DOM형식 설계를 이용한 마크업 언어연구

이돈양*, 최한용**
*경인여자대학 전산정보학과
**한북대학교 컴퓨터공학과
e-mail:dylee62@empal.com*, hychoi@hanbuk.ac.kr**

A Study Markup Language using Design of DOM Form

Don-Yang Lee*, Han-Yong Choi**
*Dept of Computer Engineering, Kyung-In Woman's College
**Dept of Computer Engineering, Han-Buk University

요 약

DOM은 기본적으로 XML 문서를 구조적으로 표현한 것이다. 그리고 DOM은 XML문서를 노드의 트리로 인식하며, 이 노드는 동작이 가능한 오브젝트들로 구성되었다. 여기서 각 엘리먼트는 노드이며, 이 노드는 서브 트리를 구성할 수 있다. 본 논문에서는 DOM 트리생성을 이용한 XML 스키마의 생성방법 중 기본적인 사용형태인 사용자 정의 심플타입 DOM 트리 설계의 모든 노드 요소들은 IXMLDOMElement의 형식으로 엘리먼트들을 정의하여 클래스내의 단위 엘리먼트의 속성부여와 모델내의 클래스 관계를 표현할 수 있도록 하였다. 마크업언어의 생성에서는 XML 스키마를 이용하여 세부적인 데이터타입의 선언이 가능하도록 하고 있다.

1. 서론

XML에서 데이터를 이용한 어플리케이션 개발에서는 두 가지의 기능에 대해서 언급할 수 있는데, 하나는 마크업언어 설계에 관한 것이고, 다른 하나는 적절한 클래스의 생성에 대한 것이다. 마크업언어로는 일반적으로 DTD[1]와 XML 스키마[2]를 많이 사용하고 있다. DTD는 현재 널리 사용되고 있으며 광범위한 툴(tools)의 지원을 받고 있고, XML 스키마는 트리 구조의 문법을 사용하여 Document와 다양한 데이터 타입을 표현할 수 있다. 본 연구에서는 XML 스키마를 가지고 연구하였다. 그리고 XMI 메타모델로 정의된 세부적인 클래스의 메타데이터에 대한 생성방법정의, 생성도구설계 및 구현에 중심을 두었다. 현재 사용되고 있는 메타데이터설계 및 생성도구로 XML SPY[3], PIXEE[4], 그리고 다산의 XML Builder[5] 등이 있으나 이들 대부분이 일반적인 XML Document에서 요구되어지는 것들에 중점을 두고 있어 클래스에 대한 데이터타입(data type), 어트리뷰트(attribute), 엘리먼트(element), 인허리턴스(inheritance) 등의 세부적인 정의에 대해서는 어려움을 가지고 있다. 이를 보완하기 위한 방법으로 본 연구에서는 클래스내의 단위 엘리먼트의 속성을 부여할 수 있고 모델내의 클래스의 관계를 표현할 수 있는 SuperClass와 SubClass에 대한 적절한 타입의 속성을 표현할 수 있도록 DOM 형식을 이용하였다.

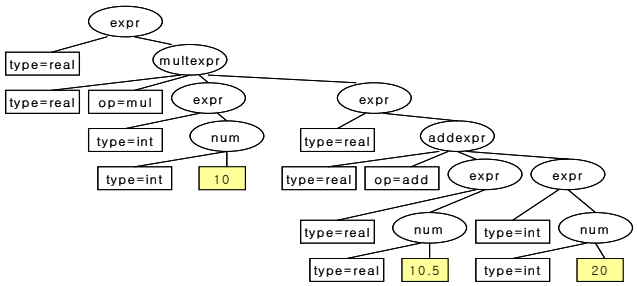
2. 관련연구

2.1 파서

파서는 어플리케이션 작성에서 XML 문서 안에 있는 정보에 접근을 해야 할 경우 사용된다[6]. 대표적인 파서로 SAX(Simple API for XML)와 DOM(Document Object Model)이 있으며, 두 가지 모두 XML 문서의 정보에 접근하여 어플리케이션을 개발할 수 있도록 API를 제공한다.

