XML Technologies and Applications

Rajshekhar Sunderraman

Department of Computer Science
Georgia State University
Atlanta, GA 30302
raj@cs.gsu.edu

IV. XML Parsing APIs

December 2005
Outline

- Introduction
- XML Basics
- XML Structural Constraint Specification
  - Document Type Definitions (DTDs)
  - XML Schema
- XML/Database Mappings
- XML Parsing APIs
  - Simple API for XML (SAX)
  - Document Object Model (DOM)
- XML Querying and Transformation
  - XPath
  - XQuery
  - XSLT
- XML Applications
What is a parser?

- A program that analyses the grammatical structure of an input, with respect to a given formal grammar
- The parser determines how a sentence can be constructed from the grammar of the language by describing the atomic elements of the input and the relationship among them
XML-Parsing Standards

We will consider two parsing methods that implement W3C standards for accessing XML:

- **SAX (Simple API for XML)**
  - event-driven parsing
  - “serial access” protocol
  - Read only API

- **DOM (Document Object Model)**
  - convert XML into a tree of objects
  - “random access” protocol
  - Can update XML document (insert/delete nodes)
SAX Parser

• SAX = Simple API for XML

• XML is read sequentially

• When a parsing event happens, the parser invokes the corresponding method of the corresponding handler

• The handlers are programmer’s implementation of standard Java API (i.e., interfaces and classes)

• Similar to an I/O-Stream, goes in one direction
Sample Data

Orders Data in XML:
several orders, each with several items
each item has a part number and a quantity

```xml
<orders>
  <order>
    <onum>1020</onum>
    <takenBy>1000</takenBy>
    <customer>1111</customer>
    <recDate>10-DEC 94</recDate>
    <items>
      <item>
        <pnum>10506</pnum>
        <quantity>1</quantity>
      </item>
      <item>
        <pnum>10507</pnum>
        <quantity>1</quantity>
      </item>
      <item>
        <pnum>10508</pnum>
        <quantity>2</quantity>
      </item>
      <item>
        <pnum>10509</pnum>
        <quantity>3</quantity>
      </item>
    </items>
  </order>
  ...
</orders>
```
Sample Data

Orders Data in XML:
several orders, each with several items
each item has a part number and a quantity

```xml
<orders>
  <startDocument/>
  <order>
    <onum>1020</onum>
    <takenBy>1000</takenBy>
    <customer>1111</customer>
    <recDate>10-DEC 94</recDate>
    <items>
      <item>
        <pnum>10506</pnum>
        <quantity>1</quantity>
      </item>
      <item>
        <pnum>10507</pnum>
        <quantity>1</quantity>
      </item>
      <item>
        <pnum>10508</pnum>
        <quantity>2</quantity>
      </item>
      <item>
        <pnum>10509</pnum>
        <quantity>3</quantity>
      </item>
    </items>
  </order>
  ...
  <endDocument/>
</orders>
```
Sample Data

Orders Data in XML:
several orders, each with several items
each item has a part number and a quantity

<orders>
<order>
    <onum>1020</onum>
    <takenBy>1000</takenBy>
    <customer>1111</customer>
    <recDate>10-DEC-94</recDate>
    <items>
        <item>
            <pnum>10506</pnum>
            <quantity>1</quantity>
        </item>
        <item>
            <pnum>10507</pnum>
            <quantity>1</quantity>
        </item>
        <item>
            <pnum>10508</pnum>
            <quantity>2</quantity>
        </item>
        <item>
            <pnum>10509</pnum>
            <quantity>3</quantity>
        </item>
    </items>
</order>
...
Sample Data

Orders Data in XML:
several orders, each with several items
each item has a part number and a quantity

<orders>
  <order>
    <onum>1020</onum>
    <takenBy>1000</takenBy>
    <customer>1111</customer>
    <recDate>10-DEC-94</recDate>
    <items>
      <item>
        <pnum>10506</pnum>
        <quantity>1</quantity>
      </item>
      <item>
        <pnum>10507</pnum>
        <quantity>1</quantity>
      </item>
      <item>
        <pnum>10508</pnum>
        <quantity>2</quantity>
      </item>
      <item>
        <pnum>10509</pnum>
        <quantity>3</quantity>
      </item>
    </items>
  </order>
  ...
</orders>
SAX Parsers

<?xml version="1.0"?>

When you see the start of the document do ...

When you see the start of an element do ...

When you see the end of an element do ...

SAX Parser
Used to create a SAX Parser

Handles document events: start tag, end tag, etc.

Handles Parser Errors

Handles DTD

Handles Entities
SAX API

Two important classes in the SAX API: SAXParser and HandlerBase.

