DTD 역 구성을 통한 XML 문서에서의 정보추출

정종석[†] · 오동익[††]

요 약

분산된 환경에서 정보를 교환하기 위한 수단으로의 XML문서는, 그 자료의 구성을 정의하는 DTD를 통해서만 정확한 의미가 파악될 수 있다. 하지만 인터넷에서 수집된 XML 문서에 항상 DTD가 제공되리라는 보장은 없으며, 이러한 경우에는 수집된 XML 문서의 구조를 파악한 후 정보를 추출해야 한다. 본 연구에서는 DTD가 알려지지 않은 XML 문서를 바탕으로 적합한 DTD를 구성하고, 이를 이용해 XML 정보를 구조적인 형태로 하부 DB에 저장할 수 있는 방법에 대해 설명하고자 한다. 특히, 본 연구를 통해 개발된 DTD 추출기는 XML 파일을 1-Path로 스캔하기에 기존에 나와있는 다른 방식보다 더 효율적으로 DTD를 구축할 수 있다.

Extracting Information from XML Documents by Reverse Generating DTDs

Jeong-suk Jung[†] and Dong-ik Oh[††]

ABSTRACT

XML documents are widely used for exchanging information in the distributed computing environment. Here, actual information contained in the document can only be interpreted with the provision of a proper DTD. However, XML documents collected from the web may not always be accompanied by corresponding DTD, so that it may not be easy to extract information from such sources. In this study, we reverse construct a DTD from DTD-unknown XML sources. We then use the DTD to extract information from XML inputs and to store it into the underlying DB. The DTD construction module developed is designed in such a way that, it scans input XML files in 1-path, where most other implementations use 2-path approach. The information extraction module provides clean Java programming interfaces as well, so that it can be easily integrated with other web applications.

Key words: XML, DTD, E-Commerce, Information Extraction, DTD Construction

1. 서 론

본 연구에서 개발한 DTD 구축기는 SECOS (Soonchunhyang E-COmmerce System)[1] 개발 사업의 일부로 진행되었다. 이 시스템은 다양한 웹 지향 정보검색 시스템 구축을 위한 정형화된 모델을 제공하고자 개발되어지고 있는 통합형 웹기반 정보 저장/검색 시스템으로서, 기존의 다양한 소프트웨어 컴포넌트들을 통합하고 새로운 컴포넌트를 개발함으로써 검색사이트, 쇼핑몰과 같은 인터넷 정보활용 사이트를 구축하는데 활용될 수 있다. 그림 1은 이러한 SECOS의 전체적인 구조를 보여주고 있다.

SECOS에서는 6개의 서버측 컴포넌트들이 3가지 Division으로 분리되어 제공된다. 이들 중 Gathering Division은 분산된 환경에서의 정보를 수집하는 역할을 담당한다. Gathering Division에서 제공하는 정보수집기는 Affiliated Gatherer와 Regular Gatherer 및 Meta Gatherer의 세 가지이다. 이들을 통해 SECOS 시스템은 분산된 환경에서 정보 수집 활동을 할 수 있

본 연구과제는 2002년 순천향대학교 학술연구 조성비 일반연구 과제로 지원을 받아 수행하였음.
접수일 : 2002년 5월 31일, 완료일 : 2002년 12월 4일
[†] 준회원, 주)ECO 개발부 연구원
[††] 정회원, 순천향대학교 정보기술공학부 조교수

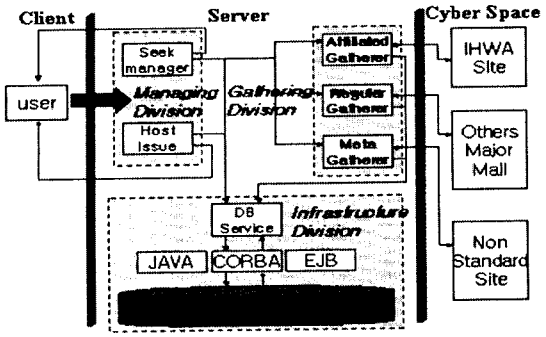


그림 1. SECOS 시스템 구성도

게 된다. Affiliated Gatherer와 Regular Gatherer는 각각 동일한 SECOS시스템과 DTD가 공개된 XML 문서에 대한 정보교환 및 수집을 담당한다[1]. Meta Gatherer는 웹 상에서 표준화 되어있지 않은 정보, 즉, 일반적인 웹 문서(HTML)나 DTD가 정의되어 있지 않은 XML 문서에서 제공되는 정보를 추출하기 위하여 필요한 Gatherer로서, 아직까지 웹에서 제공되는 자료의 대부분이 이러한 범주의 표현방식으로 제공되고 있는 것을 감안할 때[3], 정확하고 포괄적인 정보를 제공해야 하는 검색 서비스에 반드시 필요한 모듈이라 할 수 있다. 본 논문에서는 이러한 Meta Gatherer의 기능 중, DTD가 정의되어 있지 않은 XML 문서에서의 정보추출 기능의 구현 및 추출된 정보의 활용방법에 대해 설명하고자 한다.

