An Introduction to XML and Web Technologies

XML Programming

Anders Møller & Michael I. Schwartzbach

Objectives

- How XML may be manipulated from general-purpose programming languages
- How streaming may be useful for handling large documents

General Purpose XML Programming

- Needed for:
  - *domain-specific* applications
  - implementing *new generic tools*
- Important constituents:
  - *parsing* XML documents into XML trees
  - *navigating* through XML trees
  - *manipulating* XML trees
  - *serializing* XML trees as XML documents

The JDOM Framework

- An implementation of *generic* XML trees in Java
- Nodes are represented as *classes* and *interfaces*
- DOM is a *language-independent* alternative

JDOM Classes and Interfaces

- The abstract class `Content` has subclasses:
  - `Comment`
  - `DocType`
  - `Element`
  - `EntityRef`
  - `ProcessingInstruction`
  - `Text`
- Other classes are `Attribute` and `Document`
- The `Parent` interface describes `Document` and `Element`

A Simple Example

```
int xmlHeight(Element e) {
  java.util.List contents = e.getContent();
  java.util.Iterator i = contents.iterator();
  int max = 0;
  while (i.hasNext()) {
    Object c = i.next();
    int h;
    if (c instanceof Element)
      h = xmlHeight((Element)c);
    else
      h = 1;
    if (h > max)
      max = h;
  }
  return max+1;
}
```


Another Example

```
static void doubleSugar(Document d)
  throws DataConversionException {
  Namespace rcp =
    Namespace.getNamespace("http://www.brics.dk/ixwt/recipes");
  Filter f = new ElementFilter("ingredient",rcp);
  java.util.Iterator i = d.getDescendants(f);
  while (i.hasNext()) {
    Element e = (Element)i.next();
    if (e.getAttributeValue("name").equals("sugar")) {
      double amount = e.getAttribute("amount").getDoubleValue();
      e.setAttribute("amount",new Double(2*amount).toString());
    }
  }
}
```


A Final Example (1/3)

- Modify all elements like

```
<ingredient name="butter" amount="0.25" unit="cup"/>
```

into a more elaborate version:

```
<ingredient name="butter">
  <ingredient name="cream" unit="cup" amount="0.5" />
  <preparation>
    Churn until the cream turns to butter.
  </preparation>
</ingredient>
```

A Final Example (2/3)

```
void makeButter(Element e) throws DataConversionException {
  Namespace rcp =
    Namespace.getNamespace("http://www.brics.dk/ixwt/recipes");
  java.util.ListIterator i = e.getChildren().listIterator();
  while (i.hasNext()) {
    Element c = (Element)i.next();
    if (c.getName().equals("ingredient") &&
        c.getAttributeValue("name").equals("butter")) {
      Element butter = new Element("ingredient",rcp);
      butter.setAttribute("name","butter");
```


A Final Example (3/3)

```
      Element cream = new Element("ingredient",rcp);
      cream.setAttribute("name","cream");
      cream.setAttribute("unit",c.getAttributeValue("unit"));
      double amount = c.getAttribute("amount").getDoubleValue();
      cream.setAttribute("amount",new Double(2*amount).toString());
      butter.addContent(cream);
      Element churn = new Element("preparation",rcp);
      churn.addContent("Churn until the cream turns to butter.");
      butter.addContent(churn);
      i.set((Element)butter);
    } else {
      makeButter(c);
    }
  }
}
```


Parsing and Serializing

```
public class ChangeDescription {
  public static void main(String[] args) {
    try {
      SAXBuilder b = new SAXBuilder();
      Document d = b.build(new File("recipes.xml"));
      Namespace rcp =
        Namespace.getNamespace("http://www.brics.dk/ixwt/recipes");
      d.getRootElement().getChild("description",rcp)
                        .setText("Cool recipes!");
      XMLOutputter outputter = new XMLOutputter();
      outputter.output(d,System.out);
    } catch (Exception e) { e.printStackTrace(); }
  }
}
```


Validation (DTD)

```
public class ValidateDTD {
  public static void main(String[] args) {
    try {
      SAXBuilder b = new SAXBuilder();
      b.setValidation(true);
      String msg = "No errors!";
      try {
        Document d = b.build(new File(args[0]));
      } catch (JDOMParseException e ) {
        msg = e.getMessage();
      }
      System.out.println(msg);
    } catch (Exception e) { e.printStackTrace(); }
  }
}
```

Validation (XML Schema)

```
public class ValidateXMLSchema {
  public static void main(String[] args) {
    try {
      SAXBuilder b = new SAXBuilder();
      b.setValidation(true);
      b.setProperty(
        "http://java.sun.com/xml/jaxp/properties/schemaLanguage",
        "http://www.w3.org/2001/XMLSchema");
      String msg = "No errors!";
      try {
        Document d = b.build(new File(args[0]));
      } catch (JDOMParseException e ) {
        msg = e.getMessage();
      }
      System.out.println(msg);
    } catch (Exception e) { e.printStackTrace(); }
  }
}
```


XPath Evaluation

```
void doubleSugar(Document d) throws JDOMException {
  XPath p = XPath.newInstance("//rcp:ingredient[@name='sugar']");
  p.addNamespace("rcp","http://www.brics.dk/ixwt/recipes");
  java.util.Iterator i = p.selectNodes(d).iterator();
  while (i.hasNext()) {
    Element e = (Element)i.next();
    double amount = e.getAttribute("amount").getDoubleValue();
    e.setAttribute("amount",new Double(2*amount).toString());
  }
}
```

