5. Document Transformations

- XSLT (W3C Rec. 11/1999; XSLT 2.0 WD)
  - A language for transforming XML documents
  - representative of tree-based transformation languages
    (DSSSL, MetaMorphosis, TranSID, ...)
  - designed to be used
    - primarily as part of XSL formatting
    - also as an independent transformation language
- Our goal: to understand the basic model and central features of the language
  - Overview and an example
  - Data model and processing model

XSLT: Overview

- XSLT uses XML syntax for expressing transformations
  - of a document source tree into a result tree
    - result and source are separate trees
    - by template rules
- Each (ordinary) template rules has
  - a pattern (matched against nodes of the source tree)
  - a template as a body
    - instantiated to create fragments of the result tree

Overview of XSLT Transformation

Style Sheets and Template Rules

- An xsl:stylesheet (or xsl:transform) consists of template rules:
  <xsl:template match="Pattern"> Template</xsl:template>
- Rule applied to nodes of the source tree matched by the Pattern
  - expressed using XPath (XML Path Language)
- Template consists of
  - literal result tree fragments (elements, text), and
  - XSLT instructions for creating further result tree fragments

XPath in a Nutshell

- XPath 1.0 W3C Rec. 11/1999 (XPath 2.0 WD)
  - a compact non-XML syntax for addressing parts of XML documents
  - used also in other W3C languages
    - Specs for hyperlinks in XML: XLink (Rec. -01) and XPointer (Rec. -03)
    - XQuery (WD, Apr '05; extends XPath 2.0)
  - also typical primitives for manipulating strings, numbers and truth values

An XSL transformation example

- Transform below document to HTML:
  <?xml-stylesheet type="text/xsl" href="walsh.xsl" ?>
  <!-- Modified from an example by Norman Walsh -->
  <doc><title>My Document</title>
  <para>This is a <em>short</em> document.</para>
  <para>It only exists to <em>demonstrate a <em>simple</em> XML document</em>.</para>
  <figure><title>My Figure</title><graphic fileref="myfig.jpg"/></figure>
</doc>

Result (edited for readability)

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
<html><head><title>A Document</title></head>
<body>
<H1>My Document</H1>
<P>This is a <I>short</I> document.</P>
<P>It only exists to <I>demonstrate a <B>simple</B>XML document</I>.</P>
<DIV>
<B>Figure 1. </B>  
<IMG src="myfig.jpg"><B>My Figure</B>
</DIV>
</body>
</html>

Example style sheet begins

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/"
<!-- rule for root -->
<N61/NAME="My Document"/61>
<!-- process root's children here: -->
<xsl:apply-templates />
</BODY>
</HTML>
</xsl:template>
<xsl:template match="doc/title">
<N61/NAME="My Figure"/61>
</BODY>
</HTML>
</xsl:template>
<xsl:template match="doc/title">
<N61/NAME="My Figure"/61>
</BODY>
</HTML>
</xsl:template>
<xsl:template match="doc/title">
<N61/NAME="My Figure"/61>
</BODY>
</HTML>
</xsl:template>
Example (paras and emphs)

```xml
<xsl:template match="para">
  <p><xsl:apply-templates /></p>
</xsl:template>

<xsl:template match="em">
  <i><xsl:apply-templates /></i>
</xsl:template>

<xsl:template match="em/em">
  <b><xsl:apply-templates /></b>
</xsl:template>
```

Example (figures)

```xml
<xsl:template match="figure">
  <!-- Insert a bold caption of form 'Figure Num.' by counting all figures in the document: -->
  <div><b>Figure <xsl:number level="any" count="figure" />.</b><br />
  <!-- Process the children of figure, -->
  <xsl:apply-templates select="graphic" />
  <!-- then the 'title' child: -->
  <xsl:apply-templates select="title" />
  </div>
</xsl:template>
```

Example (end of style sheet)

```xml
<xsl:template match="graphic">
  <img src="{@fileref}" />
</xsl:template>

<xsl:template match="figure/title">
  <b><xsl:apply-templates /></b>
</xsl:template>
```

Result (edited for readability)