비 정형화 된 문서에서 상품정보를 추출하기 위해 필요한 가장 핵심적인 기술은 문서내의 정보를 계층구조로 표현하고 이를 통해 정보를 추출하는 기술이다. 비 정형화된 데이터 소스에서의 정보 추출을 위해서는 Data Mining분야에서 활발한 연구가 진행되고 있고, 특히 우리의 연구와 직접적인 관련성을 가진 HTML 문서에서의 정보추출에 관한 작업도 Wrapper[4]를 활용한 방식 등을 통하여 활발히 진행되고 있으나 아직까지 만족 할 만한 정보 추출이 이루어지고 있지는 못하다. 특히 본 연구에서 진행한 DTD가 제공되지 않는 XML 문서에서 DTD를 추출하는 연구는 Bell 연구소의 XTRACT[2]와 Singapore의 Nanyang Technological University[3], 그리고 IBM Alphaworks의 DDbE[9]등에서도 이루어졌으나, 3가지 연구 방법은 모두 반 정형화된 데이터 소스에서 DTD를 추출하기 위하여 트리나 그래프를 이용하여 계층적인 구조를 생성하고 이 구조에 보다 효율적이고 함축적인 DTD를 생성하기 위한 알고리즘을 적용하는 방법을 채택하고 있다. 이들은 모두 반 정형화된 데이터 소스들에서 유효한 정보를 추출할 수 있는 계층적인 데이터 구조를 생성하는 연구로서 우리가 필요로 하는 정보추출기와 직접적인 연관성을 가지고 있으나 전체적인 DTD 문법을 활용할 수 없다거나, 다른 프로그램 모듈에서 인터페이스가 될 수 없다는 제약점을 가지고 있다. 또한 이들은 XML문서들의 공통된 구조를 추출하기 위하여 먼저 XML 문서 각각의 구조를 트리나 그래프를 이용하여 표현하고 이들을 통합하는 2단계 알고리즘을 채택하고 있다. 따라서 본 연구에서는 독립된 모듈로 구현되지만 인터페이스를 통해 다른 모듈에서 활용이 가능할뿐더러 기존의 2단계 DTD추출 알고리즘에서 각 XML 문서의 구조를 트리나 그래프로 변환하는 단계를 삭제한 향상된 1-path 알고리즘을 사용한 DTD 추출기를 구현 할 필요성을 느끼게 되었다.

본 논문의 구성은 다음과 같다. 2장에서는 수집된 XML 문서의 구조를 트리로서 표현하는 방법에 대해 설명한다. 3장에서는 2장에서 제시된 알고리즘을 통해 실제적으로 트리를 구성하는 방법에 대해 설명하고, 이 구조에서 DTD를 추출하는 방법을 예제를 통해 설명한다. 4장에서는 개발된 DTD 추출기가 어떻게 인터넷 정보수집을 위한 전체적 시스템과 연동되어 질 수 있는지에 대해 설명한 후, 전체적인 Meta Gatherer의 동작을 Pseudo Code로 나타낸다. 5장에서는 본 연구의 결과 및 향후 연구방향에 대해 살펴본다.

2. 정보추출을 위한 XML 정보의 구조화

여러 XML 데이터를 위한 단일 DTD를 생성하기 위해서는 입력되는 XML 정보를 계층적 자료구조인 트리로 표현하는 것이 필요하다. XML 데이터는 중첩된 구조로 구성되고 (well-formed XML데이터), 이는 n-ary 트리로서 잘 표현 될 수 있으며, 이러한 트리에서 문서의 구조를 파악하기가 용이하기 때문이다. 본 장에서는 DTD 추출기가 XML 파일에 기술된 내용을 통해 DTD를 생성해 낼 때에 필요한 알고리즘 및 사용되는 자료구조에 대해 설명하고자 한다.

2.1 XML 정보표현을 위한 자료구조

트리는 XML 문서의 각 element를 표현하는 노드 및 중첩된 구조를 나타내는 링크로 구성된다. 트리노드는 XML 문서에서 각 element가 가질 수 있는 관계

들, 즉 Mandatory, Optional, OR, Repetition을 가리키는 속성들을 가지게 된다. 또한 element의 처리여부를 판별할 수 있는 *isChecked* 속성, 그리고 해당 트리노드가 현재 처리중인 XML 문서의 부모 element로 돌러싸인 단락을 처리하는 도중에 새로이 생성되었는지 아니면 이미 존재하는 트리노드인지를 나타내기 위한 *isNewCreated* 속성 등을 가진다. *isTemp* 속성은 임시노드를 나타내기 위한 속성인데, 임시노드는 XML 문서 element간의 OR, Repetition, Optional 관계를 효과적으로 나타내기 위해 사용되는, XML element와는 직접적으로 연관되지 않는 노드이다. 표 1은 이러한 트리노드의 속성에 대한 요약이다.

