메타데이터 인터페이스를 이용한 DTD 기반 XML 문서 변환기의 골격 원시 코드 생성

최 귀 자[†] · 남 영 광[††]

요 약

본 논문에서는 목표 문서의 DTD에 정의된 구조에 따라 원시문서를 목표 문서로 변환하는 XML 문서변환기와 골격 원시 프로그램을 생성하는 방법을 제안한다. 사용자는 생성된 프로그램을 이용하여 코드를 추가하거나 프로그램을 변경하고, 외부 클래스나 라이브러리 파일들을 연결하여 자신이 원하는 방법으로 쉽게 문서를 변환할 수 있다. 생성된 프로그램은 목표 DTD 경로(path)를 기준으로 코드를 생성하기 때문에 가독성이 높다. 현재까지의 대부분의 XML 문서변환은 XSLT를 이용하거나 XQuery를 이용하여 Java 프로그램을 생성하고 있으나, 각 요소별로 코드를 조작할 수 없는 단점이 있다.

본 논문에서 제안된 방법은 사용자가 원시/목표 문서에서 제공되는 DTD 혹은 문서에서 자동으로 추출된 DTD를 이용하여 원소 사이의 관계를 지정하면 자동적으로 문서가 변환되고 Java 원시 프로그램을 생성하기 때문에 코드 생성과정이 매우 단순하다. 메타데이터 인터페이스는 Java GUI를 이용하여 트리 형태로 표현된 DTD의 원소를 클릭하여 쉽게 생성할 수 있다. 문서변환을 위한 매핑은 원소의 특성에 따라 1:1, 1:N, N:1로 구분되며 데이터의 분해 혹은 합성을 위한 사용자 지정 함수가 메타데이터 인터페이스에 저장된다. 본 시스템은 실제 사용되고 있는 논문 XML 문서를 서지문서로 변환한 결과 및 프로그램 코드를 예제로 구현하여 결과를 제시하였다.

키워드 : XML, DTD, 문서변환, 메타데이터 인터페이스

Skeleton Code Generation for Transforming an XML Document with DTD using Metadata Interface

Guija Choe[†] · Young-Kwang Nam[††]

ABSTRACT

In this paper, we propose a system for generating skeleton programs for directly transforming an XML document to another document, whose structure is defined in the target DTD with GUI environment. With the generated code, the users can easily update or insert their own codes into the program so that they can convert the document as the way that they want and can be connected with other classes or library files. Since most of the currently available code generation systems or methods for transforming XML documents use XSLT or XQuery, it is very difficult or impossible for users to manipulate the source code for further updates or refinements. As the generated code in this paper reveals the code along the XPaths of the target DTD, the result code is quite readable.

The code generating procedure is simple; once the user maps the related elements represented as trees in the GUI interface, the source document is transformed into the target document and its corresponding Java source program is generated, where DTD is given or extracted from XML documents automatically by parsing it. The mapping is classified 1:1, 1:N, and N:1, according to the structure and semantics of elements of the DTD. The functions for changing the structure of elements designated by the user are amalgamated into the metadata interface. A real world example of transforming articles written in XML file into a bibliographical XML document is shown with the transformed result and its code.

Key Words : XML, DTD, Document Translation, Metadata Interface

1. 서 론

XML이 문서 혹은 데이터 표현의 형태로 자리 잡은 지 약 10년이 경과하고 있으나 단지 XML 태그를 이용하여 데이터를 표시하기 위한 공통적인 수단으로 성공했을 뿐 아직도 해결해야 할 과제가 많다. 그 중에서 중요하게 고려되고 있는 문제점의 하나는 서로 다른 형태를 가진 문서 사이의 자료 변환, 즉 주어진 문서를 사용자가 원하는 형태로 손쉽게 변환할 수 있는 기능에 관한 것이다. 데이터베이스 혹은 소프트웨어 공학 분야에서 많은 표준화 노력을 통하여 문서를 작성하거나 데이터를 표현하는데 CDIF[1] 혹은 XMI[2]

† 정 회 원 : 연세대학교 전산학과 박사과정
†† 정 회 원 : 연세대학교 전산학과 교수
논문접수 : 2006년 1월 31일, 심사완료 : 2006년 6월 2일

를 이용하여 표준화 노력을 기하고 있으나 그 작업에는 한계가 있다. 각 벤더나 사용자의 자료 처리 결과, 저장형태, 태그 혹은 속성 이름까지도 완전히 표준화하고 통일 할 수 없다. 따라서 많은 사용자가 자료교환을 위한 데이터나 문서의 표현 형태는 XML이라는 것에는 공감하지만 표현된 문서간의 이질성 즉 태그나 구조적 차이점을 극복하여 사용자가 원하는 형태로 변환하는 작업이 필요하다.

문서변환을 자동화하기 위한 여러 가지 방법들이 제안되고 있으며, 크게 두 가지 종류로 나눌 수 있다. 첫 번째는 XML 문서간의 자료변환을 위하여 변환 결과가 XSLT[3]로 나타나거나 작성하는 것으로써 XSLT 스크립트 혹은 태그를 이용하여 다시 XML 문서를 작성해야 하는 단점이 있다. 단순히 XML 태그 형태를 이용하여 XML 문서를 생성하고 사용하는 이용자라도 XSLT를 배우고 프로그래밍 수준의 XML 스크립트를 작성해야 한다. 두 번째는 중간언어나 스크립트를 사용하는 방법[4, 5]으로 문서변환을 위한 규칙을 정의해야 하고 처리기를 필요로 하므로 이 또한 일반 사용자가 배우고 적용하기 어렵다.

이러한 문제점을 극복하기 위해서 본 논문에서는 원시(source) XML 문서를 목표(target) 문서 DTD의 구조로 변형하기 위한 프로그램 코드를 생성하여 사용자 자신의 클래스와 시스템으로 쉽게 변경할 수 있는 방법을 제안한다. 골격 원시 코드는 목표 DTD의 각 요소에 대해 XPath 형태로 생성되므로 프로그램의 구조를 쉽게 파악할 수 있다. 문서변환을 위해서 사용자는 단지 원시 DTD와 목표 DTD를 파악하여 어떤 *SDE (Source DTD Element)*가 어떤 *TDE(Target DTD Element)*와 매핑되며, 분해되거나 합성될 때 필요한 함수이름이 무엇인지를 지정하면 자동적으로 원시 XML 문서 (*SXD : Source XML Document*)가 목표 XML 문서 (*TXD: Target XML Document*)로 변환되고 골격 원시 코드가 생성된다. 이때 SDE와 TDE 간의 매핑은 Java GUI 환경에서 해당 원소를 마우스를 클릭하여 쉽게 지정할 수 있다.

