XSL

- eXtensible Stylesheet Language
  - XSLT
    - An XML vocabulary for transforming an XML document into another XML document
      - "Unlike with a programming language, you don't need to be a programmer to successfully describe how to transform your information. XSLT implements transformation "by example", not just "by program logic", and builds in support for the kinds of transformation typically needed to present information.
        - » Ha! (see quote next slide)
  - XPath
    - A language for addressing parts of XML documents
  - XSL Formatting Objects
    - An XML vocabulary to define XML display
    - very high-quality

When someone says,

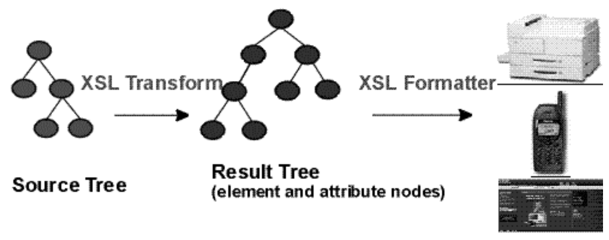
> ``I want a programming language in which I need only say what I wish done,"

give him a lollipop.

ALAN PERLIS, Epigrams on Programming (1982)

XSL Stages

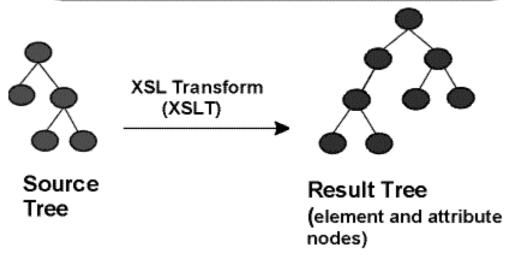
XSL Two Processes: Transformation & Formatting



Result XML tree is the result of XSLT processing.

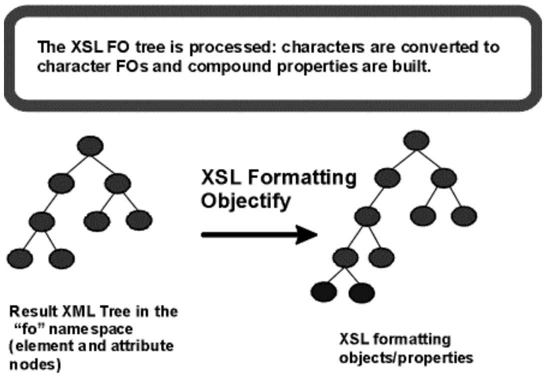
Tree Transformations

With tree transformation, the structure of the result tree can be quite different from the structure of the source tree



In constructing the result tree, the source tree can be filtered and reordered, and arbitrary structure and generated content can be added.

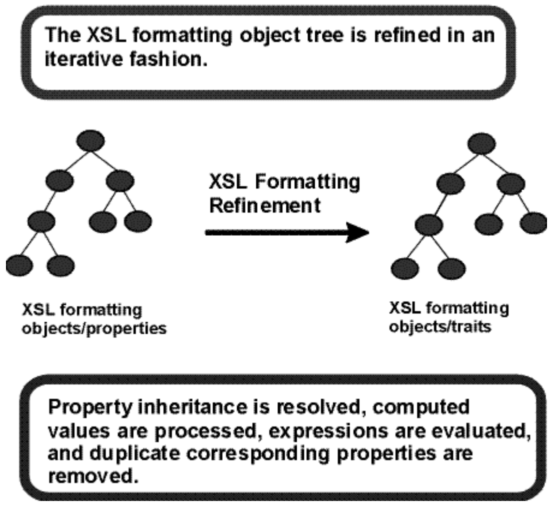
XSL Formatting



The XSL FO tree is processed: characters are converted to character FOs and compound properties are built.

XSL Formatting Objectify
Result XML Tree in the "fo" namespace (element and attribute nodes)
XSL formatting objects/properties

Refining the FO Tree



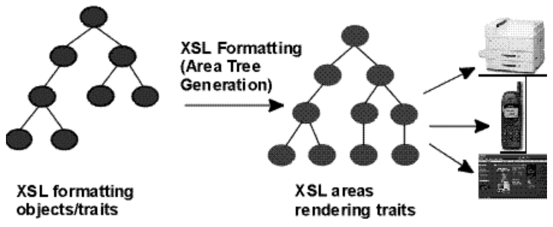
The XSL formatting object tree is refined in an iterative fashion.

