (enumeration_type,
  select_type);
END_TYPE;
.....
TYPE info_object_id = STRING;
END_TYPE;
ENTITY schema_definition;
  name            : express_id;
  identification : OPTIONAL info_object_id;
INVERSE
  entities        : SET [0:?] OF entity_definition FOR
parent_schema;
  types       : SET [0:?] OF defined_type FOR parent_schema;
  global_rules  : SET [0:?] OF global_rule FOR parent_schema;
  external_schemas : SET [0:?] OF external_schema FOR native
  _schema;
UNIQUE
  UR1 : identification;
END_ENTITY;
ENTITY interface_specification;
  current_schema_id : express_id;
  explicit_items    : SET [1:?] OF explicit_item_id;
  implicit_items    : SET [0:?] OF implicit_item_id;
END_ENTITY;
```

그림 10 dict.exp

그림 10의 dict.exp를 바탕으로 하는 엔티티 테이블과 TYPE 테이블의 부분은 각각 표 1 및 표 2와 같이 생성되었다. 이들을 바탕으로 XML DTD generator에 의해 XML DTD로 변환된 일부는 그림 10과 같다. 이 파일에서는 최상위 엘리먼트로 EXPRESS에서의 스키마이름인 SDAI_dictionary_schema가 들어가게 된다. 여기에 EXPRESS에서의 엔티티 이름이 모두 서브엘리먼트로서 (이름 1 | 이름 2 | ..)*로 들어가게 됨을 볼 수 있다.

표 1 dict.exp로부터 생성된 엔티티 table의 부분

entity이름	attribute이름	attribute 타입	집단데이타기본타입	optional 여부	low	high
schema_definition	name	express_id	null	null	null	null
schema_definition	identification	info_object_id	null	optional	null	null
schema_definition	inverse:	null	null	null	null	null
schema_definition	unique:identification	null	null	null	null	null
interface_specification	current_schema_id	express_id	null	null	0	?
interface_specification	explicit_items	set	explicit_item_id	null	1	?
interface_specification	implicit_items	set	implicit_item_id	null	0	?

다음으로 각 엔티티는 <!ELEMENT ..>로 나타내어지며, 이 엔티티의 애트리뷰트는 <!ATTLIST ..>로 표현되었다. 엔티티 schema_definition은 집단 데이타 타입이 없으므로, 엘리먼트 선언이 EMPTY로 끝나게 된다.

표 2 dict.exp로부터 생성된 type table의 부분

type_name	type	aggregation_base
base_type	select	simple_type
null	null	aggregation_type
null	null	named_type
constructed_type	select	enumeration_type
null	null	select_type

한편, dict.exp로 XML 스키마 generator인 stran slate에 의해 XML 스키마로 변환한 결과는 그림 12와 같다. 생성된 이 dict.xsd를 살펴보면, 머리 부분에는 스키마 네임스페이스 정의가 나와 있어서, 여기서 정의된 엘리먼트은 "my:"라는 접두어를 붙이게 하였다. 다음

```
<!ELEMENT SDAI_dictionary_schema
(schema_definition|interface_specification|interfaced_item|explicit
_item_id|.....|bag_type|list_type|array_type|integer_bound)*>
.....
<!ELEMENT interface_specification (interface_specification.explicit
_items,interface_specification.implicit_items)>
<!ATTLIST interface_specification
        e-id            CDATA       #REQUIRED
        current_schema_id           CDATA       #REQUIRED>
<!ELEMENT interface_specification.explicit_items (interface_
  specification.explicit_items-item)*>
<!ATTLIST interface_specification.explicit_items
        unique          (true|false)  #FIXED      "true"
        ordered         (true|false)  #FIXED      "false"
        low_bound       CDATA       #FIXED      "1"
        high_bound      CDATA       #FIXED      "unbounded">
<!ELEMENT interface_specification.explicit_items-item EMPTY>
<!ATTLIST interface_specification.explicit_items-item
        value           CDATA       #REQUIRED
        index           CDATA       #REQUIRED>
<!ELEMENT interface_specification.implicit_items
        (interface_specification.implicit_items-item)*>
```

그림 11 dict.dtd의 부분

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
targetNamespace="http://www.hyejin.pe.kr/namespaces/EXPRESS"
xmlns:xsi="http://www.w3c.org/2000/10/XMLSchema"
xmlns:my="http://www.hyejin.pe.kr/namespaces/EXPRESS">
<simpleType name="logical">
        <restriction base="string">
                <enumeration value="true"/>
                <enumeration value="false"/>
                <enumeration value="unknown"/>
        </restriction>
</simpleType>
<simpleType name="e-id">
        <restriction base="string">
                <maxlength value="5"/>
                <pattern value="#Wd{4}"/>
        </restriction>
</simpleType>
<element name="SDAI_dictionary_schema">
  <complexType>
    <choice minOccurs="0" maxOccurs="unbounded">
      <element ref="my:schema_definition" minOccurs="0"
       maxOccurs="unbounded"/>
          <element ref="my:interface_specification"
           minOccurs="0" maxOccurs="unbounded"/>
        .....
          <element ref="my:referenced_item" minOccurs="0"
           maxOccurs="unbounded"/>
        .....
          <element ref="my:integer_bound" minOccurs="0"
           maxOccurs="unbounded"/>
    </choice>
  </complexType>
</element>
```

