XSLT 2.0 vs XSLT 1.0

Jean-Michel HUFFLEN
LIFC — University of Franche-Comté
BachoTEX, 1st May 2008

Contents

Tools used

How XSLT does work?

Extended XPath expressions

Grouping elements

XSLT functions

Datatype management

Character maps

And now. . .

Tools used

1.0 libxslt, library of the GNOME project, written in C.

Tools used

1.0 libxslt, library of the GNOME project, written in C.

2.0 Saxon's open-source version:

exists in Java and C#.

How XSLT does work?

An XML node is matched by the `match` attribute of a template

How XSLT does work?

An XML node is matched by the `match` attribute of a template

in which case the template is invoked.

How XSLT does work?

An XML node is matched by the `match` attribute of a template

in which case the template is invoked.

The result expresses the transformation of the source XML text.

How XSLT does work?

An XML node is matched by the `match` attribute of a template

in which case the template is invoked.

The result expresses the transformation of the source XML text.

Possible modes: `text`, `xml`, `html`

How XSLT does work?

An XML node is matched by the `match` attribute of a template

in which case the template is invoked.

The result expresses the transformation of the source XML text.

Possible modes: `text`, `xml`, `html` $\oplus$ `xhtml`.

Version 1.0

Has succeeded,

Version 1.0

Has succeeded, is now widely used. OK. . .

Version 1.0

Has succeeded, is now widely used. OK. . . but:

some operations difficult to perform with XSLT 1.0's basic constructs:

Version 1.0

Has succeeded, is now widely used. OK. . . but:

some operations difficult to perform with XSLT 1.0's basic constructs:

$\Longleftarrow$ complicated methods or non-portable extension functions,

Version 1.0

Has succeeded, is now widely used. OK. . . but:

some operations difficult to perform with XSLT 1.0's basic constructs:

$\Longleftarrow$ complicated methods or non-portable extension functions,

$\Longrightarrow$ datatype management is only dynamic!

Version 1.0

Has succeeded, is now widely used. OK. . . but:

some operations difficult to perform with XSLT 1.0's basic constructs:

$\Longleftarrow$ complicated methods or non-portable extension functions,

$\Longrightarrow$ datatype management is only dynamic!

$\Longrightarrow$ replacing some characters: only a function

$$1 \xrightarrow{\texttt{translate}} 0 \text{ or } 1$$

Sequences in XPath 2.0

Every value is a sequence.

Atomic value $\Longleftarrow$ one-element sequence.

Sequences in XPath 2.0

Every value is a sequence.

Atomic value $\Longleftarrow$ one-element sequence.

XPath 2.0's expressions processing the whole of a sequence.

(exs in txpaths.xsl)

More expressive power

Conditional expressions.

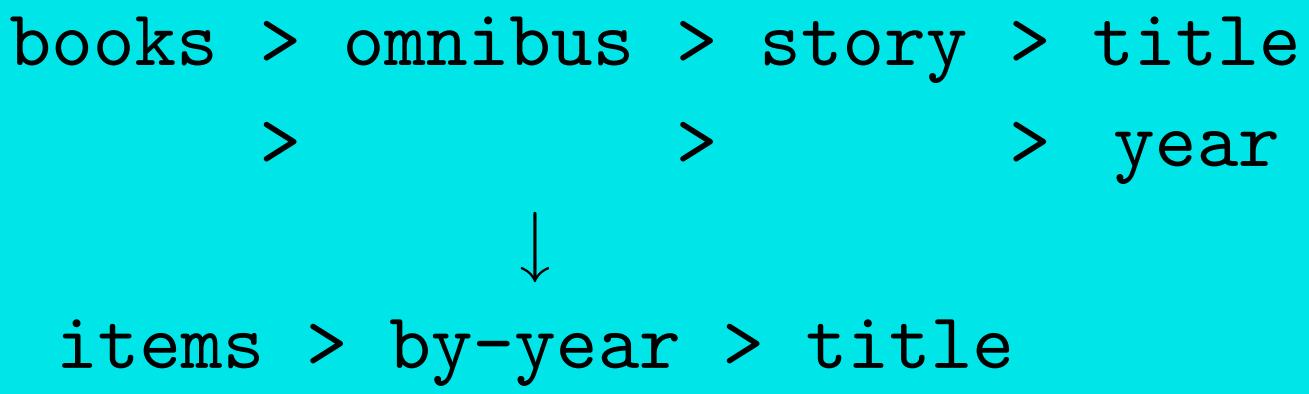
More expressive power

Conditional expressions.

Using regular expressions as in Perl.

(ex. in strip-out-f.xsl)

Changing elements' organisation



Grouping elements

XSLT 1.0 only way $\Longleftarrow$ using *keys*

Grouping elements

XSLT 1.0 only way $\Longleftarrow$ using *keys*

(as many keys as group levels!!)

ex. in grouping.xsl

Grouping elements

XSLT 1.0 only way $\Longleftarrow$ using *keys*

(as many keys as group levels!!)

ex. in grouping.xsl

XSLT 2.0 `xsl:for-each-group`

ex. in grouping-plus.xsl

Functions in XSLT 2.0

Maybe called within XPath expressions.

