**Overview of XSLT 1**

- **XSLT (1.0 W3C Rec. 11/1999; XSLT 2.0 Rec. 1/07)**
  - A language for transforming XML documents
  - Initial main purpose to support XSL formatting
  - Used mainly for independent transformations (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*
    - 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

**Style Sheets and Template Rules**

An `xsl:stylesheet` (or `xsl:transform`) consists of template rules:

```xml
<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 controlling further processing

**XPath in a Nutshell**

- **XPath 1.0 W3C Rec. 11/99 (2.0 Rec. 1/07)**
  - 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)
    - XQuery (Rec. 1/07; extends XPath 2.0)
  - Also typical operations on strings, numbers and truth values

**XSLT 2.0**

- Somewhat more advanced & flexible than 1.0
- Complexity of 1.0 vs. 2.0 specs (as pages):
  - Data Model ~ 80
  - XPath 1.0 ~ 30
  - XPath 2.0 ~ 100
  - Funcs & opers ~ 160
  - XSL 1.0 ~ 90
  - XSL 2.0 ~ 280
  - ~ 120
  - ~ 620

**An XSL transformation example**

Transform below document to HTML:

```xml
<?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 simple</em> XML document.</para>
<figure><title>My Figure</title>
  <graphic fileref="myfig.jpg"></graphic>
</figure>
</doc>
```

**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 simple</i> XML document.</p>
  <div>
    <b>Figure 1.</b> <br>
    <img src="myfig.jpg" alt="My Figure">
  </div>
</body>
</html>
```
Example style sheet begins

```xml
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <HTML>
    <HEAD><TITLE>A Document</TITLE></HEAD>
    <BODY>
      <!-- process root's children here: -->
      <xsl:apply-templates />
    </BODY>
  </HTML>
</xsl:template>
<xsl:template match="doc/title">
  <H1><xsl:apply-templates /></H1>
</xsl:template>
<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>
</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>
</xsl:stylesheet>
```

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, -->
    <!-- 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}" />
</xsl:template>
<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>
```

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 latest MS and Mozilla browsers
    » transformation of XML to HTML on the fly in browser
- 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?
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

XSLT/XPath document trees
- Defined in Sect. 5 of XPath 1.0 spec
  - for XSLT/XPath 2.0 & XQuery in their joint Data Model
- 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

Main Aspects of XSLT
- Data model
- Selection mechanism
  - How are document parts selected for processing?
  - A: With XPath expressions
- Matching
- Processing model

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

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 Axes and Their Orientation

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

XPath Axes and Their Orientation

- ancestor::
- ancestor-or-self::

XPath Axes and Their Orientation

- descendant::
- descendant-or-self::

XPath Axes and Their Orientation

- preceding-sibling::
- following-sibling::

XPath Axes and Their Orientation

- following::
- preceding::

Location paths: Node tests

- Node tests (slightly simplified)
  - Name: any element node with name Name
  - *: any element (any attribute node with name Name)
  - 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/()` can be omitted
  - `/ancestor::node/()` can be shortened to `/..`
  - `@` can be omitted

- Syntax resembles slightly Linux/Unix file path names

Location path examples (1)

- `chap` children of current node:
  . /chap (or simply chap, or `chap::[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 of type `A` or `B`:
  //chapter[@type='A' or @type='B']
- The previous `chapter` sibling:
  preceding-sibling::chapter[1]

Main Aspects of XSLT

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

XSLT Patterns

- A location path pattern \( P \) is of form
  \[
  \text{Step}_1 \odot \text{Step}_2 \odot \cdots \text{Step}_n \odot \text{Step}_r
  \]
  where each separator \( \odot \) is either `/` or `//`
  - may also begin with `//`; Pattern `/` matches only the root
- Else \( P \) matches a node \( v \), iff there are nodes \( v_1, \ldots, v_n \) such that each \( v_i \) satisfies the node test and possible predicates of \( \text{Step}_i \), and which form a path towards the root:
  - If \( P \) begins with a single `/`, node \( v_i \) must be child of the root
  - in case of \( \text{Step}_i // \text{Step}_r \), node \( v_i \) is the parent of \( v_r \)
  - in case of \( \text{Step}_i // \text{Step}_r \), node \( v_i \) is an ancestor of \( v_r \)

XSLT Patterns: Semantics

- \( \text{match} = \text{sect-head | section/head} \) 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 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
2. Serialize the result tree (as XML, HTML or text)

Overview of XSLT Transformation

- Stylesheet
- Source Document
- Source Tree
- Transformation Process
- Result Tree
- Output Process
- XML
- HTML
- Text

Result Tree Construction (roughly)

ResultTree = ApplTemps([root of the source tree]);

proc ApplTemps(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
  ApplTemps(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 traversal if no matching rules are found

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>
    <B>
      <C>
        <D>
          b1
        </D>
      </C>
      cc
      <B>b2</B>
    </B>
  </A>
  with the below rules:
  <xsl:template match="/"> <!-- Rule 1 -->
    <C>:<D>:<B>
  </xsl:template>
  <xsl:template match="C"> <!-- Rule 2 -->
    <New>C:<New>:<D>:<B>
  </xsl:template>
  <xsl:template match="D">
    <New>:<New>:<D>:<B>
  </xsl:template>
  <xsl:template match="/">
  </xsl:template>
Processing example (2)

- The result
  \[
  \langle R \rangle \langle \text{New} \rangle \text{New: b1b3ccb2} \langle /\text{NewC} \rangle /\rangle
  \]

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

Processing example (3)

3. Instruction \( <\text{xsl:apply-templates select="../B" />} \) 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 \( \text{NewC} \).

4. Instruction \( <\text{xsl:apply-templates />} \) in the context of element node \( C \) selects its children, "cc" and \( \langle B \rangle b2 \langle /B \rangle \), 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)

Source

Result

Is it Really So Tricky?

- Seldom, fortunately
  - but a computer scientist wants to understand how a model works

- Often sufficient to give simple declarative rules for different cases, like

\[
\langle xsl:template match="para">\langle P \rangle <\text{xsl:apply-templates />} \langle /P \rangle \langle /xsl:template \rangle
\]

SDPL 2007 5: Overview of XSLT 49

SDPL 2007 5: Overview of XSLT 50

SDPL 2007 5: Overview of XSLT 51

SDPL 2007 5: Overview of XSLT 52