Property inheritance is resolved, computed values are processed, expressions are evaluated, and duplicate corresponding properties are removed.

XSL Formatting Refinement
XSL formatting objects/properties
XSL formatting objects/traits

Generate the Area Tree

The last part of formatting describes the generation of a tree of geometric areas. These areas are positioned on a sequence of one or more pages.


XSL Formatting (Area Tree Generation)
XSL formatting objects/traits
XSL areas rendering traits

Each geometric area has a position on the page, a specification of what to display in that area and may have a background, padding, and borders.

XSLT Transformation

- Definitive specification
  - **XSL Transformations (XSLT) Version 1.0**

    W3C Recommendation 16 November 1999

    http://www.w3.org/TR/xslt
  - XSLT is a language for transforming XML documents into other XML documents
  - XSLT is designed to be used independently of XSL.
    - However, XSLT is not intended as a completely general-purpose XML transformation language.
    - Rather it is designed primarily for the kinds of transformations that are needed when XSLT is used as part of XSL.
  - (Target is really HTML)

Example

- cd_catalog.xml
- cd_catalog.xsl
- cd_catalog_with_xsl.xml

XSLT Structure

- XSLT Namespace

  `http://www.w3.org/1999/XSL/Transform`

  - 1999 refers to the year in which the URI was allocated by the W3C, not to the version of XSLT
- Stylesheet root element

  `<xsl:stylesheet version="1.0">`

  or

  `<xsl:transform version="1.0">`
- Combining stylesheets

  `<xsl:include href=`*uri-reference*`/>` (as if defined here)

  `<xsl:import href=`*uri-reference*`/>` (importer has priority)

Output Methods

- Various types of *output methods*
  - input
    - trivial.xml, trivial.xsl
    - html
      - default when first tag emitted is <html>
      - output of saxon
    - xml
      - output of saxon
    - text
      - output of saxon

XSL Templates

- *Templates* define how to output XML events
  - A *match* attribute is used to associate the template with an XML element
    - The value of the match attribute is a (super/subset) of an XPath expression.

```
<xsl:template match="/">
```

  - A template is always instantiated with respect to a *current node* and a *current node list*.
    - both are specified by the match

Template Processing Model

- A list of source nodes is processed to create a result tree fragment
  - by concatenating the result tree fragments for each node in the list.
- The entire result tree is constructed by processing a list containing just the root node.
- A node is processed by finding all the template rules with patterns that match that node, and choosing the best amongst them.
  - the chosen rule's template is then instantiated with the node as the current node, and with the list of source nodes as the current node list.
  - There is a default template if no matches found

Template Matching Example

- match.xml
- matchxsl.html

- "/" matches the root node
  - applies templates for all child nodes
- child node "/top" matches
  - emits h1
  - applies templates for selected child nodes
    - establishes a current node list as context

Defining templates

```
<xsl:template
  match="…"
  name="…"
  mode="…"
  >

  …

</xsl:template>
```

Invoking Templates

```
<xsl:apply-templates select="…"/>
```

- changes the current node and the current node list

```
<xsl:call-template name="…">
  <xsl:with-param name="…" select="…"/>
</xsl:call-template>
```

- does not change the current node or the current node list

Template Match Pattern

- Is a super/subset of the XPath language
- XPath is a separate standard intended for use by both XSLT and XPointer.
  - http://www.w3.org/TR/xpath

XPath

- Models an XML document as a tree of nodes.
  - root node
    - implicit top-level element (the *document root* is a child of this node, as are usually some PI nodes).
  - element nodes
  - attribute nodes
  - namespace nodes
  - processing instruction nodes
  - comment nodes
  - text nodes (maximal groupings of characters)

String-Value of Nodes

- Root Node , Element Nodes
  - concatenation of the SVs of all text-node descendants in document order
- Attribute Nodes
  - the normalized value of the attribute
- Namespace Nodes
  - the namespace URI
- Processing Instruction Nodes
  - the part following the target, up to and not including the terminating ?>
- Comment Nodes
  - all comment text not including the delimiters
- Text Nodes
  - the character data

XPath Context

- An XPath expression is evaluated to yield an object of type
  - node-set
    - an unordered collection of nodes without duplicates
  - boolean
  - number
  - string
- Expression evaluation occurs w.r.t. a context (determined by XSLT)
  - the context node
  - context position
  - context size

XPath Location Paths

LocationPath:

RelativeLocationPath | AbsoluteLocationPath

AbsoluteLocationPath:

'/' RelativeLocationPath?

