SDXML VT2024
Models and languages for semi-structured data and XML

XSLT

XSLT

nikos dimitrakas
nikos@dsv.su.se
08-161295

Corresponding reading
- Section 7.2.1 of the course book
- Chapter 15 of XML 1.1 Bible
- Compendium "Introduction to XSLT"

XSLT

- **XSL Transformations**
  - **XSL: eXtensible Stylesheet Language**
- **Transformations**
  - **From XML**
  - **To XML, HTML, text, etc.**
- **XSLT is an XML-based language**
  - **An XSLT document is and XML document**
  - **Often with the file extension .xsl**
  - **Namespace and defined semantics**
- **Similar to programing languages**
  - **Recursion**
  - **Iteration**
  - **Flow control**
  - **Variables**

XSL transformations
XSL:extensible stylesheet language
Transformations
from XML
to XML,HTML,text,etc
XSLT is an XML based language

Often with file extension .xsl
Namespace and defined semantics

Similar to programming languages
Recursion
Iteration
Flow control
Variables

XSLT versions

- **XSLT 1**
  - Based on XPath 1
  - Supported in web browsers like Internet Explorer, Safari, Chrome, FireFox, Netscape, Opera, Vivaldi, Edge

Internet Explorer,Chrome,Firefox,Edge

- **XSLT 2**
  - Based on XPath 2
  - Extra options for among other things grouping
  - More output formats
  - Not supported in any web browser yet, but there are server-side modules, e.g. Saxon (that is used by xsltransform.net, xslttest.appspot.com and in BaseX)

Based on Xpath 2
Extra options for grouping

- **XSLT 3**
  - Together with XPath 3 and XQuery 3
  - Handling of streaming data
  - JSON (input and output)

XSLT document

- **Root element**
  - xsl:transform or xsl:stylesheet (synonyms)
  - the attribute "version" in the root element decides the XSLT version

version in root element decides XSLT version

- **Namespace**
  - http://www.w3.org/1999/XSL/Transform
  - Recommended prefix: xsl
- **Linking to/from an XML document**
  - <?xml-stylesheet type="text/xsl" href="???.xsl"?>
  - or dynamically in the application

XSLT - Top level elements

- **Elements directly under the root element**
  - **"Declarations"**
- **XSLT 1**
  - **import, include, strip-space, preserve-space, output, key, decimal-format, namespace-alias, attribute-set, variable, param, template**
  - **The element "template" is were it all happens**
  - **The rest are configurations**
- **XSLT 2 (on top of the previous)**
  - **character-map, function, import-schema**
- **XSLT 3 (on top of the previous)**
  - **mode, accumulator**

elements directly under root element

XSLT 1 - include,strip-space,preserve-space, ,output,key,decimal-format, namespace-alias,attribute-set,variable,param,template

XSLT - Instructions

elements inside element template

- **Elements inside the element "template"**
- **XSLT 1**
  - **Node creation: element, attribute, comment, processing-instruction, value-of, text, copy, copy-of**
  - **Flow control, iteration: if, choose (and when, otherwise), for-each**
  - **Variables: variable, param**
  - **Template calls: apply-templates, call-template, apply-imports**
  - **Other specialized instructions like message and number**
- **XSLT 2 (on top of the previous)**
  - **for-each-group, next-match, sequence, namespace, document, result-document, analyze-string**
- **XSLT 3 (on top of the previous)**
  - **source-document, iterate, merge, fork, where-populated, on-empty, on-non-empty, try, evaluate, assert**

Sample data

Books
<Book title Language
Year Publisher Genre>
<Author Name
YOB Country>

According to the following DTD:

```
<!ELEMENT Books (Book+)>
<!ELEMENT Book (Author+)>
<!ATTLIST Book Title CDATA #REQUIRED
        Language CDATA #REQUIRED
        Year CDATA #REQUIRED
        Publisher CDATA "N/A"
        Genre CDATA "N/A">
<!ELEMENT Author EMPTY>
<!ATTLIST Author Name CDATA #REQUIRED
        YearOfBirth CDATA #REQUIRED
        Country CDATA #REQUIRED>
```


