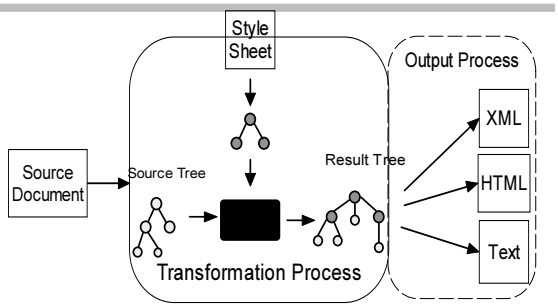
3 Document Transformations

- XSLT 1.0 (W3C Rec. 11/1999; 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**
    - » 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:

```
<xsl:template match="Pattern">
    Template    <!-- NB: well-formed! -->
</xsl:template>
```

conventional XSLT namespace prefix

- 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. 11/99 (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)
    - » XQuery (WD, Sept '05; extends XPath 2.0)
  - also typical operations on *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>  <BR>
      <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 -->
  <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>
```

Example (paras and emphs)

```
<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)

```
<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)

```
  <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>

</xsl:stylesheet>
```


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>  <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 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

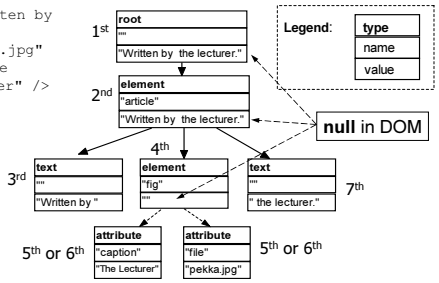
- Defined in Sect. 5 of the XPath specification
- Element nodes have elements, text nodes, comments and processing instructions of their (direct) **content** as their children
  - **NB**: attribute nodes are *not* children (but have a parent)
  - the **value** of an element node is the concatenation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

XSLT/XPath trees: Example

