

5.1 Additional features of XPath & XSLT

- XPath support for
 - arithmetics
 - processing ID/IDREF cross-references
 - manipulation of strings
- Generating text
 - for content
 - for attribute values
- Repetition, sorting and conditional processing
- Generating numbers

SDPL 2004

Notes 5.1: Additional XPath & XSLT

1

XPath: Arithmetical Operations

- Operators for double-precision (64 bit) floating-point arithmetics
 - $+$, $-$, $*$, div , mod (same as $\%$ in Java)
- Mapping numbers to integers:
 - » $\text{floor}(x) = \max\{n \in \mathbb{Z} \mid n \leq x\}$
 - » $\text{ceiling}(x) = \min\{n \in \mathbb{Z} \mid n \geq x\}$
 - » $\text{round}(x) = \text{floor}(x+0.5)$
- Mapping numbers to formatted strings (example):
 - » $\text{format-number}(-1.2534, "0.0") = "-1.3"$

SDPL 2004

Notes 5.1: Additional XPath & XSLT

2

Aggregate Functions

- Counting nodes
 - » $\text{count}(\text{node-set})$
 - and summing them as numbers
 - » $\text{sum}(\text{node-set})$
- Example:
 - Course grade average for the first student:
 $\text{sum}(\text{//student}[1]/\text{course}/\text{@grade}) \text{ div } \text{count}(\text{//student}[1]/\text{course})$

SDPL 2004

Notes 5.1: Additional XPath & XSLT

3

Cross-referencing

- Function `id` selects elements by their unique ID
 - **NB:** ID attributes need to be declared (in DTD or its internal subset; See an example later)
- Examples:
 - $\text{id}(\text{'sect:intro'})$
selects the element with unique ID "sect:intro"
 - $\text{id}(\text{'sect:intro'})/\text{para}[5]$
selects the fifth para child of the above element
 - $\text{id}(\text{'sect1 sect2 sect3'})$ selects 3 sections (if they have the corresponding ID values)

SDPL 2004

Notes 5.1: Additional XPath & XSLT

4

String manipulation

- Equality and inequality of strings can be tested with operators `=` and `!=`
 - $\text{"foo"} = \text{'foo'}$; $\text{"foo"} \neq \text{"Foo"}$
- Testing for substrings:
 - $\text{starts-with}(\text{"dogbert"}, \text{"dog"}) = \text{true}()$
 - $\text{contains}(\text{"dogbert"}, \text{"gbe"}) = \text{true}()$
- Concatenation (of two or more strings),
 - $\text{concat}(\text{"dog"}, \text{"bert"}) = \text{"dogbert"}$

SDPL 2004

Notes 5.1: Additional XPath & XSLT

5

XPath: more string functions

- $\text{substring-before}(\text{"dogbert"}, \text{"bert"}) = \text{substring-before}(\text{"dogbert"}, \text{"b"}) = \text{"dog"}$
- $\text{substring-after}(\text{"dogbert"}, \text{"g"}) = \text{"bert"}$
- $\text{substring}(\text{string}, \text{startpos}, \text{length?})$:
 - » $\text{substring}(\text{"dogbert"}, 1, 3) = \text{"dog"}$
 - » $\text{substring}(\text{"dogbert"}, 3) = \text{"gbert"}$
- $\text{string-length}(\text{"dogbert"}) = 7$
- $\text{translate}(\text{Str}, \text{ReplacedChars}, \text{ReplacingChars})$:
 - » $\text{translate}(\text{"dogbert"}, \text{"dgo"}, \text{"Dli"}) = \text{"Dilbert"}$

SDPL 2004

Notes 5.1: Additional XPath & XSLT

6

Generating Text

- The string-value of an expression can be inserted in the result tree by instruction $\text{<xsl:value-of select="Expr" />}$
 - if *Expr* selects more than one nodes, the **value of the first node** in document order is used
- Consider transforming source elements like
 - $\text{<name alias="Bird">}$
 $\text{<first>Charlie</first><last>Parker</last>}$
 </name>
 - to the form
Charlie ("Bird") Parker

SDPL 2004

Notes 5.1: Additional XPath & XSLT

7

Computing generated text (2)

- This can be specified by template rule
 - $\text{<xsl:template match="name">}$
 $\text{<xsl:value-of select="first" />}$
 $\text{("}<xsl:value-of select="@alias" />\text{")}$
 $\text{<xsl:value-of select="last" />}$
 <xsl:text>
 </xsl:text>
 </xsl:template>
- Verbatim text (like the white-space above) can be inserted using `xsl:text`

SDPL 2004

Notes 5.1: Additional XPath & XSLT

8

Attribute value templates

- The string-value of an expression can be inserted in an attribute value by surrounding the expression by braces { and }

- Consider transforming source element

```
<photo>
  <file>Mary.jpg</file>
  <size width="300"/>
</photo>
into form

```

SDPL 2004

Notes 5.1: Additional XPath & XSLT

9

Attribute value templates (2)

- This can be specified by template rule

```
<xsl:template match="photo">
  
</xsl:template>
```