SDE와 TDE간의 매핑은 1:1, 1:N, N:1의 세 종류로 구분된다. 본 논문에서의 접근 방법은 메타데이터 인터페이스인 XDMI (XML Documents Metadata Interface)에 저장된 매핑 정보를 기반으로 시스템을 구현하는 것으로써, XDMI 파일은 XML 문서의 이름이나 위치정보 및 경로정보, DTD의 요소나 경로에 관한 의미정보를 포함한다.

본 시스템은 문서를 변환하기 위해서 요소들에 관한 매핑을 지원하기 위한 GUI 도구와 변환을 실행하는 변환기 및 골격 원시 코드 생성기로 구성되어 있다. 매핑 지원 도구는 목표 문서의 DTD 혹은 문서 자체를 파싱해서 각 요소에 대한 경로를 생성하며, 사용자가 클릭하면 트리를 펼치거나 접을 수 있는 동적인 경로 트리를 제공한다[6]. 목표 요소들은 대응하는 SDE 및 변환 함수 이름과 연결되는 매핑 요소를 클릭하여 가시적인 색인번호가 할당되며, 시스템에서 색인번호를 내부적으로 수집 해서 얻은 매핑 정보를 기반으로 XDMI가 생성 된다.

프로토타입 시스템은 Windows 환경에서 Java와 DOM을 이용하여 개발하였다. 본 논문의 구성은 다음과 같다. 2장에서는 XML 문서 변환 및 코드생성에 대한 관련 연구를 조사하였으며, 3장에서는 XDMI 생성기와 XML 및 코드 생성기로 구성된 문서변환기 전체 시스템의 구조를 설명하고, 4장에서는 XDMI 생성 방법과 XML 변환과 코드 생성 알고리즘에 대하여 상세하게 기술한다. 5장에서는 구현결과를 보이기 위해서 실제 사용되고 있는 석박사논문 XML 문서를 논문정보 XML 문서로 변환하는 실행 결과를 나타내었다.

2. 관련 연구

XML 문서를 변환하기 위해서 제시된 방법들 중에서 매핑을 이용한 방법을 먼저 살펴본다. [7]에서는 메타데이터 레지스트리를 이용하여 매핑 테이블과 그에 대한 트리 및 트리에 대한 노드 이름을 변환하는 방법을 제시하였다. [8]에서는 데이터 레지스트리를 이용하여 비슷한 태그를 찾아내고 XTF라는 중간 파일을 이용하여 문서를 변환한다. 위의 두 시스템은 모두 레지스트리를 이용해야 하며 변환을 부분적으로 지원하는 단점이 있다. [9]의 변환기에서는 원시 스키마와 목표 스키마에 대한 매핑을 GUI를 이용하여 수행하지만 문서를 직접 변환하지 않고 XSL 스타일 시트를 생성한다. 이 방법은 매핑시 속성처리기를 참조하여 매핑이 이루어지기 때문에 매핑 과정이 복잡한 단점을 가진다.

XTGen[10] 시스템은 DTD를 분석하여 매핑정보를 (노드, 부모노드, 자식노드, 키 리스트)로 표현한다. 이 방법은 매핑 정보를 자동적으로 추출하기 때문에 비슷한 문서에 대해서만 변환이 가능하고 1:N 혹은 N:1 매핑에 의한 데이터 변환은 적용할 수 없는 단점을 가진다.

XSLT를 이용한 여러 방법들이 소개되고 있다. [4]에서는 원서 문서와 목표 문서의 원소 사이의 관계를 합성과 특성화에 의해서 관계를 정의하여 DTD 원소를 변환하는 방법으로 제시하였으며, [5]에서는 *renaming, production, deletion* 연산에 의해서 XML 문서를 변환하는 알고리즘을 제시하였다. [11]에서는 XSLT로 표현된 변환 규칙을 사용하여 문서를 변환하였으며, [12]에서는 XT3D라는 고급 XML 명세 언어를 이용하여 스키마에 대한 구현 시스템을 이용하여 XML 변환 방법을 제시하였다. [13]에서는 변환단계를 구조식별 (*structure identification*), 변환명세 (*transformation specifications*), 변환수행 (*performing transformation*)의 세 단계로 나누어서 변환하지만 단지 규칙 명세에 의해서 변환할 수 있다는 것만을 보였다. [14]에서는 스키마를 변환하기 위한 연산, 예를 들어, *add, insert, delete* 와 같은 연산을 사용하여 스키마를 변환하고 변환된 스키마를 바탕을 XSLT 스크립트를 생성하는 방법을 취하였다. [16, 17]에서는 여러 유형의 문서를 XSLT 및 XQuery를 이용하여 XML 문서로 변환하고 Java 코드를 생성하고 있으나 변환 자체를 위한 코드의 생성이며, 목표 문서의 각 요소별로 코드를 다룰 수 있는 방법은 제공하지 않는다.

이상의 방법들을 정리해보면 문서를 변환하기 위해 대개

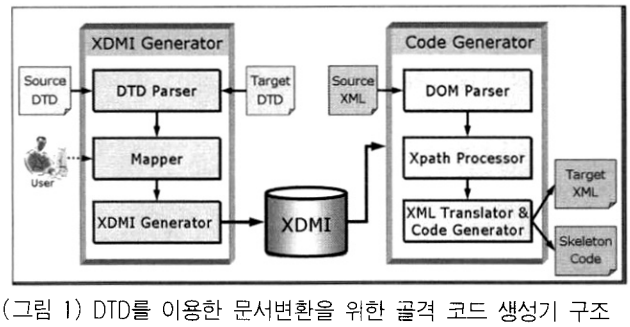
새로운 규칙 혹은 연산을 정의하고 정의된 규칙을 이용하여 문서변환을 위한 XSLT 스크립트를 생성하는 방법으로 요약된다.