RelativeLocationPath:

Step | RelativeLocationPath '/' Step

XPath Location Steps

- Selects a set of nodes
  - an axis
    - specifies the tree relationship between the nodes selected by the location step and the context node
  - a node test
    - specifies the node type and the names of the nodes selected by the location step
  - zero or more predicates
    - arbitrary expressions to further refine the set of nodes selected by the location step

Step:

AxisSpecifier NodeTest Predicate*

Axis Specifier

AxisSpecifier:

AxisName '::'

Axes

• child	all children of the current node
• descendant	no attribute or namespace nodes
• parent	single parent, if there is one
• ancestor	parent, parent's parent, …, root node
• following-sibling	empty for attr & namepsace nodes
• preceding-sibling	"
• following	no descendants, attr or namespace nodes
• preceding	no ancestors, attr or namespace nodes
• attribute	attributes of the current node empty for non-element nodes
• namespace	namespace defined in the current node
• self	just the context node itself
• descendant-or-self	self + descendants
• ancestor-or-self	self + ancestors

NodeTests

NodeTest:

NameTest | NodeTypeTest

namespaces

NameTest:

'*' | name ':*' | (name ':')? name

NodeTypeTest:

'comment()' | 'text()' |

'processing-instruction(' [literal]? ')' |

'node()' *- true for any type of node whatsoever*

13 - XSLT CSC309 25

Examples

- child::para selects the para element children of the context node
- child::* selects all element children of the context node
- child::text() selects all text node children of the context node
- child::node() selects all the children of the context node, whatever their node type
- attribute::name selects the name attribute of the context node

13 - XSLT CSC309 26

More Examples

- attribute::* selects all the attributes of the context node
- descendant::para selects the para element descendants of the current node
- ancestor::div selects all div ancestors of the context node
- ancestor-or-self::div selects the div ancestors of the context node and, if the context node is a div element, the context node as well

More Examples

- descendant-or-self::para selects the para element descendants of the context node and, if the context node is a para element, the context node as well
- self::para selects the context node if it is a para element, and otherwise selects nothing
- child::chapter/descendant::para selects the para element descendants of the chapter element children of the context node
- child::*/child::para selects all para grandchildren of the context node

More Examples

- / selects the document root (which is always the parent of the document element)
- /descendant::para selects all the para elements in the same document as the context node
- /descendant::olist/child::item selects all the item elements that have an olist parent and that are in the same document as the context node

Predicates

Predicate:

'[' PredicateExpn ']' *(full expression language here)*

- Usually will wind up with comparisons to function values, e.g.,
  - number last()
  - number position()
  - boolean starts-with(string,string)
  - string name()

Expressions as Predicates

- A general expression can be evaluated as a predicate
  - First converted to boolean as follows:
    - if the result is a number:
      - true iff number = context position
      - para[3] is equivalent to para[position()=3]
    - if the result is node-set
      - true iff non-empty
    - if the result is a string
      - true iff length is not zero

Predicate Examples

- child::para[position()=1] selects the first para child of the context node
- child::para[position()=last()] selects the last para child of the context node
- child::para[position()=last()-1] selects the last but one para child of the context node
- child::para[position()>1] selects all the para children of the context node other than the first para child of the context node

More Examples

- following-sibling::chapter[position()=1] selects the next chapter sibling of the context node
- preceding-sibling::chapter[position()=1] selects the previous chapter sibling of the context node
- /descendant::figure[position()=42] selects the forty-second figure element in the document
- /child::doc/child::chapter[position()=5]/child::section[position()=2] selects the second section of the fifth chapter of the doc document element

