INFO/CS 4302
Web Information Systems
FT 2012
Week 5: Web Architecture: Structured Formats – Part 3 (XML Manipulations)
(Lecture 8)

Theresa Velden
RECAP
## XML & Related Technologies overview

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Structured content</th>
<th>Define Document Structure</th>
<th>Access Document Items</th>
<th>Transform Document</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>XML Schema</td>
<td>XPath</td>
<td>XSLT</td>
<td></td>
</tr>
<tr>
<td>JSON</td>
<td>RELAX NG</td>
<td>DOM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YAML</td>
<td>DTD</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- XML: Extensible Markup Language
- XML Schema: XML Schema
- XPath: XPath
- XSLT: XSL Transformations
- JSON: JavaScript Object Notation
- RELAX NG: RELAX Next Generation
- DOM: Document Object Model
- YAML: YAML
- DTD: Document Type Definition
XML Meta Documents

• Express constraints on an xml document:
  – What element names and attributes to use
  – How often an element may occur
  – How elements are nested (complex elements)
  – What values attributes may have
  – What content elements may have... etc.

• Examples:
  – DTD (Document Type Definition developed for SGML)
  – XML Schema
  – RELAX NG
XML Schema & RELAX NG

• XML Schema
  – W3C recommendation
  – Powerful & complex (3-part recommendation)

• RELAX NG
  – Tree constraint language written in XML (with additional compact notation)
  – Integrate well with data type libraries (such as from XML Schema)
  – Supports namespaces
  – For many purposes equivalent to XML Schema
RECAP: XPath

• Context Node (starting point)
  – current node in XML document that is basis of path evaluation
  – Default: root element (which is the ‘document’)

• Location steps (selection of node sets)
  – Sequence of node specifications
  – Evaluation of each node specification creates a new context (within the previous context)
  – Like file paths

• Predicates: used to define a node with a specific value (i.e. to restrict the node set returned)

• Axes: define directions of movement of a step relative to current node

• Location path expressions can consist of several steps:
  – A step: axisname::nodetest[predicate]
  – As a default, child axis is used; hence when axis name missing, assume child of the node named in the node test part of the expression
  – child::actor/child::name[@gender="female"]
    → actor/name[@gender="female"]

• Result is node, set of nodes, subtree, set of subtrees
RECAP: XPath

- Like regular expressions for text, XPath finds specific portions within an XML document tree

axisname::nodetest[predicate]

AxisNames:
1. ancestor
2. ancestor-or-self
3. attribute
4. child
5. descendant
6. descendant-or-self
7. following
8. following-sibling
9. namespace
10. parent
11. preceding
12. preceding-sibling
13. self

Xpath Expressions
/catalogue/movie/actors/child::node()
/catalogue/movie/actors/child::text()
/catalogue/movie/actors/node()
/catalogue/movie/actors/text()
/catalogue/movie/child::*/child::actor
/catalogue/movie/child::*/child::node()
XPath expressions

• Demo XPath in oxygen
XPath expressions

Additional XPath Resource:
Article by Tobias Schlitt and Jacob Westhoff
html: http://schlitt.info.opensource/blog/0704_xpath.html#clarification-of-terms
pdf: http://westhoffswelt.de/data/portfolio/xpath.pdf
XSLT (XSL TRANSFORMATIONS)
XSLT

Is an XML oriented programming language
• Uses XML as its syntax
• Uses XPath to select subsets of an XML document
• Weakly typed
• Not designed for large programming tasks
• **Standard language for XML-to-XML transformations**
• Is a functional programming language (hard to get used to for people trained in procedural languages)
  – Functions are first-class citizens, supports passing functions as arguments to other functions
  – Iteration usually done by recursion
XSLT

• A transformation in the XSLT language is expressed in the form of an XSL stylesheet
  – root element: `<xsl:stylesheet>`
  – an xml document using the XSLT namespace, i.e. tags are in the namespace [http://www.w3.org/1999/XSL/Transform](http://www.w3.org/1999/XSL/Transform)
• The body is a set of templates or rules
  – The ‘match’ attribute specifies an XPath of elements in source tree
  – Body of template specifies contribution of source elements to result tree
• You need an XSLT processor to apply the style sheet to a source XML document
XSLT

