CS193k, Stanford
Spring, 2000-01

Handout #13
Nick Parlante

XML

XML Introduction

Just a format standard

XML is just a way of describing a bunch of bindings in a textual form. It's a simple thing, but by being standard, it makes many boring incompatibilities go away.

Trying to make things work together more easily -- a data exchange format.

http://www.xml.org/

DTD

DTD -- formal structure description -- meta.

The parser or other tool can formally check that a document meets the DTD structure definition. In theory, just regular code does not need to worry about structure errors -- it's handled by the parser/DTD system.

Backward/Forward/Round-Trip

With intelligent use of XML, a new version of an app will be able to read the docs of the old version -- just don't get confused if certain new nodes are not there

With intelligent use of XML, an old version of an app may be able to read the new docs -- just ignore nodes you don't understand.

Round-trip (this is hard), the versions can read each others docs, and write them out again without affecting the other version's state. This requires the old version ignore the new nodes, yet preserve them and write them back out again after editing.

Strict XML

Bad standards: TIFF, RTF, HTML somewhat-- different vendors implement it different ways, which destroys the network effect advantage of having a standard

XML has learned the lesson: behavior in all cases is defined. Where, in C, the def might say that a behavior in a weird case, like divide by 0, "undefined", XML will say that a correct implementation **must** throw an error and halt.

Nodes

Nodes

Like HTML tags

<dots> ..... </dots>

Can enclose other things with the <node>....</node> form

A node can start and end with one tag like this: <node/>

Attributes

A node can have name/value attributes defined inside its tag
<node foo="bar" pi="3.14">

1. Text Markup

Can have free form text between the <node>...</node> pairs -- like HTML

2. Tree Shape

If just have nodes inside the nodes with no free text, then it's just a big tree.
This is the way I have used XML in programming projects -- save out the internal state as an XML tree

Attributes vs. Child Nodes

Suppose we want to have a "dot" that stores an x and a y
There is **not** wide agreement about which of these is better, but I prefer the "attribute" way for fixed number of children
If a node can have an arbitrary number of children, then certainly the <parent> <child>..</child> <child>..</child> </parent> style is best
Attributes can be used where the number of children is fixed

1. Attributes

XML

```
<dot x="27" y="13">
```

2. Children

XML

```
<dot>
  <x>27</x>
  <y>13</y>
</dot>
```

XML Example

The "Dots" XML format -- a set of (x,y) points
root node : "dots" -- parent of dot nodes
child nodes : "dot" -- each with "x" and "y" attributes

```
<?xml version="1.0" encoding="UTF-8"?>
<dots>
  <dot x="72" y="101" />
  <dot x="170" y="164" />
  <dot x="184" y="158" />
  <dot x="194" y="146" />
  <dot x="191" y="133" />
  <dot x="164" y="84" />
  <dot x="119" y="89" />
</dots>
```

Java XML

JAXP project
- http://java.sun.com/xml/

SAX
- Simple parser

DOM
- Tree of nodes
- Can iterate over the tree to look at the nodes
- Can edit the tree: add/remove nodes

Jar Files
- jaxp.jar and crimson.jar -- the code we're using from Jaxp
- These are in the cs108/jars directory

DOM Document

In memory rep of the whole tree
Has a pointer to the root node
Building the Document tree is a little expensive -- it's a java object equivalent of the whole XML tree

Node

The nodes that make up the XML tree
Nodes contain other nodes -- "children"
Nodes can have attribute/value bindings
There can be free-form text in between nodes (like HTML), but we're not using that feature.

Root

The root node that contains all the content
The root is the one child of the document

DOM Reading

Our technique
- Use the DocumentBuilder.parse() method to read the XML and build the DOM in memory.
- Traverse it and examine the nodes to get the data out

Alternatives
- The SAX interface (below) will show you, one at a time, the nodes of the XML document. It does not build the tree in memory, but it's faster.
- Another technique would be to use the DOM tree as our data model itself, so there is no translation step for reading or writing.

Read DOM Into Memory

```
// The following is the standard incantation to get a Document object
  DocumentBuilderFactory dbf =
      DocumentBuilderFactory.newInstance();

  dbf.setValidating(false);

  DocumentBuilder db = null;
  try {
      db = dbf.newDocumentBuilder();
  } catch (ParserConfigurationException pce) {
      pce.printStackTrace();
  }

// Parse the XML to build the whole doc tree
Document doc  = db.parse(file);
```

DOM Traversal

Element root = doc.getDocumentElement()
NodeList list = root.getElementsByTagName(tag-string)
list.getLength() -- number of children
Element elem = (Element) list.item(i)
elem.getAttribute(attr-string)

Dot example

```
// This parses the XML file and builds the XML doc in memory
XmlDocument doc = XmlDocument.createXmlDocument(in, false);

// Get the root
Element root = doc.getDocumentElement();

// Get all the DOT children
NodeList dots = root.getElementsByTagName("dot");

// Iterate through them
for (int i = 0; i<dots.getLength(); i++) {
   Element dot = (Element) dots.item(i);

   // Get the X and Y attrs out of the dot node
   // Integer.parseInt(dot.getAttribute(X)) --> x
   // Integer.parseInt(dot.getAttribute(Y)) --> y
}
```