본 논문에서 제시한 방법은 앞서 사용된 방법들과는 달리 원시 문서와 목표 문서에서 사용된 DTD의 의미만 알면 이를 매퍼를 이용하여 매핑하고 데이터 변환을 위한 함수를 동시에 추가하여 쉽게 문서를 변환할 수 있는 장점을 가지고 있다. 사용자가 새로이 배우거나 추가해야 할 규칙이 없으며, XSLT를 이용하지도 않기 때문에 별도의 처리기가 필요없다.

3. 코드 생성기 구조

본 시스템의 구조는 (그림 1)과 같다. 구조적, 의미적으로 상이한 원시문서를 목표문서로 변환하기 위해 필요한 메타정보를 포함하는 XDMI 생성기(XDMI Generator)와 원시문서를 XDMI에 저장된 메타 정보를 이용하여 목표 문서로 변환하고 골격 원시 코드를 생성해 주는 코드생성기(Code Generator)로 구성된다. XDMI 생성기는 DTD 파서, 매퍼, XDMI 생성기를 포함한다.

XDMI 생성기는 원시문서의 DTD가 없을 경우에는 원시 XML 문서를 파싱하여 자동적으로 경로 트리를 제공하며, 사용자 매핑 과정을 통해 XDMI를 생성한다. 문서 변환기는 사용자가 생성한 메타정보와 원시 XML 문서를 이용하여 목표 XML 문서를 생성한다. XDMI 생성기와 코드 생성기는 각각 4.2절과 4.3절에서 상세히 설명한다.



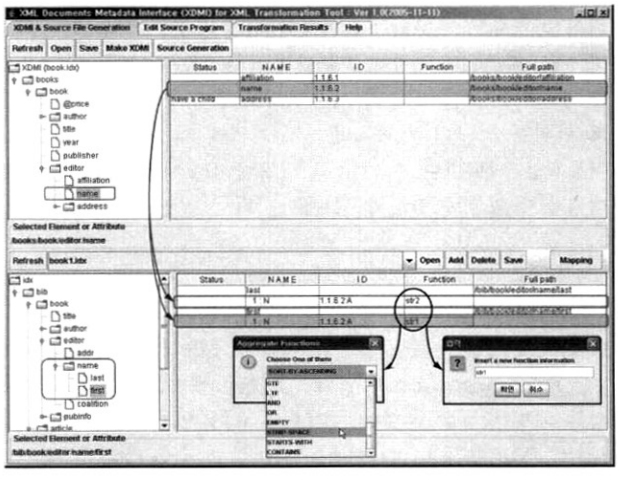
(그림 1) DTD를 이용한 문서변환을 위한 골격 코드 생성기 구조

각 문서에 대한 매핑 정보는 각각의 XML 문서 자체 혹은 문서의 DTD 파일을 파싱하여 생성된다. 원시 DTD의 각 노드에 식별 번호를 부여하고, 이 번호를 목표 경로 트리의 동일한 의미를 가진 노드에 할당함으로써 목표 문서의 노드 집합과 매치한다. 동일한 번호를 갖는 모든 노드를 모아 <source>와 <target> 경로의 집합들로 자동 생성하고, XDMI는 같은 색인 번호를 갖는 경로를 묶어서 생성된다.

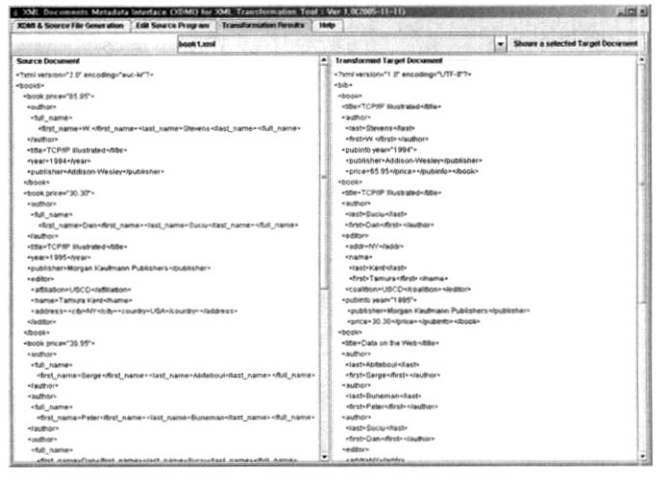
원시 문서와 목표 문서의 노드 사이의 매핑을 쉽게 하기 위해서 (그림 2)와 같은 GUI를 제공한다. (그림 2)의 좌측 상위 프레임에 원시 문서 *'book.idx'*를 루트 이름으로 갖는 동적인 경로 트리를 *'book.DTD'*로 부터 생성하고, 좌측 하위 프레임의 *'bib.idx'* 트리는 목표 문서 *'bib.DTD'*의 DTD를 파싱하여 생성한다.

우측 상위 프레임은 트리의 각 노드에 대한 색인 번호와 전체 경로를 포함한다. 사용자가 원시 노드와 매핑될 목표 노드를 선택 한 후 'Mapping' 버튼을 누르면, 연관된 색인 번호가 'ID' 필드에 복사되며, 1:N과 N:1 매핑을 위한 함수 벡터를 팝업 윈도우에 입력하면, 입력된 함수가 'Function' 필드에 복사된다. 이러한 매핑 과정은 모든 목표 문서 파일에 대해 필요하며, 사용자는 수정 및 새로운 매핑 작업을 필요 시마다 수행할 수 있다. 모든 연관된 색인 번호를 설정한 후 'Make XDMI'와 'Source Generation' 버튼을 누르면 XDMI 파일과 골격 원시 프로그램이 생성된다.

문서변환은 (그림 2)의 화면에서 생성된 XDMI를 기반으로 사용자가 원시 XML에 대해 (그림 11)의 골격 코드 편집기 화면에서 'XML Transformation' 버튼을 누르면 문서가 변환되며, (그림 3)의 화면에서와 같이 변환된 목표 문서의 결과를 확인할 수 있다.