- The transformation is achieved by a set of **template rules**
- A template rule associates a **pattern** with a **sequence constructor**
  - The pattern matches nodes in the source document “for nodes satisfying pattern X do this”
  - The resulting nodes and atomic values can be used to produce parts of a result tree

```xml
<xsl:template match="/">
  <html>
    ...
  </html>
</xsl:template>
```

**Simple Template**

Meaning: process root element of source XML document & generate HTML document for the root

Template body will usually contain xslt instructions
<xsl:template match="/">
  <html>
    ...
    <xsl:for-each select="actor/name">
      <h3>Name:</h3>
      <p><xsl:value-of select="." /></p>
    </xsl:for-each>
  </html>
</xsl:template>

Process all name nodes that are children of actor nodes
- Output a h3 heading
- Generate a paragraph element
- Output the string value of the current node (i.e. the text content of the name element)

Template Body w XSLT instructions
XPath expressions!

Extracts only the names of female actors
Non-xsl element are literals

Elements from xsl namespace are transform instructions

match attribute value is XPath expression setting context for execution of body

Sequential execution within template

<xsl:apply-templates/>
  
  – Set context to next tree step (default: depth first)
  
  – Re-evaluate rules (recurse)
XSLT

• The transformation is achieved by a set of template rules
• A template rule associates a pattern, which matches nodes in the source document
• The output is part of a result tree
• Default behavior of an XSL stylesheet is tree traversal
  – The text of the ‘document’ it outputs is produced technically by concatenating all text nodes
  – By default XSLT traverses the entire document and copies all text nodes
  – Note: attributes are not children of element nodes! (think of them as properties of elements with a value)
• The output format can be specified
  – E.g. `<xsl:output encoding="UTF-8" indent="yes" method="html" />`
  – Default output is xml
XSLT – In-class Exercise

• Example 1
• Example 2
• Example 3
• Example 4
XSLT: Example 1 – Default Behavior

Example: Minimal XSL

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  </xsl:stylesheet>
</?xml version="1.0" encoding="UTF-8"?>

<?xml version="1.0" encoding="UTF-8"?>
<!-- catalogue_snippet.xml  Created: 2012-09-08 17:09 -->
<catalogue>
  <movie>
    <title lang="en">The Others</title>
    <actors>
      <actor>
        <name gender="female">Nicole Mary Kidman</name>
      </actor>
      <actor>
        <name gender="female">Elaine Cassidy</name>
      </actor>
    </actors>
  </movie>
</catalogue>
```

Simple XML Source Document incl. elements and element attributes
XSLT: Example 1 – Default Behavior

- The text of the ‘document’ it outputs is produced technically by concatenating all text nodes
- By default XSLT traverses the entire document and copies all text nodes
- It works its way through the document recursively

```xml
<?xml version="1.0" encoding="utf-8"?>

  The Others

    Nicole Mary Kidman

    Elaine Cassidy

XML Result Document
simple text content
```
XSLT: Example 2

```xml
<xs1:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xs1:output method="text"/>
  <xs1:template match="*">
    <xs1:text>( Element:\</xs1:text>
    <xs1:value-of select="local-name()"/>\</xs1:text>
    <xs1:apply-templates select="* | @*"/>\</xs1:text>
  </xs1:template>
  <xs1:template match="@*">
    <xs1:text>Attribute:\</xs1:text>
    <xs1:value-of select="local-name()"/>\</xs1:text>
  </xs1:template>
</xs1:stylesheet>
```

[optional] defines output format (default is xml)

Not necessarily needed to generate text in output; can be useful to control line breaks and white space
XSLT: Example 2

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- catalogue_snippet.xml Created: 2012-09-08 17:09 -->
<catalogue>
  <movie>
    <title lang="en">The Others</title>
    <actors>
      <actor>
        <name gender="female">Nicole Mary Kidman</name>
      </actor>
      <actor>
        <name gender="female">Elaine Cassidy</name>
      </actor>
    </actors>
  </movie>
</catalogue>
```
XSLT: Example 2

