Xerces를 이용한 GML 파서의 개발

김영창[O] 장재우
전북대학교 컴퓨터공학과
{yckim[O], jwchang}@dblab.chonbuk.ac.kr

Development of GML Parser using Xerces

Youngchang Kim[0] Jaewoo Chang
Computer Engineering, Chonbuk National University

요 약

최근, 지리 정보 사용에 대한 관심과 응용 분야에 대한 개발이 증가함에따라서, 지리 정보의 공유 및 상호운용성에 대한 필요성이 증가하고 있다. 이에 따라 OGC(Open GIS Consortium)에서는 지리 정보의 교환 표준으로 GML 언어를 제시하였다. GML은 지리 정보를 전송 및 저장하기 위해 XML로 인코딩한 언어이다. 따라서, 지리 정보를 위한 공간 데이터베이스에 GML 문서를 저장하기 위해서는 효과적인 GML 문서의 파싱이 필수적이다. 본 논문에서는 대표적인 XML 파서인 Xerces를 확장하여 GML 문서를 효과적으로 파싱할 수 있는 GML 파서를 개발한다. 이를 위해 GML 스키마에서 제공하는 지리 정보 데이터 타입을 Xerces 파서의 내부 데이터 타입으로 제공하여, GML 응용 문서의 스캔(scan) 및 Validation을 위해 소요되는 GML 스키마 문서의 파싱 비용을 효과적으로 줄일 수 있다.

1. 서 론

최근, 웹 상에서의 지리 정보 사용에 대한 관심과 응용 분야가 증가하면서, 수집된 지리 정보의 공유 및 상호운용성에 대한 필요성이 점점 증가하고 있다[1,2,3]. 이에 따라 OGC(Open GIS Consortium)에서는 분산 환경에서 지리 정보의 전송 및 저장을 위한 표준으로 GML(Geographic Markup Language)[4]를 제안하였으며, 이러한 GML 언어로 작성된 GML 문서를 사용하여 지리 정보를 효율적으로 저장하기 위한 연구가 활발히 수행중에 있다[5,6,7,8]. 따라서, 지리 정보를 위한 GML 기반 저장 시스템을 위해, GML 문서를 파싱하여 효과적으로 데이터를 추출할 수 있는 GML 파서의 연구가 필수적이다. GML은 XML(eXtensible Markup Language)[9]로 인코딩된 언어로서 기존 XML 파서를 이용하여 파싱하는 것이 가능하다. 하지만, 기존의 파서는 GML 문서를 파싱할 때마다 GML 스키마 문서 또한 파싱하여야 한다는 단점을 가지고 있다. 이러한 단점을 극복하기 위해, 본 논문에서는 GML 스키마의 엘리먼트, 애트리뷰트, 타입 데이터를 XML 파서의 내부 데이터 타입으로 제공하여 확장함으로써 GML 문서의 파싱 시간을 단축할 수 있는 GML 파서를 개발한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 XML 문서 처리 방법인 DOM(Document Object Model), SAX(Simple API for XML)와 기존 XML 파서를 소개한다. 3장에서는 본 논문에서 설계한 GML 파서에 대해 기술하고, 4장에서는 기존 XML 파서와 제안하는 GML 파서의 성능분석을 수행하고, 마지막으로 5장에서 결론 및 향후 연구를 제시한다.