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html><head><title>A Document</title></head>
<body>
  <h1>My Document</h1>
  <p>This is a <i>short</i> document.</p>
  <p>It only exists to <i>demonstrate a <b>simple</b>XML document</i>.</p>
  <div>
    <b>Figure 1. </b>  
    <br />
    <img src="myfig.jpg"><b>My Figure</b>
  </div>
</body>
</html>
```

What use of XSL or XSLT?

- XSL can be used in different ways
  - for offline document formatting
    - produce, say, PDF from XML by an XSL style sheet (using XSLT + XSL formatting objects)
  - for offline document manipulation
    - transform XML into other form (XML/HTML/text) using XSLT
  - for online document delivery
    - on a Web server
    - in a Web browser (if the browser supports)

XSLT in online document delivery

- XSLT in a browser
  - defines rendering of XML documents
  - approach of Microsoft Internet Explorer
    - transformation of XML to HTML on the fly in browser
  - NB: Microsoft’s XSLT implementation may differ from XSLT 1.0; at least it used to
- XSLT on a Web server
  - an HTTP request served by transforming XML on the fly to HTML (or other format) on the server

Main Aspects of XSLT

- Data model
  - How is document data viewed in XSLT?
- Selection mechanism
  - How are document parts selected for processing?
- Matching
  - How are the template rules selected?
- Processing model
  - How does the XSLT execution proceed?
XSLT/XPath document trees

- Element nodes have elements, text nodes, comments and processing instructions of their (direct) content as their children
  - NB: attribute nodes are not children
  - the value of an element node is the concatenation of its text-node descendants
- Nodes have a complete document order
  - root node first, otherwise according to the order of the first character of the XML markup for each node
  - > element node precedes it’s attribute nodes, which precede any content nodes of the element

XSLT/XPath Trees

- Similar to the DOM structure model, with slight differences:
  - value of an element: its full textual content (In DOM: null)
  - no names for text nodes, comment nodes, etc. (In DOM: "#text", "#comment", etc.)

XSLT/XPath trees: Example

