Introduction to XSLT

Sebastian Rahtz

July 2001
What is the XSL family?

- **XPATH**: a language for expressing paths through XML trees
- **XSLT**: a language for expressing transformation of XML
- **XSL FO**: an XML vocabulary for describing formatted pages

The XSLT language is

- Expressed in XML; uses namespaces to distinguish output from instructions
- Purely functional
- Reads and writes XML trees
- Designed to generate XSL FO
How is XSLT used? (1)

-With a command-line program to transform XML (eg to HTML)

-Downside: no dynamic content, user sees HTML
  -Upside: no server overhead, understood by all clients

-In a web server servlet, eg serving up HTML from XML

-Downside: user sees HTML, server overhead
  -Upside: understood by all clients, allows for dynamic changes
How is XSLT used? (2)

- In a web browser, displaying XML on the fly
  - Downside: most clients do not understand it
  - Upside: user sees XML
- Embedded in specialized program
- As part of a chain of production processes, performing arbitrary transformations
What do you mean, ‘transformation’?

Take this

<recipe>
  <title>Pasta for beginners</title>
  <ingredients>
    <item>Pasta</item>
    <item>Grated cheese</item>
  </ingredients>
  <cook>Cook the pasta and mix with the cheese</cook>
</recipe>

and make this

<html>
  <h1>Pasta for beginners</h1>
  <p>Ingredients: Pasta Grated cheese</p>
  <p>Cook the pasta and mix with the cheese</p>
</html>
How do you express that in XSL?

```xml
<xsl:stylesheet
   xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
   version="1.0">
<xsl:template match="/recipe">
   <html>
   <h1><xsl:value-of select="title"/></h1>
   <p>Ingredients: <xsl:apply-templates
   select = "ingredients/item"/></p>
   <p><xsl:apply-templates select = "cook"/></p>
   </html>
</xsl:template>
</xsl:stylesheet>
```
XSL Resources

Open source processors XT, Saxon, Xalan, libxslt, Sablotron, Transformiix; Perl/Python etc

Commercial (but free) processors Oracle XML, uxsl

Web browsers Internet Explorer 5 (upgrade), Mozilla (nearly)

FO processors FOP, PassiveTeX, XEP, Unicorn
How an XSLT processor works (1)

- An XSLT stylesheet has rules for transforming a source tree into a result tree.
- The transformation is achieved by associating patterns with templates.
- A pattern is matched against elements in the source tree.
- A template is instantiated to create part of the result tree, which is separate from the source tree.
- In constructing the result tree, elements from the source tree can be filtered and reordered, and arbitrary structure can be added.
How an XSLT processor works (2)

arResult when a template is instantiated, each instruction is executed and replaced by the result tree fragment that it creates. Instructions can select and process descendant source elements.

Note that elements are only processed when they have been selected by the execution of an instruction.

The result tree is constructed by finding the template rule for the root node and instantiating its template.
How an XSLT processor works (3)

When finding the applicable template rule, more than one template rule may have a pattern that matches a given element. However, only one template rule will be applied.

XSLT makes use of the XPath expression language for selecting elements for processing, for conditional processing and for generating text.

*Extension mechanisms* are defined for extending the set of instruction elements used in templates and for extending the set of functions used in XPath expressions.
<?xml version='1.0'?><cemetery><stone number="31">
  <person sex="m">
    <name>
      <fnm status="2">John</fnm><snm status="2">Keats</snm>
    </name>
    <born>
      <date><day>0</day><mon>0</mon><yr>0</yr></date>
    </born>
    <died>
      <date><day>24</day><mon>2</mon><yr>1821</yr></date>
    </died>
    <age>0</age>
    <nat status="1" idref="GB" />
  </person>
  <inscrip face="f_S" cond="c_1" manner="m_IF" type="t_P" lang="l_EN">
    <l p="c"><i>This Grave</i></l>
    <l p="c"><i>contains all that was Mortal, </i></l>
    <l p="c"><i>of a </i></l>
    <l p="c">YOUNG ENGLISH POET</l>
  </inscrip>
  <deco>
    <icon face="f_S" occs="1" type="it_B" key="i58"></icon>
  </deco>
</stone></cemetery>
Complete example: stylesheet

```xml
<?xml version='1.0'?>
<xsl:stylesheet
    xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
    version="1.0">
    <xsl:template match="/cemetery">
        <html>
            <head> <title>Protestant Cemetery Catalogue </title> </head>
            <body> <xsl:apply-templates/> </body>
        </html>
    </xsl:template>

    <xsl:template match="stone">
        <h1>Stone <xsl:value-of select="@number"/></h1>
        <ul>
            <xsl:apply-templates select="person"/>
        </ul>
    </xsl:template>

    <xsl:template match="person">
        <li><xsl:apply-templates select="name"/></li>
    </xsl:template>

</xsl:stylesheet>
```
XSLT constructs (1)

