5 Document Transformations

- XSLT 1.0 (W3C Rec. 11/99; XSLT 2.0 Candidate Rec. 11/05)
  - A language for transforming XML documents
  - Initial main purpose to support XSL formatting
  - Currently mainly (?) used as an independent transformation language (esp. XML → HTML)

- Our goal: to understand the basic model and central features of XSLT
  - 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
  - And result and source are separate trees
  - By template rules
- Each template rule 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:
  - 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 controlling further processing

XPath in a Nutshell

- XPath 1.0 W3C Rec. 11/99 (2.0 Cand.Rec. 11/05)
  - A compact non-XML syntax for addressing parts of XML documents (as node-sets)
  - Used also in other W3C languages
    - Specs for hyperlinks in XML:
      - XLink (Rec. '01) and XPointer (Rec. '03)
    - Also typical operations on strings, numbers, and truth values

An XSL transformation example

Example style sheet begins
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>
    <!-- Process the children of figure, -->
    <!-- the 'graphic' child first: -->
    <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}" />
  <!-- Assign the value of attribute 'fileref' to attribute 'src' -->
</xsl:template>

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

Result (edited for readability)

```html
<a>My Document</a>

This is a <i>short</i> document.

It only exists to <i>demonstrate a <b>simple</b>XML document</i>.

<div>
  <b>Figure 1.</b>  
  <br>
  <img src="myfig.jpg">My Figure</div>
```

What use of XSL(T)?

- 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
  - supported by MS IE, and Netscape/Mozilla (7.0/1.7)
    - transformation of XML to HTML on the fly in browser
  - NB: Microsoft's implementation used to differ from XSLT 1.0
- 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?

Data Model of XSLT and XPath

- Documents are viewed as trees
  - made of seven types of nodes:
    - root (additional parent of document element)
    - element nodes
    - attribute nodes
    - text nodes
    - comments, processing instructions and namespaces
- NB: Entity references are expanded
  - no entity nodes
XSLT/XPath document trees

- Defined in Sect. 5 of the XPath specification
- Element nodes have elements, text nodes, comments and processing instructions of their (direct) content as their children
  - NB: attribute nodes are not children (but have a parent)
  - the value of an element node is the concatenation of its text-node descendants

XSLT/XPath Trees

- Similar to the DOM, with slight differences:
  - 7 vs 12 node types
  - value of an element: its full textual content (In DOM: null)
  - no names for text nodes, comment nodes, etc. (In DOM: "#text", "#comment", etc.)
- Document order of nodes:
  - root node first, otherwise according to the order of the first character of the XML markup for each node
  - > element node precedes its attribute nodes, which precede any content nodes of the element

XSLT/XPath trees: Example

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

Location paths

- Consist of location steps separated by '/'
  - each step produces a set of nodes
  - steps evaluated left-to-right, each node in turn as context node
    - path begins with '/\' \(\Rightarrow\) root is the first context node
- Complete form of a location step:
  - `AxisName::NodeTest([PredicateExpr])`
    - axis specifies the tree relationship between the context node and the selected nodes
    - node test restricts the type and and name of nodes
    - filtered further by 0 or more predicates

Location steps: Axes

- In total 13 axes (~ directions in tree)
  - 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

- Ordinary axes oriented away from context node
  - `@attribute` and `namespace` axes are unordered
  - the `position()` for the closest node = 1
  - for the most remote node, `position()` = `last()`
- The simplest axis, `self:`

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
    - often the current node matched by the template pattern
  - result: set of nodes selected by the location path
### XPath Axes and Their Orientation

- **parent::** (exists for every node except the root)
  - Context node: 
  - Diagram:

- **ancestor::**
  - Diagram:

- **ancestor-or-self::**
  - Diagram:

- **child::**
  - Context node: 
  - Diagram:

- **descendant::**
  - Diagram:

- **descendant-or-self::**
  - Diagram:

- **preceding-sibling::**
  - Diagram:

- **following-sibling::**
  - Diagram:

- **preceding::**
  - Diagram:

### Location paths: Node tests

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

### Location paths: Abbreviations

- **Abbreviations in location steps**
  - `child::` can be omitted
  - `attribute::` can be shortened to `@`
  - `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]`
  - `/descendant-or-self::node() // shortened to //`

- Syntax resembles slightly Linux/Unix file path names
Semantics of Location Paths (example)

Location path examples (1)

Location path examples (2)

Main Aspects of XSLT

XSLT Patterns

XSLT Patterns: Semantics

XSLT Patterns: Examples

Main Aspects of XSLT


XSLT Processing Model

0. Parse the document into a source tree
1. Construct the result tree by applying template rules to the source tree
2. Serialize the result tree (as XML, HTML or text)

Overview of XSLT Transformation

Result Tree Construction (approximately)

ResultTree := AppiTempls([root of the source tree]);
proc ApplTempls(CNL: list of Nodes) returns list of Nodes:
ResList := emptyNodeList();
for each Node cn in CNL do // current node in current node list
  Find matching template rule (of highest priority; See next)
  Instantiate its template T in context (cn, CNL), and add to ResList
  Replace each <apply-templates select="E"/> in T by ApplTempls(L), where L = value of expr E in context (cn, CNL);
end for;
return ResList;

Selecting one of matching rules

- Priority of a rule can be specified explicitly:
  <xsl:template priority="2.0" ...>
- Default priorities based on the match pattern:
  - 0 for simple name tests (like para, @href)
  - negative for less specific patterns e.g., *, @*, node()
  - 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()")
  <xsl:apply-templates /> processes all children of current node -> "default traversal": top-down
- Selected nodes are processed in document order (if not sorted with xsl:sort)
- Built-in rules support the top-down

Built-In Default Rules

- For the root and element nodes:
  <xsl:template match="/ | *"><xsl:apply-templates /></xsl:template>
- For text and attribute nodes:
  <xsl:template match="text() | @*"> <!-- Insert the string value of current node: -->
    <xsl:value-of select="." />
  </xsl:template>
- Low priority -> can be overridden

A (Tricky) Processing Example

- Consider transforming document
  <A>
  <BbB1><B>c</B><C>c</C><D>d</D><B>b</B></BbB1></A>
with the below rules:
  <xsl:template match="/"> <!-- Rule 1 -->
    <B><xsl:apply-templates select="/C" /></B>
  </xsl:template>
  <xsl:template match="/C"> <!-- Rule 2 -->
    <NewC><xsl:apply-templates select="/B" />
    <xsl:apply-templates />
  </NewC>
</xsl:template>

Processing example (2)

- The result
  <B><New: b1b3ccb2</NewC></B>
  is obtained as follows:
  1. Rule 1 matches the root node -> Element node B is added to the result; Instruction <xsl:apply-templates select="/C" /> selects the (only) C element for processing (which will produce the contents of node C).
  2. Rule 2 with pattern "C" creates into result tree a NewC element node with text node "New: " as its first child.
Processing example (3)

3. Instruction `<xsl:apply-templates select="../B" />` selects element 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 NewC.

4. Instruction `<xsl:apply-templates />` in the context of element node C selects its children, "cc" and <B>b2</B>, for processing. The built-in text rule inserts value "cc" to the result tree, and the B element node becomes "b2" in the result (similarly to step 3).

Processing example (4)

Is it Really So Tricky?

- Seldom, fortunately
  - but a computer scientist wants to understand the working of a model
- XSLT is a high-level declarative language for describing transformations
  - Normally suffices to give simple rules for different cases, like
    ```xml
    <xsl:template match="para">
      <p><xsl:apply-templates /></p>
    </xsl:template>
    ```