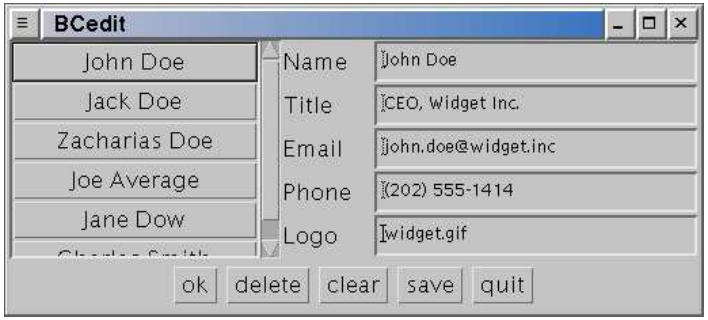

XSLT Transformation

```
public class ApplyXSLT {
  public static void main(String[] args) {
    try {
      SAXBuilder b = new SAXBuilder();
      Document d = b.build(new File(args[0]));
      XSLTransformer t = new XSLTransformer(args[1]);
      Document h = t.transform(d);
      XMLOutputter outputter = new XMLOutputter();
      outputter.output(h,System.out);
    } catch (Exception e) { e.printStackTrace(); }
  }
}
```


Business Cards

```
<cardlist xmlns="http://businesscard.org"
          xmlns:xhtml="http://www.w3.org/1999/xhtml">
  <title>
    <xhtml:h1>My Collection of Business Cards</xhtml:h1>
    containing people from <xhtml:em>Widget Inc.</xhtml:em>
  </title>
  <card>
    <name>John Doe</name>
    <title>CEO, Widget Inc.</title>
    <email>john.doe@widget.com</email>
    <phone>(202) 555-1414</phone>
  </card>
  <card>
    <name>Joe Smith</name>
    <title>Assistant</title>
    <email>thrall@widget.com</email>
  </card>
</cardlist>
```

Business Card Editor



Class Representation

```
class Card {
  public String name,title,email,phone,logo;

  public Card(String name, String title, String email,
              String phone, String logo) {
    this.name=name;
    this.title=title;
    this.email=email;
    this.phone=phone;
    this.logo=logo;
  }
}
```


From JDOM to Classes

```
Vector doc2vector(Document d) {
   Vector v = new Vector();
   Iterator i = d.getRootElement().getChildren().iterator();
   while (i.hasNext()) {
     Element e = (Element)i.next();
     String phone = e.getChildText("phone",b);
     if (phone==null) phone="";
     Element logo = e.getChild("logo",b);
     String uri;
     if (logo==null) uri="";
     else uri=logo.getAttributeValue("uri");
     Card c = new Card(e.getChildText("name",b),
                       e.getChildText("title",b),
                       e.getChildText("email",b),
                       phone, uri);
     v.add(c);
   }
   return v;
 }
```


From Classes to JDOM (1/2)

```
Document vector2doc() {
   Element cardlist = new Element("cardlist");
   for (int i=0; i<cardvector.size(); i++) {
     Card c = (Card)cardvector.elementAt(i);
     if (c!=null) {
       Element card = new Element("card",b);
       Element name = new Element("name",b);
       name.addContent(c.name); card.addContent(name);
       Element title = new Element("title",b);
       title.addContent(c.title); card.addContent(title);
       Element email = new Element("email",b);
       email.addContent(c.email); card.addContent(email);
```

From Classes to JDOM (2/2)

```
    if (!c.phone.equals("")) {
       Element phone = new Element("phone",b);
       phone.addContent(c.phone);
       card.addContent(phone);
     }
     if (!c.logo.equals("")) {
       Element logo = new Element("logo",b);
       logo.setAttribute("uri",c.logo);
       card.addContent(logo);
     }
     cardlist.addContent(card);
   }
  }
  return new Document(cardlist);
}
```


A Little Bit of Code

```
void addCards() {
   cardpanel.removeAll();
   for (int i=0; i<cardvector.size(); i++) {
     Card c = (Card)cardvector.elementAt(i);
     if (c!=null) {
       Button b = new Button(c.name);
       b.setActionCommand(String.valueOf(i));
       b.addActionListener(this);
       cardpanel.add(b);
     }
   }
   this.pack();
 }
```


The Main Application

```
public BCedit(String cardfile) {
   super("BCedit");
   this.cardfile=cardfile;
   try {
     cardvector = doc2vector(
        new SAXBuilder().build(new File(cardfile)));
   } catch (Exception e) { e.printStackTrace(); }
   // initialize the user interface
   ...
 }
```


XML Data Binding

- The methods `doc2vector` and `vector2doc` are tedious to write
- XML *data binding* provides tools to:
  - *map* schemas to class declarations
  - automatically generate *unmarshalling* code
  - automatically generate *marshalling* code
  - automatically generate *validation* code

Binding Compilers

- Which *schemas* are supported?
- *Fixed* or *customizable* binding?
- Does *roundtripping* preserve information?
- What is the support for *validation*?
- Are the generated classes implemented by some generic *framework*?

