XML 프로그래밍

충남대학교 이경하* · 이강찬* · 이규철**

1. XML 개요

1990년대 인터넷의 발전에 가장 큰 영향을 주었던 것은 바로 웹(WWW)이며, 이 웹 상에 문서를 기술하는 방법은 HTML(Hyper Text Markup Language)이었다. HTML은 누구나 사용할 수 있을 만큼 간단하며, 보편적이며, 특별한 데이터 타입이 사용되지 않고 단순한 텍스트이기 때문에 이식성과 사용이 편리하다.

또한 HTML은 SGML(Standard Generalized Markup Language)의 한 응용으로써, 이미 표준으로 정해진 DTD(Document Type Definition)에 따라 사용자는 그 인스턴스만 작성하면 된다. 즉, 사용할 수 있는 모든 태그(tag)는 전부 HTML의 DTD에 정의되어 있으므로 사용자는 정의된 태그만을 사용하기만 하면 웹 브라우저에서 볼 수 있다.

그러나, HTML은 이러한 여러 가지 장점에도 불구하고 그 한계가 있다. 첫째로, HTML은 고정된 태그 집합들을 이용하여 페이지들의 레이아웃(layout) 정보만을 지원하고 문서의 구조적 정보를 포함하지 않으며, 둘째로 잘못된 태그들은 레이아웃에서 배제시키는 정도의 검증만을 지원한다[12].

따라서, HTML은 현재와 같은 네트워크 상에서 복잡한 정보나 효과적인 검색, 재사용, 검증과 같은 능력에는 취약점을 드러내고 있다. W3C는 이러한 HTML의 취약점을 보완하기 위하여 새로운 마크업 언어(Markup Language)인 XML (eXtensible Markup Language)을 1996년 W3C WG에서 공개하였으며, 1998년 2월에는 W3C의 권고안(recommendation)으로서 XML 1.0을 발표하였다[10].

XML은 웹 상에서 구조화된 문서를 전송 가능토록 한 마크업 언어로써 HTML과 SGML의 장점을 수용하고, 단점을 극복한 표준언어이다. 특히 XML은 기능 면으로는 구조화된 문서의 작성을 지원하므로 보다 복잡한 문서의 작성을 용이하게 하며, 인터넷 기반의 마크업 언어이기 때문에 SGML보다 인터넷 언어로서 사용이 용이하다.

본 고에서는 이와 같은 차세대 인터넷 문서 표준인 XML을 기반으로 새로운 응용을 만들 때, 필요한 기본 도구인 XML 프로세서에 대해 살펴보고, 프로그래밍을 위해 이 프로세서들이 지원하는 API인 SAX와 DOM에 대해 기술한다. 마지막으로 XML을 기반으로 할 때, 변화될 컴퓨팅 환경을 소개하도록 한다.

2. XML 프로세서

2.1 XML 브라우저와 프로세서

XML 프로세서는 "XML 문서를 읽어서 문서 내용과 문서 구조에 대한 접근을 제공하는 소프트웨어"이다.

HTML 브라우저는 HTML문서를 읽고 문서 내의 엘리먼트(element)를 파싱하고 그에 따른 문서 내용을 레이 아웃하여 디스플레이할 수 있

* 학생회원
** 종신회원

도록 HTML 프로세서를 내장하고 있다. 마찬가지로, XML 브라우저는 XML로 작성된 문서나 데이터를 해석하고 보여주기 위해서 XML 프로세서를 내장하고 있다. XML 프로세서가 HTML 프로세서와 구별되는 차이점은 똑같은 XML 문법으로 작성된 문서라 하더라도 사용자가 작성한 DTD에 따라 서로 다르게 해석되어져야 하는 엘리먼트를 처리할 수 있는 방법을 제공한다는 것이다.

XML 프로세서를 이용하는 방법에는 크게 두 가지의 방법이 존재한다. 하나는 XML 브라우저를 이용하여, 문서를 파싱, 해석하며, XSL이나 CSS같은 스타일 쉬트 정보를 이용하여 적절히 표시/인쇄하는 프로그래밍이 필요 없는 방법이다. 이 경우의 예로는 기존의 웹 출판 시스템과 같이 문서로서의 보급 및 출판 등의 작업이 해당된다. 둘째는 XML 프로세서가 제공하는 API를 이용하여 XML 문서에 대한 접근, 갱신, 처리 작업을 할 수 있도록 XML 응용을 프로그래밍하는 방법이다. 프로세서는 문서와 DTD를 읽어 문서의 정확성을 검증하고, 문서 내의 엘리먼트와 컨텐트 트리, 카탈로그 등을 생성해 내며, 이를 이용하여 XML 응용은 해당 엘리먼트에 대한 특정 동작을 기술함으로서 XML 문서의 처리를 수행토록 한다.

