Information Systems

XSLT and XPath

Temur Kutsia

Research Institute for Symbolic Computation
Johannes Kepler University of Linz, Austria
kutsia@risc.uni-linz.ac.at

Outline

XSLT

XPath

XSLT

- Almost all applications that processes XML perform a transformation of some kind.
- Extensible Stylesheet Language Transformations (XSLT): The key technology to perform these transformations.
- XSLT can be used:
  - to transform one kind of XML grammar to another kind
  - to map XML documents to output documents that do not strictly follow the rules of proper XML document (e.g., HTML).

How to Use XSLT

- Write rules, called templates.
- Match templates against elements in your input XML.
- The templates work by mapping XML tags and data from your input document to new and different tags of your choice in an output document

Simple Application of XSLT

- Transform an XML grammar into an HTML document.
- The resulting document is viewable in an web browser.
- Three players in the transformation: The input XML, XSLT stylesheet, the output document.

The Input XML

```
<?xml version="1.0"?>
<message>Howdy!</message>
```

XML input file (data.xml)

The XSLT Stylesheet

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <!-- one rule, to transform the input root (/) -->
  <xsl:template match="/">
    <html><body>
      <!-- select message text using an XPath statement -->
        <h1><xsl:value-of select="./message/text()"/></h1>
    </body></html>
  </xsl:template>
</xsl:stylesheet>
```

XSLT stylesheet for transforming XML into HTML (render.xsl)

We can verify that both data.xml and render.xsl are valid XML documents.

Generating Output

- XSLT processor has to be installed.
- We use Saxon (on the .NET platform): http://saxon.sourceforge.net/.
- Command that transforms data.xml by render.xsl into HTML. The output is written in file out.html:

```
bin\Transform -t data.xml render.xsl > out.html
```

- (Full path information has to be included for the files involved.)
- Output of the transformation:

```
<html><body><h1>Howdy!</h1></body></html>
```

- Output can be viewed in a browser.

How XSLT Works

- XSLT templates act as rules that match against a source XML.
- When matched, the templates create fragments of output, usually based on values from the XML input document.
- In render.xsl, the template generates HTML elements `<html>`, `<body>`, and `<h1>`.
- The message text came from the source document.
- To retrieve the message, we used the XSLT instruction `xsl:value-of`.
- The instruction finds values based on an XPath query (explained later).

How XSLT Works

- Templates use the `xsl:apply-templates` instruction to request that additional parts of the input XML be transformed.
- It might work a a cascade: firing more templates, generating more output and firing more templates, and so on.
- This cascade of activity is the basic mechanism by which XSLT generates output given an input document.

How XSLT Works

- XSLT stylesheet: A list of rules.
- Can be accompanied by a few top-level instructions for general issues such as what type of character encodings the document should use.

Schematic Layout of an XSLT File

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Top level instructions (e.g., encoding of output data)

```
  <xsl:template match="first match condition">
    instructions for the first rule
  </xsl:template>

  <xsl:template match="second match condition">
    instructions for the second rule
  </xsl:template>
...
</xsl:stylesheet>
```

XPath in XSLT

- The XPath query language is used throughout XSLT.
- Examples:
  - The value of the `xsl:template match` attribute which starts the processing chain at the root of the XML document:
    `<xsl:template match="/">`
  - The value of the `xsl:value-of select` attribute, which actually extracts the character data between `<text>` and `</text>` tags:
    `<xsl:value-of select="./message/text()">`
- These values are XPath expressions, called location paths.
- `text()` is the XPath function used to specify the text contained within the `<message>` element.

Namespaces in XSLT

- XSLT instructions are prefixed with `xsl:`.
- Any template element without this prefix will simply be written to the destination element rather than executed.
- The `xsl:stylesheet` line in your XSLT file should read exactly as

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

- URI `http://www.w3.org/1999/XSL/Transform` is case-sensitive!
- `xsl` prefix is just a convention: Any prefix can be used to qualify XSLT instructions as long as they are associated with the correct XSLT namespace.

Basic XSLT

- Fundamentals of XSLT:
  - how to write templates;
  - how they work together to create output.
- XSLT instructions are identified using the `xsl` prefix (convention).
- An element in the stylesheet not tagged with `xsl` is considered an output element.

Concepts: Applying Templates

Example

- Given: a list of messages in an XML document.
- Goal: write them out as an HTML numbered list.
- The messages:

```
<?xml version="1.0" ?>
<system>
  <stamp>12-03-02 23:13</stamp>
  <msgs>
    <msg type="info">System started</msg>
    <msg type="info">Logging in user maryk</msg>
    <msg type="info">User 'bobm' not found</msg>
  </msgs>
  ...
</system>
```

Concepts: Applying Templates

Example (Cont.)

- The root element can be matched automatically by the XSLT processor. Let's use this rule as a starting point:

```
<xsl:template match="/system">
 <html><body style="font:normal larger tahoma">
 <h3>Log started:
    <xsl:value-of select="./stamp/text()"/>
 </h3>
 <ol><xsl:apply-templates select="./msgs/msg"/></ol>
 </body></html>
</xsl:template>
```

- `<system>`: the base element of the template.
- The contents of the `<stamp>` element is outputted using the `xsl:value-of select` command within an `<ol>` element.
- `xsl:apply-templates` kicks off the transformation template responsible for the nodes matching the `select` attribute (i.e., all `<msg>` statements).

Concepts: Applying Templates

Example (Cont.)

- The instruction

  ```
  <xsl:apply-templates select="./msgs/msg"/>
  ```

  is saying, "fire the template that handles the `<msg>` elements that are descendents of `<msgs>`."

- The corresponding "match" for this select is

  ```
  <xsl:template match="msg">
    <li><xsl:value-of select="./text()"/></li>
  </xsl:template>
  ```

- This template is called by the `xsl:apply-templates` every time a `<msg>` element is encountered by the processor.

Concepts: Applying Templates

Example (Cont.)

- Final XSLT stylesheet:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/system">
  <html><body style="font:normal larger tahoma">
  <h3>Log started:
       <xsl:value-of select="./stamp/text()"/></h3>
  <ol>
    <xsl:apply-templates select="./msgs/msg"/>
  </ol>
  </body></html>
</xsl:template>
<xsl:template match="msg">
  <li><xsl:value-of select="./text()"/></li>
</xsl:template>
</xsl:stylesheet>
```

Concepts: Applying Templates

Example (Cont.)

- Result of transformation:

```
<html>
  <body style="font:normal larger tahoma">
    <h3>Log started:  12-03-02 23:13</h3>
    <ol>
      <li>System started</li>
      <li>Logging in user maryk</li>
      <li>User 'bobm' not found</li>
    </ol>
  </body>
</html>
```

Concepts: Applying Templates

Example (Cont.)

Summarizing the example:

- The task was to render a given XML document in a browser.
- To accomplish the task, we avoided any programming and instead used XSLT to create a transformation stylesheet.
- When our XML file is passed through our XSL file, the <msg> elements in the source file are translated into HTML tags.
- These tags are output into an HTML file.

Context

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/system">
  <html><body style="font:normal larger tahoma">
  <h3>Log started:
       <xsl:value-of select="./stamp/text()"/></h3>
  <ol>
    <xsl:apply-templates select="./msgs/msg"/>
  </ol>
  </body></html>
</xsl:template>
<xsl:template match="msg">
  <li><xsl:value-of select="./text()"/></li>
</xsl:template>
</xsl:stylesheet>
```

Context

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/system">
  <html><body style="font:normal larger tahoma">
  <h3>Log started:
       <xsl:value-of select="./stamp/text()"/></h3>
  <ol>
    <xsl:apply-templates select="./msgs/msg"/>
  </ol>
  </body></html>
</xsl:template>
<xsl:template match="msg">
  <li><xsl:value-of select="./text()"/></li>
</xsl:template>
</xsl:stylesheet>
```

- `"/system"` fixes the context at which the template is fired.
- Navigating the context is like navigating in a file directory.

Context

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/system">
  <html><body style="font:normal larger tahoma">
  <h3>Log started:
       <xsl:value-of select="./stamp/text()"/></h3>
  <ol>
    <xsl:apply-templates select="./msgs/msg"/>
  </ol>
  </body></html>
</xsl:template>
<xsl:template match="msg">
  <li><xsl:value-of select="./text()"/></li>
</xsl:template>
</xsl:stylesheet>
```

- `"./stamp/text()"`: We are interested in the data stored in the `<stamp>` element, which also resides in `<system>`.
- `"."` indicates relative to the current context.

Context

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/system">
  <html><body style="font:normal larger tahoma">
  <h3>Log started:
       <xsl:value-of select="./stamp/text()"/></h3>
  <ol>
    <xsl:apply-templates select="./msgs/msg"/>
  </ol>
  </body></html>
</xsl:template>
<xsl:template match="msg">
  <li><xsl:value-of select="./text()"/></li>
</xsl:template>
</xsl:stylesheet>
```

- `<xsl:apply-templates select="./msgs/msg"/>` says: “Relative to where I am now, there are is am `<msgs>` element containing one or more `<msg>` elements.”
- “Find a template that knows how to handle `<msg>` elements, and fire it.”

