5. Document Transformations

- **XSLT (W3C Rec. Nov-99)**
  - A language for transforming XML documents
  - representative of features common to 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 to learn 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 (normal) 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>`
  - `<xsl:template>` NB: well-formed XML...
- **Template consists of**
  - literal result tree fragments (elements, text), and
  - XSLT instructions for creating further result tree fragments
- **Rule applied only to nodes of source tree matched by the Pattern**
  - expressed using XML Path Language (XPath)

XPath in Short

- **W3C Recommendation (16-Nov-99)**
  - a compact non-XML syntax for addressing parts of XML documents
  - used also by XLink and XPointer languages
  - W3C drafts for describing hyperlinks in XML
  - provides typical basic operations for manipulation of strings, numbers and truth values

An XSL transformation example

- **Transform below XML document to HTML:**
  - `<xsl:stylesheet type="text/xsl" href="walsh.xsl">`<
xsl:template match="*"> Template</xsl:template>`<
</xsl:stylesheet>`
  - `<xsl:template>` NB: well-formed XML...
- **Rule applied only to nodes of source tree matched by the Pattern**
  - expressed using XML Path Language (XPath)
Example (style sheet begins)

```xml
<xsl:stylesheet xsl:version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <!-- rule for root -->
  <HTML><HEAD><TITLE>A Document</TITLE></HEAD>
  <BODY>
    <!-- process root's children here: -->
    <xsl:apply-templates />
  </BODY>
</xsl:template>
</xsl:stylesheet>
```

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> <!-- Create new attribute 'src': -->
  <xsl:attribute name="src">
    <!-- and assign the value of current element's fileref attribute to it: -->
    <xsl:value-of select="@fileref" />
  </xsl:attribute>
</IMG>
<xsl:template match="figure/title">
  <B> <xsl:apply-templates /> </B>
</xsl:template>
```

Result (edited for readability)