A new SAXParser object is created by:

```
public SAXParser()
```

Register a SAX handler with the parser object to receive notifications of various parser events by:

```
public void setDocumentHandler(DocumentHandler h)
```

A similar method is available to register an error handler:

```
public void setErrorHandler(ErrorHandler h)
```
The `HandlerBase` class is a default base class for all handlers. It implements the default behavior for the various handlers. Application writers extend this class by rewriting the following event handler methods:

```java
public void startDocument() throws SAXException
public void endDocument() throws SAXException
public void startElement() throws SAXException
public void endElement() throws SAXException
public void characters() throws SAXException
public void warning() throws SAXException
public void error() throws SAXException
```
Creating a SAX Parser

```java
import org.xml.sax.*;
import oracle.xml.parser.v2.SAXParser;
public class SampleApp extends HandlerBase {
    // Global variables declared here
    static public void main(String [] args){
        Parser parser = new SAXParser();
        SampleApp app = new SampleApp();
        parser.setDocumentHandler(app);
        parser.setErrorHandler(app);
        try {
            parser.parse(createURL(args[0]).toString());
        } catch (SAXParseException e) {
            System.out.println(e.getMessage());
        }
    }
}
```
SAX API – Sample Application

Write a SAX Parser that reads the `orders.xml` file and extracts the various data items and creates SQL insert statements to insert the data into a relational database table.

```java
//Global Variables
Vector itemNum = new Vector();
int numberOfRows, numberOfItems;
String elementEncountered, orderNumber, takenBy,
    customer, receivedDate, partNumber, quantity;
//elementEncountered holds most recent element name

public void startDocument() throws SAXException {
    //Print SQL comment, initialize variable
    System.out.println("--Start of SQL insert Statements");
    itemNum.setSize(1);
    numberOfRows = 0;
    //}//end startDocument
```
public void startElement(String name, 
                      AttributeList atts) throws SAXException {
    elementEncountered = name;
    if (name.equalsIgnoreCase("items")) {
        numberOfItems = 0;
    } //end if statement
} //end startElement

public void characters(char[] cbuf, int start, int len) {
    if (elementEncountered.equals("orderNumber")) {
        orderNumber = new String(cbuf,start,len);
    } else if(elementEncountered.equals("takenBy")) {
        takenBy = new String(cbuf,start,len);
    } ... 
} //end characters
public void endElement(String name) throws SAXException{
    if (name.equalsIgnoreCase("item")) {
        numberOfItems++;
        if (numberOfItems == 1) { // first item; create orders row
            System.out.println(
                "insert into orders values('"+ 
                orderNumber +"','"+ customer +"','"+ 
                takenBy +"','"+ receivedDate +"','null');");
        }
        System.out.println("insert into odetails values('"+ 
                orderNumber +"','"+ partNumber +"','"+ 
                quantity +"');");
    } //end if statement
    if (name.equalsIgnoreCase("items")) {
        System.out.println("--------");
    }
} //end endElement
public void endDocument()
{
    System.out.println("End of SQL insert statements.");
}//end endDocument

public void warning(SAXParseException e)
    throws SAXException {
    System.out.println("Warning:"+ e.getMessage());
}

public void error(SAXParseException e)
    throws SAXException{
    throw new SAXException(e.getMessage());
}
DOM Parser

- DOM = Document Object Model

- Parser creates a tree object out of the document

- User accesses data by traversing the tree
  - The tree and its traversal conform to a W3C standard

- The API allows for constructing, accessing and manipulating the structure and content of XML documents
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <name>Israel</name>
    <population year="2001">6,199,008</population>
    <city capital="yes"><name>Jerusalem</name></city>
    <city><name>Ashdod</name></city>
  </country>
  <country continent="&eu;">
    <name>France</name>
    <population year="2004">60,424,213</population>
  </country>
</countries>
The DOM Tree

- Document
- Continent
- Country
- City
- Name
Using a DOM Tree

XML File → DOM Parser → DOM Tree → Application
The Node Interface

- The nodes of the DOM tree include
  - a special root (denoted `document`)
  - element nodes
  - text nodes and CDATA sections
  - attributes
  - comments
  - and more ...

- Every node in the DOM tree implements the **Node** interface
Node Navigation

- Every node has a specific location in tree
- **Node** interface specifies methods for tree navigation
  - `Node getFirstChild();`
  - `Node getLastChild();`
  - `Node getNextSibling();`
  - `Node getNextSibling();`
  - `Node getParentNode();`
  - `NodeList getFirstChild();`
  - `NamedNodeMap getAttributes();`
Node Navigation (cont)

- getPreviousSibling()
- getFirstChild()
- getLastChild()
- getNextSibling()
- getParentNode()
- getChildNodes()
DOM Parsing Example