- Expressions {file} and {size/@width} are evaluated in the context of the current node (the photo element)

SDPL 2004

Notes 5.1: Additional XPath & XSLT

10

XSLT: Repetition

- Nodes can be "pulled" from source for processing using

```
<xsl:for-each select="Expr">
  Template
</xsl:for-each>
```

- the template is applied to each of the selected nodes (0, 1 or more), each node in turn as the current node

- » in document order, unless sorted using `xsl:sort` instructions (see later)

SDPL 2004

Notes 5.1: Additional XPath & XSLT

11

Example (of `xsl:for-each`)

- Consider formatting the below document as HTML:

```
<!DOCTYPE document [ <!ATTLIST section id ID #IMPLIED> ]>
<document> <title>The Joy of XML</title>
<section id="Intro"><title>Getting Started</title>
  <name><first>Helen</first> <last>Brown</last></name>
  says that processing XML documents is fun.
  <name><first>Dave</first> <last>Dobrik</last></name> agrees.
</section>
<section><title>Family affairs</title>
  <name><first>Bob</first> <last>Brown</last></name> is the
  husband of <name><first>Helen</first>
  <last>Brown</last></name>. </section>
<section><title>Finishing Up</title>
As we discussed in <title-ref idref="Intro" />, processing XML
documents is fun. </section></document>
```

SDPL 2004

Notes 5.1: Additional XPath & XSLT

12

Example: Table of contents

- A table of contents can be formed of section titles:

```
<xsl:template match="/">
<HTML><HEAD> <TITLE><xsl:value-of
  select="document/title"/></TITLE></HEAD>
<BODY>
  <H2>Table of Contents</H2>
  <OL> <!-- Pull each section title: -->
    <xsl:for-each select="//section/title">
      <LI><xsl:apply-templates /></LI>
    </xsl:for-each>
  </OL> <!-- then process the sections: -->
  <xsl:apply-templates select="document/section"/>
</BODY> </HTML>
</xsl:template>
```

SDPL 2004

Notes 5.1: Additional XPath & XSLT

13

Example (cont; Cross references)

- Cross references (to sections) can also be processed using `xsl:for-each`:

```
<xsl:template match="title-ref">
  <xsl:for-each select="id(@idref)">
    Section (<xsl:value-of
      select="substring(title, 1, 8)" />...)
  </xsl:for-each>
</xsl:template>
```

- With this rule the source fragment

As we discussed in <title-ref idref="Intro"/>

becomes

As we discussed in Section (Getting ...)

SDPL 2004

Notes 5.1: Additional XPath & XSLT

14

XSLT Sorting

- A sorted order for the processing of nodes with `xsl:for-each` and `xsl:apply-templates` can be specified by `<xsl:sort/>`

- controlled by attributes of `xsl:sort` like

- select: expression for the sort key
- data-type: "text" (default) or "number"
- order: "ascending" (default) or "descending"

- The first `xsl:sort` specifies the primary sort key, the second one the secondary sort key, and so on.

SDPL 2004

Notes 5.1: Additional XPath & XSLT

15

Example (cont; Sorted index of names)

- All names can be collected in a last-name-first-name order using the below template

```
<H2>Index</H2> <UL>
  <xsl:for-each select="//name">
    <xsl:sort select="last" />
    <xsl:sort select="first" />
    <LI><xsl:value-of select="last" />,
    <xsl:value-of select="first"/></LI>
  </xsl:for-each>
</UL>
```

- This creates an UL list with items

```
<LI>Brown, Bob</LI>
<LI>Brown, Helen</LI>
<LI>Brown, Helen</LI>
<LI>Dobrik, Dave</LI>
```

SDPL 2004

Notes 5.1: Additional XPath & XSLT

16

What about duplicates?

- Is it possible to eliminate duplicate values like
`Brown, Helen`
`Brown, Helen` ?
- Yes (but not that straightforward)
- Using conditional instructions
 - See next

SDPL 2004

Notes 5.1: Additional XPath & XSLT

17

Conditional processing

- A template can be instantiated or ignored based on the value of a test Boolean expression, using
`<xsl:if test="Expression">`
 Template
`</xsl:if>`
- Example: a comma-separated list of names:
`<xsl:template match="namelist/name">`
 `<xsl:apply-templates/>`
 `<xsl:if test="position() < last()"`
 `>, </xsl:if>`
`</xsl:template>`

SDPL 2004

Notes 5.1: Additional XPath & XSLT

18

Conditional processing (2)

- Also a case-like construct (~ switch in Java):
`<xsl:choose>`
 `<!-- The first 'when' whose test=true() is`
 `instantiated: -->`
 `<xsl:when test="Expr1"> ... </xsl:when>`
 `<xsl:when test="Expr2"> ... </xsl:when>`
 ...
 `<!-- If no 'when' applies, an optional`
 `'otherwise' is instantiated: -->`
 `<xsl:otherwise> ... </xsl:otherwise>`
`</xsl:choose>`

