XML Query Languages

© Leonidas Fegaras
University of Texas at Arlington

Query Languages for XML

- Need a language for XML data for
  - extracting fragments (querying)
  - restructuring (data transformation)
  - integrating (e.g., combining multiple XML documents)
  - browsing
  - presentation (e.g., 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

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

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
- 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
- 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
- .. is short for parent::node()
  - ..title is short for parent::node()/child::title

Most Common Steps

<table>
<thead>
<tr>
<th>XPath step</th>
<th>full XPath syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>/tagname</td>
<td>/ child::tagname</td>
</tr>
<tr>
<td>*</td>
<td>/ child::any</td>
</tr>
<tr>
<td>/tagname</td>
<td>/ descendant::tagname</td>
</tr>
<tr>
<td>/</td>
<td>/ descendant::any</td>
</tr>
<tr>
<td>@attrname</td>
<td>/ attribute::attrname</td>
</tr>
<tr>
<td>@*</td>
<td>/ attribute::any</td>
</tr>
<tr>
<td>.</td>
<td>self::node()</td>
</tr>
<tr>
<td>..</td>
<td>parent::node()</td>
</tr>
</tbody>
</table>

Examples:

/book/chapter/section
/chapter/*
/book/author/@*
Example 2

XPath: /bib/vendor/book
Result:

Example 3

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
      ```xml
      /book[author/name | price]
      ```

- **XPath functions**
  - They are all from namespace fn
    - `fn:function_name(arg1,...,argn)`
  - See: [http://www.w3schools.com/xpath/xpath_functions.asp](http://www.w3schools.com/xpath/xpath_functions.asp)
  - Examples:
    ```xml
    /book[contains(title,'XML')]/price
    distinct-values(/book/author/lastname)
    /book[count(author) > 1]//title
    /book[position()>10]//title
    /book[last()-2]/price
    ```

Predicates

- **Existential semantics:** true, if the resulting sequence is not empty
- **Many variations**
  - `/book[10]`
  - `/book[last()]`
  - `/book[@name='Smith']`
  - `/book[position()=10]`
  - `/book[last()]`
  - `/book[@name='John'][name/lastname='Smith']`
  - `/book/@price < 100`
Using XPath in Java

```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, etc)
- 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:

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

XSLT Templates

- XSLT 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>
</xsl:stylesheet>
```

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" .../>
- 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="/book">
    <b>
      <xsl:apply-templates select="/book/author"/>
    </b>
  </xsl:template>
  <xsl:template match="/book/author">
    <b>
      <xsl:value-of select="/book/author"/>
    </b>
  </xsl:template>
</xsl:stylesheet>
```
Copy the Entire Document

```xml
<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="*">
    <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.
  ```xml
  <xsl:template name="..." mode="A">
    <xsl:apply-templates mode="B"/>
  </xsl:template>
  ```
- **Conditional and loop statements:**
  ```xml
  <xsl:if test="XPath predicate"> body </xsl:if>
  <xsl:for-each select="XPath"> body </xsl:for-each>
  ```
- Variables can be used to name data:
  ```xml
  <xsl:variable name="x" select="value"/>
  Variables are used as `{x}` in XPaths.
  ```

XLST Example

```xml
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>  
  <xsl:template match="/">
    <html>
      <body>
        <table border="1">
          <tr><td>Name</td><td>Address</td><td>Phone</td><td>Email</td></tr>
          <tr><td>A2-Graduate Student</td><td>123 Example Street</td><td>(123) 456-7890</td><td>example@example.com</td></tr>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Transforming XML

- **The cs.xml file:**
  ```xml
  <xsl:stylesheet type="text/xsl" href="cs-example.xsl"/>
  <xsl:variable name="department" select="'Computer Science'"/>
  <xsl:variable name="graduates" select="'A2-Graduate Student'"/>
  <xsl:template match="/">
    <html>
      <body>
        <table border="1">
          <tr><td>Name</td><td>Address</td><td>Phone</td></tr>
          <tr><td><xsl:value-of select="graduates"/></td><td><xsl:value-of select="address"/></td><td><xsl:value-of select="phone"/></td></tr>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```
Using XSLT in Java

```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[] args ) throws Exception {
        File stylesheet = new File("xsl-example.xsl");
        File xmlfile = new File("cs.xml");
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document document = db.parse(xmlfile);
        StreamSource stylesheet = new StreamSource(stylesheet);
        TransformerFactory tf = TransformerFactory.newInstance();
        Transformer transformer = tf.newTransformer(styleSheet);
        DOMSource source = new DOMSource(document);
        StreamResult result = new StreamResult(System.out);
        transformer.transform(source, result);
    }
}
```