```xml
<article>Written by the lecturer.</article>
```

Legend:
- type: element
- name: attribute
- value: text

<table>
<thead>
<tr>
<th>Context node</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
<th>5th or 6th</th>
<th>7th</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

XPath Expressions

- Used for selecting source tree nodes, conditional processing, and generating new text content
  - return node-sets, truth values, numbers or strings
  - can select any parts of source tree (node-set) for processing, using ...

Location paths

- the most characteristic of XPath expressions
  - evaluated with respect to a context node
  - result: set of nodes selected by the location path

Location steps: Axes

- In total 13 axes
  - for staying at the context node:
    » self
  - for going downwards:
    » child, descendant, descendant-or-self
  - for going upwards:
    » parent, ancestor, ancestor-or-self
  - for moving towards start/end of the document:
    » preceding-sibling, following-sibling, preceding, following
  - "Special" axes
    » attribute, namespace

XPath Axes and Their Orientation

- Axes are oriented away from the context node (except attribute and namespace axes, which are unordered sets)
  - the position() for the closest node = 1
  - for the most remote node, position() = last()

The simplest axis, self::

XPath Axes and Their Orientation

- **parent::**
- **ancestor::**
- **ancestor-or-self::**
- **child::**
- **descendant::**
- **descendant-or-self::**
- **preceding-sibling::**
- **following-sibling::**
- **following::**
- **preceding::**

**Location paths: Node tests**

- **Name**: any element node with name *Name*
  (on an attribute axis, any attribute node with name *Name*)
- **::**: any element node (on an attribute axis, any attribute node)
- **text()**: any text node
  - comment(): any comment node
  - processing-instruction(): any processing instruction node
- **node()**: any node of any type

**Location paths: Abbreviations**

- **Abbreviations in location steps**
  - 'child::' can be omitted
  - 'attribute::' can be shortened to 'a'
  - 'self::node()' can be shortened to '.' (period)
  - 'parent::node()' can be shortened to '..'
  - Predicate '[position()=n]' for testing occurrence
    position n can be shortened to '[n]' (n)
- '/descendant-or-self::node()' shortened to '//'
- Syntax resembles slightly Linux/Unix file path names
Semantics of Location Paths (example)

{2, 5, 7}
A
B
context node

C

43

2

8

/*node()/parent::B[child::A]

{3, 4, 6, 8} {2, 5, 7}

value after each step:

2

Location path examples (1)

- chap children of current node:
  /chap (or equivalently: chap, or
  ./child::*[name()='chap'])
- The document element
  (child element of root node): /*
- Elements chapter anywhere (below the root):
  //chapter (.//chapter -> anywhere below
  the context node)
- All chapters with attribute type="intro":
  //chapter[@type='intro']
- the previous chapter sibling:
  preceding-sibling::chapter[1]

Location path examples (2)

- All child elements having an attribute type:
  *[@type]
  NB: use of node sets as truth values:
  empty - false; non-empty - true
- All child elements of any author child:
  author/*
- sections whose type attribute equals style
  attribute of the document element:
    //sect[@type = /*/@style]
- First author child, and previous to the last:
  author[1], author[last()-1]

Location path examples (3)

- Predicate expressions can be Boolean
  combinations:
    – child elements author with a
      publ child, but without
      degree or award children:
      author[publ and not(degree or award)]
- Get author(s) of the highest sect ancestor that
  is contained in an appendix:
  ancestor:sect[ancestor::app][last()]/author

Main Aspects of XSLT

- Data model
- Selection mechanism
- Matching
  – How are the rules selected?
  – A: With Patterns
- Processing model

XSLT Patterns

- Main use in match attributes of template rules:
  <xsl:template match="Pattern"/>
  also used for numbering
- Restricted location path expressions:
  – steps with child and attribute axes only,
    separated by ‘/’ or ‘//’
    » but arbitrary predicates in [Expr] allowed
  – may begin with 'id('IdVal')
    (for selecting element nodes by ID attribute values)
  – alternative patterns separated by ‘|’ (~ node-set union)

XSLT Patterns: Semantics

- A location path pattern P is of form
  Step₁@₁ Step₂@₂ ... Stepₙ@ₙ
  where each separator @ₖ is either ‘/’ or ‘//’
  – may also begin with ‘/’; Pattern ‘//’ matches only the root
- Else P matches a node vᵢ if there are nodes vᵢ₋₁, ..., vᵢ such that
  each vᵢ satisfies the node test and possible predicates
  of Stepᵢ, and which form a path towards the root:
  – If P begins with a single ‘/’, node vᵢ must be child of the root
    – in case of Stepᵢ@₁ Stepᵢ node vᵢ₋₁ is the parent of vᵢ
    – in case of Stepᵢ@₁ Stepᵢ node vᵢ₋₁ is an ancestor of vᵢ

XSLT Patterns: Examples

- match="sect-head | section/head"
  – matches any element with name sect-head, and any
    head elements directly below a section
- Pattern
  /appendix/ulist/item[1]
  matches the first item element in a ulist element which
  is contained in an appendix, which is the document
  element
Main Aspects of XSLT

- Data model
- Selection mechanism
- Matching
- Processing model
  - How does the XSLT execution proceed?

XSLT Processing Model

0. Parse the document into a source tree
1. Construct the result tree by applying template rules to the source tree
   - may contain text and arbitrary elements (including XSL formatting objects)
2. Serialize the result tree (as XML, HTML or text), or interpret it to produce formatted output (in case of full XSL processing)

Overview of XSLT Transformation

XSLT Processing Model

0. Parse the document into a source tree
1. Construct the result tree by applying template rules to the source tree
   - may contain text and arbitrary elements (including XSL formatting objects)
2. Serialize the result tree (as XML, HTML or text), or interpret it to produce formatted output (in case of full XSL processing)

Overview of XSLT Transformation

Selecting among multiple matching rules

- Priority of a rule can be specified explicitly:
  
- Default priorities based on the match pattern:
  - 0 for simple name tests (like para, @href)
  - negative for less specific patterns
  - 0.5 for more complex patterns
- Multiple matching rules with the same maximum priority is an error - Processor may (quietly!) choose the last one of them

Application of template rules

- Without a select attribute (~ select="node()")
  
- Selected nodes are processed in document order (if not sorted with xsl:sort)
- Built-in rules allow recursive traversal to proceed gracefully in the absence of matching rules

Default rules for elements and content

- Built-in rule for the root and element nodes:
  
- Built-in rule for text and attribute nodes:
  
- Low priority --> can be overridden

A (Tricky) Processing Example

- Consider transforming document
  
- with the below rules:

Selecting among multiple matching rules

- Priority of a rule can be specified explicitly:
  
- Default priorities based on the match pattern:
  - 0 for simple name tests (like para, @href)
  - negative for less specific patterns
  - 0.5 for more complex patterns
- Multiple matching rules with the same maximum priority is an error - Processor may (quietly!) choose the last one of them

Application of template rules

- Without a select attribute (~ select="node()")
  
- Selected nodes are processed in document order (if not sorted with xsl:sort)
- Built-in rules allow recursive traversal to proceed gracefully in the absence of matching rules

Default rules for elements and content

- Built-in rule for the root and element nodes:
  
- Built-in rule for text and attribute nodes:
  
- Low priority --> can be overridden

A (Tricky) Processing Example

- Consider transforming document
  
- with the below rules:
Processing example (2)

- The result
  \[ <R><NewC>New: b1b3ccb2</NewC></R> \]

is obtained as follows:
1. Rule 1 matches the root node \(--\) Element node \(R\) is added to the result; Instruction \(<\text{xsl:apply-templates select="//\text{C}" />}\) selects the (only) \(\text{C}\) element for processing (which will produce the contents of node \(R\)).
2. Rule 2 with pattern "\(\text{C}\)" creates a \(\text{NewC}\) element node with text node "New: " as its first child.

Processing example (3)

3. Instruction \(<\text{xsl:apply-templates select="../\text{B}"/>}\) selects element \(\text{B}\) siblings of current node (C). The built-in element rule applies to these, and the built-in text rule to their children.
   Result: text nodes "b1" and "b3" become the next children of \(\text{NewC}\).
4. Instruction \(<\text{xsl:apply-templates />}\) in the context of element node \(\text{C}\) selects its children, "cc" and \(<\text{B}>b2</\text{B}>\) for processing. The built-in text rule inserts value "cc" to the result tree, and the \(\text{B}\) element node becomes "b2" in the result (similarly to step 3).

Is it Really So Tricky?

- In practice: Seldom
  - A computer scientist wants to understand how a model really works …
- XSLT is a high-level declarative language for describing transformations
  - Normally no need to think so hard about execution; Often sufficient just to specify declarative rules to handle different cases, like
    \(<\text{xsl:template match="\text{para}"}>\text{<P><xsl:apply-templates /></P>}</xsl:template>\)