Context

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/system">
  <html><body style="font:normal larger tahoma">
  <h3>Log started:
       <xsl:value-of select="./stamp/text()"/></h3>
  <ol>
    <xsl:apply-templates select="./msgs/msg"/>
  </ol>
  </body></html>
</xsl:template>
<xsl:template match="msg">
  <li><xsl:value-of select="./text()"/></li>
</xsl:template>
</xsl:stylesheet>
```

- `<xsl:template match="msg">`: Start tag for the template that that knows how to transform `<msg>` elements.

Concepts: Accessing Attributes

Example

- Process inventory records having the following form in XML:

```
<item id="31741", q="12">tshirt</item>
<item id="31752", q="19">banner</item>
```

- Goal: output these records as HTML:

```
<div>31741 (tshirt):  12</div>
<div>31752 (banner):  19</div>
```

- How? Extract values of attributes:

```
<xsl:template match="item">
  <div>
    <xsl:value-of select="@id"/>
    (<xsl:value-of select="./text()"/>):
    <xsl:value-of select="@q"/>
  </div>
</xsl:template>
```

Concepts: Wildcards

Example

- The wildcard character “*” is used when we wish to match against any element or attribute.
- Given: An XML file where the <name> element occurs in different places:

```
<employees>
  <manager><name>Jennifer Lo</name></manager>
  <vp><name>Caldera Peng</name></vp>
  <developer><name>Familia Muesli</name></developer>
</employees>
```

- Handle all employee names in a single template in an identical fashion:

```
<xsl:apply-templates select="/employees/*/name"/>
```

Concepts: Default Templates

- What happens when `xsl:apply-templates` is invoked but no matching template exists?
- Default templates “catch all”: Take all values from the unmatched nodes and pass them through as output.
- Implicit default rule for elements and the root node:

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

- Implicit default rule for text nodes and attributes:

```
<xsl:template match="text()|@*">
  <xsl:value-of select="."/>
</xsl:template>
```

Concepts: Accessing Parent Elements

Example

- The parent of the current context node is referenced by "..".
- Given:

```
<mentors>
  <mentor>sallym<sales>dough</sales></mentor>
  <mentor>samp<ops>peters</ops></mentor>
  <mentor>bobg<sales>jillp</sales></mentor>
</mentors>
```

- Output a list that includes only the new sales employees and their mentors:

```
<xsl:template match="/">
  <xsl:apply-templates select="mentors/mentor/sales"/>
</xsl:template>
<xsl:template match="sales">
  <div><xsl:value-of select="./text()"/>'s mentor is
  <xsl:value-of select="../text()"/></div>
</xsl:template>
```

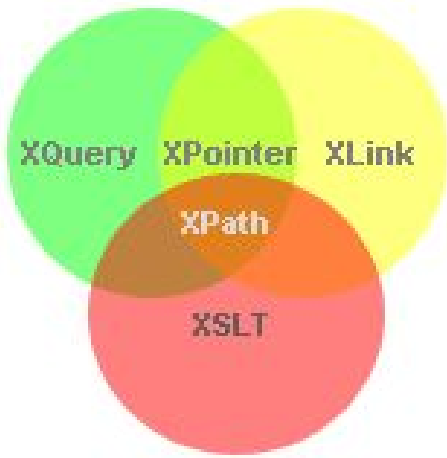
Concepts: Recursive Descent

- The recursive decent operator “//” can be used to find all nodes regardless of their notation.
- `<xml:apply-templates select="//"/>`: Act on all nodes in the document.
- `<xml:apply-templates select=".//"/>`: Act on all descendants of the current context node.
- “//” is used occasionally. An expensive operation.

XSLT: Brief Summary

- A template's context determines the location in the XML in order to
  - extract values using `xsl:value-of`,
  - apply additional templates using `xsl:apply-templates`,
  - etc.
- When `xsl:apply-templates` is used, an XSLT template is "fired" or, if none matches, the default template rules are used.
- XPath is used extensively in XSLT to select nodes.

XPath



What is XPath?

- XPath is a language whose primary purpose is to provide common syntax and functionality to address parts of XML documents.
- XPath uses path expressions to navigate in XML documents.
- XPath contains a library of standard functions.
- XPath is a major element in XSLT.

XPath

- XPath operates on the logical structure of an XML document and uses a syntax that resembles to the path constructions in URIs.
- XPath models an XML document as a tree of nodes (e.g. elements, attributes, namespaces, etc.)
- XPath expressions can compute strings, numbers, sets of nodes from the data of XML documents.

XPath Data Model

- XPath models an XML document as a tree of nodes of the types: root, element, text, attribute, namespace, processing instruction, comment.
- The tree structure is an extension of the tree structure of the element nesting of the XML document.
- The nodes can be considered in document order, which is given by the order of the first characters of the nodes in the XML document.