2.2 XML 프로세서의 종류

XML 문서를 파싱하고 접근하는 방법에는 세 가지 방법이 존재할 수 있다.

1) 정규 식(Regular Expression)을 이용하여 패턴 대조에 따른 접근
2) XML 프로세서가 제공하는 자체 API를 이용한 문서의 접근 및 변경
3) 표준화된 API를 이용한 문서의 접근 및 변경

정규 식을 그대로 표현하여 수행하는 파싱은 단순한 XML 문서의 경우, 파싱을 쉽게 할 수 있고, 파싱 속도가 빠른 이점이 있으나, 이는 단일 DTD에 대한 응용프로그램이며, 다른 DTD에 대해서는 재 사용할 수 없다는 단점이 존재한다. 또한 이 경우, 정규 식을 지원하지 않는 프로그래밍 언어로는 이를 처리하기 어렵다는 점 등 많은 문제점이 존재한다. 따라서 XML 프로세서는 여러 DTD에 대한 XML 문서의 파싱을 위해 보다 특별하고 일반화된 방법을 제공할 수 있도록 하며, 프로세서에 대한 접근은 API를 이용토록 하고 있다. API는 프로세서에 따라 자체 API를 제공하는 방법과 표준화된 API를 제공하는 방법이 있으며, 표준화된 대표적인 API로 SAX[3]나 DOM[9,13]이 존재한다.

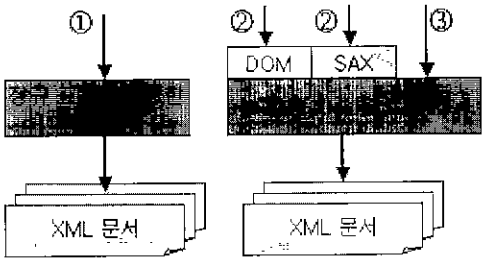

그림 1 XML 프로세서의 분류

또한, XML 프로세서는 적정 형식(well-formeness)에 대한 검증과 유효성(validity)에 대한 검증을 제공하는가에 따라 valid 프로세서와 well-formed 프로세서로 구분되어 진다. 즉, valid 프로세서의 경우, 태그의 쌍이 제대로 이루어져 있는가 등에 대한 검증뿐만 아니라 해당 태그 또는 엘리먼트들이 지정된 DTD에 정의되어 있는가에 대한 검증도 수행한다. 표 1은 현재 사용 가능한 XML 프로세서의 리스트이며, 이 중 Java로 작성된 프로세서의 정확성 검증에 대한 성능 검사는 [2]에서 행해진 실험 결과를 참고토록 한다. 더불어, Java로 작성된 XML 프로세서들의 성능 검사는 [1, 8]을 참고토록 한다.

3. XML API

표 1의 XML 프로세서 및 API들은 크게 두 가지의 접근 방법을 제공한다. 하나는 이벤트에 기반한 접근 방법(event-driven approach)이고 다른 하나는 오브젝트 모델에 기반한 접근 방법(object model approach)[5, 7]이다.

이벤트에 기반한 접근 방법은 XML 문서를 파싱할 때, 문서의 전체 구조 정보를 가지지 않으며 단순히 문서 내에 특정 엘리먼트를 만났을 때, 프로그래머가 수행토록 요구된 작업을 행하

표 1 XML 프로세서의 종류

이름	제작자	Validating	언어	DOM Level 1	SAX	Namespace	XLink XPointer
Ælfred	Microstar	N	Java	N	Y	N	N
DXP	Datachannel	Y	Java	Y	N	Y	N
expat	James Clark	N	C	N	N	N	N
HEX	Anders Kristensen	N	Java	Y	Y	N	N
Lark	Timbray	N	Java	N	Y	N	N
LT XML	Language Technoogy Group	Y	C	N	N	Y	N
SP	James Clark	N	C++	N	N	N	N
xmlproc	Lars Marius Garshol	Y	Python	N	Y	N	N
XML4j	IBM Alphaworks	Y	Java	Y	Y	Y	Y
XP	James Clark	N	Java	N	Y	N	N
XParse	Jeremie Miller	N	Javascr ipt	N	N	N	N
SXP	Silfide project	Y	Java	Y	Y	Y	Y
Java Project X	Sun Microsystems	Y	Java	Y	Y	Y	N
Oracle XML	Oracle	Y	Java	Y	Y	Y	N
MSXML for Java	Microsoft	Y	Java	Y	N	Y	N
XML::Parser	Clark Cooper	Y	Perl	N	N	N	N
TclXML	Steve Ball	Y	Tcl/Tk	N	N	N	N

도록 이벤트를 발생시키는 것을 목적으로 한다. 이런 접근 방법은 문서의 전체 구조 정보를 알 필요가 없는 경우, 즉 문서의 구조 정보보다는 문서 내에 특정 엘리먼트의 추출 연산이 중요한 메세징 서비스에 적합하며, 문서 내 구조 정보를 가지지 않으므로 좀 더 규모가 큰 문서를 메모리 상에 가지고 있을 수 있다는 이점이 있다.