Consider the following XML data file describing geographical information about states in U.S.

```xml
<?xml version="1.0" standalone="no"?>
<!DOCTYPE geography SYSTEM "geo.dtd">
<geography>
  <state id="georgia">
    <scode>GA</scode>
    <sname>Georgia</sname>
    <capital idref="atlanta"/>
    <citiesin idref="atlanta"/>
    <citiesin idref="columbus"/>
    <citiesin idref="savannah"/>
    <citiesin idref="macon"/>
    <nickname>Peach State</nickname>
    <population>6478216</population>
  </state>
  ...
  <city id="atlanta">
    <ccode>ATL</ccode>
    <cname>Atlanta</cname>
    <stateof idref="georgia"/>
  </city>
  ...
</geography>
```
<!ELEMENT geography (state|city)>
<!ELEMENT state (scode, sname, capital, 
citiesin*, nickname, population)>
  <!ATTLIST state id ID #REQUIRED>
<!ELEMENT scode (#PCDATA)>
<!ELEMENT sname (#PCDATA)>
<!ELEMENT capital EMPTY>
  <!ATTLIST capital idref IDREF #REQUIRED>
<!ELEMENT citiesin EMPTY>
  <!ATTLIST citiesin idref IDREF #REQUIRED>
<!ELEMENT nickname (#PCDATA)>
<!ELEMENT population (#PCDATA)>
<!ELEMENT city (ccode, cname, stateof)>
  <!ATTLIST city id ID #IMPLIED>
<!ELEMENT ccode (#PCDATA)>
<!ELEMENT cname (#PCDATA)>
<!ELEMENT stateof EMPTY>
  <!ATTLIST stateof idref IDREF #REQUIRED>
create type city_type as object ( 
    ccode   varchar2(15),
    cname   varchar2(50)
);

create type cities_in_table as table of city_type;

create table state ( 
    scode   varchar2(15),
    sname   varchar2(50),
    nickname varchar2(100),
    population number(30),
    capital   city_type,
    cities_in cities_in_table,
    primary key (scode)) 
    nested table cities_in store as cities_tab;
import org.w3c.dom.*;
import org.w3c.dom.Node;
import oracle.xml.parser.v2.*;

class CreateGeoData {
    static public void main(String[] argv)
        throws SQLException {
        // Get an instance of the parser
        DOMParser parser = new DOMParser();
        parser.setErrorStream(System.err);
        parser.setValidationMode(true);
        parser.showWarnings(true);
        parser.parse(url);
        // Obtain the document.
        XMLDocument doc = parser.getDocument();
    }
}
Traverse DOM Tree

```java
NodeList sl = doc.getElementsByTagName("state");
NodeList cl = doc.getElementsByTagName("city");

XMLNode e = (XMLNode) sl.item(j);
scode = e.valueOf("scode");
sname = e.valueOf("sname");
nickname = e.valueOf("nickname");
population = Long.parseLong(e.valueOf("population"));

XMLNode child = (XMLNode) e.getFirstChild();
while (child != null) {
    if (child.getNodeName().equals("capital"))
        break;
    child = (XMLNode) child.getNextSibling();
}

NamedNodeMap nnm = child.getAttributes();
XMLNode n = (XMLNode) nnm.item(0);
```
Node Manipulation

- Children of a node in a DOM tree can be manipulated - added, edited, deleted, moved, copied, etc.

- To construct new nodes, use the methods of Document
  - createElement, createAttribute, createTextNode, createCDATASection etc.

- To manipulate a node, use the methods of Node
  - appendChild, insertBefore, removeChild, replaceChild, setNodeValue, cloneNode(boolean deep) etc.
The DOM object built by DOM parsers is usually complicated and requires more memory storage than the XML file itself.
- A lot of time is spent on construction before use.
- For some very large documents, this may be impractical.

SAX parsers store only local information that is encountered during the serial traversal.

Hence, programming with SAX parsers is, in general, more efficient.
Programming using SAX is Difficult

- In some cases, programming with SAX is difficult:
  - How can we find, using a SAX parser, elements \textit{e1} with ancestor \textit{e2}?
  - How can we find, using a SAX parser, elements \textit{e1} that have a descendant element \textit{e2}?
  - How can we find the element \textit{e1} referenced by the IDREF attribute of \textit{e2}?
Node Navigation

- SAX parsers do not provide access to elements other than the one currently visited in the serial (DFS) traversal of the document.

- In particular,
  - They do not read backwards
  - They do not enable access to elements by ID or name

- DOM parsers enable any traversal method

- Hence, using DOM parsers is usually more comfortable
More DOM Advantages

- DOM object ⇔ compiled XML

- You can save time and effort if you send and receive DOM objects instead of XML files
  - But, DOM object are generally larger than the source

- DOM parsers provide a natural integration of XML reading and manipulating
  - e.g., “cut and paste” of XML fragments
Which should we use? DOM vs. SAX

- If your document is very large and you only need to extract only a few elements – use SAX
- If you need to manipulate (i.e., change) the XML – use DOM
- If you need to access the XML many times – use DOM (assuming the file is not too large)