그림 12 dict.xsd의 부분

e-id와 logical 데이타타입의 정의가 있은 후, dict.exp에서의 스키마 이름인 SDAI_dictionary_schema가 최상위 엘리먼트로서 <element name = "SDAI_dictionary_schema">로 보임을 알 수 있다. 이의 하위 엘리먼트는 모두 <choice minOccurs="0" maxOccurs="unbounded">와 </choice> 사이에 <element ref= " . . . " />로 정의되어 있음을 알 수 있다. 그 다음에는 dict.exp에서 정의되어 있는 엔티티인 schema_definition이 <element name="schema_definition">으로 정의되었고, 이 엔티티의 애트리뷰트들은 집단 데이타 타입을 가지고 있지 않으므로 서브엘리먼트는 존재

하지 않는다. 따라서 <comlexType content="empty">가 붙게 된다. 이 엔티티가 가지고 있는 애트리뷰트들은 모두 <attributeGroup name="schema_definition">과 </attributeGroup> 사이에 정의된다. 기본적인 애트리뷰트로 <attribute name="e-id" type="my: "e-id" use="required"/>가 선언되고, 다음 이 엔티티가 원래 가지고 있던 name, identification, INVERSE, UNIQUE가 각각의 타입과 함께 선언된다. INVERSE와 UNIQUE는 특별한 값을 요구하는 것이 아니므로 use="optional"로 선언됨을 알 수 있다.

schema_definition 엔티티를 두고 볼 때, e-id 애트리뷰트의 경우에 XML DTD에서는 CDATA로만 표현이 되었으나, XML 스키마에서는 e-id로 그 범위가 좀더 정확히 표현되었으며, inverse, unique는 구체적 정보가 손실되며, 이들의 데이타 타입이 XML DTD에서는 (true| false| unknown)으로 나타나지만, XML 스키마에서는 logical이라는 사용자 정의 데이타 타입으로 나타나게 되어 그 의미를 좀더 정확히 전달할 수 있다.

이렇게 변환된 DTD 및 XML 스키마에 기초한 XML 인스턴스 파일은 교환하는 직접적인 제품 정보가 될 수 있다.

5. 결론 및 향후 연구

본 연구에서는 EXPRESS를 XML로 표현하는 방법을 연구하고, 이로 변환함으로써 EXPRESS로 이루어진 자료를 웹 상에서 교환할 수 있는 방법을 제시하였다.

EXPRESS로 쓰여진 하나의 스키마가 ISO 10303-21를 따르는 여러 교환 파일의 틀 역할을 한다는 것과 같은 맥락에서, EXPRESS를 XML로 나타내기 위한 방법으로 EXPRESS 스키마 각각이 하나의 XML DTD 및 XML 스키마가 되도록 하는 early binding 방법을 사용하였다. 이러한 방법에 기초하여 EXPRESS의 XML DTD 및 XML Schema로의 변환 형태 및 변환 규칙을 세웠다. 이를 바탕으로 자바 파서 생성도구인 JavaCC에 의해 생성된 EXPRESS 파서와 XML DTD 생성기, XML Schema 생성기로 구성된 EXPRESS-to-XML 번역기를 개발하였다.

EXPRESS를 XML DTD 및 XML Schema로 표현할 경우, EXPRESS를 사용하는 소프트웨어로 이루어졌던 제품 정보 교환의 폭 및 수단이 웹으로 인해 더 확장될 수 있다.

또한 본 연구에서 이루어진 번역은 전자상거래에서 기업들이 각각의 제품의 정보를 효율적으로 교환할 수 있는 바탕이 될 수 있을 것이다. 또, 번역된 데이타를 바탕으로 제품 정보의 데이타베이스 구축을 하여 관리하고 사용할 수 있을 것이다.

본 연구에서 제안한 번역기를 통해 만들어진 XML DTD 및 XML 스키마에 맞는 XML 인스턴스도 필요하다. 따라서 ISO 10303-21형식의 파일을 XML 인스턴스로 변환하는 번역기도 제작된다면, 제품 정보 교환을 웹 상에서 할 수 있는 환경이 될 것이다.

참 고 문 헌

[1] Douglas Schenck and Peter Wilson, Information Modeling the EXPRESS Way, Oxford University, 1994.
[2] ISO, "Part 21: Implementation methods: Clear text encoding of the exchange structure," 1994.
[3] ISO TC184/SC4/WG11 N101, "Part 28: Implementation methods: XML representation of EXPRESS-drive data," 1999.
[4] PDML.org, http://www.pdml.org/xmlref.html
[5] FirstStep EXML, http://www.pdml.org/exmlintro.html
[6] STEP Tools, Inc., http://www.steptools.com/projects/xml/Compare_binding.pdf
[7] 한순흥, 이현찬, "디지털 제조를 위한 STEP", 인터넷 버전, 2000년 http://kstep.kaist.ac.kr/kstep_introduction/step_book/
[8] metamata's JavaCC homepage, http://www.metamata.com/javacc/
[9] 미국 NIST SOLIS, http://www.nist.gov/sc4/ 구현중 예 : http://www.nist.gov/sc/step/parts/part022/dis/dict.exp

이 기 호
정보과학회논문지 : 컴퓨팅의 실제
제 8 권 제 1 호 참조

김 혜 진
1999년 2월 이화여자대학교 환경공학과/컴퓨터학과(복수전공) 학사. 2001년 2월 이화여자대학교 과학기술대학원 컴퓨터학과 석사. 2001년 3월 ~ 현재 (주)삼성전자 영상디스플레이사업부 SW그룹 연구원. 관심분야는 XML, 인터넷 언어, 실시간 시스템, 데이타베이스