XSLT and XPath
without the pain!

Bertrand Delacrétaz, bdelacretaz@apache.org
Senior R&D Developer, www.day.com
Day

www.codeconsult.ch/bertrand (blog)

slides revision: 2007-11-10

Goal

Learning to learn XPath and XSLT

because it takes time...
also applies to teaching it
and includes some patterns

What is XML?

Procedural code does not apply here...

photo:aameris on morguefile.com

XPath

First things first...but do we use XPath?

XPath

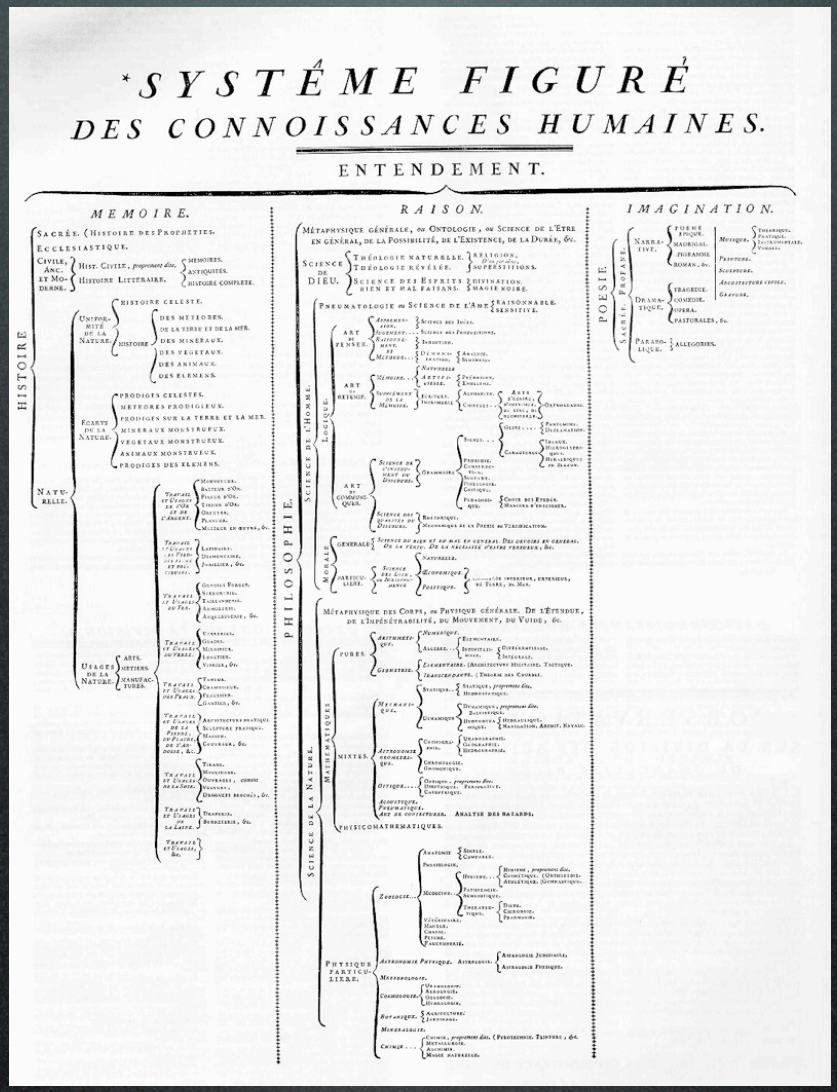
http://www.w3.org/TR/xpath says:

...The purpose of XPath is to address parts of an XML document...

...XPath operates on the abstract, logical structure of an XML document, rather than its surface syntax....

divide et impera!

XPath works on trees



wikipedia:Tree_structure

The XPath data model

Seven types of nodes in our tree:

4

root node
element nodes
text nodes
attribute nodes

3

comment nodes
namespace nodes
processing instruction nodes

...adressed with various non-obvious notations

Common node types

4

r

e

a

c

t

```
<document>
 <paragraph style="heading">
  <!-- some comment here -->
  Once upon a time...
 </paragraph>
</document>
```

r "root" is the root of the XPath tree, *not* the root element

a attributes are "under" elements in the tree: p/@id

XPath: relative and absolute “paths”

/document/section
//section
./section
.//section
//section/para/text()

paths are actually queries in the tree!

the dot “.” is the all-important “context node”

XPath predicates

```
/document[@id='12']
section[para]
section[para]//div[style='abc']
para[contains(. , 'help')]
*[self::para and @style]
```

The “where clause” of XPath

XPath axes

parent::section[1]
ancestor::section[@style]
following-sibling::para
preceding-sibling::section/para
following::para[@id='42']
following::*[1]/parent::p

And more axes: hierarchy, flat list, siblings

are we painless enough?