element의 반복되어지는 패턴을 추출하기 위해서는 정규 표현식을 그래프 형태로 표현한 - 상태전이도와 같은 형태의 수평적 관계를 나타내는 - 속성이 동일 레벨상의 트리노드를 위해 필요하다. 이와 같은 관계를 나타내기 위해 트리노드는 몇 개의 포인터를 가지게 된다. 첫째로, *refNodeId*는 각 트리 노드가 포함하고 있는 자식노드의 구조를 중복해서 정의하는 것을 피하기 위해 사용된다. 둘째로 트리노드의 *chain* 포인터는 동일 레벨상의 트리노드들 간의 반복 관계 및 연속 관계를 나타낸다. 즉, 서로 반복되어지는 패턴을 *chain* 속성에 의해 연결하여 사이클을 추출함으로써 동일 패턴의 반복 여부를 판별하도록 하였다. 셋째로, 트리노드의 *link* 포인터는 *chain* 포인터와는 반대의 개념으로 사용되어진다. *chain*은 형성된 사이클의 추출을 위해 사용되지만, *link*는 사이클이 형성되어지지 않았을 경우 그 동안 처리되어진 element이름을 추출하는데 사용된다. 즉, *chain* 포인터는 연속되어지는 다음 sibling 노드를 가리키고, *link* 포인터는 연속되는 이전 sibling 노드를 가리킨다.

표 1. 트리노드의 속성과 역할

속 성 명	역 할
ele_name	해당 노드의 XML element 이름
isTemp	해당 노드가 임시노드임을 나타냄
isMandatory	해당 노드의 관계 중 Mandatory와 Optional을 나타냄
isRepeat	해당 노드의 관계 중 Repetition을 나타냄
isOR	해당 (임시)노드의 관계 중 OR 관계를 나타냄
isChecked	해당 노드의 Repetition관계 확인을 위해, 또한 트리의 구성중 해당 노드의 방문 여부 및 추후의 트리 수정 작업에서 사용
isNewCreated	해당 노드가 XML 문서를 처리하는 중에 새로이 생성되었는지 아니면 이미 생성되어진 트리노드인지를 나타냄

XML 문서에 포함되어진 각 XML element를 노드화하여 트리에 추가/갱신하기 위해서는 노드가 삽입되어질 위치를 가리키고 있는 전역 포인터가 필요하게 된다. *startPointer*는 *chain*을 통해 사이클을 추출할 때 사이클의 시작점을 가리키는 지시자의 역할을 하고, *currentPath*는 트리노드가 삽입되거나 검색되어야 할 부모노드를 가리킨다. *insertPointer*는 *currentPath*의 자식노드들 중 검색의 시작점을 지시하는 역할을 담당한다.

XML 문서를 트리로 구축하는 과정에서는 여러 개의 임시노드를 사용하여 각 XML 문서에 포함되어진 element간의 순서관계와 Repetition 및 Optional관계를 표현한다. 이러한 방법 때문에 트리상에서 하나의 element에 대한 복수개의 트리노드가 존재하고 또한 그러한 각각의 트리노드는 자식노드를 가질 수 있다. 이러한 중복정의 문제를 해결하기 위해서 본 연구에서는 트리상에서 실제 해당 트리노드의 구조를 표현하는 원본 트리노드에 대한 경로를 *refNodeID*라는 이름으로 각 트리노드가 필드로서 저장하도록 설계하였다. 그리고 원본 트리노드들의 리스트를 테이블로서 관리하여 새로운 노드가 트리에 추가되어질 때나 트리노드의 *isChecked* 속성을 갱신할 때 원본 트리노드 테이블에서 경로정보를 추출하여 *currentPath*와 *insertPointer*의 위치를 설정하는데 사용할 수 있도록 하였다. 앞서 설명한 트리노드와 트리노드의 속성을 이용하여 표현된 XML DTD의 각 연산자들은 표 2와 같다.

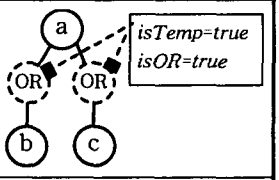
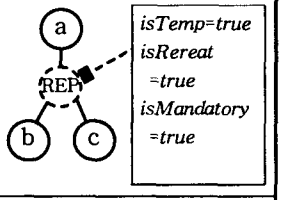
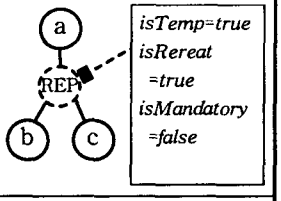
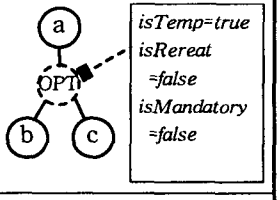
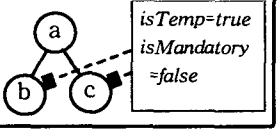
2.2 트리 구축 알고리즘

DTD 추출기는 앞서 설명한 트리 노드 및 데이터 구조를 사용하여 XML 문서내의 중첩된 구조를 트리로 표현하게 된다. 이러한 과정은 세 단계로 구분되어질 수 있으며 각 단계에 대한 설명은 다음과 같다.

Step 1. 초기화

우선 모든 XML element의 부모인 "root" 노드를 생성한다. 그리고 트리 내에서 검색 및 삽입될 위치를

표 2. DTD연산자의 트리표현

기호	용도	예제	트리표현
|	선택	<!ELEMENT a (b | c)>	
+	반복 (최소1)	<!ELEMENT a (b, c)+>	
*	반복	<!ELEMENT a (b, c)*>	
?	생략 가능	<!ELEMENT a (b, c)?>	
,	순서 지정	<!ELEMENT a (b, c) >	

가리키는 *currentPath*와 *insertPointer*가 "root" 노드를 가리키도록 초기화한다.