The JAXB Framework

- It supports most of XML Schema
- The binding is customizable (annotations)
- Roundtripping is almost complete
- Validation is supported during unmarshalling or on demand
- JAXB only specifies the interfaces to the generated classes

Business Card Schema (1/3)

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:b="http://businesscard.org"
        targetNamespace="http://businesscard.org"
        elementFormDefault="qualified">

 <element name="cardlist" type="b:cardlist_type"/>
 <element name="card" type="b:card_type"/>
 <element name="name" type="string"/>
 <element name="email" type="string"/>
 <element name="phone" type="string"/>
 <element name="logo" type="b:logo_type"/>

 <attribute name="uri" type="anyURI"/>
```


Business Card Schema (2/3)

```
 <complexType name="cardlist_type">
  <sequence>
    <element name="title" type="b:cardlist_title_type"/>
    <element ref="b:card" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
 </complexType>

 <complexType name="cardlist_title_type" mixed="true">
  <sequence>
    <any namespace="http://www.w3.org/1999/xhtml"
         minOccurs="0" maxOccurs="unbounded"
         processContents="lax"/>
  </sequence>
</complexType>
```

Business Card Schema (3/3)

```
  <complexType name="card_type">
    <sequence>
      <element ref="b:name"/>
      <element name="title" type="string"/>
      <element ref="b:email"/>
      <element ref="b:phone" minOccurs="0"/>
      <element ref="b:logo" minOccurs="0"/>
    </sequence>
  </complexType>

  <complexType name="logo_type">
    <attribute ref="b:uri" use="required"/>
  </complexType>
</schema>
```


The org.businesscard Package

- The binding compiler generates :
  - *Cardlist*, *CardlistType*
  - CardlistImpl, CardlistTypeImpl
  - ...
  - *Logo*, *LogoType*
  - LogoImpl, LogoTypeImpl
  - ObjectFactory
- The Title element is not a class, since it is declared as a local element.

The *CardType* Interface

```
public interface CardType {
    java.lang.String getEmail();
    void setEmail(java.lang.String value);
    org.businesscard.LogoType getLogo();
    void setLogo(org.businesscard.LogoType value);
    java.lang.String getTitle();
    void setTitle(java.lang.String value);
    java.lang.String getName();
    void setName(java.lang.String value);
    java.lang.String getPhone();
    void setPhone(java.lang.String value);
}
```


A Little Bit of Code

```
void addCards() {
   cardpanel.removeAll();
   Iterator i = cardlist.iterator();
   int j = 0;
   while (i.hasNext()) {
     Card c = (Card)i.next();
     Button b = new Button(c.getName());
     b.setActionCommand(String.valueOf(j++));
     b.addActionListener(this);
     cardpanel.add(b);
   }
   this.pack();
 }
```

The Main Application

```
public BCedit(String cardfile) {
   super("BCedit");
   this.cardfile=cardfile;
   try {
     jc = JAXBContext.newInstance("org.businesscard");
     Unmarshaller u = jc.createUnmarshaller();
     cl = (Cardlist)u.unmarshal(
             new FileInputStream(cardfile)
          );
     cardlist = cl.getCard();
   } catch (Exception e) { e.printStackTrace(); }
   // initialize the user interface
   ...
}
```


Streaming XML

- JDOM and JAXB keeps the entire XML tree in memory
- Huge documents can only be *streamed*:
  - movies on the Internet
  - Unix file commands using pipes
- What is streaming for XML documents?
- The SAX framework has the answer...

Parsing Events

- View the XML document as a stream of *events*:
  - the document starts
  - a start tag is encountered
  - an end tag is encountered
  - a namespace declaration is seen
  - some whitespace is seen
  - character data is encountered
  - the document ends
- The SAX tool *observes* these events
- It reacts by *calling* corresponding methods specified by the programmer

Tracing All Events (1/4)

```
public class Trace extends DefaultHandler {
  int indent = 0;

  void printIndent() {
    for (int i=0; i<indent; i++) System.out.print("-");
  }

  public void startDocument() {
    System.out.println("start document");
  }

  public void endDocument() {
    System.out.println("end document");
  }
```

Tracing All Events (2/4)

```
  public void startElement(String uri, String localName,
                           String qName, Attributes atts) {
    printIndent();
    System.out.println("start element: " + qName);
    indent++;
  }

  public void endElement(String uri, String localName,
                         String qName) {
    indent--;
    printIndent();
    System.out.println("end element: " + qName);
  }
```


Tracing All Events (3/4)

```
public void ignorableWhitespace(char[] ch, int start, int length) {
  printIndent();
   System.out.println("whitespace, length " + length);
 }

 public void processingInstruction(String target, String data) {
   printIndent();
   System.out.println("processing instruction: " + target);
 }

 public void characters(char[] ch, int start, int length){
   printIndent();
   System.out.println("character data, length " + length);
 }
```


Tracing All Events (4/4)

```
 public static void main(String[] args) {
    try {
      Trace tracer = new Trace();
      XMLReader reader = XMLReaderFactory.createXMLReader();
      reader.setContentHandler(tracer);
      reader.parse(args[0]);
    } catch (Exception e) { e.printStackTrace(); }
  }
}
```


Output for the Recipe Collection

```
start document
start element: rcp:collection
-character data, length 3
-start element: rcp:description
--character data, length 44
--character data, length 3
-end element: rcp:description
-character data, length 3
-start element: rcp:recipe
--character data, length 5
--start element: rcp:title
---character data, length 42
...
--start element: rcp:nutrition
--end element: rcp:nutrition
--character data, length 3
-end element: rcp:recipe
-character data, length 1
end element: rcp:collection
end document
```

A Simple Streaming Example (1/2)

```
public class Height extends DefaultHandler {
  int h = -1;
  int max = 0;

  public void startElement(String uri, String localName,
                           String qName, Attributes atts) {
    h++; if (h > max) max = h;
  }

  public void endElement(String uri, String localName,
                         String qName) {
    h--;
  }

  public void characters(char[] ch, int start, int length){
    if (h+1 > max) max = h+1;
  }
```


A Simple Streaming Example (2/2)

