XSLT advanced & in use

Ulrike Henny
University of Cologne / University of Würzburg
ulrike.henny@web.de

XSLT advanced

- Grouping, sorting, numbering
- Copying
- Input & output documents
- Templates, variables & parameters

XSLT advanced. Grouping, sorting, numbering

- Grouping
  - ***xsl:for-each-group***
  - a set of items is selected and arranged into groups based on specific criteria (for example common values); then each group is processed in turn
  - special XPath functions within for-each-group: ***current-grouping-key(), current-group()***
  - Format:
    - <xsl:for-each-group select="*XPath expression (items)*" group-by="*XPath expression (grouping-key)*">
  - Example:
    - <xsl:for-each-group select="//tei:persName" group-by="substring(.,1,1)">

```
<h2><xsl:value-of select="current-grouping-key()"/></h2>
<ul><xsl:for-each select="current-group()">
        <xsl:sort />
        <xsl:value-of select="."/>
    </xsl:for-each></ul>
</xsl:for-each-group>
```

XSLT advanced. Grouping, sorting, numbering

- Grouping
  - Typical use of for-each-group: creation of indexes
  - Example: Index of place names for “The Discovery of Guiana“
    - Source file: raleigh-discovery-of-guiana.xml
    - Approach:
      - Part 1: creation of a TEI index of places
        - get all the place names occurring in the text
        - group those place names referring to the same place
        - sort the places alphabetically, write them into the result document (template: TEI-list-places.xml)
      - Part 2: creation of an HTML index of places
        - get all the place names occurring in the text
        - group those place names referring to the same place
        - group the places according to the letters of the alphabet
        - sort the places alphabetically
        - create HTML lists

XSLT advanced. Grouping, sorting, numbering

- Grouping
  - Example: Index of place names for “The Discovery of Guiana“
    - Part 3: enhancement of the place index
      - What are the actual place names that occur in the text?
        - e.g. Puerto de los Españoles = Port of Spain
        - Goal: list the different names as a sublist below each place entry in the index
      - Where in the text do the place names occur?
        - Goal: indicate the page numbers of place name occurrences as a list after each place name entry

XSLT advanced. Grouping, sorting, numbering

- Grouping
  - Variants of <xsl:for-each-group>:
    - <xsl:for-each-group select="*XPath expression (items)*" group-adjacent="*XPath expression (grouping-key)*">
      - adjacent items are allocated to the same group if they have common values for the grouping key
    - <xsl:for-each-group select="*XPath expression (items)*" group-starting-with="*XPath expression (pattern)*">
      - Whenever an item matches the pattern, a new group is started with this item
    - <xsl:for-each-group select="*XPath expression (items)*" group-by="*XPath expression (pattern)*">
      - Whenever an item matches the pattern, a new group is started after this item

XSLT advanced. Grouping, sorting, numbering

- Sorting
  - ***xsl:sort***
  - can be used inside of <xsl:for-each>, <xsl:for-each-group>, <xsl:apply-templates> and <xsl:perform-sort> (must appear first!)
  - defines the order in which the data is processed by the instruction
  - several subsequent sort keys can be defined
  - Format:
    - <xsl:sort select="*XPath expression (items)*" order="ascending|descending" data-type="text|number" case-order="upper-first|lower-first" lang="*language code*" collation="*URI*">
  - Example:
    - <xsl:for-each select="//tei:idno" >
      <xsl:sort data-type="number" order="descending" />
      <xsl:value-of select="."/>
      </xsl:for-each>

XSLT advanced. Grouping, sorting, numbering

- Sorting
  - Using collations
  - Example:
    - <xsl:for-each select="//tei:persName" >

```
<xsl:sort collation="http://saxon.sf.net/collation?rules={encode-for-uri('&lt; A,a &lt; Ae=Ä,ae=ä &lt; B,b &lt; C,c &lt; D,d &lt; E,e &lt; F,f &lt; G,g &lt; H,h &lt; I,i &lt; J,j &lt; K,k &lt; L,l &lt; M,m &lt; N,n &lt; O,o &lt; Oe=Ö,oe=ö &lt; P,p &lt; Q,q &lt; R,r &lt; S,s &lt; ß=ss &lt; T,t &lt; U,u &lt; Ue=ü,ue=ü &lt; V,v &lt; W,w &lt; X,x &lt; Y,y &lt; Z,z')}" />
<xsl:value-of select="."/>
</xsl:for-each>
```