(그림 2) 원시문서와 목표문서 경로를 매핑하기 위한 GUI 인터페이스



(그림 3) 목표 문서 결과 화면

4. XDMI 생성과 문서변환 알고리즘

4.1 원시 DTD와 목표 DTD 간의 매핑

*SD(Source DTD)*는 원시 DTD를, *TD(Target DTD)*는 목표 DTD를 의미한다. *S*를 원시 DTD *SD*의 경로 트리 *ST*의 노드 집합, *T*를 목표 DTD *TD*의 경로 트리 *TT*의 노드 집합이라 가정하고, *PS*와 *PT*를 각각 *S*와 *T*의 멱집합이라 하자. 각 트리의 노드 o_i 는 노드명 ol_i 와 노드값 ov_i 로 구성된 객체 (ol_i, ov_i)가 된다. (그림 4)에서 '*title*' 노드는 노드명 '*title*'과 노드값 '*TCP/IP Illustrated*'를 갖는다.

[정의 1] *PS*와 *PT*의 카티션곱 $CP = PT \times PS$는 (t, s) 원소들의 집합이다. 여기서 $s = (sn_1, sn_2, ..., sn_m)$ 이고 $sn_i \in L$ 이며, $t = (tn_1, tn_2, ..., tn_m)$이고 $tn_i \in G$ 이다.

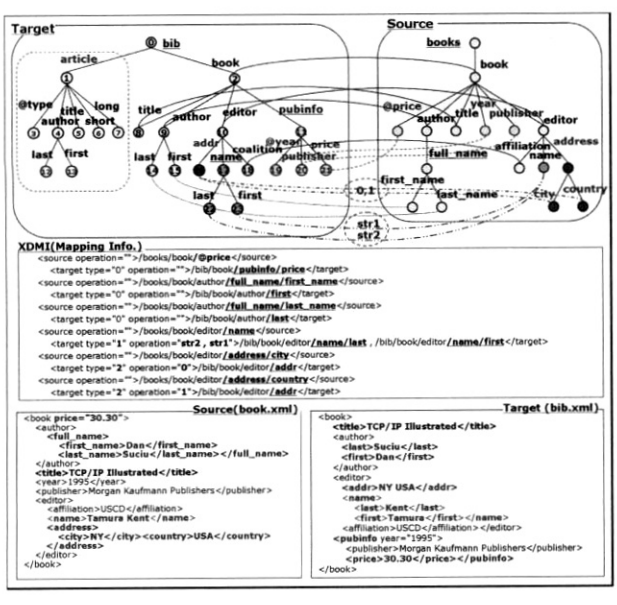
[정의 2] *TD*와 *SD*간의 매핑 $M(TD, SD) \subset CP(TD, SD)$는 $M_{11}(TD, SD) \cup M_{1N}(TD, SD) \cup M_{N1}(TD, SD)$ 이며, 다음 조건을 만족한다.

i) $M_{11}(TD, SD)$는 일대일 매핑 원소들의 집합 으로서, 단일원소집합 (singleton) 이고, $t \in PT$ 에 대해 $m_{11}(t) = sl$ 인 원소 $m_{11} = (t, s) \in M(TD, SD)$의 집합이다.

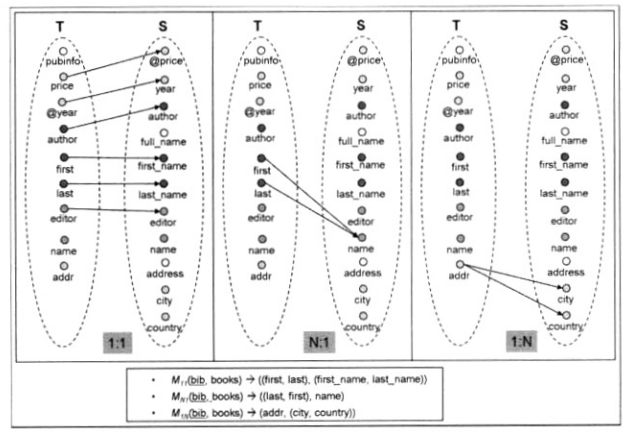
ii) $M_{N1}(TD, SD)$는 다대일 매핑 원소들의 집합 으로서, 단일원소집합 $s \in PS$ 이고, $t = (tn_1, tn_2, ..., tn_m)$ (단 $m \geq 2$)인 $t \in PT$ 에 대하여 $m_{N1}(t) = s$ 인 $m_{N1} = (t, s) \in M(TD, SD)$의 원소들의 집합이다.

iii) $M_{1N}(TD, SD)$는 일대다 매핑 원소들의 집합 으로서, $s = (sn_1, sn_2, ..., sn_m)$ (단 $m \geq 2$)은 $s \in PS$ 이고, 단일원소집합 $t \in PT$ 에 대해 $m_{1N}(t) = s$ 인 원소 $m_{1N} = (t, s) \in M(TD, SD)$의 집합이다.

iv) 이외의 다른 원소 $t \not\supset (domain(M_{11}) \cup domain(M_{N1}) \cup domain(M_{1N}))$에 대해서 $m_0(t) = \phi$ 이다.



(그림 4) 원시 문서와 목표 문서 간의 매핑



(그림 5) 그림 4에서의 노드를 집합으로 표현한 매핑

만약 문맥상 이러한 의미가 명백하다면, 간단히 M 과 m 으로 표현한다.

sn_i와 tn_i는 각각 트리 *ST*, *TT*의 내부 혹은 외부 노드일 수 있다. '*books*'를 원시 문서, '*bib*'를 목표 문서의 DTD 이름이라 하자. 예를 들어, (그림 4)에서, $M_{11}(bib, books)$는 *((first, last), (first_name, last_name))*, $M_{N1}(bib, books)$는 *((last, first), name)*, 그리고 $M_{1N}(bib, books)$는 *(addr, (city, country))*가 된다.

[정의 3] $m \in M(TD, SD)$라 하자. $m = (t, s)$에 대한 변환 T_m은 T_m:$s \rightarrow t$로 정의되고 $T_m(s) = t$ 이다. 여기서 T_m은 벡터이고, T_m의 길이 $|T_m|$은 $|g|$이며, 객체로부터 적절한 값을 얻기 위해 l 객체의 값에 적용되는 함수 벡터를 의미한다.