SDPL 2004

Notes 5.1: Additional XPath & XSLT

19

Example (cont; Eliminating duplicate names)

- No access to other nodes (except `current()`) in the list of `xsl:for-each`
 - But can refer to other nodes in the source tree
 - Process just the first one of duplicate names:
- ```
<xsl:for-each select="//name">
 <xsl:sort select="last"/>
 <xsl:sort select="first" />
 <xsl:if test="not(
 preceding::name[first=current()/first
 and last=current()/last])">
 <xsl:value-of select="last"
 />, <xsl:value-of select="first"/>
 </xsl:if>
</xsl:for-each>
```

SDPL 2004

Notes 5.1: Additional XPath & XSLT

20

## Generating Numbers

- Formatted numbers can be inserted in the result tree by element `xsl:number`
  - number can be specified by attribute `value="Expr"`
  - otherwise the number generated based on the position of the current node in the source tree
- We'll consider typical cases through examples
  - The complete rules of the Spec are rather complex
- Example 1: Numbering list items

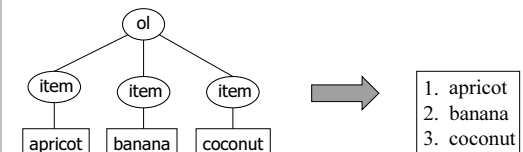
SDPL 2004

Notes 5.1: Additional XPath & XSLT

21

## Generating numbers: Example 1

- `<xsl:template match="ol/item">`  
    `<!-- default: count like siblings (items) -->`  
    `<xsl:number format="1. " />`  
    `<xsl:apply-templates/>`  
`</xsl:template>`



SDPL 2004

Notes 5.1: Additional XPath & XSLT

22

## Generating numbers: Example 2

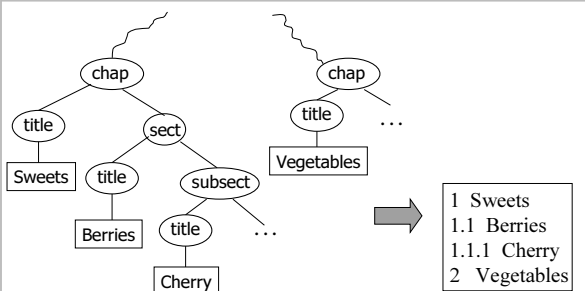
- Hierarchical numbering (1, 1.1, 1.1.1, 1.1.2, ...) for titles of chapters, titles of their sections, and titles of subsections:  
`<xsl:template match="title">`  
    `<xsl:number level="multiple"`  
        `count="chap|sect|subsect"`  
        `format="1.1 " />`  
    `<xsl:apply-templates/>`  
`</xsl:template>`

SDPL 2004

Notes 5.1: Additional XPath & XSLT

23

## Generating numbers: Example 2



SDPL 2004

Notes 5.1: Additional XPath & XSLT

24

## Example 2: Variation

- As above, but number titles within appendices with A, A.1, A.1.1, B.1 etc:

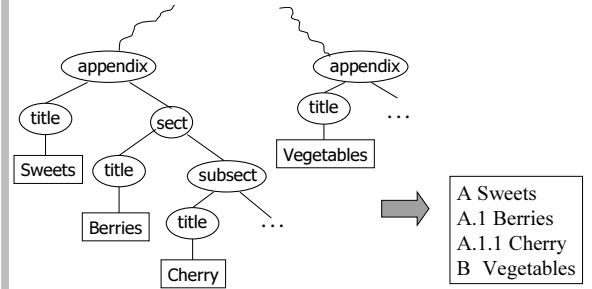
```
<xsl:template match="appendix//title">
 <xsl:number level="multiple"
 count="appendix|sect|subsect"
 format="A.1 " />
 <xsl:apply-templates/>
</xsl:template>
```

SDPL 2004

Notes 5.1: Additional XPath & XSLT

25

## Example 2: Variation



SDPL 2004

Notes 5.1: Additional XPath & XSLT

26

## Generating numbers: Example 3

- Sequential numbering of notes within chapters:  
(more precisely: starting anew at the start of each chapter)

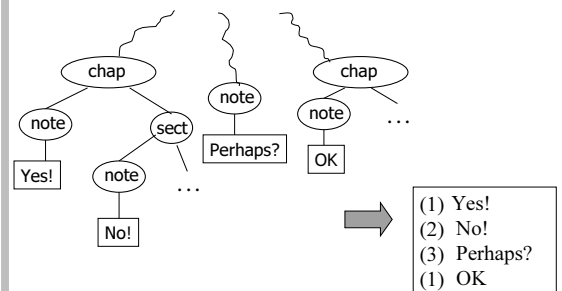
```
<xsl:template match="note">
 <xsl:number level="any"
 from="chap"
 format="(1) " />
 <xsl:apply-templates/>
</xsl:template>
```

SDPL 2004

Notes 5.1: Additional XPath & XSLT

27

## Ex 3: Sequential numbering from chaps



SDPL 2004

Notes 5.1: Additional XPath & XSLT

28