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="text"/>
  <xsl:template match="*">
    <xsl:text>
      Element: </xsl:text>
      <xsl:value-of select="local-name()"/>
      <xsl:apply-templates select="* | @*"/>
    </xsl:text>
  </xsl:template>
  <xsl:template match="@*">
    <xsl:text>Attribute: </xsl:text>
    <xsl:value-of select="local-name()"/>
  </xsl:template>
</xsl:stylesheet>
```
XSLT: Example 2

( Element:catalogue( Element:movie
( Element:titleAttribute:lang)( Element:actors
( Element:actor( Element:nameAttribute:gender))
( Element:actor( Element:nameAttribute:gender)))))

XML Result Document
simple text content
Examples 3 + 4

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!-- catalogue_simple.xml  Created: 2012-09-08 17:09 -->
<catalogue>
  <movie>
    <title>The Others</title>
    <actors>
      <actor>
        <name>Nicole Mary Kidman</name>
      </actor>
      <actor>
        <name>Elaine Cassidy</name>
      </actor>
      <actor>
        <name>Christopher Eccleston</name>
      </actor>
      <actor>
        <name>Alakina Mann</name>
      </actor>
      <actor>
        <name>Eric Sykes</name>
      </actor>
      <actor>
        <name>Fionnula Flanagan</name>
      </actor>
    </actors>
  </movie>
</catalogue>
```

Simple XML Source Document:

Catalogue of movies (title > text) and actors in those movies (name > text)
XSLT: Example 3 (Pull Model)

**Pull Model** = all patterns are pulled into the root template

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output encoding="UTF-8" indent="yes" method="html"/>
  <xsl:template match="/">
    <html>
      <body>
        <h2>Movies</h2>
        <xsl:for-each select="/catalogue/movie">
          <h4><xsl:value-of select="."/></h4>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Non-XSLT elements are literal result elements (may contain XSLT or other literal result elements)
XSLT: Example 3 (Pull Model)

**Pull Model** = all patterns are pulled into the root template

```html
<html>
  <body>
    <h2>Movies</h2>
    <h4>The Others</h4>
    <h4>The Sea Inside</h4>
    <h4>Waiting for the Hearse</h4>
    <h4>The Animatrix</h4>
    <h4>My Darling Clementine</h4>
    <h4>Vicky Cristina Barcelona</h4>
    <h4>The Godfather Part II</h4>
    <h4>The Spiderwick Chronicles</h4>
    <h4>Day of the Fight</h4>
    <h4>The Son’s Room</h4>
  </body>
</html>
```
XSLT: Example 4 (Push Model)

Push Model = template pushes nodes out to be handled by other templates

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output encoding="UTF-8" indent="yes" method="html" />

    <xsl:template match="/">
        <html>
            <body>
                <h2>Movies</h2>
                <xsl:apply-templates/>
            </body>
        </html>
    </xsl:template>

    <xsl:template match="movie">
        <xsl:apply-templates select="title"/>
    </xsl:template>

    <xsl:template match="title">
        <h4><xsl:value-of select="."/></h4>
    </xsl:template>