```
  public static void main(String[] args) {
    try {
      Height handler = new Height();
      XMLReader reader = XMLReaderFactory.createXMLReader();
      reader.setContentHandler(handler);
      reader.parse(args[0]);
      System.out.println(handler.max);
    } catch (Exception e) { e.printStackTrace(); }
  }
}
```


Comments on The Example

- This version is less intuitive (stack-like style)
- The JDOM version:
  `java.lang.OutOfMemoryError`
  on 18MB document
- The SAX version handles 1.2GB in 51 seconds

SAX May Emulate JDOM (1/2)

```
public void startElement(String uri, String localName,
                         String qName, Attributes atts) {
  if (localName.equals("card")) card = new Element("card",b);
  else if (localName.equals("name"))
    field = new Element("name",b);
  else if (localName.equals("title"))
    field = new Element("title",b);
  else if (localName.equals("email"))
    field = new Element("email",b);
  else if (localName.equals("phone"))
    field = new Element("phone",b);
  else if (localName.equals("logo")) {
    field = new Element("logo",b);
    field.setAttribute("uri",atts.getValue("","uri"));
    }
  }
```

SAX May Emulate JDOM (2/2)

```
public void endElement(String uri, String localName,
                       String qName) {
  if (localName.equals("card")) contents.add(card);
  else if (localName.equals("cardlist")) {
    Element cardlist = new Element("cardlist",b);
    cardlist.setContent(contents);
    doc = new Document(cardlist);
  } else {
    card.addContent(field);
    field = null;
  }
}

public void characters(char[] ch, int start, int length) {
  if (field!=null)
    field.addContent(new String(ch,start,length));
}
```


Using Contextual Information

- Check forms beyond W3C validator:
  - that all form `input` tags are inside `form` tags
  - that all `form` tags have distinct `name` attributes
  - that `form` tags are not nested
- This requires us to keep information about the *context* of the current parsing event

Contextual Information in SAX (1/3)

```
public class CheckForms extends DefaultHandler {
  int formheight = 0;
  HashSet formnames = new HashSet();

  Locator locator;
  public void setDocumentLocator(Locator locator) {
    this.locator = locator;
  }

  void report(String s) {
    System.out.print(locator.getLineNumber());
    System.out.print(":");
    System.out.print(locator.getColumnNumber());
    System.out.println(" ---"+s);
  }
```


Contextual Information in SAX (2/3)

```
public void startElement(String uri, String localName,
                         String qName, Attributes atts) {
   if (uri.equals("http://www.w3.org/1999/xhtml")) {
     if (localName.equals("form")) {
       if (formheight > 0) report("nested forms");
       String name = atts.getValue("","name");
       if (formnames.contains(name))
         report("duplicate form name");
       else
         formnames.add(name);
       formheight++;
     } else
     if (localName.equals("input") ||
         localName.equals("select") ||
         localName.equals("textarea"))
       if (formheight==0) report("form field outside form");
   }
 }
```

Contextual Information in SAX (3/3)

```
public void endElement(String uri, String localName,
                       String qName) {
   if (uri.equals("http://www.w3.org/1999/xhtml"))
     if (localName.equals("form"))
       formheight--;
 }

 public static void main(String[] args) {
   try {
     CheckForms handler = new CheckForms();
     XMLReader reader = XMLReaderFactory.createXMLReader();
     reader.setContentHandler(handler);
     reader.parse(args[0]);
   } catch (Exception e) { e.printStackTrace(); }
 }
}
```


SAX Filters

- A SAX application may be turned into a *filter*
- Filters may be *composed* (as with pipes)
- A filter is an event handler that may pass events along in the chain

A SAX Filter Example (1/4)

- A filter to remove processing instructions:

```
class PIFilter extends XMLFilterImpl {
  public void processingInstruction(String target, String data)
  throws SAXException {}
}
```


A SAX Filter Example (2/4)

- A filter to create unique `id` attributes:

```
class IDFilter extends XMLFilterImpl {
  int id = 0;
  public void startElement(String uri, String localName,
                           String qName, Attributes atts)
    throws SAXException {
    AttributesImpl idatts = new AttributesImpl(atts);
    idatts.addAttribute("","id","id","ID",
                        new Integer(id++).toString());
    super.startElement(uri,localName,qName,idatts);
  }
}
```

A SAX Filter Example (3/4)

- A filter to count characters:

```
class CountFilter extends XMLFilterImpl {
  public int count = 0;
  public void characters(char[] ch, int start, int length)
    throws SAXException {
    count = count+length;
    super.characters(ch,start,length);
  }
}
```


A SAX Filter Example (4/4)

```
public class FilterTest {
  public static void main(String[] args) {
    try {
      FilterTest handler = new FilterTest();
      XMLReader reader = XMLReaderFactory.createXMLReader();
      PIFilter pi = new PIFilter();
      pi.setParent(reader);
      IDFilter id = new IDFilter();
      id.setParent(pi);
      CountFilter count = new CountFilter();
      count.setParent(id);
      count.parse(args[0]);
      System.out.println(count.count);
    } catch (Exception e) { e.printStackTrace(); }
  }
}
```


Pull vs. Push

- SAX is known as a **push** framework
  - the parser has the initivative
  - the programmer must react to events
- An alternative is a **pull** framework
  - the programmer has the initiative
  - the parser must react to requests
- XML Pull is an example of a pull framework

Contextual Information in XMLPull (1/3)

```
public class CheckForms2 {
  static void report(XmlPullParser xpp, String s) {
    System.out.print(xpp.getLineNumber());
    System.out.print(":");
    System.out.print(xpp.getColumnNumber());
    System.out.println(" ---"+s);
  }

  public static void main (String args[])
    throws XmlPullParserException, IOException {
    XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
    factory.setNamespaceAware(true);
    factory.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, true);