2.2 DOM

일반적으로 많이 사용되는 SAX 인터페이스는 매우 단순한 형태를 구성되어, 엘리먼트나 어트리뷰트를 사용하지 않는 own data structure를 만들 때 유리하다. 그러나 ID나 IDREF와 같은 internal cross-reference가 많으면 사용이 힘들 뿐만 아니라 구조적으로 복잡한 인헤리턴스(inheritance)등의 기능을 찾기가 어렵다. 그러나 DOM은 DOM tree라는 하나의 트리를 구성하여 파서를 생성되며, 모든 엘리먼트와 어트리뷰트가 하나의 vertex로 표현이 되어, 트리의 root는 Document의 root 엘리먼트가 되고, edge는 상호 종속관계로 표현이 가능하다.



(그림 1) DOM Tree

그리고 언어형식의 문법을 이용하여 일반적인 어휘의 명세방법을 표기할 수 있다[7]. 형식문법(formal grammer)은 G=(N,T,S,P) 집합으로 표시한다. 여기서 N은 비터미널(nonterminal) 심볼, T는 터미널(terminal) 심볼, S는 시작(start) 심볼, 그리고 P는 트랜잭션 규칙의 집합을 나타낸다. 아래의 문법에서는 왼쪽부분을 단지 한 개의 비터미널로 구성하고 있으며, 이를 CF(Context Free) 문법이라 한다.

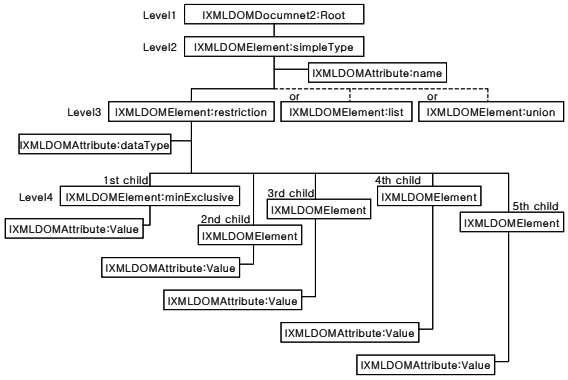
```
N = {expr, multexpr, addexpr, num}
S = expr
T = {"ADD", "MUL", NUM}
P :
 expr -> num | multexpr | addexpr
 addexpr -> expr "ADD" expr | expr "SUB" expr
 multexpr -> expr "MUL" expr | expr "DIV" ex
 num      -> NUM
```

3. 사용자 정의 심플타입을 이용한 XML 스키마 DOM 트리 설계

DOM은 기본적으로 XML 문서를 구조적으로 표현한 것이다. 그리고 DOM은 XML문서를 노드의 트리로 인식하며, 이 노드는 동작이 가능한 오브젝트들로 구성되었다. 여기서 각 엘리먼트는 노드이며, 이 노드는 서브 트리를 구성할 수 있다. MSXML 파서의 DOM 트리생성을 이용한 XML 스키마의 생성방법 중 기본적인 사용형태인 사용자 정의 심플타입 DOM 트리 설계의 모든 노드 요소들은 IXMLDOMElement의 형식으로 엘리먼트들을 정의하였다.

(그림 2)에서와 같이 Level1에 해당하는 Root 노드의 자식(child)노드로 Level2의 심플타입 노드를 생성하였으며 IXMLDOMAttribute를 이용하여 name의 속성 값을 부여할 수 있도록 하였다 그리고 Level3에서 restriction, list, union 엘리먼트를 사용하여 심플타입의 자식 엘리먼트를 생성할 수 있으나 동시에 여러 개의 엘리먼트를 사용할 수가 없으므로 반드시 한 개의 엘리먼트만을 환경에 따라 선택하여 생성하여야 한다.

Level3의 엘리먼트의 속성 값으로 데이터형식을 지정할 수 있다. 여기서는 이미 정의되어 사용가능한 심플타입을 이용하였으며 마크업언어의 네임스페이스 접두사인 "xsd"를 붙였다. Level4는 Level3의 자식노드로 패싯(facet) 엘리먼트에 해당하는 패싯에 대한 특정범위의 값을 정수(integer) 기반으로 입력한다.



