Gentle introduction to XSLT

Magdalena Turska
@magdaturska

April 2015

What is the XSL family?

- XPath: a language for expressing paths through XML trees
- XSLT: a programming language for transforming XML

These pdf slides were produces from XML with XSLT!

What is a transformation?

Take this:

```
<persName>
  <forename>Milo</forename>
  <surname>Casagrande</surname>
</persName>
<persName>
  <forename>Corey</forename>
  <surname>Burger</surname>
</persName>
<persName>
  <forename>Naaman</forename>
  <surname>Campbell</surname>
</persName>
```

and make this:

```
<item  n="1">
  <name>Burger</name>
</item>
<item  n="2">
  <name>Campbell</name>
</item>
<item  n="3">
  <name>Casagrande</name>
</item>
```

A text example

Take this

```
<div type="recipe" n="34">
 <head>Pasta for beginners</head>
 <list>
  <item>Pasta</item>
  <item>Grated cheese</item>
 </list>
 <p>Cook the pasta and mix with the cheese</p>
</div>
```

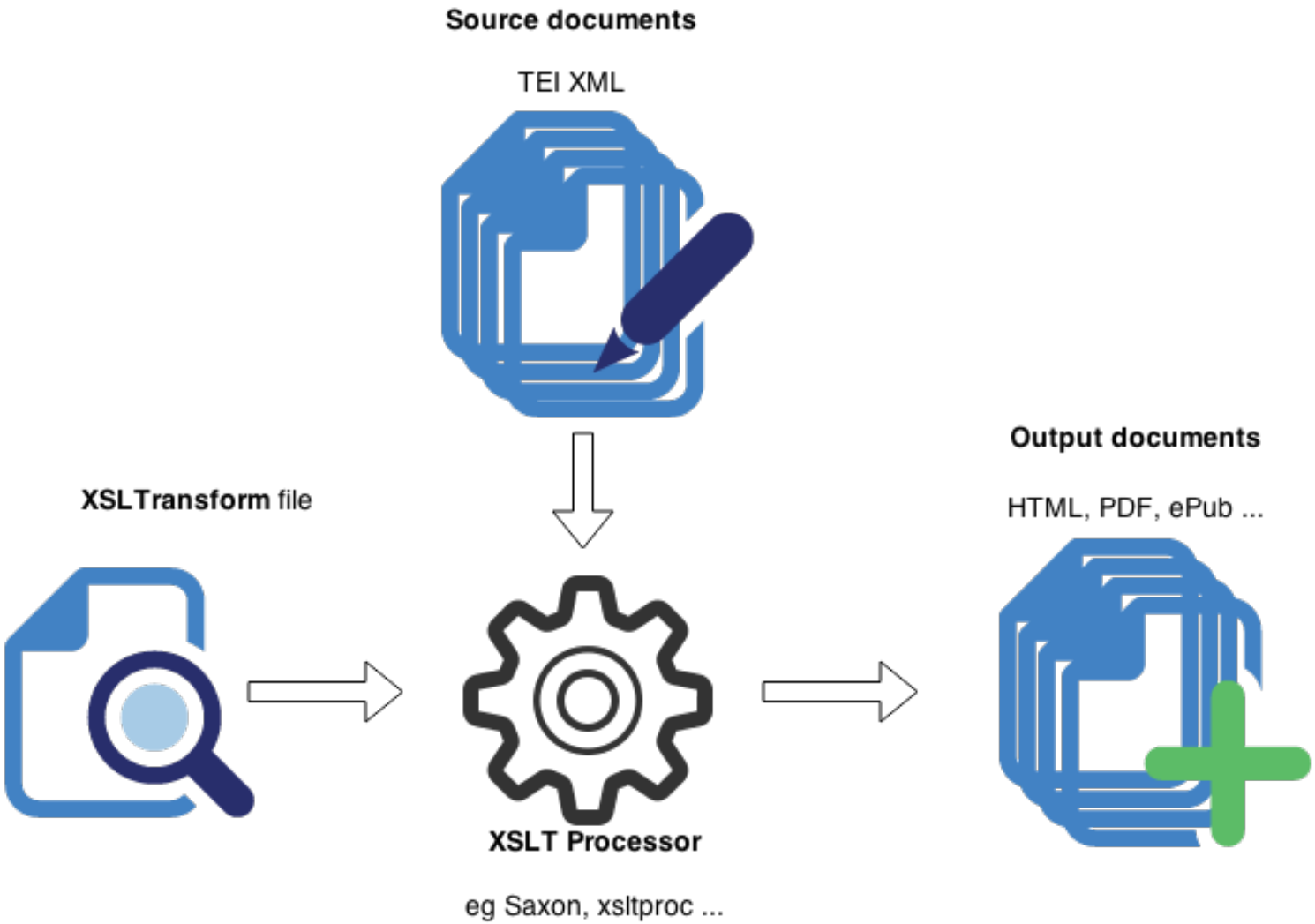
and make this

```
<html>
 <h1>34: 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?