DOM Creation Methods

document.createElement(tag-string) -> returns Node
node.appendChild(node)
node.setAttribute(attr-string, value-string)

DOM Writing

Construct the DOM Document tree in memory
Trick: downcast the Document object to an XmlDocument
XmlDocument responds to a write() message where it writes itself out in text form.
Alternatives
- Someday Jaxp may have a better, official way to write out XML, but our trick works for now.
- A faster technique would be to write the XML out just using println(), but then you need to make sure you're writing valid XML -- be careful about <, >, and " in the text

DOM Writing Code

```
        /*
         TRICK: as of JAXP1.1, there is no documented way to write a Document
         object out. However, by digging around a little, I found this
         unsupported way.
        */

        // 1. Cast the doc down to an XmlDocument
        XmlDocument x = (XmlDocument) doc;

        // 2. XmlDocument knows how to write itself out Woo Hoo!
        x.write(out, "UTF-8");
```

SAX Parsing

Faster
Does not build the whole tree
Reads the doc from start to end
Sends notifications at each node

Notifications

startElement(name, attrs) -- <dot>
endElement(name) -- </dot> -- often you can ignore this one
characters() -- called for characters between

State Machine Strategy

Have ivars to keep track of a current state -- what nodes have been started but not ended
Each notification updates the state
Assume the structure is correct -- let the parser notice structural errors for you. In this case, notice that if there were <foo>...</foo> nodes added in the doc, we just ignore them. Also, we don't check that the <dots> node is there.

SAX Example -- Dots

```
// XMLReader.java
import java.io.*;

import org.xml.sax.*;

import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;


import com.sun.java.util.collections.*;

/**
 This is a simple class that can read state out of an XML file
 using a SAX state-machine parser.

 In this case, we support data like this, where the flip
 node switches x,y...

   <?xml version="1.0" encoding="UTF-8"?>

   <dots>
      <dot x="81" y="67" />
      <dot x="175" y="122" />
      <flip>
         <dot x="175" y="122" />
         <dot x="209" y="71" />
      </flip>
      <dot x="209" y="71" />
   </dots>

*/
public class XMLReader extends HandlerBase
{

   public static void main (String argv [])
   {
      if (argv.length != 1) {
         System.err.println ("Usage: cmd filename");
         System.exit (1);
      }


      try {
         XMLReader xr = new XMLReader();
         InputStream in = new BufferedInputStream( new FileInputStream(new
File(argv[0])));
         xr.read(in);
      } catch (Throwable t) {
         t.printStackTrace ();
      }

   }


   //============================================================
   // SAX DocumentHandler methods
   //==============================================================

   public void setDocumentLocator (Locator l)
   {
   }

   public void startDocument ()
   throws SAXException
   {
      //System.out.println("startDocument");
   }

   public void endDocument ()
   throws SAXException
   {
      //System.out.println("startDocument");
   }



   public void read(File file) {
      try {
         InputStream in = new BufferedInputStream( new FileInputStream(file));
         read(in);
      }
      catch (IOException ioe) {
         ioe.printStackTrace();
      }
   }


   /**
    Read the XML in the given file
   */
   public void read(InputStream stream)  {
      try {
         SAXParserFactory factory = SAXParserFactory.newInstance();
         SAXParser saxParser = factory.newSAXParser();
         clear();
         saxParser.parse(stream, this);
      }
      catch (Exception e) {
         e.printStackTrace();
      }
   }


   public XMLReader() {
      clear();
   }


   // State we keep track of -- like a state machine,
   // where startElement() etc. keep getting called
   private int x;
   private int y;
   private boolean flip;
```

```
    public void clear() {
        x = -1;
        y = -1;
        flip = false;
    }


    /**
     Called for each node
     -look at the node name
     -process that node if appropriate
     -or, update our state to affect future calls to startElem()
     or characters()
    */
    public void startElement (String name, AttributeList attrs)
    throws SAXException
    {
        //System.out.println("start element:" + name);
        if (name.equals("dot")) {
            x = Integer.parseInt(attrs.getValue("x"));
            y = Integer.parseInt(attrs.getValue("y"));

            if (flip) {
                int temp = x; x = y; y = temp;
            }

            // do something with our x,y state (could wait for endElement)
            System.out.println(x);
            System.out.println(y);
        }
        else if (name.equals("flip")) {
            flip = true;
        }
    }

    public void endElement (String name)
    throws SAXException
    {
        //System.out.println("end element:" + name);

        if (name.equals("flip")) {
            flip = false;
        }
    }

    // Called for characters between nodes
    public void characters (char buf [], int offset, int len)
    throws SAXException
    {
        //String s = new String(buf, offset, len);
        //s = s.trim();
        //if (!s.equals("")) {
            //System.out.println("characters:" + s);
        //}
    }
}
```