CSE@UTA

XML Query Languages

© Leonidas Fegaras
University of Texas at Arlington

CSE@UTA

Query Languages for XML

- Need a language for XML data for
  - extracting fragments (querying)
  - restructuring (data transformation)
  - integrating (eg, combining multiple XML documents)
  - browsing
  - presentation (eg, from XML to HTML)
- We will first learn XPath
  - used in extracting fragments from a single document
  - many XML query languages are based on XPath
- We will then learn XSLT
  - for extracting, restructuring, and presentation over a single document
- We will focus later on XQuery
  - a full-fledged query language
  - much like SQL

CSE@UTA

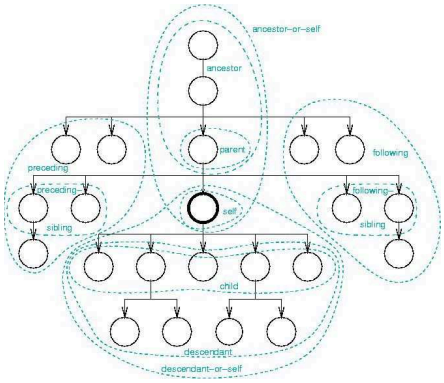
XPath

- Describes a single navigation path in an XML document
- Selects a sequence of nodes reachable by the path
  - the order of nodes is the *document order* (which is the preorder of the XML tree: every node occurs before its children)
- Main construct: *axis navigation*
- The / step returns the document root (the entire document)
- An XPath consists of one or more navigation steps separated by /
- A navigation step is a triplet
  axis :: node-test [predicate]*
- Each navigation path is evaluated relative to a **context node**
- Examples:
  `/ child::bib / descendant::author`
  `/ descendant::book [ child::author / child::name = 'Smith' ] / child::title`
- Most people use shorthands
  `/bib//author`
  `//book[author/name='Smith']/title`

CSE@UTA

Axis

- Forward axes
  - child
  - descendant
  - attribute
  - self
  - descendant-or-self
  - following-sibling
  - following
- Reverse axes
  - parent
  - ancestor
  - preceding-sibling
  - preceding
  - ancestor-or-self



Node Test

person	any element node whose name is person
*	any element node regardless of its name
@price	any attribute whose name is price
@*	any attribute, regardless of its name
node()	any node
text()	any text node
string()	the text content of the node at any depth
element()	any element node
element(person)	any element node whose tagname is person
element(person, surgeon)	any element node whose tagname is person, and whose type annotation is surgeon
element(*, surgeon)	any element node whose type annotation is surgeon, regardless of its name
attribute()	any attribute node
attribute(price)	any attribute whose name is price
attribute(*, xs:decimal)	any attribute whose type annotation is xs:decimal, regardless of its name.

Abbreviated Syntax

- The attribute axis child:: can be omitted
  - section/para is an abbreviation for child::section/child::para,
  - section/@id is an abbreviation for child::section/attribute::id
- … unless the axis step contains an attribute test; then the default axis is attribute
  - section/attribute(id) is shorthand for child::section/attribute::attribute(id)
- The attribute axis attribute:: can be abbreviated by @
  - para[@type="warning"] is shorthand for child::para[attribute::type="warning"]
- // is replaced by /descendant-or-self::node()/
  - div//para is shorthand for child::div/descendant-or-self::node()/child::para, or better child::div/descendant::para
- .. is short for parent::node()
  - ../title is short for parent::node()/child::title

Most Common Steps

XPath step	*full XPath syntax*
/tagname	/ child::tagname
/*	/ child::any
//tagname	/ descendant::taganme
//*	/ descendant::any
/@attrname	/ attribute::attrname
/@*	/ attribute::any
.	self::node()
..	parent::node()

- Examples:

  /book/chapter/section

  //chapter/*

  //book/author/@*

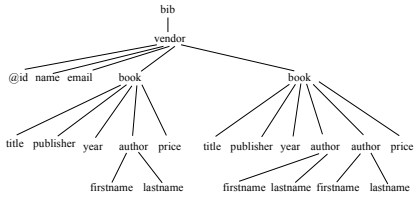
Semantics

step1 / step2

- Evaluate step1 to get a sequence of nodes
- Bind current context (.) to each node in the sequence
- Evaluate step2 using this binding; get a new sequence of nodes
- Concatenate all these sequences into one sequence
- Eliminate duplicates
- Sort by document order