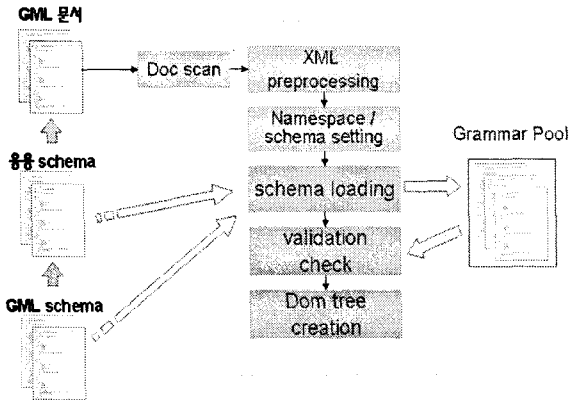
2. 관련 연구

GML은 XML 언어로 인코딩된 언어로써, 기존 XML 문서 처리에 대한 W3C의 표준인 DOM과 SAX 방법을 이용하여 GML 문서를 처리할 수 있다[10]. 첫째, DOM은 XML의 구조와 데이터로의 접근을 용이하게 하고, 데이터의 조작을 가능하게 하는 표준 인터페이스를 제공하며, XML 문서를 계층적인 트리구조의 형태로 나타낸다. 하지만, 전체 문서를 스캔하여 메모리에 유지해야하기 때문에 속도가 느리고, 메모리 사용량이 많은 단점이 존재한다. 둘째, SAX는 DOM 대신 사용할 수 있는 대안으로 DOM에 비하여 단순한 인터페이스를 제공하며, 속도가 빠른 장점을 가지고 있다. 하지만, 데이터의 내용을 조작하는 기능은 상대적으로 적다. 본 논문에서는 문서의 구조적인 접근의 효율성과 데이터의 조작 측면에서 우수한 DOM 방법을 사용하여 GML 문서를 파싱한다.

기존의 대표적인 XML 파서로는 Expat[11], libXML[12], Xerces[13], OracleXDK[14] 등이 있다. 이들 중 GML 문서를 위한 파서개발을 위해 엘리먼트 단위별 파싱능력, 문서의 validation, 지원 가능한 최대 문서 크기, DOM 지원 측면에서 가장 우수한 Xerces를 확장하여 GML 파서를 개발한다.

Xerces는 Apache Software Foundation에서 개발한 XML 파서로서, GML 문서의 파싱 및 validation을 위해 해당 스키마 문서의 파싱 및 validation을 먼저 수행한다. 이때, 해당 스키마 문서의 validation을 위해 외부 스키마 즉 GML 스키마 문서를 import하여 validation을 위한 grammar pool을 생성한다. 해당 스키마 문서의

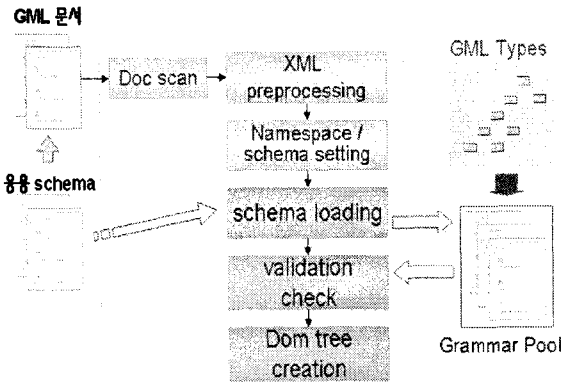
validation이 종료되면 이를 grammar pool에 추가하고 GML 문서의 파싱 및 grammar pool을 이용한 validation을 수행하여 DOM 트리를 생성한다. (그림 1)은 Xerces의 전체적인 파싱 구조를 나타낸다. 하지만, Xerces를 GML 파서로 사용할 때 파싱 구조상 외부 스키마인 GML 스키마 문서를 항상 파싱 및 validation 하여야 한다는 단점을 가지고 있다. OGC에서 제안된 GML은 현재 버전 3.1.1이며, 이는 33개의 스키마 문서로 구성된다. 즉 하나의 GML 문서를 파싱하기 위하여 33개의 GML 스키마 문서를 파싱하여야 한다는 단점을 가지고 있다. 따라서, 본 논문에서는 이를 해결하기 위하여 GML 스키마의 엘리먼트, 애트리뷰트, 타입 데이터를 Xerces의 내부 데이터 타입으로 제공하여 파싱 및 validation을 위한 비용을 감소할 수 있는 GML 파서를 제안한다.