Sample data

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Books SYSTEM "books.dtd">
<Books>
        <Book Title="Misty Nights" Year="1997" Language="English" Publisher="Kingsly" Genre="Thriller">
                <Author Name="John Craft" YearOfBirth="1948" Country="England"></Author>
        </Book>
        <Book Title="Archeology in Egypt" Year="1992" Publisher="KLC" Language="English" Genre="Educational">
                <Author Name="Arnie Bastoft" YearOfBirth="1971" Country="Austria"></Author>
                <Author Name="Meg Gilmand" YearOfBirth="1968" Country="Australia"></Author>
                <Author Name="Chris Ryan" YearOfBirth="1944" Country="France"></Author>
        </Book>
        <Book Title="Database Systems in Practice" Year="2000" Language="English" Genre="Educational">
                <Author Name="Alan Griff" YearOfBirth="1972" Country="USA"></Author>
                <Author Name="Marty Faust" YearOfBirth="1970" Country="USA"></Author>
                <Author Name="Celine Biceau" YearOfBirth="1969" Country="Canada"></Author>
        </Book>
        <Book Title="Contact" Language="English" Year="1988" Genre="Science Fiction">
                <Author Name="Carl Sagan" YearOfBirth="1913" Country="USA"></Author>
        </Book>
        <Book Title="The Fourth Star" Year="2001" Language="English" Publisher="Bästa Bok" Genre="Science Fiction">
                <Author Name="Leslie Brenner" YearOfBirth="1945" Country="USA"></Author>
        </Book>
        <Book Title="Våren vid sjön" Year="1982" Language="Swedish" Genre="Novel">
                <Author Name="Marie Franksson" YearOfBirth="1937" Country="Sweden"></Author>
        </Book>
        <Book Title="Dödliga Data" Year="1993" Language="Swedish" Genre="Thriller">
                <Author Name="Jakob Hanson" YearOfBirth="1946" Country="Sweden"></Author>
        </Book>
        <!-- more Book elements -->
</Books>
```

XSLT declaration output

- Used to specify the format of the result of the transformation
- Has many attributes
  - method: xml, html, text (and xhtml in XSLT 2, json in XSLT 3)
  - encoding
  - more attributes for XML declarations and configurations
    - » e.g. indent, media-type, omit-xml-declaration

used to specify format or result of transformations

- Example

```
<xsl:output method="xml" />
```

XSLT declaration output
used to specify format of result of transformations.

```
<xsl:output method="xml"/>
```


XSLT declaration variable

- Used to declare variables and assign them values
- Attributes
  - name: the name of the variable
  - select: the value of the variable (may be an XPath expression whose result becomes the value of the variable)

- Example

```
<xsl:variable name="x" select="5" />
```

- Variables can then be used in XPath expressions with the prefix $

XSLT declaration template

- Used to work with data and construct output
- Named templates have the attribute name
- Rule-based templates have the attribute match that contains a pattern expression (a specific type of XPath expression) that decides when the template is applicable

templates - used to work with data and construct output

- Example