예를 들어, $m_{N1} = ((last, first), name)$ 일 때 $(str1, str2)m_{N1}(name, 'Tamura\ Kent') = ((last, 'Tamura'), (first, 'Kent'))$ 이고 $m_{1N} = (addr, (city, country))$ 일 때 $mergepath_{m1N}((city, 'NY'), (country, 'USA')) = (addr, 'NY\ USA')$이다. 여기서 '*mergepath*'는 문자열을 합병하는 함수이고, 'str_i'는 노드의 값에서 공백으로 구분되는 i 번째 단어를 추출하기 위한 함수이다. 때로 $(div(100))\ ^{o}(str_i)$와 같이 문자 함수에 나누기 연산자를 추가하는 복합 함수가 사용되기도 한다. 이 복합 함수는 화폐 단위를 바꾸기 위해 i 번째 문자열에 나누기 연산자를 재적용함을 의미한다. (그림 5)는 (그림 4)에서의 각 노드 간의 매핑을 유형별로 집합의 형태로 나타낸 것이다.

4.2 매핑에 대한 구현

XDMI 문서를 위한 DTD는 (그림 6)과 같다. 원시 문서 DTD에 존재하는 요소를 <source> 요소라 하고, 목표 문서의 요소는 <target> 요소라 한다. <target>에 기술된 <type> 속성은 매핑 종류를 표현하는 것으로, 0, 1, 2의 값을 가지며 각각 1:1, N:1 그리고 1:N 매핑을 의미하고, <operation> 속성의 값은 변환된 값을 얻기 위해 목표 값에 적용되어 <target> 노드로 전달되는 함수를 의미한다.

```
<!ELEMENT XDMI (XDMI.header, XDMI.isequivalent)>
<!ELEMENT XDMI.header (documentation,version,date,authorization)>
<!ELEMENT documentation (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT authorization (#PCDATA)>
<!ELEMENT XDMI.isequivalent (source, target*)*>
<!ELEMENT source (#PCDATA)>
<!ATTLIST source operation CDATA #IMPLIED>
<!ELEMENT target (#PCDATA)>
<!ATTLIST target
    type CDATA #REQUIRED
    operation CDATA #IMPLIED
>
```

(그림 6) XDMI의 DTD

목표 DTD를 위한 원시 경로 트리와 목표 경로 트리 사이의 매핑 정보는 다음과 같이 생성된다. 색인된 목표 경로 트리의 노드가 생성되면, 사용자는 각 목표 경로 트리의 색인 번호와 동일한 의미를 갖는 원시 경로 트리의 노드와 매핑시킨다. 동일한 번호를 갖는 모든 원시 DTD와 목표 DTD의 경로들을 같은 경로 집합으로 모아 각각 <source>와 <target> 태그의 세트로 구성하는 XDMI 파일을 자동으로 생성한다. (그림 4)의 'XDMI' 부분에 이러한 개념을 적용해서 생성된 원시 문서인 *'books'*와 목표 문서인 *'bib'*를 위한 매핑 정보의 일부가 XML 형식으로 표현되어 있다.

4.3 문서변환 알고리즘

앞 절에서 정의된 매핑과 변환함수를 바탕으로 원시 문서를 목표 문서로 변환한다. 문서를 변환하기 위해서 먼저 목표 DTD를 파싱하면 (그림 2) 하단에 있는 *'bib'*라는 문서에 대한 경로 트리가 생성된다. *SD*로부터 *TD*를 생성하기 위해서는 *TD*에 의해 생성된 *TT*의 루트 노드부터 시작하여 루트 노드에 해당하는 *SDE*의 이름을 노드 매핑에 의해 XDMI로부터 찾아낸다. 만약 해당되는 이름이 *SD*에 있으면 그 이름을 *TXD*에 생성함과 동시에 *SDE*의 값이 존재하면 그 값도 함께 *TXD*에 넣는다. 이와 같은 방법으로 *TT*의 모든 노드를 방문하여 해당되는 *SD*의 값을 *TXD*에 넣는다. 이때 만약 매핑의 종류가 1:N이면 XDMI에 저장된 함수를 적용하여 *SDE*의 값을 N 개의 *TDE*로 분해하는 함수를 적용하여 그 값을 얻고, 매핑의 종류가 N:1인 경우에도 마찬가지로 *TDE*에 해당하는 요소의 값을 얻는다.

위와 같은 방법으로 *TT*를 깊이우선탐색 (depth-first search) 방법에 의해 모든 노드를 방문하고 해당 노드에 대한 값을 *SD*로부터 가져와 *TDE*와 함께 문서를 생성한다. 이와 같은 절차를 알고리즘으로 작성해보면 다음과 같다.

Document Translation Algorithm

Input : *SD*, *XDMI*, *TT*(Target path Tree)
Output : Target XML Document
While (traversing *TT* by depth-first traversal) do
  let p be a node name of *TT*
    Case i where mapping $m_i(p) = q$
(1) 0 : do nothing;
(2) 1:1 : generate q tag and get the value of p if p's value exists;
(3) N:1 : apply T_m over p and generate q tag and get the value of $T_m(p)$ if p's value exists;
(4) 1:N :
    For i = 1 to N
      generate q_i tag;
      get the value of $T_m(p) = q_i$ value if p's value exists
    Endfor
EndCase
End

4.4 골격 원시 코드 구조

골격 원시 코드의 구조는 4.3절의 알고리즘을 기반으로 생성한다. 각 매핑 형태에 대한 문장은 <표 1>과 같다. <표 1>의 *while* 문의 모든 노드는 노드 p_i의 모든 값 $v(p_i)$를 나타내기 위해 골격 원시 코드 내에서 *if* 문장의 형태로 명시적으로 표현된다. 목표 DTD의 절대 경로에 대한 원시 문서의 요소 값을 가시적으로 다룰 수 있도록 목표 DTD 요소와 관련된 요소 값을 포함하는 변수를 원시 코드에 나타낸다. 원시 코드는 매핑 형태에 따라 다양하게 생성된다. *TT*의 루트 노드부터 p_i 노드까지의 절대 경로를 tp_i라 하고, *ST*의 sp_i를 *TT*의 tp_i 경로와 매핑된 경로라 하자. $v_i(sp_i)$와 $v_i(tp_i)$는 각각 원시 요소 sp_i 경로와 목표 요소 tp_i 경로의 요소값을 의미한다. 1:1 매핑의 경우는 $v(sp_i)$를 $v(tpi)$에 적용하기 위한 문장이 필요하며, 1:N 매핑 시에는, $v_i(sp_i)$ 값이 사용자 함수에 의해 토큰화 되거나 분리되고, 각 tp_{ij} ($1 \le j \le N$)는 현재의 노드 sp_i의 값을 N개의 값 $v_{ij}(tp_{ij})$로 분리하기 위한 코드가 필요하다. N:1 매핑의 경우는, 원시 요소 N개의 값 $v_{ij}(sp_{ij})$를 병합하여 목표 요소 $v_i(tp_i)$의 값으로 생성하기 위한 코드가 필요하다.