Step 2. 트리 구축

XML문서에서 순서대로 element를 추출하고 각각의 추출된 element에 대하여 같은 이름을 가지는 트리 노드가 현재까지 구축된 트리 내에 존재하는지를 검색한다. 이 검색의 결과와 삽입되는 노드의 트리에서의 위치에 따라 새로운 노드가 추가되거나 기존의 노드가 조정될 수 있다. 표 3은 이러한 검색 결과의 종류에 따라 트리에 어떠한 수정이 가해지는지를 정리하고 있다.

추가되어질 위치를 나타내는 *currentPath*와 *insertPointer*의 설정정보와 *startPointer*의 설정정보를 바탕으로 노드를 새로이 생성하여 트리에 추가하거나 기존에 존재하는 트리노드의 속성들을 갱신한다. 참고적으로 일곱 번 째 비교인 next sibling과의 비교에서는 next sibling이 임시노드인 경우에는 임시노드의 첫 번째 자식 노드와의 비교가 수행된다. 표 3의 비교 결과에 따라 11 종류의 작업이 이루어 질 수 있는데, 이러한 노드 삽입/수정의 알고리즘이 표 4에 기술되어 있다.

트리노드의 추가나 갱신 시에는 원본 트리노드 테이블에서의 노드가 사용된다. 이 테이블의 노드와 여러 전역 포인터들의 관계를 살펴보면 다음과 같다. 우선, 새로운 트리노드를 추가할 때 먼저 추출되어진 element이름으로 원본 트리노드 테이블에서의 검색이 실시된다. 검색결과 이미 해당 이름이 등록되어 있다면 이름과 관련된 원본 트리노드의 경로정보를 추출하여 생성되어진 트리노드의 *refNodeID*필드를 설정한다. 그리고 *currentPath*와 *insertPointer*의 위치는 추출되어진 링크정보로 설정한다. 그렇지 않고 원본 트리노드 테이블에 해당 이름으로 저장되어진 정보가 없다면, 생성된 트리노드의 이름과 트리상의 경로 정보를 원본 트리노드 테이블에 저장하고 *currentPath*와 *insertPointer*의 위치는 추가되어진 트리노드를 가리키도록 한다. 이미 존재하는 트리노드의 속성을 변경할 때는 해당 트리노드의 필요한 속성을 변경한다. 변경한 후에 *currentPath*와 *insertPointer*의 위치를 설정해야 하는데, 추출되어진 element이름을 원본 트리노드 테이블에서 검색, 추출되어진 경로정보로 이들의 위치를 조정한다.

Step 3. 트리 수정 및 보완

Step 2에 의해 구성된 트리는 상당한 복잡도를 갖는 형태가 된다. 이 단계에서는 추출되어질 DTD의 가독성을 높이기 위해 구축되어진 트리를 보다 간략화 하는 작업이 수행된다.

그림 2의 예제 XML문서는 Step 2까지를 거치면서 왼쪽의 트리를 구축하게 될 것이다. 여기서 OR관계로 설정되어진 두 subtree에는 공통되는 부분이 존재하지만 이것을 하나로 묶지 않은 형태로 트리가 구축된다. Step 3에서는 구축되어진 원본 트리노드 테이블을 입력으로 하여, 테이블의 각 원소를 순회하며 각 원소의 자식노드들 중에서 OR관계로 구축되어진 두 개의 subtree 중 공통된 노드들을 추출하여 더 간략화 된 트리를 구축한다. 이 결과 생성된 트리가 그림 2의 오른쪽 트리이다.

표 3. 트리 구축 알고리즘

*currentPath*의 자식유뮤	*currentPath*의 *isNewCreated* 필드	*currentPath*의 임시노드 여부	임시노드 종류	*insertPointer* 노드와 비교	*startPointer*의 설정 유무	*insertPointer*의 next sibling과의 비교	*insertPointer*의 이전 sibling들과의 비교	결과
단말노드	true							①
	false							②
비 단말노드	true	임시노드	OR					③
			Optional					③
			Repetition		설정되어짐	동일		④
						다름		⑤
					초기값 유지	없음	동일노드 존재	⑥
							존재하지 않음	⑦
		비 임시노드		동일	설정되어짐			⑤
					초기값 유지			⑧
				다름	설정되어짐	동일		④
						다름		⑤
					초기값 유지	없음	동일노드존재	⑥
							존재하지 않음	⑦
	false	임시노드	OR					⑨
			Optional					⑨
			Repetition		설정되어짐	동일		④
						다름		⑤
					초기값 유지	동일		⑩
						다름		⑪
						없음	동일노드 존재	⑥
							존재하지 않음	⑦
		비 임시노드		동일	설정되어짐			⑤
					초기값 유지			⑧
				다름	설정되어짐	동일		④
						다름		⑤
					초기값 유지	동일		⑩
						다름	동일노드존재	⑥
							존재하지 않음	⑪
						없음	동일노드존재	⑥
							존재하지 않음	⑦