오브젝트 모델에 기반한 접근 방법은 XML 문서를 파싱하여 문서의 구조 정보와 컨텐트 모두를 오브젝트로서 메모리 상에 올려놓고 이를 이용하는 방법이다. 이 경우, 문서 전체에 대한 구조 정보를 트리에 기반한 오브젝트로서 이용 가능하므로 XML 에디터와 같이 XML 문서를 구조적으로 변경하는 작업 등에 적합하며, 여러 응용들이 XML 문서를 메모리 상에서 공유할 수 있다는 이점이 존재한다. 그러나, 문서 전체에 대한 정보를 메모리 상에 올려놓음으로써 크기가 큰 XML 문서를 처리할 시에는 그에 따른 메모리 사용량도 늘어나는 단점이 존재한다. 그림 2는 이벤트 기반 접근 방법과 오브젝트 모델 기반 접근 방법의 비교를 나타낸 것이다.

3.1. SAX(Simple API for XML)

SAX[3]는 XML 문서를 파싱하기 위한 이벤트 기반의 API이다. 이 API는 XML 문서를 파싱할 때 특정 작업을 수행하는 필요한 메소드들을 포함하는 몇 개의 핸들러(handler) 인터페이스들과 이에 관련한 클래스들을 정의하고 있다.

SAX 1.0이 제공하는 클래스들과 인터페이스는 다음과 같다

1) 프로세서에 의해 구현되는 인터페이스인 AttributeList, Locator

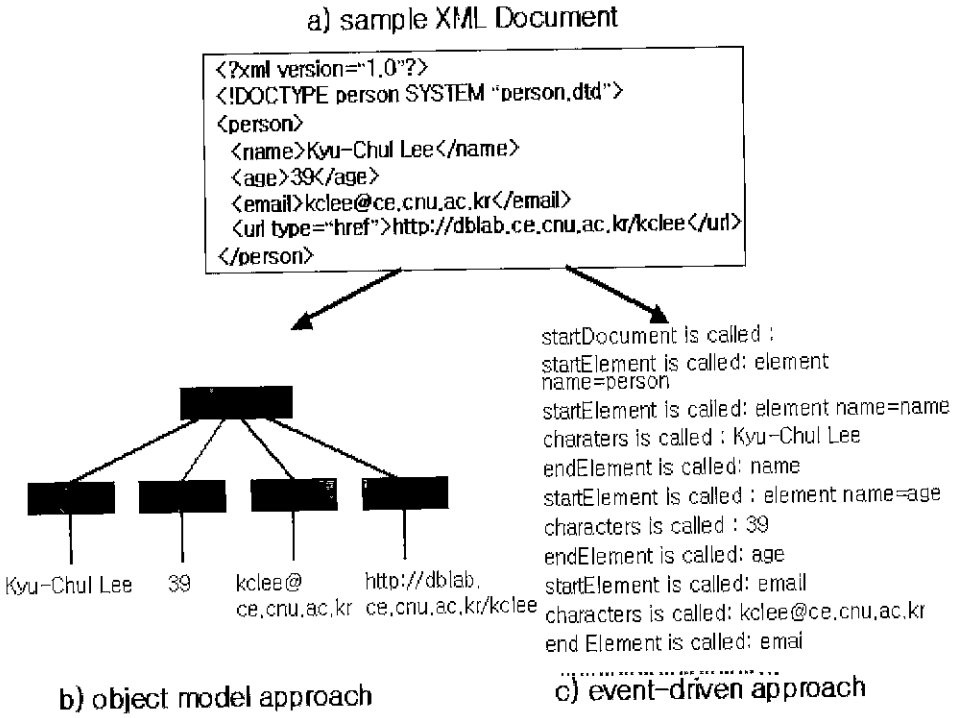

그림 2 object model approach와 eventdriven approach의 비교

2) 응용에 의해 구현되는 인터페이스인 DocumentHandler, ErrorHandler, DTDHandler, EntityResolver
3) SAX 표준 지원 클래스인 InputSource, SaxException, SAXParserException, HandlerBase