    XmlPullParser xpp = factory.newPullParser();

    int formheight = 0;
    HashSet formnames = new HashSet();
```

Contextual Information in XMLPull (2/3)

```
xpp.setInput(new FileReader(args[0]));
int eventType = xpp.getEventType();
while (eventType!=XmlPullParser.END_DOCUMENT) {
  if (eventType==XmlPullParser.START_TAG) {
    if (xpp.getNamespace().equals("http://www.w3.org/1999/xhtml")
        && xpp.getName().equals("form")) {
      if (formheight>0)
        report(xpp,"nested forms");
      String name = xpp.getAttributeValue("","name");
      if (formnames.contains(name))
        report(xpp,"duplicate form name");
      else
        formnames.add(name);
      formheight++;
    } else if (xpp.getName().equals("input") ||
               xpp.getName().equals("select") ||
               xpp.getName().equals("textarea"))
      if (formheight==0)
        report(xpp,"form field outside form");
  } }
```


Contextual Information in XMLPull (3/3)

```
      else if (eventType==XmlPullParser.END_TAG) {
        if (xpp.getNamespace().equals("http://www.w3.org/1999/xhtml")
            && xpp.getName().equals("form"))
          formheight--;
      }
      eventType = xpp.next();
    }
  }
}
```


Using a Pull Parser

- Not that different from the push version
- More direct programming style
- Smaller memory footprint
- Pipelining with filter chains is not available
  (but may be simulated in languages with higher-order functions)

Streaming Transformations

- SAX allows the programming of streaming applications "by hand"
- XSLT allows high-level programming of applications
- A broad spectrum of these could be streamed
- But XSLT does not allow streaming...

- Solution: use a domain-specific language for streaming transformations

STX

- STX is a variation of XSLT suitable for streaming
  - some features are not allowed
  - but every STX application can be streamed
- The differences reflect necessary limitations in the control flow

Similarities with XSLT

- `template`
- `copy`
- `value-of`
- `if`
- `else`
- `choose`
- `when`
- `otherwise`
- `text`
- `element`
- `attribute`
- `variable`
- `param`
- `with-param`
- Most XSLT functions

Differences with XSLT

- `apply-templates` is the main problem:
  - allows processing to continue anywhere in the tree
  - requires moving back and forth in the input file
  - or storing the whole document
- **mutable** variables to accumulate information

STXPath

- A subset of XPath 2.0 used by STX
- STXPath expressions:
  - look like restricted XPath 2.0 expressions
  - evaluate to sequences of nodes and atomic values
  - but they have a **different** semantics

STXPath Syntax

- Must use abbreviated XPath 2.0 syntax
- The axes `following` and `preceding` are not available
- Extra node tests: `cdata()` and `doctype()`

STXPath Semantics

- Evaluate the corresponding XPath 2.0 expression
- Restrict the result to those nodes that are on the `ancestor` axis
-
```
<A>
  <B/>
  <C><D/></C>
</A>
```
- Evaluate `count(//B)` with `D` as the context node
- With XPath the result is `1`
- With STXPath the result is `0`

Transformation Sheets

- STX use `transform` instead of `stylesheet`
- `apply-templates` is not allowed
- Processing is defined by:
  - `process-children`
  - `process-siblings`
  - `process-self`
- Only a single occurrence of `process-children` is allowed in each template (to enable streaming)

A Simple STX Example

- Extract comments from recipes:

```
<stx:transform xmlns:stx="http://stx.sourceforge.net/2002/ns"
               version="1.0"
               xmlns:rcp="http://www.brics.dk/ixwt/recipes">

  <stx:template match="rcp:collection">
    <comments>
      <stx:process-children/>
    </comments>
  </stx:template>

  <stx:template match="rcp:comment">
     <comment><stx:value-of select="."/></comment>
  </stx:template>
</stx:transform>
```

SAX Version (1/2)

```
public class ExtractComments extends
  DefaultHandler {
   bool chars = true;

   public void startElement(String uri,
   String localName,
                              String
   qName, Attributes atts) {
      if
   (uri.equals("http://www.brics.dk/ixwt
   /recipes")) {
         if
```


SAX Version (2/2)

```
  public void characters(char[] ch, int start, int length) {
    if (chars)
      System.out.print(new String(ch, start, length));
  }

  public void endElement(String uri, String localName,
                                      String qName) {
    if (uri.equals("http://www.brics.dk/ixwt/recipes")) {
      if (localName.equals("collection"))
        System.out.print("</comments>");
      if (localName.equals("comment")) {
        System.out.print("</comment>");
        chars = false;
      }
    }
  }
}
```


The Ancestor Stack

```
<stx:transform xmlns:stx="http://stx.sourceforge.net/2002/ns"
               version="1.0">
  <stx:template match="*">
    <stx:message select="concat(count(//*),' ',local-name())"/>
    <stx:process-children/>
  </stx:template>
</stx:transform>
```

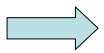
```
<A>
 <B/>
 <B><C/></B>
 <A/>
 <B><A><C/></A></B>
</A>
```

```
1 A
2 B
2 B
3 C
2 A
2 B
3 A
4 C
```


Using `process-siblings`

```
<stx:transform xmlns:stx="http://stx.sourceforge.net/2002/ns"
               version="1.0">
  <stx:template match="*">
    <stx:copy>
      <stx:process-children/>
      <stx:process-siblings/>
    </stx:copy>
  </stx:template>
</stx:transform>
```