```
<xsl:stylesheet xpath-default-namespace="http://www.tei-
c.org/ns/1.0"
 version="2.0">
 <xsl:template match="div">
  <html>
   <h1>
    <xsl:value-of select="@n"/>:
   <xsl:value-of select="head"/>
   </h1>
   <p>Ingredients:
   <xsl:apply-templates select="list/item"/>
   </p>
   <p>
    <xsl:value-of select="p"/>
   </p>
  </html>
 </xsl:template>
</xsl:stylesheet>
```

How does it work?



XSLT

The XSLT language is

- expressed in XML
- uses namespaces to distinguish output from instructions
- purely functional
- reads and writes XML trees

Widely used to generate HTML, but can be used to generate other formats as well: pdf, docx, KML.

How could XSLT be used?

- With a command-line program to transform XML (eg to HTML)
- In a web server, eg serving up HTML from XML (eg Cocoon, eXist)
- In a web browser, displaying XML on the fly (beware: not all clients understand it)
- Embedded in specialized program
- As part of a chain of production processes, performing arbitrary transformations

Structure of an XSL file

```
<xsl:stylesheet xpath-default-namespace="http://www.tei-
c.org/ns/1.0"
 version="2.0">
 <xsl:template match="div">
<!-- .... do something with div elements....-->
 </xsl:template>
 <xsl:template match="p">
<!-- .... do something with p elements....-->
 </xsl:template>
</xsl:stylesheet>
```

The `div` and `p` are *XPath expressions*, which specify which bit of the document is matched by the template.

Any element not starting with `xsl:` in a template body is put into the output.

Doesn't make sense? Let's try analogy!

Little house (on a prairie)

You want to build a house. What do you do?

- choose your project
- find your builder, give her your project
- wait for happy end

Project aka source document