(그림 1) Xerces의 전체적인 파싱 구조

3. GML 파서

본 논문에서 제안하는 GML 파서는 기존 XML 파서인 Xerces의 단점을 극복하기위해, GML 문서를 스캔하고 이에 대한 응용 스키마 문서를 파싱할 때, import 구문을 만나면 네임스페이스 검사를 수행한다. 만약, GML 네임스페이스(namespace)일 경우 GML 스키마 문서를 import 하지않고 이를 우회하여 GML 스키마에서 제공되는 데이터들을 엘리먼트, 애트리뷰트, 타입 데이터형태로 grammar pool에 추가한다. 응용 스키마 문서의 validation은 GML 데이터 타입이 추가된 grammar pool을 이용하여 이루어지고, 마지막으로 GML 문서의 validation이 성공적으로 수행되면 파싱된 데이터를 이용하여 DOM 트리를 생성한다. 이를 통해 GML 문서를 파싱 및 validation 할 때마다 소요되는 GML 스키마 문서에 대한 파싱 비용을 제거함으로써 전체적인 파싱 성능을 향상 시킬수 있다. (그림 2)는 GML 파서의 전체적인 파싱 구조를 나타내다.



(그림 2) GML 파서의 전체적인 파싱 구조

GML 스키마의 데이터는 지리 정보를 표현하기 위한 엘리먼트, 애트리뷰트, 타입 데이터로 나눌 수 있다. 엘리먼트는 GML 문서 및 응용 스키마 문서에서 사용되며, 애트리뷰트 및 타입 데이터는 응용 스키마 문서에서 응용을 위한 새로운 데이터 타입의 정의에 사용된다. 첫째, 엘리먼트는 GML 스키마에서 name, type, substitutionGroup 등의 속성 정보를 사용하여 구성되며 이외의 엘리먼트가 갖는 속성값은 XML의 default 값으로 제공한다. name은 엘리먼트의 이름을 나타내며, type은 해당 엘리먼트의 데이터 타입을, 마지막으로 substitutionGroup은 대체할 엘리먼트의 정규화된 이름을 나타낸다. 따라서, GML 스키마의 엘리먼트는 (그림 3)과 같은 형태의 데이터타입으로 나타낼 수 있다.

```
<element
        abstract = boolean : false
        block = (#all | List of (substitution | extension
                | restriction | list | union))
        default = string
        final = (#all | List of (extension | restriction))
        fixed = string
        form = (qualified | unqualified)
        id = ID
        maxOccurs = (nonNegativeInteger | unbounded) : 1
        minOccurs = nonNegativeInteger : 1
        name = NCName
        nillable = boolean : false
        ref = QName
        substitutionGroup = QName
        type = QName
        Content: (annotation?, ((simpleType | complexType)?
                , (unique | key | keyref)*))
</element>
```

(그림 3) 엘리먼트 데이터타입의 구조

둘째, 애트리뷰트는 attribute와 attributeGroup으로 나누어진다. attribute는 name, type, use 등의 속성 정보를 사용하여 구성되며 이외의 속성은 XML의 default 값으로

제공한다. 엘리먼트와 마찬가지로 name은 attribute의 이름을 나타내며, type은 attribute의 데이터 타입을, 그리고 use는 attribute가 표현될 수 있는 방법을 지시하는데 사용되며 optional, prohibited, 그리고 required 중의 하나의 값을 갖는다. attributeGroup은 name 속성 정보와 Content로 attribute 또는 attributeGroup 엘리먼트를 통해 다른 attrubute를 값으로 갖는다. (그림 4)는 애트리뷰트의 데이터 타입 구조를 나타낸다.

```
<attribute
      fixed = string
      default = string
      form = qualified | unqualified
      id = ID
      name = NCName
      ref = QName
      type = QName
      use = optional | prohibited | required
      Content: (annotation? , simpleType?)
<attribute/>

<attributeGroup
      id = ID
      name = NCName
      ref = QName>
      Content: (annotation? , (attribute|attributeGroup)*
                , anyAttribute?)
<attributeGroup/>
```