〈표 1〉 매핑 유형별 골격 코드 형식

Mapping Type	Source Code Format
1:1	If (sp_i -- tp_i) then { while(get_next_element(p_i) in SD is not null) { $v(tp_i) = v(sp_i)$; print(tp_i, $v(tp_i)$ to TD) } }
1:N	If (sp_i -- tp_i) then { while(get_next_element (p_i) in SD is not null) { for (each v_{ij} (sp_{ij}), $1 \le j \le N$) { //N is # of tokens in $v(sp_i)$ $v_{ij}(tp_i) = token_i(v(sp_i))$; print($tp_i$, $v_{ij}(tp_i)$ to TD) } } }
N:1	If (sp_i == tp_i) then { while(get_next_element (p_i) in SD is not null) { for (each v_{ij} (sp_{ij}), $1 \le j \le N$) { $v_i(tp_i) = v_i(tp_i) + v_{ij}(sp_{ij})$ print(tp_i, $v_i(tp_i)$ to TD) } } }

5. 문서변환 및 코드 생성 예

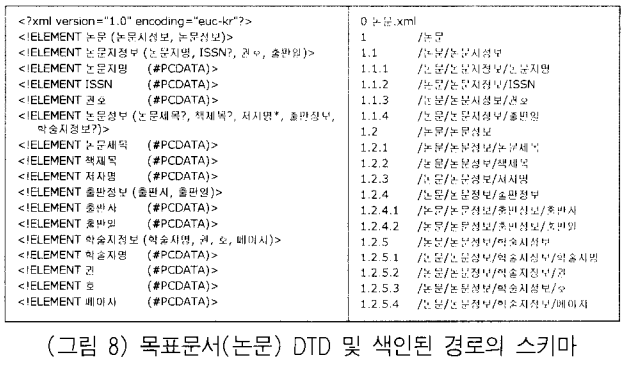
5.1 XDMI 생성 예

문서 변환에 대한 실행 결과를 얻기 위해 실제 사용되고 있는 DTD로 정의된 원시 문서인 '석박사논문.xml' 문서로부터 목표 문서인 '논문.xml'을 생성한다. 원시 DTD와 그에 대한 색인은 (그림 7)에 나타내었으며 이 색인은 XDMI 생성시 원시 경로와 목표 경로 사이의 매핑 정보로 이용된다.

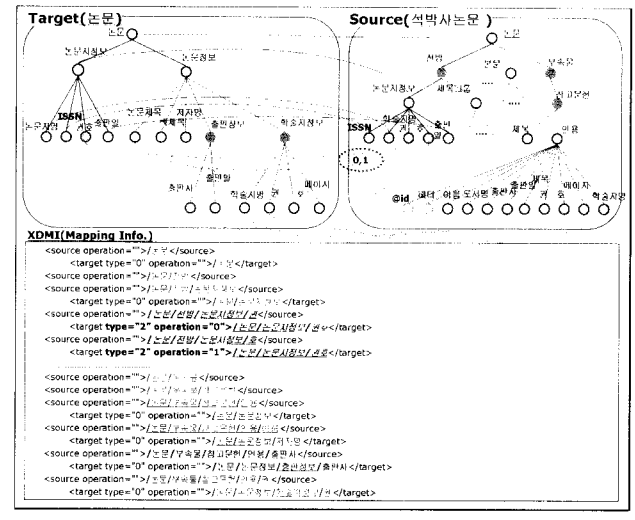
(그림 8)은 목표 DTD인 '논문.DTD'와 이에 대한 색인 및 경로 트리를 나타낸 것으로 여기서의 색인은 문서 변환시 목표 문서의 태그를 생성하는 순서를 의미한다.

원시 DTD와 목표 DTD 간의 매핑이 (그림 9)의 상단에, 매핑에 대한 XDMI의 일부가 하단에 나타나 있다. 원시 DTD와 동일한 색인 번호가 목표 DTD의 동일한 의미를 갖는 요소에 주어지고, 각 색인 순서는 서로 다르다. 원시 DTD의 색인이 목표 DTD의 색인과 매핑되지 않는 경우도 있다. 이것은 원시 문서의 요소를 목표 문서에서 사용하지 않는다는 의미이다. (그림 7)과 같은 원시 문서의 색인 할당을 기반으로 동일한 번호를 갖는 목표 문서 경로들을 같은 경로 집합으로 모아 XDMI 파일을 자동으로 생성한다. 이

(그림 7) 원시문서(석박사논문) DTD 및 색인된 경로의 스키마



(그림 8) 목표문서(논문) DTD 및 색인된 경로의 스키마



(그림 9) 원시문서(석박사논문)와 목표문서(논문) 사이의 매핑 및 XDMI 생성

XDMI 파일은 사용자가 쉽게 변경이나 수정할 수 있으며, 색인 번호를 재할당시 쉽게 재생성 할 수 있다. [15]에서의 XDMI와는 달리, 목표 문서의 색인 경로는 2장에서 언급한 매핑 형태를 지칭하는 0, 1, 2의 값을 갖는 또 다른 필드가 필요 하다.

5.2 문서변환 및 코드생성 실행 예

(그림 10)은 원시 DTD인 '석박사논문.DTD'와 목표 DTD인 '논문.DTD' 간의 색인 경로의 매핑을 표현한 것이다. 원시 문서의 색인 *'1.1.1.3'* 과 *'1.1.1.4'*는 목표 문서 *'1.1.3'*으로 매핑되는데, 이는 원시 문서의 두 경로가 목표 문서의 하나의 경로로 합성되는 1:N 매핑을 의미한다. 원시 경로의 *'1.3 /논문/부속물'*과 같이 매핑이 되지 않은 경로는 목표 문서로 변환되지 않음을 의미한다. 반면, 목표 경로의 *'1.2.4 /논문/논문정보/출판정보'*는 어떠한 원시 경로와도 매핑되지 않지만 목표 문서 변환시 목표 문서의 색인 순서에 따라 <논문정보>의 자식노드인 <출판정보> 태그로 생성된다.