표 4. 트리 구축 알고리즘 (표 3의 결과에 대한 설명)

결과번호	트리 구축관련 내용 및 설명
①	새로운 트리노드를 *currentPath* 자식노드로 추가.
②	이러한 경우는 PCDATA를 표현하는 트리노드에서만 나타날 수 있음
③	임시노드가 아닌 경우와 같음
④	*insertPointer*가 가리키는 노드의 chain 포인터와 next sibling노드의 *link* 포인터를 상호 연결시킴
⑤	*link* 포인터에 의해 연결되어진 트리노드를 추출하여 *currentPath*의 자식노드로 추가. *currentPath*가 임시노드이면 *currentPath*의 부모의 자식노드로 추가
⑥	*startPointer*에 현재 *insertPointer*의 값을 저장하고 현재 *insertPointer*가 가리키는 노드의 *chain* 포인터와 검색되어진 노드의 *link* 포인터를 연결시킴.
⑦	*isMandatory*속성이 false인 임시노드를 생성하여 추출되어진 element에 대한 트리노드를 추가
⑧	*insertPointer*가 가리키는 트리노드의 *isRepeat* 속성을 *true*로 설정
⑨	임시노드가 아닌 경우와 같음
⑩	다음 sibling노드의 *isChecked* 속성을 *true*로 설정
⑪	*isOR* 속성이 *true*인 두 개의 임시노드를 생성하여 *currentPath*의 자식노드들 중 이미 처리되어진 노드를 제외한 트리노드들과 현재 추출되어지는 element들을 분리

```
<?xml version="1.0" encoding="EUC-KR"?>
<test>
   <a>....</a>
   <b>....</b>
   <c>....</c>
</test>
<test>
   <b>....</b>
   <c>....</c>
</test>
```

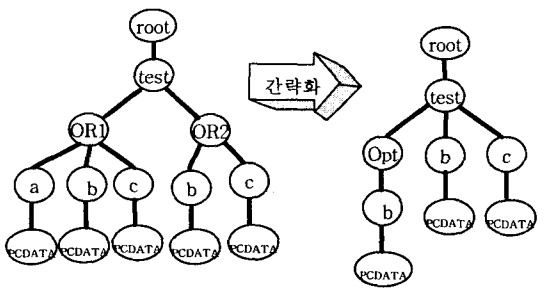


그림 2. 트리 간략화의 예

3. 트리의 구축 및 DTD 추출예제

본 절에서는 표 5의 XML 예제 파일을 트리로 구성하는 과정을 통해서 위에서 언급한 트리 구축 알고리즘이 어떻게 적용되어지는지 살펴보도록 한다.

3.1 트리구축의 예

표 5. 예제 XML 파일

```
<?xml version="1.0" encoding="EUC-KR"?>
<books>
 <book>
  <title> Harry Potter and the Sorcerer's Stone</title>
  <author> J. K. Rowling </author>
 </book>
 <book>
  <title> Hamlet </title>
  <author>
   <first-name> William </first-name>
   <last-name> Shakespeare </last-name>
  </author>
 </book>
</books>
```

Step 1의 과정으로서 "root" 노드와 *currentPath* 그리고 *insertPointer*가 생성되고 초기화된다. "books" element가 추출되면 이를 트리에 추가하기 위해 검색을 수행하게 되는데, 이 때 *currentPath*는 "root"를 가리키고 있고 "root" 노드는 단말노드이며 *isNewCreated* 속성이 *true*이기 때문에 표 3의 분류를 따라 표 4의 결과 ①이 적용되게 된다. 따라서 "books"라는 이름을 가지는 새로운 트리 노드를 생성하고 트리에 추가하게 된다. 다음으로는 "book" element가 트리 노드로 생성되어 추가되어지고 "title" element역시 트리에 추가된다. 다음으로 "title"의 PCDATA가 추출되어 트리에 삽입되고 "title" element의 EndTag가 추출되면 *currenPath*는 "book" 노드를 *insertPointer*는 "title" 노드를 가리키게 된다.

"author" element가 추출되어 검색을 수행하게 되면 표 3에서 비 단말노드, *currentPath*가 가리키는 트리노드의 *isNewCreated* 필드가 *true*, 이 노드가 임시노드가 아니며, 노드의 이름이 element이름과 다르고, 다음 sibling이 없는 경우로서, 표 4의 ⑤ 알고리즘이 적용되어 진다. 즉, 부모 노드인 "book"의 *isNewCreated* 필드가 *true*이고 *insertPointer*가 "title"을 가리키고 있고 *insertPointer*의 이전 sibling노드가 존재하지 않기 때문에 "title"과 "author"는 같은 부모 하에 존재하는 형제 관계임을 알 수 있다. 그러므로 "author"라는 이름을 가지는 새로운 트리 노드를 생성하고 이를 트리에 추가하게 된다. 새로운 노드를 생성하고 추가되어질 때마다 *currentPath*와 *insertPointer*는 새로이 추가되어지는 노드로 이동한다. 계속해서 "author"와 "book" element에 대한 EndTag가 추출되면 *currentPath*는 트리의 "books" 노드를 가리키게 되고 *insertPointer*는 "book"노드를 가리키고 있게 된다. 그림 3은 이와 같은 트리구성 과정을 그림으로 설명하고 있다.