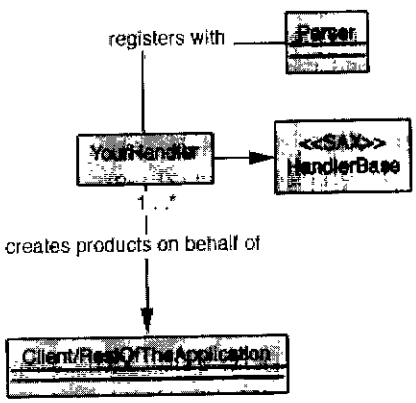

그림 3 SAX를 이용한 XML 프로그래밍

최근에 발표된 SAX 2.0에서는 위의 API의 개선과 함께 외부 엔티티 접근의 효율적인 접근, XML Namespace 지원을 위한 API를 제공하고 있다.

응용에 적합한 XML 모듈을 정의할 때 프로그래머는 HandlerBase 클래스(여러 핸들러 클래스들을 상속받은 서브 클래스)를 상속받아 특정 엘리먼트에 대한 작업을 수행토록 클래스를 오버라이드(override)한 후 이를 프로세서에 등록함으로써 쉽게 프로그래밍할 수가 있다. 그림 3은 이 과정을 도식화한 것이다.

그림 4는 그림 2 a)에서의 XML 문서를 XML4j의 SAX API를 이용하여 이를 파싱하고 해당 SAX 이벤트들을 표준 출력으로 출력하는 예이다. 예에서 보면 HandlerBase 클래스를 상속받아 각 메소드들(startDocument(), endDocument(), StartElement(), EndElement())을 오버라이드 한 후 이를 프로세서에 등록하여 사용함을 알 수 있다. 그림 4의 코드를 수행한 결과는 그림 2의 c)와 같다.

```
import org.xml.sax.*;
public class NotifyStr extends HandlerBase {
    static public void main(String[] argv) {
      try {
        Class c = Class.forName(argv[0]). //SAX 드라이버의 클
          래스 오브젝트의 획득
        // create instance of the class
        Parser parser = (Parser)c.newInstance();  //프로세서 클
          래스의 인스턴스화
        NotifyStr notifyStr = new NotifyStr(). // 도큐먼트 핸들
          러의 인스턴스화
        parser.setDocumentHandler(notifyStr),   //프로세서에 해
          당 도큐먼트 핸들러를 등록
        parser.parse(argv[1]);
      } catch (Exception e) {
        e.printStackTrace(),
      }
    }
    public NotifyStr() {   //dummy constructor
    }
    public void startDocument() throws SAXException {
      System.out.println("startDocument is called:");
    }
    public void endDocument() throws SAXException {
      System.out.println("endDocument is called:"),
    }
    public void startElement(String name, AttributeList amap)
      throws SAXException {
      System.out.println("startElement is called  element name="
        + name),
      for (int i = 0; i < amap.getLength(), i++) {
        String attname = amap.getName(i).  //엘리먼트의 어트
          리뷰트 이름 추출
        String type = amap.getType(i).  //어트리뷰트 타입 추출
        String value = amap.getValue(i),  //어트리뷰트 값 추출
        System.out.println("attribute name="+attname+"
                              type="+type+" value="+value),
      }
    }
    public void endElement(String name) throws
      SAXException {
      System.out.println("endElement is called " + name);
    }
    public void characters(char[] ch, int start, int length)
      throws SAXException { //엘리먼트의 값 추출
      System.out.println("characters is called. " + new
        String(ch, start, length)),
    }
}
```

그림 4 SAX API를 이용한 XML 프로그래밍의 예

3.2 DOM(Document Object Model)

DOM은 HTML과 XML 문서를 위한 응용 프로그래밍 인터페이스(API)이며, DOM은 문서를 접근하고 조작하기 위한 방법으로 문서의 논리적 구조를 정의한다. 따라서 DOM을 이용하여 사용자들은 문서를 생성하고 그 문서의 구조에 따라 항해(navigation)하고, 엘리먼트와 문서 내용을 추가/수정/삭제할 수 있다. DOM Level 1은 현재 W3C에 권고안으로 등록되어 있는 실정이고, 이중 DOM Core API는 HTML과 XML로 작성된 문서에 접근하는데 필요한 API를 포함하고 있으며, DOM HTML API는 CORE API에서 HTML 문서를 처리하는 필요한 인터페이스를 추가 확장한 것이다. 또한 최근에 발표된 DOM Level 2는 후보 권고안(candidate recommendation)으로 등록되어 있으며, CSS 같은 스타일 쉬트의 접근, UI에서의 이벤트 처리, 억세스 컨트롤 등과 같은 부분들을 수용하는 인터페이스들을 추가 제공한다.