```
<a>
  <b><c/></b>
  <d><e/></d>
</a>
```



```
<a>
  <b>
    <c/>
    <d><e/></d>
  </b>
</a>
```

Mutable Variables

```
<stx:transform xmlns:stx="http://stx.sourceforge.net/2002/ns"
               version="1.0"
               xmlns:rcp="http://www.brics.dk/ixwt/recipes">
  <stx:variable name="depth" select="0"/>
  <stx:variable name="maxdepth" select="0"/>

  <stx:template match="rcp:collection">
     <stx:process-children/>
     <maxdepth><stx:value-of select="$maxdepth"/></maxdepth>
  </stx:template>

  <stx:template match="rcp:ingredient">
    <stx:assign name="depth" select="$depth + 1"/>
    <stx:if test="$depth > $maxdepth">
      <stx:assign name="maxdepth" select="$depth"/>
    </stx:if>
    <stx:process-children/>
    <stx:assign name="depth" select="$depth - 1"/>
  </stx:template>
</stx:transform>
```


STX Version of `CheckForms` (1/2)

```
<stx:transform xmlns:stx="http://stx.sourceforge.net/2002/ns"
               version="1.0"
               xmlns:xhtml="http://www.w3.org/1999/xhtml">
  <stx:variable name="formheight" select="0"/>
  <stx:variable name="formnames" select="'#'"/>

  <stx:template match="xhtml:form">
    <stx:if test="$formheight&gt;0">
      <stx:message select="'nested forms'"/>
    </stx:if>
    <stx:if test="contains($formnames,concat('#',@name,'#'))">
      <stx:message select="'duplicate form name'"/>
    </stx:if>
    <stx:assign name="formheight" select="$formheight + 1"/>
    <stx:assign name="formnames"
                select="concat($formnames,@name,'#')"/>
    <stx:process-children/>
    <stx:assign name="formheight" select="$formheight - 1"/>
  </stx:template>
```


STX Version of `CheckForms` (2/2)

```
  <stx:template match="xhtml:input|xhtml:select|xhtml:textarea">
    <stx:if test="$formheight=0">
      <stx:message select="'form field outside form'"/>
    </stx:if>
    <stx:process-children/>
  </stx:template>

</stx:transform>
```


Groups (1/2)

```
<stx:transform xmlns:stx="http://stx.sourceforge.net/2002/ns"
               version="1.0"
               strip-space="yes">
  <stx:template match="person">
     <person><stx:process-children/></person>
  </stx:template>

  <stx:template match="email">
    <emails><stx:process-self group="foo"/></emails>
  </stx:template>
```

```
<person>
  <email/><email/><email/>
  <phone/><phone/>
</person>
```

```
<person>
  <emails>
    <email/><email/><email/>
  </emails>
  <phone/><phone/>
</person>
```

Groups (2/2)

```
  <stx:group name="foo">
    <stx:template match="email">
      <email/>
      <stx:process-siblings while="email" group="foo"/>
    </stx:template>
  </stx:group>

  <stx:template match="phone">
    <phone/>
  </stx:template>
</stx:transform>
```

```
<person>
  <email/><email/><email/>
  <phone/><phone/>
</person>
```

```
<person>
  <emails>
    <email/><email/><email/>
  </emails>
  <phone/><phone/>
</person>
```


Limitations of Streaming

- Something we will never write with STX:

```
<xsl:stylesheet version="2.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template name="mirror" match="/|@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*"/>
      <xsl:apply-templates select="reverse(node())"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```


STX for Recipes (1/7)

```
<stx:transform xmlns:stx="http://stx.sourceforge.net/2002/ns"
               version="1.0"
               xmlns:rcp="http://www.brics.dk/ixwt/recipes"
               xmlns="http://www.w3.org/1999/xhtml"
               strip-space="yes">

  <stx:template match="rcp:collection">
    <html>
      <stx:process-children/>
    </html>
  </stx:template>

  <stx:template match="rcp:description">
    <head>
      <title><stx:value-of select="."/></title>
      <link href="style.css" rel="stylesheet" type="text/css"/>
    </head>
  </stx:template>
```


STX for Recipes (2/7)

```
  <stx:template match="rcp:recipe">
    <body>
      <table border="1">
        <stx:process-self group="outer"/>
      </table>
    </body>
  </stx:template>

  <stx:group name="outer">
    <stx:template match="rcp:description">
      <tr>
        <td><stx:value-of select="."/></td>
      </tr>
    </stx:template>
```

STX for Recipes (3/7)

```
<stx:template match="rcp:recipe">
  <tr>
    <td>
      <stx:process-children/>
    </td>
  </tr>
</stx:template>

<stx:template match="rcp:title">
  <h1><stx:value-of select="."/></h1>
</stx:template>

<stx:template match="rcp:date">
  <i><stx:value-of select="."/></i>
</stx:template>
```


STX for Recipes (4/7)

```
<stx:template match="rcp:ingredient" >
  <ul><stx:process-self group="inner"/></ul>
</stx:template>

<stx:template match="rcp:preparation">
  <ol><stx:process-children/></ol>
</stx:template>

<stx:template match="rcp:step">
  <li><stx:value-of select="."/></li>
</stx:template>

<stx:template match="rcp:comment">
  <ul>
    <li type="square"><stx:value-of select="."/></li>
  </ul>
</stx:template>
```


STX for Recipes (5/7)

```
<stx:template match="rcp:nutrition">
  <table border="2">
    <tr>
      <th>Calories</th><th>Fat</th>
      <th>Carbohydrates</th><th>Protein</th>
      <stx:if test="@alcohol"><th>Alcohol</th></stx:if>
    </tr>
    <tr>
      <td align="right"><stx:value-of select="@calories"/></td>
      <td align="right"><stx:value-of select="@fat"/></td>
      <td align="right"><stx:value-of select="@carbohydrates"/></td>
      <td align="right"><stx:value-of select="@protein"/></td>
      <stx:if test="@alcohol">
        <td align="right"><stx:value-of select="@alcohol"/></td>
      </stx:if>
    </tr>
  </table>
</stx:template>
</stx:group>
```