```
<house>
 <foundation depth="100" unit="cm"/>
 <walls>
  <wall height="270" width="500"
   unit="cm" direction="se"/>
  <wall height="270" width="600"
   unit="cm" direction="nw"/>
  <wall height="270" width="500"
   unit="cm" direction="se"/>
  <wall height="270" width="600"
   unit="cm" direction="nw"/>
 </walls>
 <roof material="slate" angle="45"/>
 <happyend bill="gigantic"/>
</house>
```

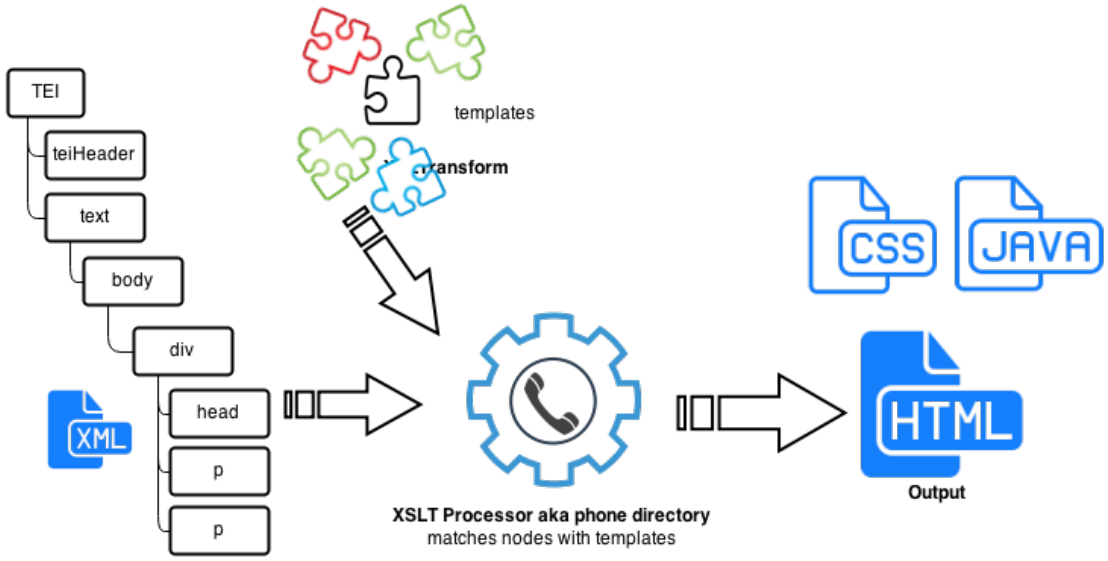
Little house - cd

Your builder builds you a house. What does she do?

- reads from the project, finds she needs to put up foundations
- calls up foundations specialist and waits until it's done
- reads further from the project, finds she needs to put up walls
- calls up walls specialist and waits until it's done
- reads further from the project, finds she needs to put up a roof (people these days)
- calls up roof specialist and waits until it's done
- reads further (or tries to) from the project but she reached the end, so tidies up, presents you with a bill and goes to Majorca for holiday

So maybe this is not how real-life construction works, but if you think your XSLT processor is your builder, source document is the project and templates are your

How does it work again?



The Golden Rules of XSLT

1. If there is no template matching an element, we process the elements inside it
2. If there are no elements to process by Rule 1, any text inside the element is output
3. Children elements are not processed by a template unless you explicitly say so:
4. `xsl:apply-templates select="XX"` looks for templates which match element "XX"; `xsl:value-of select="XX"` simply gets any text from that element
5. The order of templates in your program file is immaterial
6. You can process any part of the document from any template
7. Everything is well-formed XML. Everything!

Important magic

Our examples and exercises all start with two important attributes on <stylesheet>:

```
<xsl:stylesheet xpath-default-namespace="http://www.tei-
c.org/ns/1.0"
 version="2.0">....
</xsl:stylesheet>
```

which indicates

1. In our XPath expressions, any element name without a namespace is assumed to be in the TEI namespace
2. We want to use version 2.0 of the XSLT specification. This means that we must use the Saxon processor for our work.

A simple test file

```
<text>
 <front>
  <div>
   <p>Material up front</p>
  </div>
 </front>
 <body>
  <div>
   <head>Introduction</head>
   <p rend="it">Some sane words</p>
   <p>Rather more surprising words</p>
  </div>
 </body>
 <back>
  <div>
   <p>Material in the back</p>
  </div>
 </back>
</text>
```

Feature: apply-templates

```
<xsl:stylesheet version="2.0"
 xpath-default-namespace="http://www.tei-c.org/ns/1.0">
 <xsl:template match="/">
  <html>
   <xsl:apply-templates/>
  </html>
 </xsl:template>
</xsl:stylesheet>
```

```
<xsl:template match="TEI">
 <xsl:apply-templates select="text"/>
</xsl:template>
```