(그림 2) 사용자 정의 심플타입 엘리먼트 트리

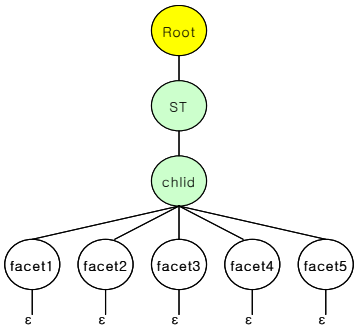
DOM 트리를 생성하기 위한 조건을 정의하기 위해서 Σ를 XML 엘리먼트 타입의 유한집합으로 놓고, D는 데이터 타입의 유한 집합, ε는 빈(empty) 데이터를 나타내면 $\varepsilon \in D$가 성립된다.

사용자 정의 심플타입 DOM 트리를 생성하기 위

해서는 다음의 정의를 기반으로 하고있다.

1. ε is a tree(ε is empty data)
2. α(ω), where α∈Σ and ω∈*D*, is a tree
   (Σ is finite set of XML element type, D is finite set of data type)
3. α(t_1,t_2, ..., t_k), κ≥1, where α∈Σ and t_i is a tree, i = 1,2, ..., κ, is a tree
4. nothing else is a tree. Let Σ = {Root, simpleType, child, facet1, facet2, facet3, facet4, facet5}.Then,Root(simpleType(child(facet1,facet2, facet3,facet4,facet5))).

그리고 이것을 Document 트리로 (그림 3)과 같이 도식할 수 있다.



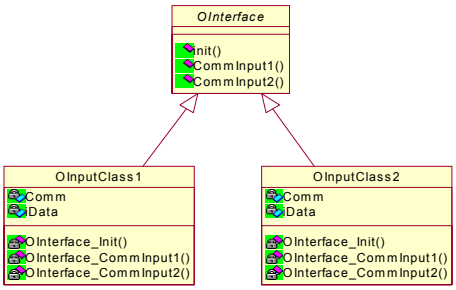
(그림 3) 사용자 정의 심플타입 DOM 트리

또한 이 Document 트리는 XML 마크업언어로 다음의 파일 형태로 생성할 수 있다. 즉 Document 트리의 각 노드(node)를 엘리먼트(element)로 표현을 하였으며 트리의 상위 레벨(level)과 하위 레벨(level)을 단계적으로 분류하여 작성할 수 있다.

```
<Root>
  <simpleType>
    <child>
      <facet1>
      <facet2>
      <facet3>
      <facet4>
      <facet5>
    </child>
  </simpleType>
</Root>
```

4. simple factory 패턴의 DOM 트리설계

본 논문에서는 사례연구로 simple factory 패턴의 구조를 가진 기본형의 디자인패턴인 (그림 4)를 적용하였다. 이는 세 개의 클래스로 구성되어 있으며 SuperClass *OInterface*는 abstract 클래스로, SubClass로 OInputClass1과 OInputClass2는 concrete 클래스로 정의하였다. 그리고 SuperClass와 SubClass는 일반관계(generalization)로 되어 클래스간의 상속(inheritance)적인 관계로 유지되었다.



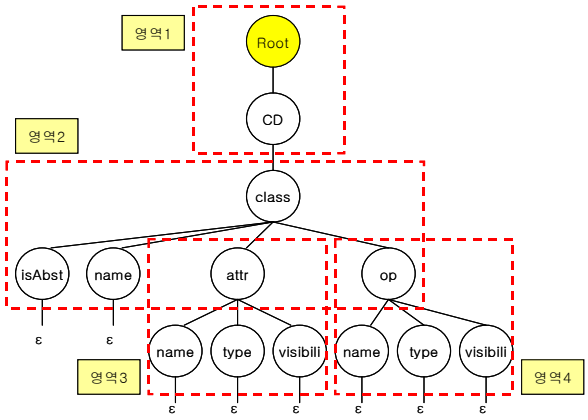
(그림 4) simple factory 패턴의 사례적용

simplefactory 패턴에 사용되고 있는 클래스를 DOM 트리의 형태로 단계적으로 생성한 것이 (그림 5)이다. 이 그림은 단순한 XML 문서형식에 대한 DOM 트리 형식을 도식한 것으로 다음과 같은 형식으로 정의할 수 있다. 메타데이터 형식으로는 <Root>와 <classDiagram> tag의 기본 구조에 따라 클래스에 대한 abstract, concrete 형식의 분류 그리고 이름과 각 어트리뷰트, 오퍼레이션에 대한 tag를 생성하였다.

```
<Root>
  <classDiagram>
    <class>
      <isAbstract> ... </isAbstract>
      <name> ... </name>
      <attribute>
        <name> ... </name>
        <type> ... </type>
        <visibility> ... </visibility>
      </attribute>
      <operation>
        <name> ... </name>
        <type> ... </type>
        <visibility> ... </visibility>
      </operation>
    </class>
  </classDiagram>
</Root>
```

그러나 본 논문에서 생성하고자 하는 XML 스키마 기반

의 마크업언어 생성을 위하여 <그림5-19>을 4개의 영역으로 구분하여 콤플렉스 타입으로 분류하면서 트리를 재구성하였다.