STX for Recipes (6/7)

```
<stx:group name="inner">
  <stx:template match="rcp:ingredient">
    <stx:choose>
      <stx:when test="@amount">
        <li>
          <stx:if test="@amount!='*'">
            <stx:value-of select="@amount"/>
            <stx:text> </stx:text>
            <stx:if test="@unit">
              <stx:value-of select="@unit"/>
              <stx:if test="number(@amount)>number(1)">
                <stx:text>s</stx:text>
              </stx:if>
              <stx:text> of </stx:text>
            </stx:if>
          </stx:if>
          <stx:text> </stx:text>
          <stx:value-of select="@name"/>
        </li>
      </stx:when>
```

STX for Recipes (7/7)

```
        <stx:otherwise>
          <li><stx:value-of select="@name"/></li>
          <stx:process-children group="outer"/>
        </stx:otherwise>
      </stx:choose>
      <stx:process-siblings while="rcp:ingredient" group="inner"/>
    </stx:template>
  </stx:group>
</stx:transform>
```


XML in Programming Languages

- SAX: programmers react to parsing events
- JDOM: a general data structure for XML trees
- JAXB: a specific data structure for XML trees

- These approaches are convenient
- But no compile-time guarantees:
  - about validity of the constructed XML (JDOM, JAXB)
  - well-formedness of the constructed XML (SAX)

Type-Safe XML Programming Languages

- With XML schemas as types
- Type-checking now guarantees validity

- An active research area

XDuce

- A first-order functional language
- XML trees are native values
- Regular expression types (generalized DTDs)

- Arguments and results are explicitly typed
- Type inference for pattern variables
- Compile-time type checking guarantees:
  - XML navigation is safe
  - generated XML is valid

XDuce Types for Recipes (1/2)

```
namespace rcp = "http://www.brics.dk/ixwt/recipes"

type Collection = rcp:collection[Description,Recipe*]
type Description = rcp:description[String]
type Recipe = rcp:recipe[@id[String]?,
                         Title,
                         Date,
                         Ingredient*,
                         Preparation,
                         Comment?,
                         Nutrition,
                         Related*]
type Title = rcp:title[String]
type Date = rcp:date[String]
```


XDuce Types for Recipes (2/2)

```
type Ingredient = rcp:ingredient[@name[String],
                                 @amount[String]?,
                                 @unit[String]?,
                                 (Ingredient*,Preparation)?]
type Preparation = rcp:preparation[Step*]
type Step = rcp:step[String]
type Comment = rcp:comment[String]
type Nutrition = rcp:nutrition[@calories[String],
                               @carbohydrates[String],
                               @fat[String],
                               @protein[String],
                               @alcohol[String]?]
type Related = rcp:related[@ref[String],String]
```


XDuce Types of Nutrition Tables

```
type NutritionTable = nutrition[Dish*]
type Dish = dish[@name[String],
                 @calories[String],
                 @fat[String],
                 @carbohydrates[String],
                 @protein[String],
                 @alcohol[String]]
```


From Recipes to Tables (1/3)

```
fun extractCollection(val c as Collection) : NutritionTable =
  match c with
    rcp:collection[Description, val rs]
      -> nutrition[extractRecipes(rs)]

fun extractRecipes(val rs as Recipe*) : Dish* =
  match rs with
    rcp:recipe[@..,
               rcp:title[val t],
               Date,
               Ingredient*,
               Preparation,
               Comment?,
               val n as Nutrition,
               Related*], val rest
    -> extractNutrition(t,n), extractRecipes(rest)
  | () -> ()
```

From Recipes to Tables (2/3)

```
fun extractNutrition(val t as String, val n as Nutrition) : Dish =
  match n with
    rcp:nutrition[@calories[val calories],
                  @carbohydrates[val carbohydrates],
                  @fat[val fat],
                  @protein[val protein],
                  @alcohol[val alcohol]]
    -> dish[@name[t],
            @calories[calories],
            @carbohydrates[carbohydrates],
            @fat[fat],
            @protein[protein],
            @alcohol[alcohol]]
```


From Recipes to Tables (3/3)

```
  | rcp:nutrition[@calories[val calories],
                  @carbohydrates[val carbohydrates],
                  @fat[val fat],
                  @protein[val protein]]
    -> dish[@name[t],
            @calories[calories],
            @carbohydrates[carbohydrates],
            @fat[fat],
            @protein[protein],
            @alcohol["0%"]]


let val collection = validate load_xml("recipes.xml") with Collection
let val _ = print(extractCollection(collection))
```


XDuce Guarantees

- The XDuce type checker determines that:
  - every function returns a valid value
  - every function argument is a valid value
  - every `match` has an exhaustive collection of patterns
  - every pattern matches some value
- Clearly, this will eliminate many potential errors

XACT

- A Java framework (like JDOM) but:
  - it is based on **immutable** templates, which are sequences of XML trees containing named gaps
  - XML trees are constructed by plugging gaps
  - it has syntactic sugar for template constants
  - XML is navigated using XPath
  - an analyzer can a compile-time guarantee that an XML expression is valid according to a given DTD