이 후 두 번째 "book"의 정의가 시작된다. 또 다시 "book" element가 추출되어지면 표.4의 ⑧의 논리가 실행되어 "book" 노드의 *isRepeat* 속성이 *true*로 설정되어지게 된다. 즉 "book"이라는 element는 반복되어 나타남을 의미한다. "title" element가 추출되어지면 이 element는 *currentPath*가 가리키는 노드의 자식으로 존재함으로 단순히 *currentPath*와 *insertPointer*의 위치만이 "title"로 변경되게 된다. 그리고 "title"의 EndTag를 만나게 되면 *currentPath*와 *insertPointer*의 위치는 "book" 노드로 변경되게 된다. 다음으로는 "author" element가 추출되는데, 이 노드는 트리에 이미 구축되어 있으므로 *insertPointer*가 "author" 노드를 가리키게 된다. 이제는 XML 문서에서 "first-name" element가 추출되어지고 트리의 "author"노드

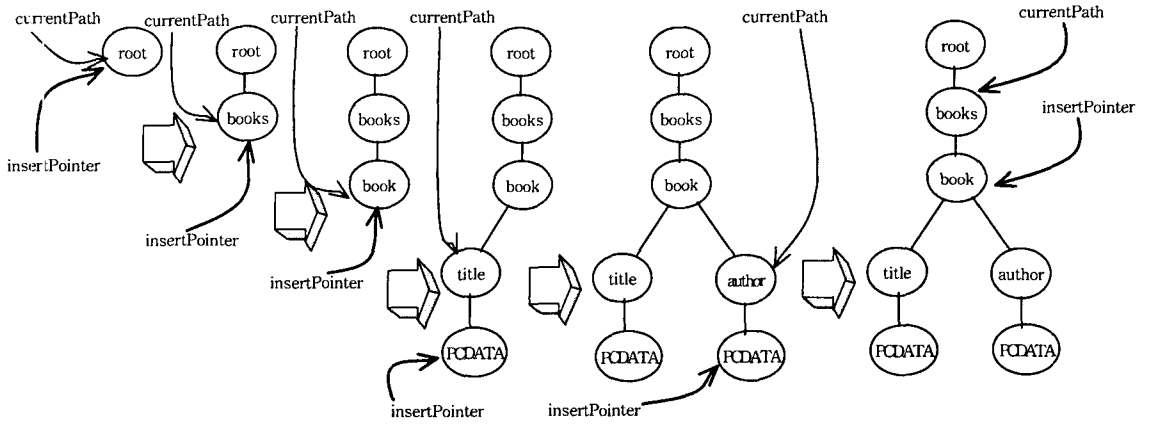


그림 3. 표 5의 첫 번째 XML 정의에 대한 트리구성 과정

의 자식으로 삽입되어야 한다. 이때 트리의 "author" 노드는 *isNewCreated* 속성이 *false*로 설정되어 있기 때문에 표 4의 ⑪ 결과가 적용되어지고 임시노드를 사용하여 트리 노드를 추가한다. "first-name"의 EndTag가 추출되면 *insertPointer*는 새로 생성된 임시노드를 가리키게 되고 *currentPath*는 그대로 "author" 노드를 가리키게 된다. 새로이 "last-name" element가 추출되면 이 임시노드의 자식이며 "first-name"의 sibling으로 노드가 추가 되게 된다. 그림 4는 현 단계까지의 트리 구축 내용을 보여준다.

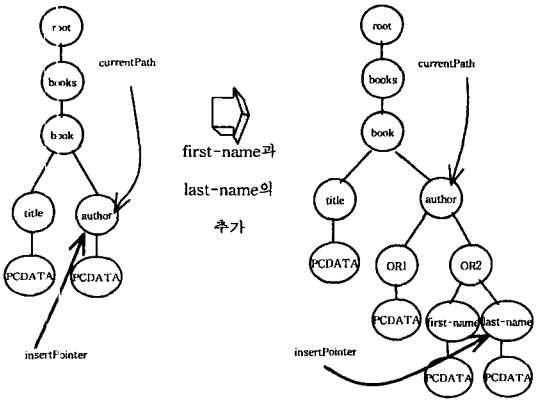


그림 4. 표 5의 두 번째 XML 정의에 대한 트리구성 과정 및 원본 트리노드 테이블의 변환과정

3.2 DTD 추출

XML 입력에 의해 트리가 생성된 후에는 입력 XML문서의 문법을 모두 만족시키는 DTD가 추출되어진다. 본 연구에서는 이러한 기능을 담당하는 모듈을 구현하였고 이는 DTD Extractor라고 불린다. 구축되어진 트리는 XML문서의 계층적 구조와 부모와 자식간의 관계를 구조적으로 잘 나타내고 있기 때문에 DTD Extractor는 트리를 레벨별로 순회하면서 노드들의 순서와 부모 element와의 관계, 즉 각 노드가 표현하는 Mandatory, Optional, OR, Repetition 같은 의미를 DTD 연산자로 변환하면서 DTD를 생성하게 된다. 단 트리 노드들 중 실제 XML 문서에는 존재하지 않지만 OR, Optional, Repetition관계를 표현하기 위해 생성된 임시 노드에 대해서는 그 임시노드 대신 노드의 서브 트리의 내용을 사용함으로써 DTD를 표현하게 된다.