- Template match against element
  
  ```xml
  <xsl:template match="person">
    <xsl:apply-templates/>
  </xsl:template>
  ```

- Process all children
  
  ```xml
  <xsl:apply-templates/>
  ```

- Process selected children
  
  ```xml
  <xsl:apply-templates select="foo/bar"/>
  ```

- Process children separately
  
  First:
  ```xml
  <xsl:apply-templates select="foo"/>
  ```
  Second:
  ```xml
  <xsl:apply-templates select="bar"/>
  ```
XSLT constructs (2)

☞ Loop around a set of children

\[
<ol>
  <xsl:for-each select="item"/>
    <li><xsl:apply-templates/></li>
  </xsl:for-each>
</ol>

☞ Print an attribute

\[
<xsl:value-of select="@number"/>
\]

☞ Put a calculated value in an output attribute

\[
<a href="#S{@number}">
  <xsl:value-of select="@number"/>
</a>
\]

☞ Put out literal text (including spaces)

\[
<xsl:apply-templates select="item"/>
<xsl:text> !</xsl:text>
<xsl:apply-templates select="bar"/>
\]
XSLT constructs (3)

- **Template match against attributes**

```xml
<xsl:template match="person[@sex='M']">
  <xsl:apply-templates/>
</xsl:template>
```

- **Simple test**

```xml
<xsl:if test="@sex='M'">
  <xsl:apply-templates/>
</xsl:if>
```

- **Case statement**

```xml
<xsl:choose>
  <xsl:when test="@sex='M'">Its a boy!</xsl:when>
  <xsl:when test="@sex='F'">Its a girl!</xsl:when>
  <xsl:otherwise>
    Error in data: sex attribute has <xsl:value-of select="@sex"/>
  </xsl:otherwise>
</xsl:choose>
```
XSLT constructs (4)

Numbering. The default is to provide the sibling number, but you can also make it document-wide:

```xml
<xsl:template match="item">
  <xsl:number/>.  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="div3">
  <xsl:number level="multiple" count="div1|div2"/>.  
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="note">
  <xsl:number level="any"/>.  <xsl:apply-templates/>
</xsl:template>
```
XSLT constructs (5)

Sorting

```xml
<xsl:template match="/">
  <xsl:apply-templates select="name">
    <xsl:sort select="surname"/>
    <xsl:sort select="forename"/>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match="/">
  <ul>
    <xsl:for-each select="name">
      <xsl:sort select="surname"/>
      <xsl:sort select="forename"/>
      <li><xsl:apply-templates/></li>
    </xsl:for-each>
  </ul>
</xsl:template>
```
XSLT constructs (6)

Using built-in functions

```xml
<xsl:apply-templates select="id(@target)"/>
<xsl:value-of select="count(item)"/>
<xsl:value-of select="sum(item)"/>
<xsl:value-of select="substring-after(item,':')"/>
<xsl:if test="count(item) > 6">
  ....
</xsl:if>
```
XSLT constructs (7)

🔗 Using axes

```xml
<xsl:if test="not(preceding-sibling::item)">
  ....
</xsl:if>

<xsl:value-of select="count(descendant::footnote)"/>

<xsl:apply-templates
  select="ancestor::teiHeader//revisionDesc/list/item[1]/date"/>
```
**XPATH axes**

- self
- attribute (shorthand form: @)
- child (shorthand form: )
- descendant (shorthand form: / /)
- descendant-or-self
- ancestor
- ancestor-or-self
- namespace
- following
- preceding
- following-sibling
- preceding-sibling
- parent (shorthand form: ..)
All children

<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>
No children

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

<xsl:template match=""/">
  <xsl:apply-templates
      select = "cemetery/stone/inscrip"/>
</xsl:template>
Death dates of people on stones

```xml
<xsl:template match="/">
  <xsl:apply-templates
      select = "cemetery/stone/person/died"/>
</xsl:template>
```
"Person inside stone"

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

<xsl:template match="stone">
  <xsl:apply-templates select = "person"/>
</xsl:template>
```
First person on each stone

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

<xsl:template match="stone">
  <xsl:apply-templates select="person[1]"/>
</xsl:template>
```
Death dates of last person on first stone

```xml
<xsl:template match="/">
  <xsl:apply-templates
      select = "cemetery/stone[1]/person[last()]/died"/>
</xsl:template>

<xsl:template match="died">
  <xsl:value-of select = "day"/>
  <xsl:value-of select = "mon"/>
  <xsl:value-of select = "yr"/>
  <xsl:apply-templates select = "parent::person/born"/>
</xsl:template>
```
Illustration of axes
Conclusions

☛ XSLT is good at expressing typical documentation transformations
☛ XPATH has expressive power sufficient for most eventualities
☛ The range and number of implementations makes the standard secure
☛ XSLT is already widely used at all levels
☛ XSLT currently has problems with big documents