```
<xsl:template name="abc">
</xsl:template>
```

Named templates have the attribute name

xsl:template match="/"

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


XSLT - Calling templates

- Default template
  - match="/"
  - starts the execution
- Calling named templates
  - <xsl:call-template name="abc" />
- Calling rule-based templates
  - <xsl:apply-templates />
  - <xsl:apply-templates select="Book" />
  - the attribute select contains an XPath expression that decided which nodes should be processed by a matching template
  - if the attribute select is missing then all the children are selected
  - if no template matches, a default template (built-in template) will be used
    - » Returns the value of the node
    - » More sets of built-in templates in XSLT 3 by using xsl:mode and on-no-match

xsl:call-template name="abc"

xsl:apply-templates select

select contains Xpath that decides which nodes should be processed by a matching template.

if select missing all children are selected.

XSLT - Constructor instructions

element - creates an element
name of element specified in attribute name

- **element**
  - Creates an element
  - The name of the element is specified in the attribute name
  - The content of the element is specified in the content
  - <xsl:element name="MyBooks">content</xsl:element>

< xsl:element name="" >

- **attribute**
  - Creates an attribute for the current element
  - The name of the attribute is specified in the attribute name
  - The value of the attribute is specified in the attribute select (XSLT 2) or in the content
  - <xsl:attribute name="Title">value</xsl:attribute>

<xsl:attribute name="" >

- **comment**
  - Creates a comment
  - The comment is specified in the attribute select (XSLT 2) or in the content
  - <xsl:comment>comment</xsl:comment>

XSLT - Constructor instructions

- **processing-instruction**
  - Creates an XML processing instruction
  - Note! Not the XML declaration that get created by xsl:output
  - The name of the PI is specified in the attribute name
  - The value of the PI is specified in the attribute select (XSLT 2) or in the content
  - <xsl:processing-instruction name="Greeting" select="'Ahoy'"/>
- **namespace (XSLT 2)**
  - Creates a namespace node (xmlns attribute)
  - The name of the namespace (the alias) is specified in the attribute name
  - The value of the namespace (the URI) is specified in the attribute select or in the content
  - <xsl:namespace name="sdxml" select="'http://ns.dsv.su.se/SDXML'" />
- **text**
  - Creates a text node
  - <xsl:text>Greetings</xsl:text>

name- alias of namespace
select-URI of namespace

XSLT - Constructor instructions

<xsl:value-of select="Book/@Title"/>
creates a text node from an XPath expression

- **value-of**
  - Creates a text node from an XPath expression
    - » Only the first value if the expression returns a sequence (XSLT 1)
    - » All values separated according to the attribute separator (XSLT 2)
  - The XPath expression is specified in the attribute select
  - <xsl:value-of select="Book/@Title" />
- **copy**
  - Creates a (shallow) copy of the current node
    - » Or the node specified in the attribute select (XSLT 3)
  - <xsl:copy />
- **copy-of**
  - Creates a (deep) copy on the node/nodes specified in the attribute select
  - <xsl:copy-of select="Book" />

copy-of - deep copy of node

<xsl:copy-of select="Book"/>

Node creation without instructions

- **Write xml code directly**
  - <Person />
  - Same effect as <xsl:element name="Person"/>
- **Attributes**
  - <Person name="James" />
  - Same effect as

```
<xsl:element name="Person">
        <xsl:attribute name="name">James</xsl:attribute>
</xsl:element>
```

- **Dynamic attribute values?**
  -

```
<xsl:element name="Person">
        <xsl:attribute name="name">
                <xsl:value-of select="@Pname" />
        </xsl:attribute>
</xsl:element>
```

xsl:value-of select=@Pname

  - <Person name="??" />

Attribute value templates

- Get the value for an attribute from the result of an XPath expression
  - <xsl:element name="Person">
    <xsl:attribute name="name">
    <xsl:value-of select="@Pname" />
    </xsl:attribute>
    </xsl:element>
  - <Person name="{@Pname}" />
- Note! This only works for attribute values. The following is invalid (in XSLT 1 and 2):
  - <Person>{@Pname}</Person>

Text value templates (XSLT 3)

XSLT 3.0
text nodes created from Xpath expression inside {}

- Text nodes can be created from an XPath expression inside {}
  - Requires activation with the attribute expand-text
    - » yes, 1 or true (no, 0 or false is default)
  - The attribute expand-text must be specified at an ancestor.