```xml
<!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>
```

Prospects of XSL or XSLT

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

- XSLT in a browser
  - defines rendering of XML documents
  - approach of Microsoft Internet Explorer 5
    - transformation of XML to HTML on the fly in browser
    - NB: Microsoft's XSLT implementation differs from XSLT 1.0
      (a new one exists but has to be installed separately)
- XSLT in a Web server
  - an HTTP request for an XML document served by
    transforming XML on the fly to HTML (or other format)
      in the server

XSLT/XPath Data Model

- XSLT and XPath have a common data model
- Documents viewed as abstract, logical tree structures
- Seven types of tree nodes
  - root (additional parent of document element)
  - element nodes
  - attribute nodes
  - text nodes
  - nodes for comments, processing instructions and
    namespaces
- NB: Entities are expanded -> no entity nodes

XSLT/XPath document trees

- Element nodes have element nodes, text nodes,
  comment nodes and processing instruction nodes
  of their (direct) content as children
  - NB: attribute nodes not viewed as 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
  - a element node precedes it's attribute nodes, which
    precede any content nodes of the element

XSLT/XPath trees: Example

- 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.)
- Example document:
  `<article>Written by <fig file="pekka.jpg"
    caption="The Lecturer"/> the
    lecturer.</article>`

XSLT/XPath trees: Example

- Used for selecting source tree nodes, conditional
  processing, and generating new text content
  - return node-sets, truth values, numbers or strings
  - XSLT can select any parts of source tree for
    processing using node-set valued expressions
- 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
Location paths (1)

- Consist of location steps separated by `/`
  - each step produces a set of nodes
  - steps evaluated left-to-right, each node in turn acting as a context node
- Each location step is of form
  \[ \text{AxisName} :: \text{NodeTest} ([\text{PredicateExpr}])^* \]
  - axis specifies the tree relationship between the context node and the selected nodes
  - node test restricts the type and and name of nodes
  - further filtered by 0 or more predicates

Location paths (2)

- In total 13 axes
  - child, descendant, parent, ancestor, following-
  - sibling, preceding-sibling, following, preceding,
  - attribute, self, descendant-or-self, ancestor-or-
  - self, namespace
- Node tests (slightly simplified)
  - Name: any element node with name \( \text{Name} \)
  - \* : any element node (on an attribute axis, any attribute node with name \( \text{Name} \))
  - text(): any text node
  - node(): any node of any type

Location paths (3)

- Abbreviations in location steps
  - 'child:' can be omitted
  - 'attribute:' can be shortened to '@'
  - '/(descendant-or-self::node())/' shortened to '//'
    ("go down any number of levels")
  - 'parent::node()' can be shortened to '..' (period)
  - 'Predicate [position()=n]' for testing occurrence position \( n \)
    can be shortened to '[n]' ('go down any number of levels')
- '/' in the beginning sets context to the root node
- Syntax resembles Linux/Unix file path names

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:

- parent:

- ancestor:

- ancestor-or-self:
XPath Axes and Their Orientation

- **child:**
  - Context node:
  - Diagram showing child nodes

- **descendant:**
  - Diagram showing descendant nodes

- **descendant-or-self:**
  - Diagram showing descendant-or-self nodes

- **preceding-sibling:**
  - Diagram showing preceding-sibling nodes

- **following-sibling:**
  - Diagram showing following-sibling nodes

Location path examples (1)

- **chap children of current node:**
  - .//chap (or equivalently: chap)

- **The document element (child element of root node):**
  - /*

- **Elements author anywhere (below the root):**
  - //author

- **All chapters with attribute type=intro:**
  - //chapter[@type='intro']

- **the previous chapter sibling:**
  - preceding-sibling::chapter[position()=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:
  - `author[publ and not(degree or award)]`
- Closest `chap` ancestor (and highest):
  `ancestor::chap[1]` (or `ancestor::chap[last()]`)
- `author` of the highest `chap` ancestor that is contained in an `app`endix:
  `ancestor::chap[ancestor::app][last()]/author`

XSLT Patterns

- Main use in match attributes of template rules:
  `<xsl:template match="Pattern">`
- Expressions specifying conditions on a node
  - `author` with a `publ` child, but without `degree` or `award` children:
  `author[publ and not(degree or award)]`
- Closest `chap` ancestor (and highest):
  `ancestor::chap[1]` (or `ancestor::chap[last()]`)
- `author` of the highest `chap` ancestor that is contained in an `app`endix:
  `ancestor::chap[ancestor::app][last()]/author`

XSLT Patterns: Semantics

- A location path pattern `P` is of form
  `Step₁§Step₂§…§Stepₙ` where each separator `§` is either `/` or `//`
- Pattern `P` matches a node `n` if the rightmost step `Stepₙ` matches `n` and a suitable node `v` matches the prefix `Step₁§Step₂§…§Stepₙ-1` of `P`:
  - if the rightmost `§` is `/`, `v` has to be the parent of `n`
  - if the rightmost `§` is `//`, `v` can be any ancestor of `n`

XSLT Patterns: Examples

- Pattern `sect-head | section/head`
  matches any element node with name `sect-head`, and any `head` elements directly below a `section`
- For a match with pattern `appendix//ulist/item[1]`
  a node has to be the first `item` element in a `ulist` element which is contained in an `appendix`
Selecting among multiple matching rules

- Priority of a rule can be specified by numerical attribute: `<xsl:template priority="2.0">`

  - Default priorities:
    - 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 recover by (silently) choosing the last one of those rules

Application of template rules

- Without a `select` attribute:
  ```xml
  <xsl:apply-templates />
  ```

  - Default behaviour: top-down traversal of source tree

- Otherwise the selected nodes are processed:
  - In document order (Order a major distinction to typical database query languages)

- 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:
  ```xml
  <xsl:template match="/ | *">
    <xsl:apply-templates />
  </xsl:template>
  ```

- Built-in rule for text and attribute nodes:
  ```xml
  <xsl:template match="text() | @*">
    <!-- Insert the string value of current node: -->
    <xsl:value-of select="." />
  </xsl:template>
  ```

  - Low priority → can be overridden

A result tree construction example

- Consider transforming document
  ```xml
  <A>
    <C>c1</C><B>bb</B><C>c2</C></B><D>dd</D><C>c3</C>
  </A>
  ```

  with below rules:
  ```xml
  <xsl:template match="/">
    <!-- Rule 1 -->
    <xsl:apply-templates select="//B" />
  </xsl:template>

  <xsl:template match="B">
    <!-- Rule 2 -->
    <NewB>New Content <xsl:apply-templates select="../C" /> <xsl:apply-templates />
  </NewB>
  </xsl:template>

  <xsl:template match="text()">
    <!-- Rule 3 -->
    <xsl:value-of select="." />
  </xsl:template>

  <xsl:template match="@*"> <!-- Rule 4 -->
    <xsl:apply-templates />
  </xsl:template>

Construction example (2)

- The result
  ```xml
  <NewB>New Content c1c3bbc2</NewB>
  ```

  is obtained as follows:
  1. Rule 1 with `match="/"` matches the root node.
  2. Rule 2 with pattern `"B"` creates into result tree a `NewB` element node with text node “New Content” as its first child.

Construction example (3)

3. Instruction `<xsl:apply-templates select="../C" />` selects element `c1` and `c3` become the next children of `NewB`

4. Instruction `<xsl:apply-templates />` in the context of element node `b` selects its children, “bb” and “c2”, for processing. The text node is copied to result tree by the built-in text rule, and the `c` element node becomes “c2” in the result (similarly to step 3).
Is it Really So Complex?

- Yes, and No
- XSLT is high-level declarative language for describing transformations
  - Normally no need to think about execution in so much detail; Often sufficient just to specify declarative rules to handle different cases
  - A computer scientist wants to understand how a model really works ...