More Examples

- child::para[attribute::type="warning"] selects all para children of the context node that have a type attribute with value warning
- child::para[attribute::type='warning'][position()=5] selects the fifth para child of the context node that has a type attribute with value warning
- child::para[position()=5][attribute::type="warning"] selects the fifth para child of the context node if that child has a type attribute with value warning

More Examples

- child::chapter[child::title='Introduction'] selects the chapter children of the context node that have one or more title children with string-value equal to Introduction
- child::chapter[child::title] selects the chapter children of the context node that have one or more title children
- child::*[self::chapter or self::appendix] selects the chapter and appendix children of the context node
- child::*[self::chapter or self::appendix][position()=last()] selects the last chapter or appendix child of the context node

Abbreviated Syntax

- child:: is the default
- attribute:: can be abbreviated to @
- // is short for /descendant-or-self::node()/
- . is short for self::node()
- .. is short for parent::node()

Abbreviated Syntax Examples

- para selects the para element children of the context node
- * selects all element children of the context node
- text() selects all text node children of the context node
- @name selects the name attribute of the context node
- @* selects all the attributes of the context node
- para[1] selects the first para child of the context node
- para[last()] selects the last para child of the context node

More Examples

- */para selects all para grandchildren of the context node
- /doc/chapter[5]/section[2] selects the second section of the fifth chapter of the doc
- chapter//para selects the para element descendants of the chapter element children of the context node
- //para selects all the para descendants of the document root and thus selects all para elements in the same document as the context node
- //olist/item selects all the item elements in the same document as the context node that have an olist parent

More Examples

- . selects the context node
- .//para selects the para element descendants of the context node
- .. selects the parent of the context node
- ../@lang selects the lang attribute of the parent of the context node
- para[@type="warning"] selects all para children of the context node that have a type attribute with value warning
- para[@type="warning"][5] selects the fifth para child of the context node that has a type attribute with value warning
- para[5][@type="warning"] selects the fifth para child of the context node if that child has a type attribute with value warning

More Examples

- chapter[title="Introduction"] selects the chapter children of the context node that have one or more title children with string-value equal to Introduction
- chapter[title] selects the chapter children of the context node that have one or more title children
- employee[@secretary and @assistant] selects all the employee children of the context node that have both a secretary attribute and an assistant attribute

Template Match Pattern

- Similar to XPath expressions, but
  - allows top-level '|' signs
  - only 'child' and 'attribute' axes allowed
  - allows '//' for descendants
  - allows id(name) to start the expression

XSL Variables and Parameters

- <xsl:variable name="…" select="…"/>
  - value of select is the value of the variable
  - if no select, then value is the tree contained within the element.
- <xsl:param name="…" select="…"/>
  - value of select if the default value of the parameter, if no other value is specified on call
  - can have top-level params, in which case the way it gets its initial value is implementation defined.
  - can declare inside templates as well

Repetition

```
<xsl:for-each select="…">
  …
</xsl:for-each>
```

Selection

```
<xsl:if test="…">
  …
</xsl:if>

<xsl:choose>
  <xsl:when test="…"> … </xsl:when>
  <xsl:when test="…"> … </xsl:when>
  …
  <xsl:otherwise> … </xsl:otherwise>
</xsl:choose>
```

Sorting

- Add one or more <xsl:sort> elements as children of <xsl:apply-templates> or <xsl:for-each>

```
<xsl:sort
   select = string-expression    (sort key)
   lang = nmtoken
   data-type = "text" | "number"
   order = "ascending" | "descending"
   case-order = "upper-first" | "lower-first" />
```

  - sort.xml, raw sort.xml, sort.xsl

Creating Nodes

```
<xsl:element name="…"/>
   …
</xsl:element>

<xsl:attribute name="…">
   …
</xsl:attribute>

<xsl:text disable-output-escaping="yes"|"no">
   …
</xsl:text>
```

Numbering

```
<xsl:number
    level = "single" | "multiple" | "any"
    count = pattern
    from = pattern
    value = number-expression
    format = { string }
    lang = { nmtoken }
    letter-value = { "alphabetic" | "traditional" }
    grouping-separator = { char }grouping-size = { number }
/>
```

- Can implement complex hierarchical numbering schemes