Functions in XSLT 2.0

Maybe called within XPath expressions.

Example: finding the rank of a month name.

ex. in months-f.xsl

Functions vs named templates

	F	*NT*
Names	namespace prefix	—
Call	XPath expression	`xsl:call-template` (`xsl:apply-templates`)
Arguments	mandatory	default values except required ones
Arity	Strict check	Additional arg. rejected except tunnel ones
Tunnel arg.	No	Possible

Datatypes

Basically:

`xsl:sequence` builds a *sequence*

`xsl:value-of .....` *text node*

Datatypes

Basically:

`xsl:sequence` builds a *sequence*
`xsl:value-of .....` *text node*

Conversions may be needed if datatypes are not declared:

Datatypes

Basically:

`xsl:sequence` builds a *sequence*
`xsl:value-of .....` *text node*

Conversions may be needed if datatypes are not declared:

`<year>2008</year>` [or `<... year="2008">...</...>`]

Don't be surprised by types!

```
<xsl:value-of select="year[1]"/> ⟸                    1999
<xsl:value-of select="year[2]"/> ⟸                       2
<xsl:value-of select="year[1] ge year[2]"/>
        ⟹
```

Don’t be surprised by types!

```
<xsl:value-of select="year[1]"/> ⇐=                    1999
<xsl:value-of select="year[2]"/> ⇐=                       2
<xsl:value-of select="year[1] ge year[2]"/>
        =⇒false
```

Don't be surprised by types!

```
<xsl:value-of select="year[1]"/> ⟸                    1999
<xsl:value-of select="year[2]"/> ⟸                       2
<xsl:value-of select="year[1] ge year[2]"/>
        ⟹false
<xsl:value-of
    select=
    "year[1] cast as xsd:integer ge xsd:integer(year[2])"/>
        ⟹ true
```

Other methods

```
<xsl:variable name="y1" select="year[1]" as="xsd:integer"/>
<xsl:variable name="y2" select="year[2]" as="xsd:integer"/>
<xsl:value-of select="$y1 ge $y2"/> ⟹ true
```

Other methods

```
<xsl:variable name="y1" select="year[1]" as="xsd:integer"/>
<xsl:variable name="y2" select="year[2]" as="xsd:integer"/>
<xsl:value-of select="$y1 ge $y2"/> ⟹ true
```

Declaring:

```
<xsd:element name="year" type="xsd:integer"/>
```

by using XML Schema.

Datatype language

No type information $\Longrightarrow$ `item()*`

Datatype language

No type information $\Longrightarrow$ `item()*`

`as` attributes $\Longleftarrow$ `xsl:function`, `xsl:param`, `xsl:variable`, `xsl:template`, `xsl:with-param`.

Datatype language

No type information $\Longrightarrow$ `item()*`

`as` attributes $\Longleftarrow$ `xsl:function`, `xsl:param`, `xsl:variable`, `xsl:template`, `xsl:with-param`.

$$X \quad \begin{matrix} \texttt{cast as} \\ \texttt{castable as} \\ \texttt{instance of} \\ \texttt{treat as} \end{matrix} \quad T$$

Character maps

```
<xsl:character-map name="TeX-map">
  <xsl:output-character character="#" string="\#"/>
  ...
</xsl:character-map>
```

Character maps

```
<xsl:character-map name="TeX-map">
  <xsl:output-character character="#" string="\#"/>
  ...
</xsl:character-map>
```

Representing ‘exceptional’ characters $\Longleftarrow$ introducing characters belong to Unicode’s private areas.

ex. in using-maps.xsl

And now. . .

Only a few processors of XSLT 2.0 at the present time: Saxon, AltovaXML™.

And now. . .

Only a few processors of XSLT 2.0 at the present time: Saxon, AltovaXML™.

Microsoft has announced that it would not develop XSLT 2.0.

XLST 2.0 & XML Schema

An XSLT 2.0 processor:

must be able to deal with basic types and relationships of XML Schema $\Longleftarrow$ Saxon's open source version;

XLST 2.0 & XML Schema

An XSLT 2.0 processor:

must be able to deal with basic types and relationships of XML Schema $\Longleftarrow$ Saxon's open source version;

may be interfaced with the whole of XML Schema's datatype library and user-defined datatypes using this language's constructs $\Longleftarrow$ Saxon's schema-aware version.

My opinion

XSLT 2.0 $\Longrightarrow$ major improvement.

Schema languages $\Longleftarrow$ XML Schema or Relax NG? (+ Schematron)

My opinion

XSLT 2.0 $\Longrightarrow$ major improvement.

Schema languages $\Longleftarrow$ XML Schema or Relax NG? (+ Schematron)

Computer scientist's viewpoint programming using new features encourages to use *datatypes* as far as possible.

Non-computer-scientist's may appear as more difficult, because of the notion of sequence.

Practically

Don't hesitate to try Saxon's open-source version:

- Java version is easy to install,
- in case of an error, messages are quite clear.

More complete documents

Michael Kay's books:

- XPath 2.0,
- XSLT 2.0.

More complete documents

Michael Kay's books:

- XPath 2.0,
- XSLT 2.0.

Sal Mangano's (O'Reilly):

$\Longleftarrow$ comparison 1.0/2.0.