```
<folder>
  <email date='20 Aug 2003'> ...  </email>
  <email date='21 Aug 2003'> ...  </email>
</folder>
```

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<folder>
  <email date='20 Aug 2003'>
    <from>robert@company.com</from>
    <to>oliver@company.com</to>
    <subject>Meeting</subject>
    Could we meet ...
  </email>
  <email date='21 Aug 2003'>
    <from>oliver@company.com</from>
    <to>rob@company.com</to>
    <cc>amy@company.com</cc>
    <subject>Re:  Meeting</subject>
    On 20 Aug 2003 rob@company.com wrote ...
  </email>
</folder>
```

String Values

Every node has a string value, which is computed as follows:

- For root and element nodes it is the concatenation of string values of the descendants in document order.
- For attribute nodes it is the value of the attribute.
- For text nodes it is just the character data contained by them.
- For namespace nodes it is the corresponding URI.
- For comment nodes it is the text of the comment

Location Paths

- Location paths are special expressions for selecting a set of nodes.
- A location path consists of location steps composed together from left to right and separated by ’/’.
- An absolute location path is one that starts with a ’/’.
- Relative location paths are defined always with respect to the context node.

Example

The node selection is analogous to the file selection in a Unix-like file system.

```
../reports/*/summary
```

Location Paths

Example

Path Expression	Result
`bookstore`	Selects all the child nodes of the `bookstore` element.
`/bookstore`	Selects the root element `bookstore` Note: If the path starts with a slash (`/`) it always represents an absolute path to an element!

Location Paths

Example

Path Expression	Result
`bookstore/book`	Selects all `book` elements that are children of `bookstore`.
`//book`	Selects all `book` elements no matter where they are in the document.
`bookstore//book`	Selects all `book` elements that are descendant of the `bookstore` element, no matter where they are under the `bookstore` element.
`//@lang`	Selects all attributes that are named `lang`.

Predicates

- Predicates are used to find a specific node or a node that contains a specific value.
- Predicates are always embedded in square brackets.

Example

Path Expression	Result
`/bookstore/book[1]`	Selects the first `book` element that is the child of the `bookstore` element.
`/bookstore/book[last()-1]`	Selects the last but one `book` element that is the child of the `bookstore` element.
`/bookstore/book[position()<3]`	Selects the first two `book` elements that are children of the `bookstore` element.

Predicates

Example

Path Expression	Result
`//title[@lang='eng']`	Selects all the title elements that have an attribute named lang with a value of `'eng'`.
`/bookstore/book[price>35.00]`	Selects all the `book` elements of the `bookstore` element that have a `price` element with a value greater than 35.00.

Selecting Unknown Nodes

XPath wildcards can be used to select unknown XML elements.

Example

Wildcard	Result
`/bookstore/*`	Selects all the child nodes of the bookstore element.
`//*`	Selects all elements in the document.
`//title[@*]`	Selects all title elements which have any attribute.

Selecting Several Paths

By using the | operator in an XPath expression you can select several paths.

Example

Path Expression	Result
`//title | //price`	Selects all the `title` AND `price` elements in the document.

Location Steps

Location steps have the following parts:

- **axis**. It specifies the (in-tree) relationship between the context node and the nodes selected by the location step:

  Available axes: child, descendant, parent, ancestor, followingsibling, preceding-sibling, following, preceding, attribute, namespace, self, descendant-or-self, ancestor-or-self. (Used explicitly in “long notation”)
- **node test**. Specifies the node type for the nodes selected by the location step (separated by :: from the axis).
- **predicate**. It specifies further expressions with boolean value, to refine the selected node set (enclosed in [ ], described on before).

Example

- This location path (using the “long notation”) selects all attributes of all the email elements.

  ```
  /child::folder/child::email/attribute::*
  ```

- This selects only the attribute of the first email element in the XML document.

  ```
  /child::folder/child::*[position()=1]/@*
  ```

- What does this do?

  ```
  /folder/email/to[string()='rob@jku.at']/../@date
  ```

XPath Expressions

- Simple expressions: numerical and string literals, variable references, function calls.
- Values can be bound to variable names in XSLT. The value of `x` can be retrieved by `$x`.
- Basic arithmetic operations are available for numbers.
- More complex expressions are location paths and boolean expressions (e.g. using $<$, $>$, $!=$, $=$ and logical connectives `and`, `or`).
- Implicit coercion works from strings to numbers and from numbers to booleans as needed.