Java APIs for XML Processing

- DOM: a language-neutral interface for manipulating XML data
  - requires that the entire document be in memory
- SAX: push-based stream processing
  - hard to write non-trivial applications
- StAX: pull-based stream processing
  - far better than SAX, but not very popular yet

XML Programming in Java

© Leonidas Fegaras
University of Texas at Arlington

DOM

The Document Object Model (DOM) is a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content and structure of XML documents.

- The following is part of the DOM interface:
  
  public interface Node {
    public String getNodeName();
    public String getNodeValue();
    public NodeList getChildren();
    public NamedNodeMap getAttributes();
  }

- public interface Element extends Node {
    public NodeList getElementsByTagName(String name);
  }

- public interface Document extends Node {
    public Element getDocumentElement();
  }

- public interface NodeList {
    public int getLength();
    public Node item(int index);
  }

Traversing the DOM Tree

- Finding all children of node n with a given tagname
  NodeList nl = n.getElementsByTagName("tagname");
  for (int i = 0 ; i < nl.getLength(); i++) {
    Node m = nl.item(i);
    if (m.getNodeName().equals("tagname"))
      ...

- ... but can also be done this way
  NodeList nl = n.getElementsByTagName("tagname");
  for (int i = 0 ; i < nl.getLength(); i++) {
    Node m = nl.item(i);
    ...

- Finding all descendants-or-self of node n with a given tagname requires recursion
Node Printer using DOM

```java
static void printNode ( Node e ) {
    if (e instanceof Text)
        System.out.println(((Text) e).getData());
    else {
        NodeList c = e.getChildNodes();
        System.out.print( "<" + e.getNodeName() + " >");
        for (int k = 0; k < c.getLength(); k++)
            printNode(c.item(k));
        System.out.print( "/>" + e.getNodeName() + "/");
    }
}
```

A DOM Example

```java
import javax.xml.parsers.*;
import org.w3c.dom.*;
import java.io.File;

class DOMTest {
    static void query ( Node n ) {
        NodeList nl = (Element) n.getElementsByTagName("gradstudent");
        for (int i = 0; i < nl.getLength(); i++) {
            Element e = (Element) nl.item(i);
            if (e.getElementsByTagName("first_name").item(0)
                .getFirstChild().getNodeValue().equals("John"))
                printNode(e);
        }
    }

    public static void main ( String args[] ) throws Exception {
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document doc = db.parse(new File("cs.xml"));
        Node root = doc.getDocumentElement();
        query(root);
    }
}
```

A Better, Modular Programming

- Want to write modular Java code to evaluate XPath queries
- One component for each XPath axis step
- An XPath query is constructed by composing these components
- Each component reads a sequence of Nodes and returns a sequence of Nodes
  - ... starting from the root: root = new Sequence("cs.xml")

Examples:
- `/department/gradstudent/name`
  - root.child("department").child("gradstudent").child("name").print();
- `/department/gradstudent[gpas="3.5"]/name`
  - Condition p = new Condition()
    - public Sequence condition ( Node x ) {
      return (new Sequence(x).child("gpas").equals("3.5");
    }
  - root.child("department").child("gradstudent").predicate(p).child("name").print();
- See: examples/dom1.java

Implementation

- Every component is a method of Sequence that returns a Sequence
  - class Sequence extends Vector<Node> {
    ...
  }
- `child`: returns the children of the current nodes that have the given tagname
  ```java
  public Sequence child ( String tagname ) {
      Sequence result = new Sequence();
      for (int i = 0; i < size(); i++) {
          Node n = elementAt(i);
          NodeList c = n.getChildNodes();
          for (int k = 0; k < c.getLength(); k++)
              if (c.item(k).getNodeName().equals(tagname))
                  result.addElement(c.item(k));
      }
      return result;
  }
  ```

Web Databases and XML  1b: XML Programming in Java  5
Web Databases and XML  1b: XML Programming in Java  6
Web Databases and XML  1b: XML Programming in Java  7
Web Databases and XML  1b: XML Programming in Java  8
Descendant must be Recursive

- `descendant`: returns the descendants of the current nodes that have the given tagname
  
  ```java
  public Sequence descendant ( String tagname )
  {
      Sequence result = new Sequence();
      for (int i = 0; i < size(); i++)
      {
          Node n = elementAt(i);
          NodeList c = n.getChildNodes();
          for (int k = 0; k < c.getLength(); k++)
          {
              if (c.item(k).getNodeName().equals(tagname))
              {
                  result.add(c.item(k));
              }
          }
          result.append((new Sequence(c.item(k))).descendant(tagname));
      }
      return result;
  }
  ```

Predicates

- The predicate method must be parameterized by a condition
  
  ```java
  abstract class Condition {
      public Condition () {}      // abstract public Sequence condition ( Node x );
  }
  ```