DOM은 이름이 암시하는 바와 같이 문서에 대한 구조적 정보를 객체화된 인터페이스를 통하여 접근하도록 하고 있다. XML에서 문서의 구조는 시작 태그와 종료 태그의 쌍으로 이루어진 엘리먼트들의 집합으로 중첩 트리(nested-tree) 구조를 가지는데 DOM은 이런 엘리먼트들을 Element 노드들로서 처리하며, 이런 문서의 구성 요소들을 오브젝트로서 표현한다. 객체지향 프로그래밍의 관점에서 보면 DOM은 실제 구현이 되어야할 인터페이스들의 집합이다. 이것은 DOM 인터페이스는 각각 프로세서에 따라 그 구현 내용이 달라질 수 있으며, 여러 언어에서 구현시 상호운영성(interoperability)을 위해 CORBA 2.2의 OMG IDL, ECMAScript, 자바 언어에 대한 언어 바인딩 또한 명시하고 있다. 또한 DOM을 저장매체에 저장하는 방법과, DOM의 메모리의 관리 방법 등은 완전히 구현할 프로세서 쪽에 일임하고 있다. 그림 5는 DOM(Core) level 1 인터페이스의 클래스/인터페이스 계층도이다. Node 인터페이스는 XML 문서의 트리 구조를 이루는 모든 구성 요소들(엘리먼트, 엔티티, 어트리뷰트 등)의 데이터 구조를 구성하는 인터페이스이며 각 인터페이스들은 이 Node 인터페이스로부터 파생되어 특정 요소들을 대표하고 있다.

그림 6은 어떤 방법으로 DOM API를 이용하여 XML 응용을 작성하는지를 도식화한 것이다. 보는 바와 같이 DOM에서는 XML 문서의 파싱을 맞는 부분은 핸들러가 아니라 프로세서로부터 생성된 DOM 표현들을 처리하는 DOM처리 모듈이다. 이는 XML 응용에서의 문서 처리가 파

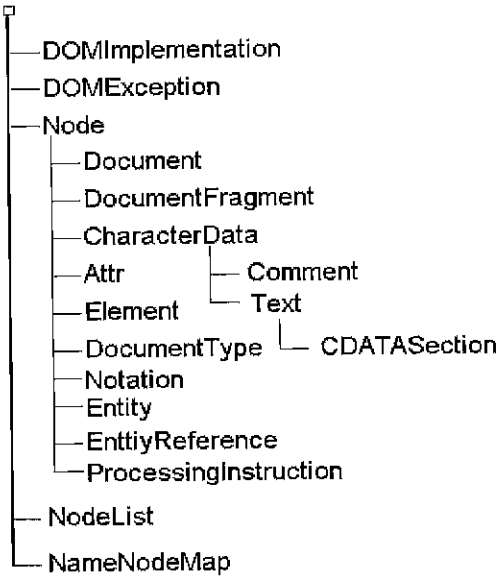

그림 5 DOM(Core) level 1의 클래스/인터페이스 계층도

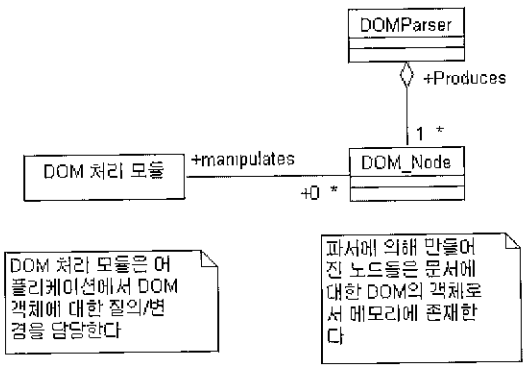

그림 6 DOM을 이용한 XML 프로그래밍

싱 타임에 수행되어지는 것이 아니라, 노드들을 처리할 별도 모듈이 처리함을 의미한다. DOM을 통한 XML 응용을 작성하는 과정은 다음과 같다.

1) 구현에 의존적인 내용들과 에러를 처리할 핸들러를 생성, 등록한다.
2) 프로세서로부터 파싱에 관한 질의를 처리 반환할 DOM 처리 모듈을 생성한다.
3) 생성한 처리 모듈을 XML 응용에 추가한다.