(그림 4) 애트리뷰트 데이터 타입의 구조

마지막으로 타입은 simpleType과 complexType으로 나누어진다. simpleType은 type의 이름을 나타내는 name 속성 정보를 갖으며, Content로 restriction 엘리먼트를 데이터로 갖는다. 이때, restriction은 base 속성 값으로 갖는 기본 데이터타입을 제한하는 역할을 하며 제한하는 방법을 위해 다양한 속성 값들을 Content 값으로 갖는다. 예를 들어, minLength와 maxLength는 기본 데이터형의 최소 길이와 최대 길이를 제한한다. (그림 5)는 simpleType의 데이터 타입 구조를 나타낸다.

```
<simpleType
   id = ID
   name = NCName>
   Content: (annotation? , ((list | restriction | union)))
</simpleType>

<restriction
   base = QNAME
   id = ID
   Content: (annotation?, (simpleType?, (minExclusive
      | minInclusive | maxExclusive | maxInclusive | totalDigits
      | fractionDigits | length | minLength | maxLength
      | enumeration | pattern | whiteSpace)*)?,
      ((attribute | attributeGroup)* , anyAttribute?))
</restriction>
```

(그림 5) simpleType의 데이터 타입 구조

complexType은 type의 이름을 나타내는 name 속성 정보를 지니며, Content로 simpleContent, complexContent, attribute, 그리고 attributeGroup 등을 사용하여 나타낸다. extension은 제한하거나 확장할 기본 데이터 타입의 이름을 base 속성값으로 갖는다. (그림 6)은 complexType의 데이터 타입 구조를 나타낸다.

```
<complexType
     abstract = boolean
     block = #all or subset of {extension, restriction}
     final = #all or subset of {extension, restriction}
     id = ID
     mixed = boolean : false
     name = NCName
     Content: (annotation? , (simpleContent | complexContent |
              ( (group | all | choice | sequence)? ,
              ( (attribute | attributeGroup)* , anyAttribute?))))
</complexType>

<extension
    base = QNAME
    id = ID
    Content: (annotation? , ((attribute | attributeGroup)* ,
             anyAttribute?))
</extension>
```

(그림 6) complexType의 데이터 타입 구조

4. 성능 분석

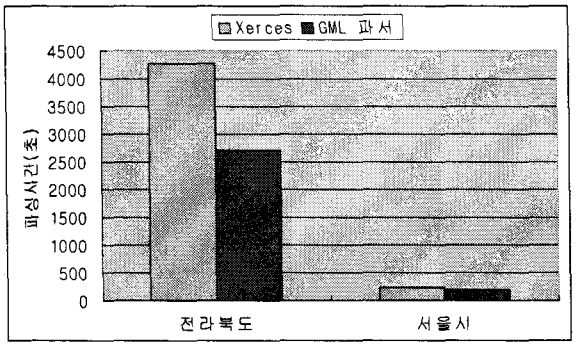
설계된 GML 파서의 성능분석을 위하여 (표 1)과 같은 환경에서 visual studio .net을 이용하여 구현하였다. GML 파서는 Xerces C++ 2.7.0 버전을 확장하여 구현하였으며, 성능분석을 위해 GML 버전 2.1.2의 데이터 타입 36가지를 추가하였다.