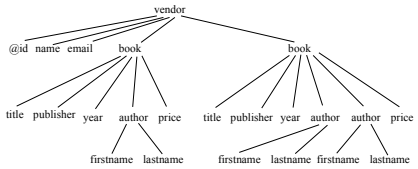
Example 1

```
<?xml version="1.0" encoding="UTF-8"?>
<bib>
 <vendor id="id0_1">
   <name>Amazon</name>
   <email>webmaster@amazon.com</email>
   <book>
      <title>Unix Network Programming</title>
      <publisher>Addison Wesley</publisher>
      <year>1995</year>
      <author>
         <firstname>Richard</firstname>
         <lastname>Stevens</lastname>
      </author>
      <price>38.68</price>
   </book>
   <book>
      <title>An Introduction to Object-Oriented Design</title>
      <publisher>Addison Wesley</publisher>
      <year>1996</year>
      <author>
         <firstname>Jo</firstname>
         <lastname>Levin</lastname>
      </author>
      <author>
         <firstname>Harold</firstname>
         <lastname>Perry</lastname>
      </author>
      <price>11.55</price>
   </book>
 </vendor>
</bib>
```



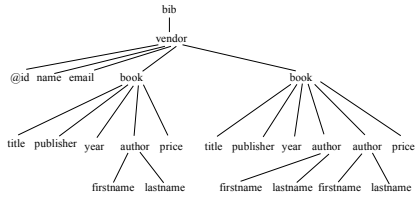
XPath: /bib/vendor

Result:



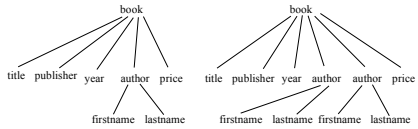
Example 2

```
<?xml version="1.0" encoding="UTF-8"?>
<bib>
 <vendor id="id0_1">
   <name>Amazon</name>
   <email>webmaster@amazon.com</email>
   <book>
      <title>Unix Network Programming</title>
      <publisher>Addison Wesley</publisher>
      <year>1995</year>
      <author>
         <firstname>Richard</firstname>
         <lastname>Stevens</lastname>
      </author>
      <price>38.68</price>
   </book>
   <book>
      <title>An Introduction to Object-Oriented Design</title>
      <publisher>Addison Wesley</publisher>
      <year>1996</year>
      <author>
         <firstname>Jo</firstname>
         <lastname>Levin</lastname>
      </author>
      <author>
         <firstname>Harold</firstname>
         <lastname>Perry</lastname>
      </author>
      <price>11.55</price>
   </book>
 </vendor>
</bib>
```



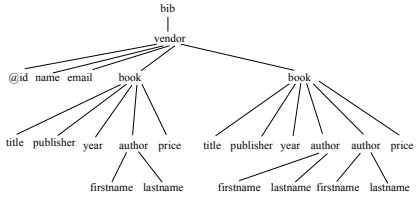
XPath: /bib/vendor/book

Result:



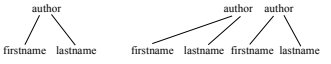
Example 3

```
<?xml version="1.0" encoding="UTF-8"?>
<bib>
 <vendor id="id0_1">
   <name>Amazon</name>
   <email>webmaster@amazon.com</email>
   <book>
      <title>Unix Network Programming</title>
      <publisher>Addison Wesley</publisher>
      <year>1995</year>
      <author>
         <firstname>Richard</firstname>
         <lastname>Stevens</lastname>
      </author>
      <price>38.68</price>
   </book>
   <book>
      <title>An Introduction to Object-Oriented Design</title>
      <publisher>Addison Wesley</publisher>
      <year>1996</year>
      <author>
         <firstname>Jo</firstname>
         <lastname>Levin</lastname>
      </author>
      <author>
         <firstname>Harold</firstname>
         <lastname>Perry</lastname>
      </author>
      <price>11.55</price>
   </book>
 </vendor>
</bib>
```



XPath: /bib/vendor/book/author

Result:



Functions

- XPath operators
  - arithmetic and boolean
    + - * div mod = != < > <= >= and or
  - selecting multiple tagnames
    - Example: return the author names and prices of all books

      `//book(author/name | price)`
- XPath system functions
  - They are all from namespace fn

    fn:function_name(arg1,...,argn)
  - See: http://www.w3schools.com/xpath/xpath_functions.asp
  - Examples:

    //book[contains(title,'XML')]/price

    distinct-values(//book/author/lastname)

    //book[count(author) > 1]/title

    //book[position() > 10]/title

    //book[last()-2]/price

Atomization

- How do we extract a value from a node?
  - //gradstudent[gpa > 3.5]/name
- arithmetic expressions/comparisons extract the value of the node before the apply the operation using fn:data
  - eg: fn:data(<a>2) = "2"
- strings are cast to numerical values
  - eg: "3" < 4 is true
- Effective boolean value:
  - () "" 0 fn:false() are all false
  - fn:true() is true
  - any nonzero number or nonempty string is true
  - any nonempty sequence where the first item is a node is true
  - otherwise, error

Predicates

- Existential semantics:
  true, if the resulting sequence is not empty
- Many variations

//book[10]	the tenth child node of the context node (tenth book) same as //book[position()=10]
//book[last()]	the last child node of the context node (last book)
//book[author]	all books that have at least one author
//book[author/name]	all books that have at least one author/name
//book[author/name='Smith']	all books authored by Smith
//book[price>35.0]	all books that have price more than 35.0

- Examples

```
/bib/book[@price < 100]/title
/bib/book[author/text()]
//author[name/firstname='John'][name/lastname='Smith']/title
/bib/*/author[name/firstname][address[zip=12345][city]]/name/lastname
```


Using XPath in Java

```
import javax.xml.xpath.*;
import org.xml.sax.InputSource;

class XPATH {
   public static void main ( String[] args ) throws Exception {
      String xpath_query = "//gradstudent[name/lastname='Smith']/name";
      XPathFactory xpathFactory = XPathFactory.newInstance();
      XPath xpath = xpathFactory.newXPath();
      InputSource inputSource = new InputSource("cs.xml");
      XPathExpression query = xpath.compile(xpath_query);
      String result = query.evaluate(inputSource);
      System.out.println(result);
   }
}
```


XSL Transformation

- XSL stands for eXtensible Stylesheet Language
- XSLT: XSL Transformations
- XSLT is a stylesheet specification language for converting XML documents into various forms (XML, HTML, plain text)
- Can transform each XML element into another element, add new elements into the output file, or remove elements
- Can rearrange and sort elements, test and make decisions about which elements to display, and much more
- Uses XPath to navigate through a document:

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http//www.w3.org/1999/XSL/Transform">
      <students>
            <xsl:copy-of select="//student/name"/>
      </students>
</xsl:stylesheet>
```

XSLT Templates

- XSL uses XPath to define parts of the source document that match one or more predefined templates.
- When a match is found, XSLT will transform the matching part of the source document into the result document.
- The parts of the source document that do not match a template will end up unmodified in the result document (they will use the default templates).

Form:

```
<xsl:template match="XPath expression">
        …
</xsl:template>
```

The default (implicit) templates visit all nodes and strip out all tags:

```
<xsl:template match="* | / ">
    <xsl:apply-templates/>
</xsl:template>
<xsl:template match="text() | @*">
    <xsl:value-of select="."/>
</xsl:template>
```


Other XSLT Elements

<xsl:value-of select="XPath expression"/>
select the value of an XML element and add it to the output stream of the transformation, e.g. <xsl:value-of select="//books/book/author"/>

<xsl:copy-of select="XPath expression"/>
copy the entire XML element to the output stream of the transformation

<xsl:apply-templates match="XPath expression"/>
apply the template rules to the elements that match the XPath expression

<xsl:element name="XPath expression"> … </xsl:element>
add an element to the output with a tag-name derived from the XPath

Example:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="employee">
        <b> <xsl:apply-templates select="node()"/> </b>
  </xsl:template>
  <xsl:template match="surname">
        <i> <xsl:value-of select="."/> </i>
  </xsl:template>
</xsl:stylesheet>
```


Copy the Entire Document

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <xsl:apply-templates/>
    </xsl:template>
    <xsl:template match="text()">
        <xsl:value-of select="."/>
    </xsl:template>
    <xsl:template match="*">
        <xsl:element name="name(.)">
            <xsl:apply-templates/>
        </xsl:element>
    </xsl:template>
</xsl:stylesheet>
```


More on XSLT

- *Conflict resolution*: more specific templates overwrite more general templates. Templates are assigned default priorities, but they can be overwritten using priority="n" in a template.
- Modes can be used to group together templates. No mode is an empty mode.