```
<article>Written by
<fig
  file="pekka.jpg"
  caption="The
      Lecturer" />
the lecturer.
</article>
```



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 paths

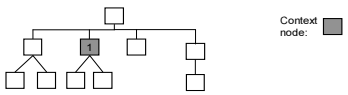
- Consist of *location steps* separated by '`/`'
  - each step produces a set of nodes
  - steps evaluated left-to-right, each node in turn as context node
    - » path begins with '`/`' –> root is the first context node
- Complete form of a location step:
  AxisName`::` *NodeTest* (`[`*PredicateExpr*`]`)*
  - *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`

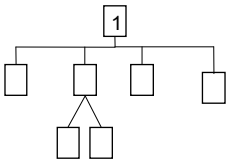
XPath Axes and Their Orientation

- 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 Axes and Their Orientation

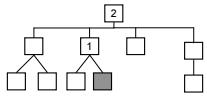
- `parent::` (exists for every node except the root)



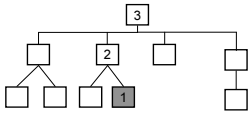
Context node: 

XPath Axes and Their Orientation

- `ancestor::`

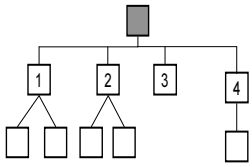


- `ancestor-or-self::`



XPath Axes and Their Orientation

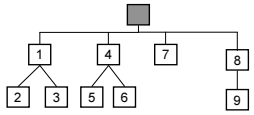
- `child::`



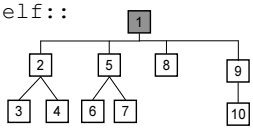
Context node: 

XPath Axes and Their Orientation

- `descendant::`

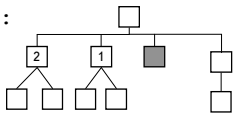


- `descendant-or-self::`

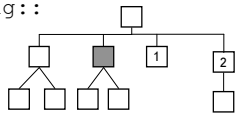


XPath Axes and Their Orientation

- `preceding-sibling::`

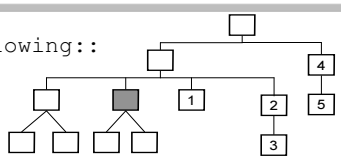


- `following-sibling::`

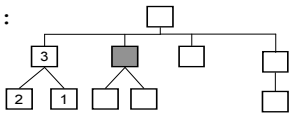


XPath Axes and Their Orientation

- `following::`



- `preceding::`



Location paths: Node tests

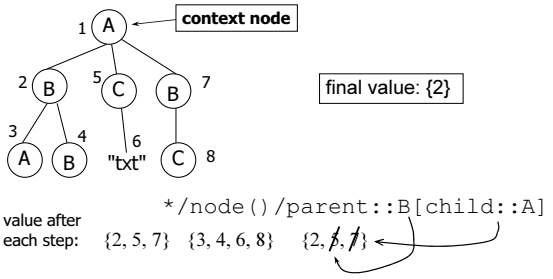
- Node tests (slightly simplified)
  - ***Name***: any **element** node with name *Name*
    (on an attribute axis, any attribute node with name *Name*)
  - **`*`**: any *element* (any *attribute* node on an attribute axis)
  - **`text()`**: any text node
    - » **`comment()`**: any comment node
  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

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

Location path examples (2)

- All child elements having an attribute `type`:
  `*[@type]`
  NB: Node sets as truth values: empty - **false**; non-empty - **true**
- All child elements of any `author` child:
  `author/*`
- `sect`ions 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]`

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 (Which parts are counted?)
- 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
  $Step_1 \oplus Step_2 \oplus \cdots Step_{n-1} \oplus Step_n$,
  where each separator ⊕ is either `/` or `//`
  - may also begin with `/`; Pattern `/` **matches** only the root
- Else *P* **matches a node** v_n iff there are nodes $v_n, \ldots, v_1$ such that each v_i satisfies the node test and possible predicates of $Step_i$, and which form a path towards the root:
  - If *P* begins with a single `/`, node v_1 must be child of the root
  - in case of $Step_{i-1}/Step_i$ node v_{i-1} is the parent of v_i
  - in case of $Step_{i-1}//Step_i$ node v_{i-1} is an ancestor of v_i

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

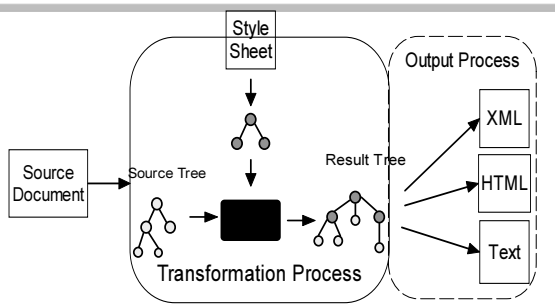
Main Aspects of XSLT

- 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
2. Serialize the result tree (as XML, HTML or text)

Overview of XSLT Transformation



Result Tree Construction (approximately)

ResultTree:= ApplTempls([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 traversal if no matching have been given rules

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>b1</B><C>cc<B>b2</B></C><D>dd</D><B>b3</B>
</A>
```
with the below rules:
```
<xsl:template match="/"> <!-- Rule 1 -->
    <R><xsl:apply-templates select="//C" /></R>
</xsl:template>

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


Processing example (2)

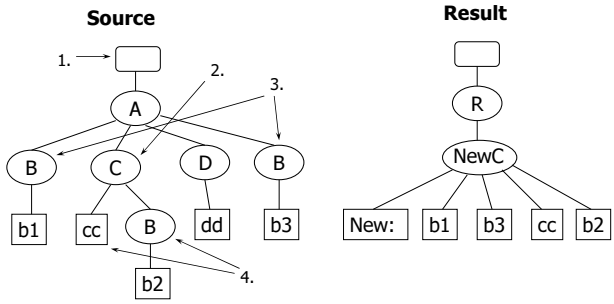
- The result
```
<R><NewC>New: b1b3ccb2</NewC></R>
```
is obtained as follows:

1. Rule 1 matches the root node –> Element node `R` is added to the result; Instruction `<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 `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?

- Fortunately seldom
  - 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

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