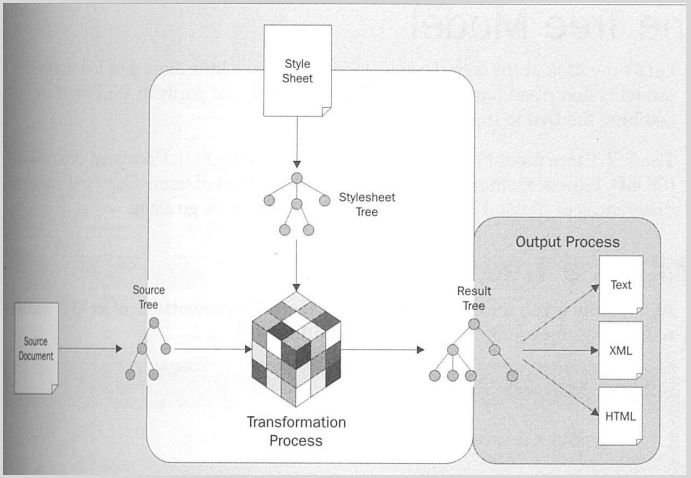
Tutorial: don't be afraid of XSLT

Berend de Boer – eurotex 2001

What an XSLT processor
XSLT basics
XSLT style sheets
Bonus
Tricks
Tips
close

What an XSLT processor



With an XSLT processor you can transform XML into:

1. XML.
2. XHTML.
3. Plain text; TEX macro's.
4. Calculate things like summaries for a 'spreadsheet'.

What an XSLT processor
XSLT basics
XSLT style sheets
Bonus
Tricks
Tips
close

XSLT basics

- ⋆ XSLT stands for ‘e**X**tensible **S**tylesheet **L**anguage: **T**ransformations.
- ⋆ XSLT is a language for transforming the structure of an XML document.
- ⋆ XSLT is itself XML.
- ⋆ XML is hierarchical, it consists of a root node, that has child nodes, that can have child nodes, etc. XSLT operates on those nodes. It can generates (result) nodes. Everything is a node.
- ⋆ Selecting or filtering things
- ⋆ It’s harder to learn than you think: basics are easy. Things you probably want to do have a significant learning curve.

What an XSLT processor
XSLT basics
XSLT style sheets
Bonus
Tricks
Tips
close

XSLT style sheets

XSLT code is written in the form of stylesheets.

There are four kinds of XSLT stylesheets:

1. Fill-in-the-blanks stylesheets.
2. Navigational stylesheets.
3. Rule-based stylesheets.
4. Computational stylesheets.

Fill-in-the-blanks

Stylesheet has the same structure as the desired output.

```
1  <xsl:stylesheet>
2  <xsl:template match="/">
3  \begin{document}

4  \chapter{<xsl:value-of
5              select="/conference/title"/>}

6  \section{<xsl:value-of
7              select="/conference/day[1]/title"/>}

8  \section{<xsl:value-of
9              select="/conference/day[1]/title"/>}

10 \end{document}
11 </xsl:template>
12 </xsl:stylesheet>
```

Navigational

Like fill-in-the-blanks, but uses named templates as subroutines to perform commonly-needed tasks.

```
1  <xsl:stylesheet>
2  <xsl:template match="/">
3  \begin{document}

4  \section{Day 1}
5  <xsl:call-templates name="presentation>
6  </xsl:call-templates>

7  \end{document}
8  </xsl:template>
9  </xsl:stylesheet>
```

Rule-based

Preferred format. Consists primary of rules describing how different features of the source document should be processed. It makes minimal assumptions about the structure of either the source document or the result document.

```
1  <xsl:stylesheet>

2  <xsl:template match="program">
3  \begin{document}
4  <xsl:apply-templates/>
5  \end{document}
6  </xsl:template>

7  <xsl:template match="day">
8  \section{<xsl:value-of select="@weekday"/>}
9  \begin{itemize}
10 <xsl:apply-templates/>
11 \end{itemize}
12 </xsl:template>
```

```
13  <xsl:template match="presentation">
14  \item <xsl:value-of select="title"/>
15  </xsl:template>

16  </xsl:stylesheet>
```

Computational

When there is a need to generate nodes in the result tree that do not correspond directly to nodes in the source tree.

This example counts the number of presentations per day:

```
1  <xsl:stylesheet>

2  <xsl:template match="program">
3  \begin{document}
4  <xsl:apply-templates/>
5  \end{document}
6  </xsl:template>

7  <xsl:template match="day">
8  \section{<xsl:value-of select="@weekday"/>}
9  Number of presentations: <xsl:value-of select="count(item/presen
10 </xsl:template>

11 </xsl:stylesheet>
```

This example outputs in XML the authors and per author the presentations they have. This XPath expression gives us all authors:

```
1 <xsl:for-each
2         select="day/item/presentation/author">
3   <xsl:value-of select="."/>
4 </xsl:for-each>
```

We want the authors only once of course.

```
1  <xsl:stylesheet>
2  <xsl:template match="program">
3  \begin{document}

4  \begin{enumerate}
5  <xsl:for-each
6         select="day/item/presentation/author
7                 [not(.=preceding::author)]">
8    <xsl:sort select="."/>
9    \item <xsl:value-of select="."/>
10     \begin{enumerate}
11       <xsl:for-each
12              select="//title
13                      [current()=preceding-sibling::author]">
14         \item <xsl:value-of select="."/>
15       </xsl:for-each>
16     \end{itemize}
17 </xsl:for-each>
18 \end{enumerate}
```

What an XSLT processor
XSLT basics
XSLT style sheets
Bonus
Tricks
Tips
close

```
19 \end{document}
20 </xsl:template>
21 </xsl:stylesheet>
```

Bonus

There are certain things you might want to do with `program.xsl`.

1. Strip of the ‘h’ or ‘u’ in the time attribute.
2. Writing a comma separated list of authors.
3. determine maximum number of columns required: this to determine final table parameter, i.e. how many l’s for example.

Strip last character

Let's strip the last character of the time attribute:

```
1  <item time="9.45h"><presentation/>
2  <break time="10.30h" type="coffee"/>
3  <item time="11.00u"><presentation/>
```

Main loop, doesn't strip anything yet:

```
1  <xsl:for-each select="//break|//item">
2    <xsl:value-of select="@time"/>
3    <xsl:text>&#xa;</xsl:text>
4  </xsl:for-each>
```

Writing a comma separated list of authors

Tricks

1. `<xsl:text>` to get white space.
2. `<xsl:output>` for text output.
3. encoding issues, so specify encoding.
4. Context versus current node is not clear to me. If one doesn't work I try the other (p437).

Tips

1. Everything in XSL is a node.
2. You can walk along any axis: children, parents, attributes (up, down, left, right).
3. A variable is a node.
4. If a variable is a node, that node is the root node, and the root node can have children.
5. You cannot update things (functional programming):
    1. Use recursion.
    2. Don't try to do two things at once (example).
6. Everything is a node, stupid.

What an XSLT processor
XSLT basics
XSLT style sheets
Bonus
Tricks
Tips
close