XPath is the most important part of XSLT!

it is.

XPath summary

The Tree Model
Relative and absolute paths
Expressions
Predicates
Axes
(some) functions

REQUIRED knowledge for painless XSLT!

XSLT

It gets easier...

XPath strikes again...

```
<xsl:template
 match="section[para[normalize-space(.)]]">

 <div class="{@style}">
  <h1><xsl:value-of select="title"/></h1>

  <xsl:apply-templates select="para[@ok='true']"/>
 </div>

</xsl:template>
```

not much XSLT in this...

XSLT learning subset

XPath and the tree model
Namespaces (basic knowledge)
The XSLT Processor Scenario

xsl:template

xsl:apply-templates
xsl:value-of, dynamic attributes

And later:
xsl:variable and xsl:param
xsl:call-template, xsl:key, modes
xsl:include and xsl:import

Forget about...

xsl:if
xsl:choose
xsl:for-each

For as long as you can: those don't climb trees well...


important stuff ahead
WAKE UP!
photo:nasirkhan on morguefile.com

The XSLT Processor Scenario

Start by visiting the root node.

For each node that is visited:
- Set the node as the “context node”
- Do we have a matching template?
  - -YES: execute the template, all done.
  - -NO: apply the default rules.

XPath!

If YES causes nodes to be visited, repeat. But in the YES case, default rules don’t apply.

How about <xsl:template match=”/” /> ?

XSLT Default Rules

If no template applies...

The root node is visited
Element nodes are visited
Text nodes are copied
All other nodes are ignored

```
<xsl:transform
 xmlns:xsl="http://...Transform"
 version="1.0"/>
```

XSLT summary

XPath
Learning subset
XSLT Processor Scenario
Default Rules

are we painless enough?

XSLT 2.0 ?

Built-in grouping

Regular Expressions

xsl:function

Sequences

Sorting with collations

Schema support, typed variables

And more...

It's all good news, but the basic principles remain!

Patterns

(and anti-patterns)

symbols:ppdigital and o0o0xmods0o0oon morguefile.com

Dynamic attributes

```
<xsl:template match="something">
  <output href="{@url}"/>
 </xsl:template>
```

xsl:attribute is most often not needed!

The Useless Template


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


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

no need to recreate the default rules!

Don’t choose: use more templates!

```
<xsl:template match="something">
 <xsl:choose>
  <xsl:when test="@id">
   ...case 1
  </xsl:when>
  <xsl:otherwise>
   ...case 2
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```


```
<xsl:template match="something[@id]">
   ...case 1
 </xsl:template>

 <xsl:template match="something">
   ...case 2
 </xsl:template>
```

Don't loop: apply-templates!


```
<xsl:for-each select="something">
   ..process something
</xsl:for-each>
```

```
<xsl:apply-templates select="something"/>

<xsl:template match="something">
   ..process something
</xsl:template>
```

in the general case at least...

I'll trade an xsl:if...


```
<xsl:template match="div">
  ...process div
  <xsl:if test="@class = 'SPECIAL'">
    ...process div having SPECIAL class
  </xsl:if>
 </xsl:template
```

it's not *that* bad with just one xsl:if...

...for a modal template

```
<xsl:template match="div">
  ...process div
  <xsl:apply-templates select="." mode="div.extra"/>
 </xsl:template>

 <xsl:template match="div" mode="div.extra"/>

 <xsl:template
  match="div[@class='SPECIAL']"
  mode="div.extra">
    ...process div having SPECIAL class
 </xsl:template>
```

much more
extensible!

Named templates as reusable blocks

```
<xsl:template match="something">
  <xsl:call-template name="somePartOne"/>
  <xsl:call-template name="somePartTwo"/>
 </xsl:template>

 <xsl:template name="somePartOne">
  ...
 </xsl:template>

 <xsl:template name="somePartTwo">
  ...
 </xsl:template>
```

Match that root!

```
<xsl:template match="/">
  <xsl:call-template name="input.sanity.check"/>

  <output-root-element>
   <xsl:apply-templates/>
  </output-root-element>
 </xsl:template>

 <xsl:template name="input.sanity.check">
  <xsl:if test="not(/something)">
   <xsl:message terminate="yes">
    Error: expected "something" as the root element
   </xsl:message>
  </xsl:if>
 </xsl:template>
```

So, where's the pain?

Where's the pain?

a.k.a "conclusion"

Not knowing XPath well
Not understanding the Processor Scenario
Not creating enough templates
Abusing "procedural" constructs

WDYT?

See you at the Day booth!

Read my blog at
www.codeconsult.ch/bertrand

Michael Kay,
my favorite XSLT book