Business Cards to Phone Lists (1/2)

```
import dk.brics.xact.*;
import java.io.*;

public class PhoneList {
  public static void main(String[] args) throws XactException {
    String[] map = {"c", "http://businesscard.org",
                    "h", "http://www.w3.org/1999/xhtml"};
    XML.setNamespaceMap(map);

    XML wrapper = [[<h:html>
                      <h:head>
                        <h:title><[TITLE]></h:title>
                      </h:head>
                      <h:body>
                        <h:h1><[TITLE]></h:h1>
                        <[MAIN]>
                      </h:body>
                    </h:html>]];
```


Business Cards to Phone Lists (2/2)

```
    XML cardlist = XML.get("file:cards.xml",
                           "file:businesscards.dtd",
                           "http://businesscard.org");
    XML x = wrapper.plug("TITLE", "My Phone List")
                   .plug("MAIN", [[<h:ul><[CARDS]></h:ul>]]);


    XMLIterator i = cardlist.select("//c:card[c:phone]").iterator();
    while (i.hasNext()) {
      XML card = i.next();
      x = x.plug("CARDS",
                 [[<h:li>
                     <h:b><{card.select("c:name/text()")}></h:b>,
                     phone: <{card.select("c:phone/text()")}>
                   </h:li>
                   <[CARDS]>]]);
    }
    System.out.println(x);
  }
}
```


XML API

- `constant(s)` build a template constant from s
- `x.plug(g,y)` plugs the gap `g` with `y`
- `x.select(p)` returns a template containing the sequence targets of the XPath expression `p`
- `x.gapify(p,g)` replaces the targets of `p` with gaps named `g`
- `get(u,d,n)` parses a template from a URL with a DTD and a namespace
- `x.analyze(d,n)` guarantees at compile-time that `x` is valid given a DTD and a namespace

A Highly Structured Recipe

```
<rcp:recipe id="117">
  <rcp:title>Fried Eggs with Bacon</rcp:title>
  <rcp:date>Fri, 10 Nov 2004</rcp:date>
  <rcp:ingredient name="fried eggs">
    <rcp:ingredient name="egg" amount="2"/>
    <rcp:preparation>
       <rcp:step>Break the eggs into a bowl.</rcp:step>
       <rcp:step>Fry until ready.</rcp:step>
    </rcp:preparation>
  </rcp:ingredient>
  <rcp:ingredient name="bacon" amount="3" unit="strip"/>
  <rcp:preparation>
    <rcp:step>Fry the bacon until crispy.</rcp:step>
    <rcp:step>Serve with the eggs.</rcp:step>
  </rcp:preparation>
  <rcp:nutrition calories="517"
                 fat="64%" carbohydrates="0%" protein="0%"/>
</rcp:recipe>
```

A Flattened Recipe

```
<rcp:recipe id="117">
  <rcp:title>Fried Eggs with Bacon</rcp:title>
  <rcp:date>Fri, 10 Nov 2004</rcp:date>
  <rcp:ingredient name="egg" amount="2"/>
  <rcp:ingredient name="bacon" amount="3" unit="strip"/>
  <rcp:preparation>
    <rcp:step>Break the eggs into a bowl.</rcp:step>
    <rcp:step>Fry until ready.</rcp:step>
    <rcp:step>Fry the bacon until crispy.</rcp:step>
    <rcp:step>Serve with the eggs.</rcp:step>
  </rcp:preparation>
  <rcp:nutrition calories="517"
                 fat="64%" carbohydrates="0%" protein="36%"/>
</rcp:recipe>
```


A Recipe Flattener in XACT (1/2)

```
public class Flatten {
  static final String rcp = "http://www.brics.dk/ixwt/recipes";
  static final String[] map = { "rcp", rcp };

  static { XML.setNamespaceMap(map); }

  public static void main(String[] args) throws XactException {
    XML collection = XML.get("file:recipes.xml",
                             "file:recipes.dtd", rcp);
    XML recipes = collection.select("//rcp:recipe");
    XML result = [[<rcp:collection>
                     <{collection.select("rcp:description")}>
                     <[MORE]>
                   </rcp:collection>]];
```


A Recipe Flattener in XACT (2/2)

```
    XMLIterator i = recipes.iterator();
    while (i.hasNext()) {
      XML r = i.next();
      result = result.plug("MORE",
        [[<rcp:recipe>
            <{r.select("rcp:title|rcp:date")}>
            <{r.select("//rcp:ingredient[@amount]")}>
            <rcp:preparation>
              <{r.select("//rcp:step")}>
            </rcp:preparation>
            <{r.select("rcp:comment|rcp:nutrition|rcp:related")}>
          </rcp:recipe>
          <[MORE]>]]);
    }
    result.analyze("file:recipes.dtd", rcp);
    System.out.println(result);
  }
}
```


An Error

```
        <rcp:ingredient>
          <{r.select("rcp:title|rcp:date")}>
          <{r.select("//rcp:ingredient[@amount]")}>
          <rcp:preparation>
            <{r.select("//rcp:step")}>
          </rcp:preparation>
          <{r.select("rcp:comment|rcp:nutrition|rcp:related")}>
        </rcp:ingredient>
```

Caught at Compile-Time

```
*** Invalid XML at line 31
 sub-element 'rcp:ingredient' of element 'rcp:collection' not declared
 required attribute 'name' missing in element 'rcp:ingredient'
 sub-element 'rcp:title' of element 'rcp:ingredient' not declared
 sub-element 'rcp:related' of element 'rcp:ingredient' not declared
 sub-element 'rcp:nutrition' of element 'rcp:ingredient' not declared
 sub-element 'rcp:date' of element 'rcp:ingredient' not declared
```


Essential Online Resources

- http://www.jdom.org/
- http://java.sun.com/xml/jaxp/
- http://java.sun.com/xml/jaxb/
- http://www.saxproject.org/