그림 7은 위의 과정으로 DOM API를 이용하여 그림 2의 a)에서 예시한 구조를 가지는 XML

```
import com.ibm.xml.parser.TXDocument;
import org.w3c.dom.*;
import java.io.PrintWriter;
public class MakeDocument{
    public MakeDocument(){
        try {
            //Document 오브젝트의 생성
            Document doc = (Document)Class.forName("com.ibm.
              xml.parser.TXDocument").newInstance();
            Element root = doc.createElement("person"); //<person>
              엘리먼트를 루트 엘리먼트로 생성

            Element item = doc.createElement("name");    //<name>
              엘리먼트를 생성
            item.appendChild(doc.createTextNode("John Doe")); //
              <name> 엘리먼트에 대한 값을 추가
            root.appendChild(item);  // <name>엘리먼트를 루트 엘리
              먼트에 추가

            // comment와 processing instruction을 생성 후 추가
            root.appendChild(doc.createComment("Processing
              Instruction for application"));
            root.appendChild(doc.createProcessingInstruction
              ("parser", " igonoreNextLine"));

            item = doc.createElement("age");          //<age> 엘리
              먼트를 생성하고 루트 엘리먼트에 추가
            item.appendChild(doc.createTextNode("35"));//
            root.appendChild(item);
            item = doc.createElement("email");        //<email> 엘리
              먼트를 생성하고 루트 엘리먼트에 추가
            item.appendChild(doc.createTextNode("John.Doe@foo.
              com"));
            item = doc.createElement("url");          //<url> 엘리먼
              트를 생성하고 루트 엘리먼트에 추가
            item.setAttribute("href", "http://www.foo.com/~John.D
            root.appendChild(item);

            // 루트 엘리먼트인 <person>을 도큐먼트 오브젝트에 추가
            doc.appendChild(root);

            // XML4j의 API를 이용해 생성한 XML 문서를 출력
            ((TXDocument)doc).setVersion("1.0");
            ((TXDocument)doc).printWithFormat(new
                PrintWriter(System.out));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

그림 7 DOM을 이용한 XML 프로그래밍

문서를 그림 2의 b)에 해당하는 DOM 트리로 생성하는 예이다. 1번째 줄에서의 TXDocument는 XML4j 프로세서가 제공하는 DOM의 Document 인터페이스의 실제 구현 클래스이며, 2번째 줄은 DOM의 인터페이스들에 대한 import 구문이다. Document는 XML 문서를 대표하는 오브젝트이며, createXXX() 메소드들을

이용하여 문서 내의 구성요소인 엘리먼트나 Comment, Processing Instruction을 생성하고, appendChild() 메소드를 이용하여 XML 문서를 이루는 트리 구조에 추가시켜줌으로써 XML 문서를 생성한다. setAttribute()는 엘리먼트가 가지는 어트리뷰트에 대한 속성과 값을 명시하는 메소드이다.

3.3 DOM과 SAX의 비교