그림 4의 예제 트리를 DTD Extractor를 통해 DTD로 변경하면 표 6과 같다.

표 6. 예제 XML문서의 DTD

```
<!ELEMENT root books>
<!ELEMENT books (book)+>
<!ELEMENT book (title, author)>
<!ELEMENT title #PCDATA>
<!ELEMENT author
 (#PCDATA | (first-name, last-name))>
<!ELEMENT first-name #PCDATA>
<!ELEMENT last-name #PCDATA>
```

4. DTD 추출기의 활용

SECOS시스템의 Gathering Division은 분산환경에서 다양한 정보를 수집하여 사용자에게 제공한다. 많

은 웹 상품정보들이 HTML 문서형식으로 혹은 DTD가 제공되지 않는 XML문서들과 같이 비 정형화 된 형태로 정보를 표현하는 것을 감안 할 때, 이러한 데이터 소스들로부터 상품정보를 수집, 추출하는 기능을 담당하는 Meta Gatherer의 구현은 완벽한 상품정보 수집기능을 제공하기 위하여 필수적이다. 우리가 현재 구현한 반 정형화 된 문서에서의 정보 추출기능은 이러한 Meta Gatherer의 기능 중의 하나이며, 이를 통해 Meta Gatherer는 DTD가 제공되지 않는 XML문서에서의 정보 추출을 수행한다.

Meta Gatherer는 "문서수집기", 2장에서 설명한 "XML 정보추출 및 그룹화 모듈"과 "DTD Extractor" 및 본 절에서 설명할 "XML Mapper and Data Storing Module"로 구성된다. 그림 5는 Meta Gatherer의 이와 같은 기능을 담당하는 모듈의 전체적인 구성도이고, 표 7은 전체적인 Meta Gatherer의 작동 과정에 대한 Pseudo Code이다.

즉, "문서 수집기"는 웹사이트들을 정기적 또는 비정기적으로 돌아다니며 XML문서를 수집하는 역할을

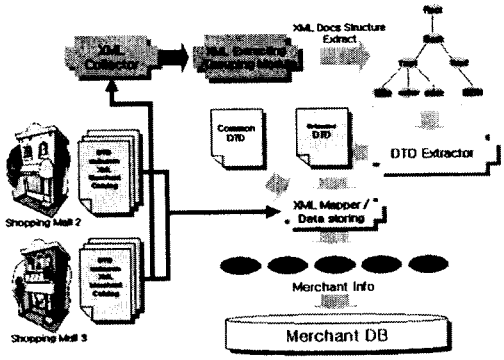


그림 5. Meta Gatherer의 시스템 구조

담당한다. 이렇게 수집된 XML 정보는 "XML 정보추출 및 그룹화 모듈"을 통해 트리로 표현되고, 이 트리는 계속적인 XML 입력에 의해 확장되고 그룹화 되어진다. 모든 XML 입력에 의해 트리가 완성되면 "DTD Extractor"가 이 트리를 읽어 입력된 XML 문서들에 적합한 DTD를 생성해 내고, 이 DTD에 기초하여 "XML 매퍼 및 데이터 저장 모듈"이 수집된 XML 문서

표 7. Meta Gatherer의 전체동작을 설명한 Pseudo Code

```
// XML_Directory_Path := 수집된 XML문서들의 저장소
// Tree_File_Path := 기 작업된 XML Tree파일의 저장소
// Current_XML_File := 현재 작업중인 XML 파일 레퍼런스
// Tree_File := 현재 작업중인 Tree_File 레퍼런스
// DTD_File_Path := 생성된 DTD파일 저장소
// Current_Generate_DTD := 생성된 DTD파일의 레퍼런스
// System_Common_DTD := 시스템 공통 DTD파일
// DTD_Mapping_Info := 추출 DTD와 공통 DTD간의 Mapping 정보 구조체

Current_XML_File = GetFirstXMLFile(XML_Directory_Path);  // XML 저장소에서 첫 번째 파일 읽음
while(Current_XML_File != NULL) {                        // 계속되는 XML 입력에 대해 반복
  if(Tree_File == NULL)
   Tree_File = MakeTreeFile(Current_XML_File);           // 첫 번째 임시 트리파일 생성
  else {
   LoadTreeFile(Tree_File);                              // 현재까지의 트리파일 인식
   AddToTreeFile(Current_XML_File);                      // 작업중인 XML 내용을 트리에 반영
  }
  Current_XML_File = GetNextXML_File(XML_Directory_Path);        // 다음 XML 파일을 일어들임
}

Current_Generate_DTD = MakeDTD(Tree_File);               // 공통 트리에서 DTD 생성

// 생성된 DTD와 하부 DB를 나타내는 DTD간의 연관관계 추출
DTD_Mapping_Info = SetMappingInfo(Current_Generate_DTD, System_Common_DTD);

Current_XML_File = GetFirstXMLFile(수집XML_Directory_Path);
while(Current_XML_File != NULL) {                        // 디렉토리의 모든 XML 문서에 대하여
  SaveDataByDTD(Current_XML_File, Current_Generate_DTD, DTD_Mapping_Info );
                                                         // DTD 연관관계 사용하여 저장
  Current_XML_File = GetNextXML_File(수집XML_Directory_Path);
}
```