(표 1) GML 구현환경

시스템	HP ML 150G2 Xeon 3.0Ghz*2, 2GB Memory, 410GB HDD
운영체제	Windows 2003 Enterprise Server
개발언어	C++ (Visual Studio .Net)

성능분석은 기존 Xerces와 제안하는 GML 파서의 문서 파싱 시간을 측정한다. 사용된 데이터는 팅크웨어(주)에서 GML 버전 2.1.2에 기반하여 작성한 실제 전국 지도 데이터 중 서울시(5개구포함)와 전라북도 데이터를 사용하였으며 각각 70, 4158개의 GML 문서로 구성된다. 파싱 성능은 서울시와 전라북도의 GML 문서를 순차적으로 파싱하는데 소요되는 시간을 합산하여 측정한다. (그림 7)은 Xerces와 GML 파서가 GML 문서를 파싱하는데 소요되는 시간을 측정한 결과를 나타낸다. 그림에 나타난 바와 같이 전라북도 문서의 경우 Xerces는 4,275초, GML 파서는 2,702초 소요되었으며, 서울시 문서의 경우 Xerces는 약 231초 GML 파서는 190초로 GML 파서가 우수함을 알 수 있다. 이는 GML 스키마의 데이터 타입을 추가하여 GML 스키마의 파싱 시간을 제거함으로써

성능의 향상이 이루어졌기 때문이다. 성능의 차이가 비교적 적은 이유는 사용된 GML 문서가 GML 버전 2.1.2로써 3개의 스키마 문서에 기반하여 생성되었기 때문이다.



(그림 7) 파싱 성능 비교

5. 결론 및 향후연구

본 논문에서는 지리 정보를 위한 GML 기반 저장 시스템을 위해 GML 문서를 효과적으로 파싱할 수 있는 GML 파서를 제안하였다. 제안된 GML 파서는 GML 문서의 효과적인 파싱을 위해 GML 스키마의 엘리먼트, 애트리뷰트, 타입 데이터를 기존 XML 파서인 Xerces의 내부 데이터 타입으로 추가하였다. 이를 통해, GML 스키마 문서의 파싱에 소요되는 비용을 제거함으로써 파싱 성능을 증가시킬 수 있다. 아울러, GML 파서와 Xerces의 GML 문서의 파싱 성능 분석을 통해 제안한 GML 파서의 성능이 우수함을 보였다.
향후 연구로는 GML의 가장 최신 버전인 3.1.1의 데이터 타입을 추가하여 개발하고, 실제로 GML 문서 저장 시스템에 직접 적용하는 연구를 수행하는 것이다.

감사의 글
본 결과물은 교육인적자원부, 산업자원부, 노동부의 출연금 및 보조금으로 수행한 최우수실험실지원사업의 연구결과입니다.

[참고 문헌]

[1] Tom, H., "GIS Standardization : The American Experience", 개방형 GIS 연구회 논문지, 1권 1호, p99-108, 1999
[2] 오병우, 한기준, "지리 정보 시스템을 위한 표준화", 한국정보과학회 정보과학회지, 제 13권 10호, p46-55, 1995
[3] 장영승, 윤재관, 한기준, "ZEUS 기반 OpenGIS 서버의 설계 및 구현", 개방형 지리 정보 시스템 학술회의 논문집, 2권 2호, p21-32, 1999
[4] http://www.opengeospatial.org/standards/gml
[5] GML 데이타를 지원하는 확장된 DOM, 반재훈, 조정희, 문상호, 홍봉희, 한국정보과학회논문지, 제8권 5호, p510-519, 2002
[6] GML 문서의 데이터베이스 저장 스키마 설계기술, 신학진, 산학연구소논문집, 제24집, p69-85, 2004
[7] Lakshmi N. Sripada, Chang-Tien Lu, Weili Wu, " Evaluating GML Support for Spatial Databases", COMPSAC'04, p74-77, 2004
[8] S.-Y. Park, Y.-S. An, J.-D. Lee, and H.-Y. Bae, "On a GML Management System for Interoper- ability of Distributed Geographic Data and Its Application", IMSA2003, 2003
[9] http://www.w3.org/XML/
[10] http://www.w3c.org
[11] http://expat.sourceforge.net
[12] http://www.xmlsoft.org
[13] http://xerces.apache.org
[14] http://www.lc.leidenuniv.nl/awcourse/oracle/appdev.920/a96621/title.htm