</xsl:stylesheet>
```
<html>
<body>
  <h2>Movies</h2>
  <h4>The Others</h4>
  <h4>The Sea Inside</h4>
  <h4>Waiting for the Hearse</h4>
  <h4>The Animatrix</h4>
  <h4>My Darling Clementine</h4>
</body>
</html>

HTML Result Document
XSLT: ‘Pull Model’ vs ‘Push Model’

• One-template XSLT (Pull Model) is an easier way to get started
• For easy matching tasks pattern is sufficient
• However, for complex tasks this is the XSLT equivalent of ‘spaghetti code’
• In homework on XSLT you will be asked to apply several templates and practice avoiding spaghetti code
(Some) XSLT elements/XSLT instructions

<table>
<thead>
<tr>
<th>XSLT Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsl:apply-templates</td>
<td>Finds correct template to apply</td>
</tr>
<tr>
<td>xsl:template</td>
<td>Defines a template</td>
</tr>
<tr>
<td>xsl:element</td>
<td>Generates an element</td>
</tr>
<tr>
<td>xsl:attribute</td>
<td>Generates attribute node</td>
</tr>
<tr>
<td>xsl:value-of</td>
<td>Evaluates select and outputs value</td>
</tr>
<tr>
<td>xsl:text</td>
<td>Generates text literalaly</td>
</tr>
<tr>
<td>xsl:copy and xsl:copy-of</td>
<td>Shallow/deep copy of a node</td>
</tr>
<tr>
<td>xsl:choose</td>
<td>Conditional testing</td>
</tr>
<tr>
<td>xsl:for-each</td>
<td>Cycle based on test</td>
</tr>
<tr>
<td>xsl:if</td>
<td>Conditional test</td>
</tr>
<tr>
<td>xsl:sort</td>
<td>Sort the elements</td>
</tr>
</tbody>
</table>

Source: W3C http://www.w3.org/Consortium/Offices/Presentations/XSLT_XPATH/#(19)
XSLT: Conditional Instructions

• Programming languages typically provide ‘if-then, else’ constructs
• XSLT provides
  – If-then: `<xsl:if>`
  – If-then-(elif-then)*-else: `<xsl:choose>`
<catalogue>
  <movie>
    <title lang="en">The Others</title>
    <actors>
      <actor>
        <name gender="female">Nicole Mary Kidman</name>
      </actor>
      <actor>
        <name gender="female">Elaine Cassidy</name>
      </actor>
    </actors>
  </movie>
  <movie>
    <title lang="fr">Les Autres</title>
    <actors>
      <actor>
        <name gender="female">Nicole Mary Kidman</name>
      </actor>
      <actor>
        <name gender="female">Elaine Cassidy</name>
      </actor>
    </actors>
  </movie>
</catalogue>
<xsl:stylesheet version="1.0" encoding="UTF-8" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output encoding="UTF-8" indent="yes" method="html" />

  <xsl:template match="/">
    <html>
      <body>
        <h2>Movies</h2>
        <xsl:for-each select="catalogue/movie/title">
          <xsl:if test="@lang='en'">
            <h4><xsl:value-of select="."/> (English title)</h4>
          </xsl:if>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
<html>
  <body>
    <h2>Movies</h2>
    <h4>The Others (English title)</h4>
  </body>
</html>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output encoding="UTF-8" indent="yes" method="html"/>
  
  <xsl:template match="/">
    <html>
      <body>
        <h2>Movies</h2>
        <xsl:for-each select="catalogue/movie/title">
          <xsl:choose>
            <xsl:when test="@lang='en'">
              <h4><xsl:value-of select="."/>
                (English title)</h4>
            </xsl:when>
            <xsl:when test="@lang='fr'">
              <h4><xsl:value-of select="."/>
                (French title)</h4>
            </xsl:when>
          </xsl:choose>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
<html>
  <body>
    <h2>Movies</h2>
    <h4>The Others (English title)</h4>
    <h4>Les Autres (French title)</h4>
  </body>
</html>
<xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output encoding="UTF-8" indent="yes" method="html"/>
  <xsl:template match="/">
    <html>
      <body>
        <h2>Movies</h2>
        <xsl:for-each select="catalogue/movie/title">
          <xsl:choose>
            <xsl:when test="@lang='en'">
              <h4><xsl:value-of select="."/>
            </h4>
            <xsl:when test="@lang='fr'">
              <h4><xsl:value-of select="."/>
            </h4>
          </xsl:choose>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
<html>
  <body>
    <h2>Movies</h2>
    <h4>The Others (en.)</h4>
    <h4>Les Autres (fr.)</h4>
  </body>
</html>
More on XSLT (& XML):