```
<xsl:template match="…" mode="A">
        <xsl:apply-templates mode="B"/>
</xsl:template>
```

- Conditional and loop statements:

```
<xsl:if test="XPath predicate"> body </xsl:if>
<xsl:for-each select="XPath"> body </xsl:for-each>
```

- Variables can be used to name data:

```
<xsl:variable name="x"> value </xsl:variable>
```

Variables are used as {$x} in XPaths.

XLST Example

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
  <html xmlns="http://www.w3.org/1999/xhtml">
   <body>
    <h2>Best Graduate Students</h2>
    <table border="1">
     <tr bgcolor="green">
      <th>Name</th>  <th>Phone</th>  <th>Address</th>  <th>Office</th>  <th>URL</th>
     </tr>
     <xsl:for-each select="//gradstudent[gpa='4.0']">
      <tr>
       <td> <xsl:value-of select="name/lastname"/>, <xsl:value-of select="name/firstname"/> </td>
       <td><xsl:value-of select="phone"/></td>
       <td> <xsl:value-of select="address/city"/>, <xsl:value-of select="address/state"/> <xsl:value-of select="address/zip"/> </td>
       <td><xsl:value-of select="office"/></td>  <td><xsl:value-of select="url"/></td>
      </tr>
     </xsl:for-each>
    </table>
   </body>
  </html>
 </xsl:template>
</xsl:stylesheet>
```


Transforming XML

- The cs.xml file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="xslt-example.xsl"?>
<department>
 <deptname>Computer Sciences</deptname>
 <gradstudent>
  <name>
   <lastname>Aboulnaga</lastname>
   <firstname>Ashraf</firstname>
  </name>
  <phone>2626623</phone>
  <email>ashraf@cs.wisc.edu</email>
  <address>
   <city>Madison</city>
   <state>WI</state>
   <zip>53705</zip>
  </address>
  <office>7360</office>
  <url>www.cs.wisc.edu/~ashraf</url>
  <gpa>4.0</gpa>
 </gradstudent>
...
```

Best Graduate Students

Name	Phone	Address	Office	URL
Aboulnaga, Ashraf	2626623	Madison, WI53705	7360	www.cs.wisc.edu/~ashraf
Ailamaki, Anastasia	2652311	Madison, WI53706	7351	www.cs.wisc.edu/~natassa
Polyzotis, Neoklis	2652311	Madison, WI53705	7351	www.cs.wisc.edu/~alkis
Reddy, Sridhar	2625596	Madison, WI53705	7360	www.cs.wisc.edu/~sridhar
Beyer, Kevin	2626629	Madison, WI53705	7390	www.cs.wisc.edu/~beyer
Butts, Adam	2626618	Madison, WI53705	6382	www.cs.wisc.edu/~butts
Bezenek, Todd	2656605	Madison, WI53706	1331	www.cs.wisc.edu/~bezenek
Deyeh, Gregory	2652455	Madison, WI53705	7351	www.cs.wisc.edu/~gdeyeh
Yang, Pan	2625596	Madison, WI53705	1343	www.cs.wisc.edu/~fyang
Donjerkovic, Donko	2624542	Madison, WI53705	7366	www.cs.wisc.edu/~donko
Pung, Glenn	2652455	Madison, WI53705	7351	www.cs.wisc.edu/~gfung
Glew, Andrew	2625596	Madison, WI53705	1343	www.cs.wisc.edu/~glew
Gupta, Anurag	2621243	Madison, WI53705	1346	www.cs.wisc.edu/~anurag

Using XSLT in Java

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

class XSLT {
   public static void main ( String argv[] ) throws Exception {
      File stylesheet = new File("xslt-example.xsl");
      File xmlfile  = new File("cs.xml");
      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
      DocumentBuilder db = dbf.newDocumentBuilder();
      Document document = db.parse(xmlfile);
      StreamSource stylesource = new StreamSource(stylesheet);
      TransformerFactory tf = TransformerFactory.newInstance();
      Transformer transformer = tf.newTransformer(stylesource);
      DOMSource source = new DOMSource(document);
      StreamResult result = new StreamResult(System.out);
      transformer.transform(source,result);
} }
```