(그림 10)의 매핑에 의해 생성된 골격 원시 코드 편집기는 (그림 11)과 같으며, 매핑시 정의한 사용자 함수 이외에

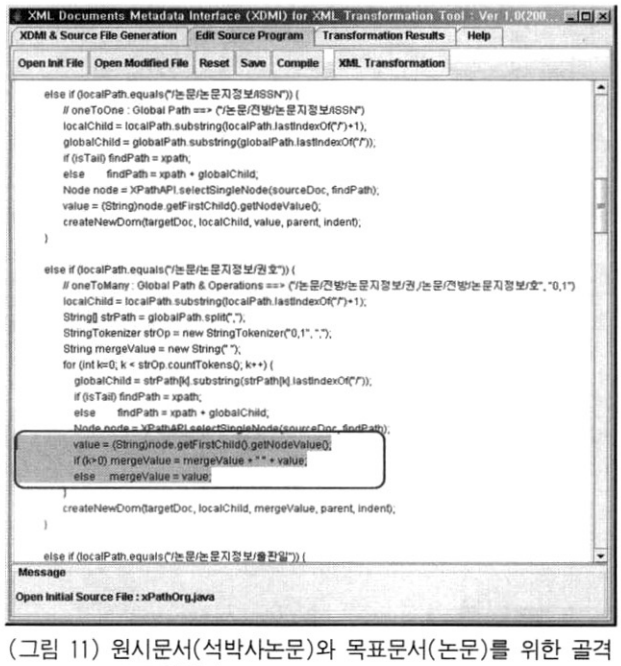
Seq	Target (논문.org)	
1	1	/논문
2	1.1	/논문/논문지정보
3	1.1.1	/논문/논문지정보/논문지원
4	1.1.2	/논문/논문지정보/ISSN
5	1.1.3	/논문/논문서명보/권호
6	1.1.4	/논문/논문지정보/출판일
7	1.2	/논문/논문정보
8	1.2.1	/논문/논문정보/논문제목
9	1.2.2	/논문/논문정보/책제목
10	1.2.3	/논문/논문정보/저자명
11	1.2.4	/논문/논문정보/출판정보
12	1.2.4.1	/논문/논문정보/출판정보/출판사
13	1.2.4.2	/논문/논문정보/출판정보/출판일
14	1.2.5	/논문/논문초록/참조문헌
15	1.2.5.1	/논문/논문초록/학술지정보/학술지명
16	1.2.5.2	/논문/논문초록/학술지정보/권
17	1.2.5.3	/논문/논문초록/학술지정보/호
18	1.2.5.4	/논문/논문초록/학술지정보/페이지

Source (석박사논문.idx)	
1	/논문
1.1	/논문/논문지정보
1.1.1	/논문/논문지정보/논문지원
1.1.1.1	/논문/논문지정보/논문지원/학술지명
1.1.1.2	/논문/논문지정보/논문지원/ISSN
1.1.1.3	/논문/논문지정보/논문지원/권
1.1.1.4	/논문/논문지정보/논문지원/호
1.1.1.5	/논문/논문지정보/논문지원/출판일
.....	.....
1.3	/논문/부속물
1.3.1	/논문/부속물/참고문헌
1.3.1.1	/논문/부속물/참고문헌/인용/저자
1.3.1.2	/논문/부속물/참고문헌/인용
1.3.1.2.1	/논문/부속물/참고문헌/인용/@id
1.3.1.2.2	/논문/부속물/참고문헌/인용/제목
1.3.1.2.3	/논문/부속물/참고문헌/인용/매체
1.3.1.2.4	/논문/부속물/참고문헌/인용/도서명
1.3.1.2.5	/논문/부속물/참고문헌/인용/출판사
1.3.1.2.6	/논문/부속물/참고문헌/인용/출판일
1.3.1.2.7	/논문/부속물/참고문헌/인용/제목
1.3.1.2.8	/논문/부속물/참고문헌/인용/권
1.3.1.2.9	/논문/부속물/참고문헌/인용/호
1.3.1.2.10	/논문/부속물/참고문헌/인용/페이지
1.3.1.2.11	/논문/부속물/참고문헌/인용/학술지명

(그림 10) 원시 DTD와 목표 DTD 간의 경로 매핑

XPath의 경로로 표현된 목표문서의 각 요소별로 어떠한 문장도 추가하여 문서를 변환할 수 있다. (그림 11)의 박스 부분은 매핑 함수에 의해 적용될 요소값을 사용자가 새로운 값으로 변경할 수 있도록 코드를 추가할 수 있는 부분을 나타낸다.

(그림 12)는 원시 문서 '석박사논문.xml'을, (그림 13)은 변환된 목표 문서 '논문.xml'의 결과를 보여 준다.



(그림 11) 원시문서(석박사논문)와 목표문서(논문)를 위한 골격 코드 편집기

(그림 12) 원시문서(석박사논문) XML

(그림 13) 변환된 목표문서(논문) XML

6. 결론 및 향후 연구

본 논문에서는 메타데이터 인터페이스를 이용하여 원시 XML 문서를 목표 XML 문서로 변환하는 문서변환기 및 코드 생성기에 대하여 기술하였다. 원시 XML 문서와 변환하기 위한 문서에 대한 DTD만 있으면 원시 문서 DTD의 존재 유무에 관계없이 구조가 다르더라도 원시 DTD와 목표 DTD를 매핑시키고 요소 변환 정보를 입력하여 변환시킬 수 있다.