• Lecture: ‘XML Foundations’ by Eric Wilde, School of Information, UC Berkeley Fall 2010
  http://dret.net/lectures/xml/
DOM (DOCUMENT OBJECT MODEL)
Well-formed XML Documents

• If an xml document is well-formed if it can be transformed deterministically (parsed) to a single infoset (DOM)
DOM – Document Object Model

• API for manipulating document tree
  – Language independent; implemented in lots of languages
  – Suitable for XML and other document formats
  – Core notion of a node

• W3C Specifications
  – Level 1: functionality and navigation within a document
  – Level 2: modules and options for specific content models (XML, HTML, CSS)
  – Level 3: further content model facilities (e.g. XML validation handlers)
DOM (Core) Level 1

DOM tree constraints
- Root must be Document
- Root children: one node and optionally Processing Instructions, DocumentType, Comment, Text
- Children of element must be either element, comment or text
- Attributes are properties of elements (not their children), and are not member of the DOM tree
- Children of attribute must be text
### W3C DOM Node Types

<table>
<thead>
<tr>
<th>Node type</th>
<th>Description</th>
<th>Children</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document</td>
<td>Represents the entire document (the root-node of the DOM tree)</td>
<td>Element (max. one), ProcessingInstruction, Comment, DocumentType</td>
</tr>
<tr>
<td>DocumentFragment</td>
<td>Represents a &quot;lightweight&quot; Document object, which can hold a portion of a document</td>
<td>Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference</td>
</tr>
<tr>
<td>DocumentType</td>
<td>Provides an interface to the entities defined for the document</td>
<td>None</td>
</tr>
<tr>
<td>ProcessingInstruction</td>
<td>Represents a processing instruction</td>
<td>None</td>
</tr>
<tr>
<td>EntityReference</td>
<td>Represents an entity reference</td>
<td>Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference</td>
</tr>
<tr>
<td>Element</td>
<td>Represents an element</td>
<td>Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference</td>
</tr>
<tr>
<td>Attr</td>
<td>Represents an attribute</td>
<td>Text, EntityReference</td>
</tr>
<tr>
<td>Text</td>
<td>Represents textual content in an element or attribute</td>
<td>None</td>
</tr>
<tr>
<td>CDATASection</td>
<td>Represents a CDATA section in a document</td>
<td>None</td>
</tr>
<tr>
<td>Node Type</td>
<td>Description</td>
<td>Subclasses</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>DocumentType</td>
<td>Provides an interface to the entities defined for the document</td>
<td>CDATASection, EntityReference</td>
</tr>
<tr>
<td>ProcessingInstruction</td>
<td>Represents a processing instruction</td>
<td>None</td>
</tr>
<tr>
<td>EntityReference</td>
<td>Represents an entity reference</td>
<td>Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference</td>
</tr>
<tr>
<td>Element</td>
<td>Represents an element</td>
<td>Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference</td>
</tr>
<tr>
<td>Attr</td>
<td>Represents an attribute</td>
<td>Text, EntityReference</td>
</tr>
<tr>
<td>Text</td>
<td>Represents textual content in an element or attribute</td>
<td>None</td>
</tr>
<tr>
<td>CDATASection</td>
<td>Represents a CDATA section in a document (text that will NOT be parsed by a parser)</td>
<td>None</td>
</tr>
<tr>
<td>Comment</td>
<td>Represents a comment</td>
<td>None</td>
</tr>
<tr>
<td>Entity</td>
<td>Represents an entity</td>
<td>Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference</td>
</tr>
<tr>
<td>Notation</td>
<td>Represents a notation declared in the DTD</td>
<td>None</td>
</tr>
</tbody>
</table>
Parsing XML Document Model (DOM)

- Using DOM for a programming language requires definition of interfaces and classes ("language bindings" provide API)
- Complete in-memory representation of entire tree

http://docstore.mik.ua/orelly/xml/jxml/ch05_01.htm
Parsing XML

• Alternative to DOM: SAX (open source software)
  – Event driven, sequential
  – Mostly hierarchical, no ‘sibling’ concept
  – More memory efficient
  – Good for quick, less intensive parsing that does not require easy-to-use, clean interface
Thursday

- Course Projects: Question & Answer Session
- Structured Formats – Part 4: JSON / YAML