- Filter out the current nodes that do not satisfy the condition pred
  
  ```java
  public Sequence predicate ( Condition pred )
  {
      Sequence result = new Sequence();
      for (int i = 0; i < size(); i++)
      {
          Node n = elementAt(i);
          Sequence s = pred.condition(n);
          boolean b = false;
          for (int j = 0; j < s.size(); j++)
          {
              b = b || (s.elementAt(j) instanceof Text) || (Text s.elementAt(j).getText).equals("false");
          }
          if (b) result.add(n);
      }
      return result;
  }
  ```

An Even Better Method

- Problem: the intermediate Sequences of Nodes may be very big
  - they are created to be discarded at the next step
- Example:
  ```xml
  <book><author="Smith"></book>
  <book> may return thousands of items to be discarded by the predicate
- Instead of creating Sequence, we can process one Node at a time
- Pull-based processing
- Method: pull-based iterators to avoid creating intermediate sequences of nodes
  - A method used extensively in stream processing
  - Also used in database query evaluation (query pipeline processing)
- See: examples/dom2.java

Iterators

- Iterator interface:
  ```java
  abstract class Iterator {
      Iterator input;  // the previous iterator in the pipeline
      public Node next () { return input.next(); }   
  }
  ```

- `department/gradstudent/name`
  ```java
  query(new Child("name",new Child("gradstudent",new Child("department",
          new Document("cs.xml")))));
  ```

- `department/gradstudent[gpa="3.5"]/name`
  ```java
  Clone n = new Clone(new Child("graduate",new Child("department",
          new Document("cs.xml")));
  query(new Child("name",new Predicate(new Equals("3.5",new Child("gpa",n)));
  ```

```
```
Node Children

Child: returns the children of the current nodes that have the given tagname

```java
public Node next () {
    while (true) {
        while (index < nodes.size()) {
            Node e = nodes.get(index);
            if (e == null) return null;
            index ++;
            if (e.getNodeName().equals(tagname)) {
                return e;
            }
        }
        return null;
    }
}
```

Cloning the Stream

For each element e, emit e, then end-of-stream (null), then e again.
The first e is consumed by the predicate while the second by the result

```java
class Clone extends Iterator {
    Node node;
    int state;
    public Node next () {
        state = (state+1) % 3;
        switch (state) {
            case 0:
            node = input.next();
            return node;
            case 1:
            return node;
            case 2:
            return node;
        }
    }
}
```

Predicate

Filters out the current nodes that do not satisfy the condition

```java
class Predicate extends Iterator {
    final Iterator condition;
    public Predicate (Iterator condition, Clone input) { ... }
    public Node next () {
        while (true) {
            Node n = condition.next();
            if (n == null)
                return null;
            else continue;
            while (n != null)
                n = condition.next();
                return input.next();
        }
    }
}
```

SAX

DOM requires the entire document to be read before it takes any action.
- Requires a large amount of memory for large XML documents.
- It waits for the document to load in memory before it starts processing.
- For selective queries, most loaded data will not be used.

SAX is a Simple API for XML that allows you to process a document as it's being read.
- The SAX API is event based.
  - The XML parser sends events, such as the start or the end of an element, to an event handler, which processes the information.
Parser Events

- Receive notification of the beginning of a document
  void startDocument()
- Receive notification of the end of a document
  void endDocument()
- Receive notification of the beginning of an element
  void startElement (String namespace,
  String localName,
  String qName,
  Attributes atts )
- Receive notification of the end of an element
  void endElement (String namespace,
  String localName,
  String qName )
- Receive notification of character data
  void characters (char[] text, int start, int length )

SAX Example: a Printer

class Printer extends DefaultHandler {
  public Printer () { super(); }
  public void startDocument () {} 
  public void endDocument () { System.out.println(); } 
  public void startElement ( String uri, String name, 
    String tag, Attributes atts ) { 
    System.out.print("<" + tag + ">"); 
  } 
  public void endElement ( String uri, String name, String tag ) { 
    System.out.print("</" + tag + ">"); 
  } 
  public void characters ( char text[], int start, int length ) { 
    System.out.print(new String(text,start,length)); 
  }
}

Parsing with SAX

import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.*;
import java.io.FileReader;

try {
  SAXParserFactory factory = SAXParserFactory.newInstance();
  factory.setValidating(false);
  factory.setNamespaceAware(false);
  XMLReader xmlReader = factory.newSAXParser().getXMLReader();
  xmlReader.setContentHandler(new Printer());
  xmlReader.parse(file);
  catch (Exception e) {
    throw new Error(e);
  }
}
A Better, Modular Method

- For each XPath query, construct a pipeline to process the SAX stream one-event-at-a-time
- Each handler in the pipeline corresponds to an XPath axis step
- A handler acts as a filter that either
  - blocks a SAX event, or
  - propagates the event to the next pipeline handler
- A handler must have a state of fixed size (independent of stream size)
- See: examples/sax.java

Examples:

```java
new Document("cs.xml",
  new Child("department",
    new Child("gradstudent",
      new Child("name", new Print()))));
```

Example

<table>
<thead>
<tr>
<th>Input Stream</th>
<th>SAX Events</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;department&gt;</code></td>
<td>StartDocument:</td>
</tr>
<tr>
<td><code>&lt;degreename&gt;</code></td>
<td>SE: department</td>
</tr>
<tr>
<td><code>&lt;department&gt;</code></td>
<td>SE: deprename</td>
</tr>
<tr>
<td><code>&lt;gradstudent&gt;</code></td>
<td>C: Computer Science</td>
</tr>
<tr>
<td><code>&lt;name&gt;</code></td>
<td>EE: deprename</td>
</tr>
<tr>
<td><code>Smith</code></td>
<td>SE: deprename</td>
</tr>
<tr>
<td><code>&lt;lastname&gt;</code></td>
<td>SE: deprename</td>
</tr>
<tr>
<td><code>&lt;firstname&gt;</code></td>
<td>C: Smith</td>
</tr>
<tr>
<td><code>John</code></td>
<td>EE: lastname</td>
</tr>
<tr>
<td><code>&lt;lastname&gt;</code></td>
<td>EE: firstname</td>
</tr>
<tr>
<td><code>&lt;firstname&gt;</code></td>
<td>SE: firstname</td>
</tr>
<tr>
<td><code>John</code></td>
<td>C: John</td>
</tr>
<tr>
<td><code>&lt;name&gt;</code></td>
<td>EE: deprename</td>
</tr>
<tr>
<td><code>&lt;gradstudent&gt;</code></td>
<td>EE: name</td>
</tr>
<tr>
<td><code>Smith</code></td>
<td>EE: gradstudent</td>
</tr>
<tr>
<td><code>John</code></td>
<td>EE: department</td>
</tr>
<tr>
<td><code>&lt;department&gt;</code></td>
<td>EndDocument:</td>
</tr>
</tbody>
</table>

Predicates

- Need new events:
  - startPredicate, endPredicate, releasePredicate
- `/department/gradstudent[gpa="3.5"]/name`

```java
XPathHandler n = new Child("name", new Print());
new Document("cs.xml",
  new Child("department",
    new Child("gradstudent",
      new Predicate(1, new Child("gpa",
        new Equals(1, "3.5", n)),
        n))));
```

```java
new Document("cs.xml"/department/gradstudent/predicate[
  gpa = 3.5], name)
```
How Predicate Works

stream A

Predicate

stream B

outcome of = 3.5

.startPredicate

.releasePredicate

.endPredicate

StartPredicate

.releasePredicate

.endPredicate

Predicate Handler

class Predicate extends XPathHandler {
    int level;
    public Predicate ( int predicate_id, XPathHandler condition, XPathHandler next ) {...
        public void startElement ( String nm, String ln, String qp, Attributes a ) {
            if (level+) += 0
                next.startPredicate[predicate_id];
            next.startElement(nm,ln,qp,a);
            condition.startElement(nm,ln,qp,a);
        }
        public void endElement ( String nm, String ln, String qp ) {
            next.endElement(nm,ln,qp);
            condition.endElement(nm,ln,qp);
            if (-level) == 0
                next.endPredicate(predicate_id);
        }
        public void characters ( char[] text, int start, int length ) {
            next.characters(text,start,length);
            condition.characters(text,start,length);
        }
    }
}

StAX

- Unlike SAX, you pull events from document
- Imports:
  import javax.xml.transform.stax.*;
  import javax.xml.stream.*;
- Create a pull parser:
  XMLInputFactory factory = XMLInputFactory.newInstance();
  XMLStreamReader reader = factory.createXMLStreamReader(new FileReader(file));
- Pull the next event: reader.getEventType()
- Type of events:
  1. START_TAG, END_TAG, TEXT
  2. START_DOCUMENT, END_DOCUMENT
- Methods:
  1. Predicates: isStartElement(), isEndElement(), isCharacters(), ...
  2. Accessors: getLocalName(), getText(), ...
- See: examples/stax.java

Better StAX Events

class Attributes {
    public String[] names;
    public String[] values;
}
abstract class Event {
    public String tag;
    public Attributes attributes;
}
class StartTag extends Event { public String tag; 
    public Attributes attributes;
}
class EndTag extends Event { public String tag; 
}
class CData extends Event { public String text; 
}
A stream iterator must conform with the following class:

```
abstract class Iterator {
    Iterator input; // the next iterator in the pipeline
    public Iterator (Iterator input) {
        this.input = input;
    }
    public Event next () {
        return input.next();
    }
    public void skip () {
        input.skip();
    }
}
```

The Child Iterator:

```
class Child extends Iterator {
    String tag;
    short next;
    boolean keep;
    public Event next () {
        while (true) {
            Event t = input.next();
            if (t == null) return null;
            if (t instanceof StartTag) {
                if (next == 1) {
                    keep = tag.equals(((StartTag)t).tag);
                    if (!keep)
                        continue;
                } else if (t instanceof EndTag) {
                    if (--next == 0 & keep) {
                        keep = false;
                        return t;
                    }
                } if (keep)
                    return t;
            }
        }
    }
    return t;
}
```