문서변환을 자동화하기 위해 제안되고 있는 방법들은, 사용자가 XSLT를 배우고 프로그래밍 수준이 필요한 XSLT 스크립트 혹은 태그를 이용하여 XML 문서를 작성하는 방법, 문서변환 규칙을 정의하고 처리기를 필요로 하는 중간 언어나 스크립트를 사용하는 방법들이 있다. 이러한 방법들은 GUI 환경에서 자동 변환을 지원하지만, 스크립트나 프로그래밍 작업 없이는 완전한 자동화나 사용자 요구에 맞는 변환이 불가능하므로 더 많은 수작업이 필요하다. 또한 상용화되고 있는 여러 제품들을 자동 변환 방법 이외에 다양한 코드 생성 기능을 제공한다. [16]에서는 XML 스키마에서 정의된 데이터 요소에 기반하여 Java, C++, Microsoft C 클래스 파일 등을 자동으로 생성하기 위한 코드 생성기를 제공한다. [17]에서는 문서 변환을 위해 XSLT나 XQuery를 사용하며, 사용자 응용에 대한 Java 원시코드 생성기가 XSLT나 XQuery를 사용하기 위해 필요한 Java 코드를 생성한다. 그러나 이러한 코드들은 목표 문서의 DTD나 스키마 요소의 속성과 같은 정적인 부분을 생성하거나, DTD나 스키마 자체의 요소를 수행하기 위한 코드이므로 문서의 개별 요소를 사용자가 변경할 수 있는 다음 단계의 변환 작업에 직접적으로 응용할 수 없다.

반면, 본 논문에서는 사용자가 원시 문서와 목표 문서에

서 사용된 DTD의 의미만 알면 이를 매퍼를 이용하여 매핑하고 데이터 변환을 위한 함수를 동시에 추가하여 쉽게 문서를 변환한다. 또한 골격 원시 코드를 동시에 생성하여 사용자에게 목표 문서의 각 요소별로 문장을 추가하거나 변경할 수 있는 유연성을 제공한다.

매핑은 요소의 의미 혹은 구조에 따라 1:1, 1:N, N:1로 나누어지며 GUI 인터페이스에서 마우스를 클릭하여 쉽게 지정할 수 있다. 요소는 경로 트리로 나타내지며, 요소를 변환하기 위한 함수도 함께 지정된다. 매핑된 정보를 이용하여 XDMI 메타데이터 인터페이스가 생성되고 이를 바탕으로 자동적으로 원시 XML 문서가 목표 XML 문서로 변환되고 동시에 골격 원시 코드가 생성된다. 본 코드생성기는 Windows 환경에서 Java와 DOM을 이용하여 구현되었으며, 구현 결과를 보이기 위해서 실제 사용되고 있는 '석박사논문.xml' 문서를 '논문.xml' 문서로 변환하여 실행 결과를 나타내었다.

현재의 문서 변환 개념은 1:1, 1:N 혹은 N:1 매핑에 대한 변환을 지원하고 있으며, 향후 연구과제로는 경로 상에서 순환 경로가 발생하는 경우와 변환 조건 내에서 부가적인 조건을 포함하는 문서의 변환을 위한 연구가 필요하다.

참 고 문 헌

[1] EIA/IS-106 "CDIF - CASE Data Interchange Format - Overview", January 1994.

[2] XML Metadata Interchange (XMI), http://www-4.ibm.com/software/ad/library/standards/xmi.html.

[3] XSL Transformations (XSLT) Version 1.0, http://www.w3.org/TR/xslt/

[4] J. Eder and W. Strametz. "Composition of XML-Transformations", EC-Web 2001, LNCS 2115, pp.71-80, 2001.

[5] M. Erwig. "Toward the Automatic Derivation of XML Transformations", ER 2003 Workshops, LNCS 2814, pp.342-354, 2003.

[6] XML Path Language (XPath) 1.0, http://www.w3.org/TR/xpath/

[7] 홍종하, 양유승, 나홍석, 백두권. "메타데이터 레지스트리를 이용한 XML-문서 교환 방법", 정보과학회 봄 학술발표 논문집 Vol.28, No.1, pp.94-96, 2001.

[8] 조정길, 조윤기, 구연설. "구조적 상이성 분석에 기반한 XML 문서 변환 시스템의 설계 및 구현", 정보처리학회 논문지D, 제 9-D권 제 2호, pp.297-306, 2002.

[9] 성길용, 강치원, 정회경. "구조적 문서 변환을 위한 XML Mapper 시스템 설계 및 구현", 정보과학회 봄 학술발표 논문집 Vol.28, No.1, pp.382-384, 2001.

[10] 심민석, 유대승, 엄정섭, 강만모, 이명재. "XTGen: XML 변환기 생성을 위한 컴포넌트 기반 시스템", 정보과학회 봄 학술발표 논문집 Vol.28, No.1, pp.310-312, 2001.

[11] F. Bendeck. "Automation of XML Documents Translator Generation", WETICE 2001, pp.37-38, 2001.

[12] S. Krishnamurthi, K. E. Gray and P. T. Graunke. "Transformation-by-Example for XML", PADL 2000, LNCS 1753, pp.249-262, 2000.

[13] T. Pankowski. "Transformation of XML Data Using an Unranked Tree Transducer", EC-Web 2003, LNCS 2738, pp.259-269, 2003.

[14] H. Su, H. Kuno, E. "Rundensteiner. Automating the Transformation of XML Documents", WIDM 2001, Atlanta.

[15] Y. K. Nam, J. Goguen, G. Wang. "A Metadata Integration Assistant Generator for Heterogeneous Distributed Databases", in Proceedings, International Conference on Ontologies, DataBases, and Applications of Semantics for Large Scale Information Systems, Springer, LNCS, Vol.2519, pp.1332-1344, 2002.

[16] ALTOVA XMLSpy, http://www.altova.com

[17] Stylus Studio XML Enterprise Edition, http://www.stylusstudio.com

최 귀 자

e-mail : gjchoe@hosu.yonsei.ac.kr
1991년 서울산업대학교(학사)
2005년 연세대학교 전산학과(석사)
2005년~현재 연세대학교 전산학과 박사과정
관심분야 : 프로그래밍언어, 소프트웨어공학, XML, 메타데이터, 정보검색

남 영 광

e-mail : yknam@yonsei.ac.kr
1978년 연세대학교 수학과(학사)
1985년 한국과학기술원 전산학과(석사)
1992년 Northwestern University 전산학과(박사)
1993년~1994년 시스템공학연구소 선임연구원
1995년~현재 연세대학교 전산학과 교수
관심분야 : 프로그래밍언어, 소프트웨어공학, 정보검색, XML