DOM은 XML 문서와 문서를 이루는 구성요소들을 트리 형식으로 구성하여 오브젝트로서 메모리 상에서의 접근과 변경을 지원하는 반면, SAX는 XML 문서를 파싱할 때 문서 내 엘리먼트에 해당하는 이벤트를 처리함을 목표로 한다. 그에 따라 DOM은 문서 구조에 대한 접근에서 SAX에 비해 나은 환경을 제공하나, 메모리 사용량이 보다 많다. GMD-IPSI의 XQL 엔진의 경우, 많은 양의 XML 문서 처리 시 메모리 사용의 효율성을 위해 DOM 객체를 인덱싱하여 저장매체에 저장한 후 이용하는 Persistent DOM을 이용하여 DOM에서의 메모리 사용량을 줄이기도 한다[6].

표 2 DOM과 SAX의 비교

API	접근 방법	문서 구조의 접근/변경	메모리 사용량	타당한 적용분야
DOM	Tree-based	쉬움	높음	적은 양의 문서 변경/갱신 작업
SAX	Event-based	어려움	낮음	많은 양의 문서 처리 및 메세징 처리

4. XML 컴퓨팅 환경

4.1 웹 환경에서의 XML

XML 프로세서를 이용한 XML 응용들은 기존의 웹 환경을 그대로 이용할 수 있다. 즉, 기존의 HTTP 프로토콜 상에서 동작하던 웹 서버를 그대로 이용하면서, 단지 "text/xml" MIME 타입[4]으로 XML 문서를 전송하는 것으로 웹 서버를 그대로 이용한다.

4.2 XML을 통한 확장된 컴퓨팅 환경

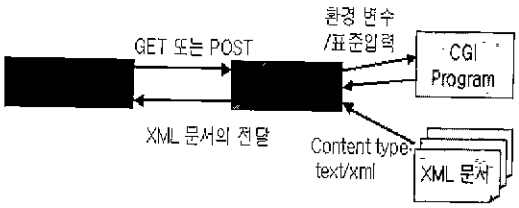

그림 8 웹에서의 XML 이용

XML은 기존의 XML을 이용하면서, 보다 확장된 컴퓨팅 환경을 제공한다. 그림 9는 이를 도식화 한 것으로 XML의 확장성과 정형성의 특징에 기반하여, XML이 단순히 웹 환경에서의 문서 표준으로 사용되어질 뿐만 아니라, 기존의 정보 매체들을 대체할 수 있는 방법을 제안한다

개발자들은 XML/XSL 에디터를 이용하여 문서의 구조와 레이아웃을 생성하여 문서의 출판/보급을 할 수 있다. 이렇게 생성된 XML 문서는 DOM이나 SAX같은 표준 API로서 XML문서의 프로그램적인 접근을 통하여 XML 데이터 서버에 기존의 데이터처럼 저장/변경이 가능하다. 저장된 문서는 XQL, XML-QL 등의 XML 질의 언어를 이용하여 문서의 구조 및 내용에 기반한 검색 기능을 지원할 수 있다. 또한, 기존 정보 자원들의 데이터들조차 XML의 정형성에 기반하여 XML 문서로 변환이 가능하다.

이 때 문서의 특정 DTD에 대한 처리는 응용이 담당할 수도 있으며, 또는 중간 계층(middle-tier)에 해당하는 웹 서버에 비즈니스 로직(business logic)을 추가하여 이 비즈니스 로직이 XML을 처리하여 해당 내용을 HTML로 제공할 수도 있다. 브라우저는 웹서버가 제공하는 XML 문서나 또는 XML 문서로부터 생성된 HTML 문서를 웹 서버에서 받아오는 것으로 모든 일을 대체할 수 있으며, 일반 응용에서의 XML 문서의 접근 및 처리 또한 XML 프로세서를 이용하여 쉽게 이용이 가능하다. 데이터 서버는 크게 XML 문서를 저장하는 XML 저장소와, HTML이나 기존의 DBMS, IRS와 같은 기존의 정보자원들을 XML로 변환하여 제공하여 주는 변환기(converter)가 존재할 수 있다. 웹 서버는 데이터 서버와 응용의 중간 계층으로서, 응용과 데이터 서버간의 상호 교환 및 공유가 빈번히 이루어지게 되는데, 여기서 XML은 데이터 교환

의 공용으로 사용되어지며, 서버와 서버간 메시지 전송에도 XML은 이용 가능하다.

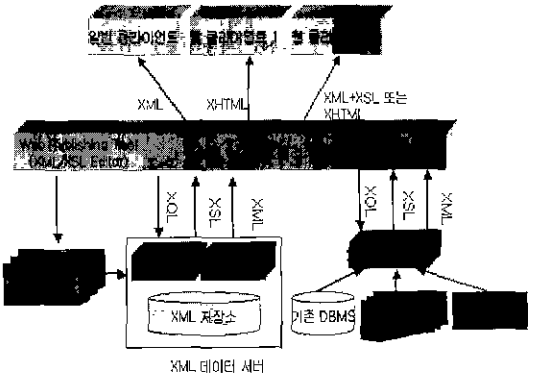

그림 9 XML을 통한 확장된 컴퓨팅 환경

5. 결 론

XML 응용들은 기존의 데스크탑 응용들이 다루었던 데이터들을 XML로 대체하면서 추가적인 이점을 제공해줄 수 있다. 즉, 확장성과 정형성 제공하는 XML의 특징은 단순한 문서로서의 기능 이외에도 기존의 데이터 모델을 대체할 새로운 수단으로 떠오를 수 있을 것이다. [11]에서 밝힌 바와 같이 XML은 기존의 정보 매체들에 대한 메타 데이터로서 사용이 가능하며, DOM, SAX와 같은 표준 API를 이용한 문서의 프로그램적인 접근을 통하여 문서 처리를 가능케 하고, 응용간의 데이터 교환/처리를 위해 사용되어질 수 있다. 더불어, XML의 정형성에 기반한 응용에서는 검색 및 데이터 변환 작성을 좀 더 효과적으로 수행할 수 있도록 하고 있다.

서버에서의 XML은 웹을 이용한 트랜잭션 처리의 확장성을 보장하면서 현존하는 웹 환경 및 도구를 최소한의 변경으로 이용할 수 있는 이점을 제공한다.

XML은 웹에서의 문서 작성을 위한 HTML의 기능을 보완하는 단순한 마크업 언어로서 머물지 않고 기존의 응용들이 다루었던 데이터 및 문서의 많은 영역들을 대체할 것이며, 그에 따른 새로운 응용들을 창출해 낼 것이라 기대된다.

참고문헌

[1] Clark Cooper, "Benchmarking XML Parsers", http://www xml.com/pub/Benchmark/article.html