```
<Person xsl:expand-text="yes">{@Pname}</Person>
```

```
<xsl:element expand-text="yes" name="People">
    <Person>{@Pname}</Person>
</xsl:element>
```

xsl:expand-text="yes" must be specified at an ancestor.

Flow control

- **if**
  - **Its content is executed only if the condition is true**
  - **The condition is specified in the attribute test**
  - **<xsl:if test="@Title='Contact' ">…</xsl:if>**
- **choose**
  - **Has one or more when and possibly an otherwise**
  - **Every when has a condition that is specified in the attribute test**
  - **Only the first matching when gets executed**
  - **If no when matches the content of otherwise gets executed**
  - **<xsl:choose>**
    <xsl:when test="$n=1">One</xsl:when>
    <xsl:when test="$n=0">Zero</xsl:when>
    <xsl:otherwise>Many</xsl:otherwise>
    </xsl:choose>

condition specified in attribute test. content executed if condition is true
<xsl:if test="@Title='Contact'"> </xsl:if>

Flow control
if - content executed only if condition is true

choose - one or more when, possibly an otherwise
every when has condition specified in attribute test.
<xsl:choose>
<xsl:when test="$n=1"> One </xsl:when> <xsl:otherwise>Many</xsl:otherwise> </xsl:choose>

Iteration and grouping

- **for-each**
  - **Loops through the items in the sequence that is the result of the XPath expression in the attribute select**
  - **<xsl:for-each select="Author">…</xsl:for-each>**
- **for-each-group (XSLT 2)**
  - **Groups the items in the sequence that is the result of the XPath expression in the attribute select according to the expression in the attribute group-by (or group-adjacent, or group-starting-with, or group-ending-with) and loops through the groups**
  - **The function current-group() returns the sequence containing all the items of the current group**
  - **The function current-grouping-key() returns the grouping value of the current group**
  - **<xsl:for-each-group select="Author" group-by="@Country">**
    …
    </xsl:for-each-group>
  - **Support for complex grouping in XSLT 3 (with the attribute "composite")**

Sorting

- sort
  - Can be used in all loops (for-each, apply-templates, for-each-group)
  - Sorts the iterations of the loop (the sequence of the loop) according to the attribute select
  - Sorts on one thing, but it is possible to have multiple xsl:sort
  - The order can be ascending (default) or descending and is specified in the attribute order
  - The attribute data-type specifies how the values should be compared Available options: text (default), number, or types like xs:date
  - <xsl:sort select="@Year" order="descending" data-type="number" />

Some more

- source-document (XSLT 3)
  - Dynamically open other XML documents as input
  - <xsl:source-document href="{$filename}"/>
- evaluate (XSLT 3)
  - Evaluates a dynamic XPath expression
  - <xsl:evaluate xpath="$xpathstring"/>
  - Supports parameters with xsl:with-param

Important functions

- current()
  - returns the current/context item
- position()
  - returns the current position in the sequence
- last()
  - returns the size of the sequence (the position on the last item)
- doc(uri) document(uri)
  - opens an XML file
- not(expression)
  - negates the boolean value of the parameter

Example - iteration with for-each

- All book titles

```
<xsl:template match="/">
  <xsl:element name="Titles">
    <xsl:for-each select="Books/Book">
      <xsl:element name="Title">
        <xsl:value-of select="@Title" />
      </xsl:element>
    </xsl:for-each>
  </xsl:element>
</xsl:template>
```

```
<xsl:template match="/">
<xsl:element name="Titles">
<xsl:for-each select="Books/Book">
```

for each to loop through the Books

value of used to get a value

Example - templates

- All book titles

```
<xsl:template match="/">
    <xsl:element name="Titles">
        <xsl:apply-templates select="Books/Book" />
    </xsl:element>
</xsl:template>
<xsl:template match="Book">
    <xsl:element name="Title">
        <xsl:value-of select="@Title" />
    </xsl:element>
</xsl:template>
```


More information

- XSLT 1, XSLT 2 and XSLT 3 specifications
- Compendium "Introduction to XSLT"
- Chapter 15 in XML 1.1 Bible

XSLT can be executed with

- Web browser (only XSLT 1)
- http://xsltransform.net
- http://xslttest.appspot.com
- Oracle, DB2, SQL Server (only XSLT 1 inside SQL)
- BaseX (with XQuery)
- Custom program using an XSLT library like Saxon

What to do next

- Quiz about XSLT (Quiz 6)
- XSLT (compendium "Introduction to XSLT")
  - Contains more examples
  - Contains information about execution environments
- Lesson exercises (3)
- Seminar exercises (Assignment 2)
  - Wait until after the lesson