를 시스템에서 제공하는 상품정보를 위해 정의되어진 공통 DTD로 변환하는 역할을 담당한다. 이렇게 자동 수정된 XML 문서는 공통 DTD에 기반하여 하부 DB에 저장되게 된다. 이러한 하부 DB로의 저장을 위해서 본 연구에서는 추출되어진 DTD와 시스템에서 데이터를 표현하는 공통 DTD간의 상이점을 파악하여 수집되어진 문서들을 공통 DTD에 유효하도록 수정하는 모듈을 (XML 매퍼) 구현하였고, 이렇게 수정되어진 XML문서에서 상품 정보의 추출 및 DB 저장을 위해서는 Oracle XDK 유틸리티를 활용하였다.

5. 결 론

SECOS시스템의 Gathering Division이 보다 광범위한 정보를 수집하도록 하기 위해서는 Meta Gatherer의 역할이 중요하다. 우리는 Meta Gatherer의 DTD생성 모듈과 DB저장 모듈의 구조를 설계한 후 Java언어를 사용하여 이를 구현하였고, 이를 위해 개발된 알고리즘을 본 논문에서 설명하였다.

전체적인 Meta Gatherer를 위해서는 상용 툴 들도 사용되었는데, XML문서의 토큰 추출을 위해서는 Oracle xmlparserv2.0의 XMLTokenizer를, XML정보추출 및 그룹화 모듈의 트리를 구성하기 위해서는 Java의 Swing 컴포넌트를 사용하였다. 또한 XML 데이터의 추출 및 저장을 위해서는 Oracle XDK 유틸리티를 활용하였다. 본 연구에서 개발한 Meta Gatherer는 현재 SECOS 시스템과 연동되어 활용되고 있고, 계속적으로 안정화 및 성능향상을 위한 수정과 갱신작업이 이루어지고 있다.

향후 연구 과제는 다음과 같다. 첫째로, 본 연구에서는 DTD attribute에 대해서는 고려하지 않았다. 하지만 attribute는 구조적인 형태를 가지지 않으므로 개발된 모듈에 쉽게 반영될 수 있을 것이다. 둘째로, 임시노드를 사용한 트리구조는 상당히 복잡하다. 때문에 트리를 DTD로 변환하면 종종 복잡하고 가독성이 떨어지는 DTD가 생성된다. 생성된 트리를 순회하면서 트리의 구조를 단순화시켜 가독성을 높일 수 있는 연구가 앞으로 계속 진행되어야 할 것이다.

참 고 문 헌

[1] D. Oh and J. Jung, "Effective Web-Based Information Gathering Services of IHWA," *Proceedings of ICEIC'2000 International Conference*, Shenyang, China, pp. 202-205, 2000.

[2] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim, "XTRACT: A System for Extracting Document Type Descriptors from XML Documents," *Bell Labs Tech. Memorandum*, 1999.

[3] C. H. Moh, E. P. Lim, and W. K. Ng, "Re-engineering Structures from Web Documents," *Proceedings of the 5th ACM International Conference on Digital Libraries (DL2000)*, San Antonio, USA, 2000.

[4] N. Ashish and C. A. Knoblock, "Semi-Automatic Wrapper Generation for Internet Information Sources," *Proceedings of Coopis Conference*, 1997.

[5] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo, "Extracting Semi-Structured Data from the Web," *Proceedings of Workshop on Management of Semi-structured Data*, pp. 18-25, 1997.

[6] N. Kushmerick, D. Weil, and R. Doorenbos, "Wrapper Induction for Information Extraction," *Proceedings of Int. Joint Conference on Artificial Intelligence (IJCAI)*, 1997.

[7] Cay S. Horstmann and Gary Cornell, *Core Java Volumn I-II*, Sun Microsystems Press, 1998

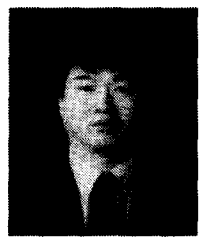
[8] Tom Valesky, *Enterprise JavaBeans - Developing Component-Based Distributed Applications*, Addison-wesley, 1999.

[9] alphaWorks, http://www.alphaworks.ibm.com/tech/DDbE

정 종 석

2000년 순천향대학교 컴퓨터학부 학사
2002년~순천향대학교 전산학과 석사
2002년~현재 (주)ECO 개발부 연구원

관심분야 : e-commerce, J2EE, 인터넷 응용 서버



오 동 익

1985년 뉴욕시립대학교 전산학 학사
1989년 플로리다 주립대학교 전산학 석사
1997년 플로리다 주립대학교 전산학 박사
1997년~현재 순천향대학교 정보기술공학부 조교수

관심분야 : m-commerce, e-commerce, 운영체제, 실시간 시스템, Ada 프로그래밍언어 등

교 신 저 자

오 동 익 336-745 충남 아산시 신창면 읍내리 646 순천향대학교 정보기술공학부