[2] David Brownell, "Conformance Testing for XML Processor", http://www.xml.com/pub/1999/09/conformance/index.html

[3] David Megginson, "Simple API for XML(SAX)", http://www.megginson.com/SAX/index.html

[4] E. Whitehead, UC Irvine "RFC 2376: XML Media Type", IETF Network Working Group.

[5] Fabio Arciniegas, "An Introduction to C++ XML programming", http://www.xml.com/pub/1999/11/cplus/index.html

[6] GMD-IPSI, "GMD-IPSI XQL Engine", http://xml.darmstadt.gmd.de/xql/, GMD's XML Competence Center.

[7] H. Maruyama, K. Tamura, N. Uramoto, "XML and Java Developing Web applications", Addison-Wesley.

[8] Steven Marcus, "Benchmarking XML Parsers on Solaris", http://www.xml.com/pub/1999/06/benchmark/solaris.html

[9] W3 Consortium, "Document Object Model(DOM)", http://www.w3.org/DOM/

[10] W3 Consortium, "W3C Recommendation: Extensible Markup Language 1.0", http://www.w3.org/TR/1998/REC-xml-19980210"

[11] W3 Consortium "Extensible Markup Language(XML) Activity" http://www.w3c.org/XML/acitivity.html

[12] 이강찬, 이규철, "XML과 그 미래", EDI/EC Magagine, 98년 가을호(15).

[13] 이강찬, "DOM 동향 기술 분석서", http://dblab.ce.cnu.ac.kr/~dolphin/xml/atoz/dom.html

이 경 하

1998 충남대학교 정보통신공학과(공학사)
2000 충남대학교 정보통신공학과(공학석사)
현재 충남대학교 컴퓨터공학과 박사과정
관심분야: 데이터베이스, Java, XML, 정보통합, WWW
E-mail bart@ce.cnu.ac.kr

이 강 찬

1994 충남대학교 컴퓨터공학과(공학사)
1996 충남대학교 컴퓨터공학과(공학석사)
1996~현재 충남대학교 컴퓨터공학과 박사과정
관심분야: 데이터베이스, 정보통합, XML, 미디에이터, WWW
E-mail dolphin@ce.cnu.ac.kr

이 규 철

1984 서울대학교 컴퓨터공학과(공학사)
1986 서울대학교 대학원 컴퓨터공학과(공학석사)
1990 서울대학교 대학원 컴퓨터공학과(공학박사)
1989.3~현재 충남대학교 컴퓨터공학과 부교수
1994.3~1994.6 미국 IBM Almaden Research Center 객원 연구원
1995.8~1996.8 미국 Syracuse University, CASE Center 객원 교수
1997.1~1998.1 학술진흥재단 부설 첨단학술정보센터파견 교수
관심분야: 데이터베이스, XML, 정보통합, 멀티미디어 시스템
E-mail:kclee@ce.cnu.ac.kr

2000년 정례회의 및 주요행사 연간일정표

월 별	이 사 회		편집위원회	총회·학술발표회		송년회
	상 임	정 례	학회지	임시·춘계	정기·추계	
1월	14일(금) 16:00		21일(금) 16:30			
2월	14일(월) 16:00	18일(금) 16:00	18일(금) 16:30			
3월	10일(금) 16:00		17일(금) 16:30			
4월	7일(금) 16:00	21일(금) 16:00	21일(금) 16:30	28(금)~29(토) 대구효성가톨릭대		
5월	12일(금) 16:00		19일(금) 16:30			
6월	9일(금) 16:00	23일(금) 16:00	16일(금) 16:30			
7월	7일(금) 16:00		21일(금) 16:30			
8월	4일(금) 16:00	25일(금) 16:00	18일(금) 16:30			
9월	8일(금) 16:00		15일(금) 16:30			
10월	6일(금) 16:00	20일(금) 16:00	20일(금) 16:30		27(금)~28(토) 숙명여대	
11월	3일(금) 16:00		17일(금) 16:30			
12월	1일(금) 16:00	15일(금) 16:00	15일(금) 16:30			12일(화) 18:00

※ 회의일정은 사정에 따라 변경될 수 있음.