(그림 5) simple factory 패턴 DOM 트리

영역1은 XML 문서형식의 DOM 트리를 XML 스키마를 생성하기 위한 DOM 트리의 형태로 변경한 것이다. Root 노드 아래에 콤플렉스타입과 sequence를 이용하여 스키마를 작성하였으며, XML 클래스에 대한 마크업언어 생성은 다음과 같은 형식으로 작성되었다. 그리고 DOM 트리 생성 방법에 따라 다음의 Class Diagram 엘리먼트 DOM 트리 정의는 ∑ = {Root, complexType, sequence, classDiagram}로 놓고 Root(complexType(sequence(classDiagram)))에서 트리의 종속의 관계를 표현하였다. 이와 같이 영역2, 영역3, 영역4에 대한 마크업언어 생성를 생성하고 (그림 6) 의 알고리즘형식을 이용하여 결합하였다.

```
// 데이터베이스를 이용하여 생성된 스키마 문서
 begin {SchemaComposition}
     input version
     input SchemaNameSpace
     input RootElement
// classDiagramType.xsd 외부스키마 결합
     if classDiagramType = " " then
        input classDiagramType
     end {if}
// classType.xsd 외부스키마 결합
     if classType = " " then
        input classType
     end {if}
// attributeType.xsd 외부스키마 결합
     if attributeType = " " then
        input attributeType
     end {if}
// operationType.xsd 외부스키마 결합
     if operationType = " " then
        input operationType
     end {if}
 end {SchemaComposition}
```

(그림 6) 주 스키마에서 외부스키마 결합 알고리즘

동일한 네임스페이스를 사용하고 있는 조건에 따라서 네임스페이스의 선언과 루트 엘리먼트에 대한 기본적인 문서형식을 미리 정의하고, 외부 스키마의 결합형태로 존재하는 classDiagramType.xsd, classType.xsd, attributeType.xsd, operationType.xsd를 include 시켰다.

본 논문에서는 이런 결합의 방법을 통해서 단위형태의 재사용이 가능하도록 하였다. 그리고 평가기준으로서 단일클래스, simple factory 패턴, 패턴합성 모델을 가지고 본 연구에서 제안한 영역별 마크업언어 생성 및 결합에 대해 측정한 결과 파일크기와 코드라인 수에서는 각각 27%감소, 33% 정도의 감소를 가질 수 있음을 알 수 있었다.

참고문헌

[1] Tim Bray, Jean Paoli, and C.M. Sperberg-McQueen, editors. Extensible Markup Language(XML) 1.0. World Wide Web Consortium, 1998.

[2] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn, editors. XML Schema Part 1: Structures. World Wide Web Consortium, 2000.

[3] xmlspy Enterprise Edition User and Reference Manual, www.xmlspy.com/document/xmlspy2004.pdf. 2004

[4] Robert Kosara, Klaus Hammermuller, and Silvia Miksch. Codesigning XML-bsaed language and classes with pontifex. Technical Report Asgaard-TR-2000-1, Vienna University of Technology, Institute of Software Technology, Vienna, Austria 2000.

[5] http://www.tagfree.com

[6] Wu, I. C.; Hsieh, S. H, "An UML-XML-RDB Model Mapping Solution for Facilitating Information Standardization and Sharing in Construction Industry", International Symposium on Automation and Robotics in Construction, 19th (ISARC). Proceedings. National Institute of Standards and Technology, Gaithersburg, Maryland. September 23-25, 2002, 317-321 pp, 2002

[7] Georg Gottlob, Micheal Schrefl, and Brigitti Rock, "Extending Object-Oriented Systems with Roles", ACM Transactions on Information Systems 14, 3, 1996, pp.268-296