```
<xsl:template match="text">
 <h1>FRONT MATTER</h1>
 <xsl:apply-templates select="front"/>
 <h1>BODY MATTER</h1>
 <xsl:apply-templates select="body"/>
</xsl:template>
```

Feature: value-of

Templates for paragraphs and headings:

```
<xsl:template match="p">
 <p>
  <xsl:apply-templates/>
 </p>
</xsl:template>
<xsl:template match="div">
 <h2>
  <xsl:value-of select="head"/>
 </h2>
 <xsl:apply-templates/>
</xsl:template>
<xsl:template match="div/head"/>
```

Notice how we avoid getting the heading text twice.

Why did we need to qualify it to deal with just <head> inside ?

Example of context-dependent matches

Compare

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

with

```
<xsl:template match="div/head"> ...
</xsl:template>
<xsl:template match="figure/head"> ....
</xsl:template>
```

Pushing and pulling

XSLT stylesheets can be characterized as being of two types:

push In this type of stylesheet, there is a different template for every element, communication via <xsl:apply-templates> and the overall result is assembled from bits in each template. It is sometimes hard to visualize the final design. Common for document-oriented processing where the input document structure varies.

pull In this type, there is a master template (usually matching /) with the main structure of the output, and specific <xsl:for-each> or <xsl:value-of> commands to grab what is needed for each part. The templates tend to get large and unwieldy. Common for data-oriented processing where the

Do you push or pull a house together?

Remember our little house example? Let's say your project looks like this:

```
<bathroom>
 <tiles/>
 <bath>
  <tap/>
 </bath>
 <sink>
  <tap/>
 </sink>
</bathroom>
```

```
<xsl:template match="bathroom">
 <xsl:apply-templates select="tiles"/>
 <xsl:apply-templates select="bath"/>
 <xsl:apply-templates select="sink"/>
 <xsl:apply-templates select=".//tap"/>
</xsl:template>
<xsl:template match="sink"> Sing before you sink!
SPLISH! SPLASH! Done!
Note no calling apply-templates whatsoever, because sink
template cares not for its children!
</xsl:template>
<xsl:template match="tap"> Tap! Tap! Tap! Dance!
```

More complex patterns

The match attribute of <template> can point to any part of the document. Using XPath expressions, we can find:

`/`	the root of document (*outside* the root element)
`*`	any element
`text()`	
`name`	an element called ‘name’
`@name`	an attribute called ‘name’

Example of complete path in <value-of>:

```
<xsl:value-
of select="/TEI/teiHeader/fileDesc/titleStmt/title"/>
```

Attribute value template

What if we want to turn

```
<ref target="http://www.it.ox.ac.uk/">IT Services</ref>
```

into

```
<a href="http://www.it.ox.ac.uk/">IT Services</a>
```

? What we cannot do is

```
<xsl:template match="ref">
 <a href="@target">
  <xsl:apply-templates/>
 </a>
</xsl:template>
```

This would give the *@href* attribute the literal value '@target'.

For example

Instead we use {} to indicate that the expression must be evaluated:

```
<xsl:template match="ref">
 <a href="{@target}">
  <xsl:apply-templates/>
 </a>
</xsl:template>
```

This would give the *@href* attribute whatever value the attribute *@target* has!

Feature: for-each

If we want to avoid lots of templates, we can do in-line looping over a set of elements. For example:

```
<xsl:template match="listPerson">
  <ul>
    <xsl:for-each select="person">
      <li>
        <xsl:value-of select="persName"/>
      </li>
    </xsl:for-each>
  </ul>
</xsl:template>
```

contrast with

```
<xsl:template match="listPerson">
  <ul>
    <xsl:apply-templates select="person"/>
  </ul>
</xsl:template>
<xsl:template match="person">
  <li>
    <xsl:value-of select="persName"/>
  </li>
</xsl:template>
```

Feature: if

We can make code conditional on a test being passed:

```
<xsl:template match="person">
 <xsl:if test="@sex='1'">
  <li>
   <xsl:value-of select="persName"/>
  </li>
 </xsl:if>
</xsl:template>
```

contrast with

```
<xsl:template match="person[@sex='1']">
 <li>
  <xsl:value-of select="persName"/>
 </li>
</xsl:template>
<xsl:template match="person"/>
```

The *@test* can use any XPath facilities.

Feature: choose

We can make a multi-value choice conditional on what we find in the text:

```
<xsl:template match="person">
 <xsl:apply-templates/>
 <xsl:choose>
  <xsl:when test="@sex='1'">(male)
  </xsl:when>
  <xsl:when test="@sex='2'">(female)
  </xsl:when>
  <xsl:when test="not(@sex)">(no sex specified)
  </xsl:when>
  <xsl:otherwise>(unknown sex)
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

Feature: number

We can produce numbering based on the position of elements in the text.

1. Position within containing element:

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

2. Position within whole document:

```
<xsl:template match="p">
 <xsl:number level="any"/>
</xsl:template>
```

3. Position within an element further up the tree:

```
<xsl:template match="l">
 <xsl:number level="any" from="lg"/>
</xsl:template>
```

TEI Stylesheets

The TEI Consortium loosely owns and manages a family of XSLT stylesheets which operate on TEI XML documents. They can be used:

- to implement an ODD processor, generating schemas and documentation from TEI sources (this is what the Consortium primarily needs)
- to do general-purpose formatting of TEI XML to ‘human-readable’ formats like HTML, ePub, LaTeX, XSL FO
- to convert between TEI XML and Microsoft Word, and between TEI and Open Office, format
- to convert between TEI and some other XML formats (TEI P4, EEBO TCP, NLM, Docbook)
- to generate JSON, RDF, BibTeX and other strange formats

There is no one right way to render TEI documents

What do the Stylesheets packages actually provide?

- A set of XSLT 2.0 transformations which read and write TEI XML
- A set of Ant scripts to package the transforms
- A set of Unix shell scripts, calling on Ant, to perform all conversions
- Use "trang" library to generate XSD from RELAXNG
- Use a TeX install to write PDF
- Use "profiles" as containers for customization

Do not assume that the conversions cover every feature of the input and output formats!

TEI stylesheet availability

The XSLT files are available:

- for download from Sourceforge (`https://sourceforge.net/projects/tei/files/Stylesheets/`)
- within oXygen (in the TEI framework which can be updated separately from main oXygen)
- as Debian packages (for Linux users); see `http://tei.oucs.ox.ac.uk/teideb/`
- in OxGarage (see later)
- on Github (`http://www.github.com/TEIC/Stylesheets`)

Usage examples

1. in oXygen, I choose the transformation scenario called "TEI P5 DOCX"
2. On a command line line, I write

```
docxtotei test11.docx test.xml
```

3. Using Ant, I write

```
ant -f docx/build-from.xml -
DinputFile=`pwd`/Test/test11.docx -DoutputFile=test.xml -
lib lib/saxon9he.jar
```

4. If I have a way of sending the file using REST, it would go to

```
http://oxgarage.oucs.ox.ac.uk:8080/ege-
webservice/Conversions/docx%3Aapplication%3Avnd.openxmlformats-
officedocument.wordprocessingml.document/TEI%3Atext%3Axml/
```

TEI Stylesheet family top-level layout

Some of the directories for output formats

docx	Converting to and from Word OOXML
epub	Converting to ePub
fo	Making XSL FO
latex	Making LaTeX
nlm	Converting from NLM
odds	Transforming TEI ODD specifications
odt	Converting to and from OpenOffice Writer
slides	Making slides (HTML and PDF)
tite	Converting from TEI Tite
html	Making HTML

Special directories

profiles	Customizations
common	Templates for any output format