XSLT advanced. Grouping, sorting, numbering

- Sorting
  - ***xsl:perform-sort***
  - to sort a sequence of items without processing them by <xsl:for-each>, <xsl:for-each-group> or <xsl:apply-templates>
  - contains one or several sort expressions
  - Format:

```
<xsl:perform-sort select="XPath expression (items)">
    <xsl:sort />
</xsl:perform-sort>
```

  - Example:

```
<xsl:perform-sort select="//tei:date">
    <xsl:sort select="substring-before(@when, '-')" data-type="number"/>
    <xsl:sort select="substring-before(substring-after(@when,'-'), '-')" data-type="number"/>
</xsl:perform-sort>
```

XSLT advanced. Grouping, sorting, numbering

- Numbering
  - ***xsl:number***
  - instruction which is used inside of a sequence constructor
  - Tasks:
    - Determines the number of an item in a sequence
    - Formats this number
    - Writes this number into the result document
  - Format:

    <xsl:number count="*items*" format="*formatting template*" level="single|multiple|any" from="*pattern*" /> → *always empty!*
  - Example:

```
<xsl:template match="//tei:div[@type='chapter']">
  Chapter <xsl:number count="." format="I."/>
</xsl:template>
```

XSLT advanced. Grouping, sorting, numbering

- Numbering
  - Formatting templates

Value	Type of numbering
1	1,2,3,4...
01	01,02,03,04,...
a	a,b,c,d,...
A	A,B,C,D,...
i	i,ii,iii,iv,...
I	I,II,III,IV,...

XSLT advanced. Grouping, sorting, numbering

- Exercises
  - **(1)** Create an index of persons for “The Discovery of Guiana“
    - XML file: raleigh/raleigh-discovery-of-guiana.xml
    - Use xsl:for-each-group and xsl:sort! (procede according to the place index example)
  - **(2)** Create an alphabetically sorted index of first verse lines of the Sonnets of Shakespeare
    - XML file: shakespeare/poetry/son.xml
    - xsl:for-each-group, xsl:sort
  - **(3)** Create an HTML list from TEI-list-persons.xml. Sort the list using xsl:sort:
    - (a) by first name in ascending order
    - (b) by last name, ascending, according to the German sort rules
    - (c) by switching letters in the alphabet (e.g. A,C,B,D,F,E...) using your own collation
  - **(4)** Sort the dates in “The Discovery of Guiana“ using xsl:perform-sort; first by year, then by month, then by day
  - **(5)** Create an HTML list which shows the structure (Acts and Scenes and how they are nested) of the Shakespeare play Macbeth by using xsl:number.
    - XML file: shakespeare/tragedies/mac.xml

XSLT advanced. Copying

- ***xsl:copy***
  - copies the current element, without attributes, without descendants
  - Format:

    <xsl:copy>
  - Example:

    <xsl:template match="//p">
    - <xsl:copy><xsl:apply-templates/></xsl:copy>
    - </xsl:template>
- ***xsl:copy-of***
  - copies the whole subtree, including all attributes, including all descendants
  - Format:
    - <xsl:copy-of select="*XPath expr*"/>
  - Example:
    - <xsl:copy-of select="//div[@type='dedication']"/>

XSLT advanced. Copying

- This can be useful:
  - „Copy all, but...“

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
    <!-- copy all -->
    <xsl:template match="node() | @* | processing-instruction() | comment()">
      <xsl:copy>
        <xsl:apply-templates select="node() | @* | processing-instruction() |
    comment()"/>
      </xsl:copy>
    </xsl:template>

    <!-- ,but... -->
    <xsl:tempate match="pattern">
      ...
    </xsl:template>
</xsl:stylesheet>
```

XSLT advanced. Input & output documents

- ***document()***
  - XSLT function; retrieves one/several external XML document(s) by means of a (set of) URI(s)
  - Format: *document("URI(s)", expr (base node)?)*
  - Example: *document("http://en.wikipedia.org/Raleigh")*
- ***doc()***
  - XPath function; retrieves an external XML document by means of a URI; simplified version of the XSLT document-function
  - Format: *doc("URI")*
  - Example: *doc("shakespeare/tragedies/mac.xml")*
- ***collection()***
  - XPath function; returns a sequence of documents / nodes, identified by a URI
  - Format: *collection("URI")*
  - Example: *collection("shakespeare/tragedies")*

XSLT advanced. Input & output documents

- ***xsl:result-document***
  - XSLT instruction; creates a new result tree; allows one transformation to produce multiple result documents!
  - Format:
    - <xsl:result-document href="*URI*">
  - Example:

```
<xsl:for-each select="//tei:div[@type='book']">
    <xsl:result-document href="books/book-{position()}.html">
        <html>
            <head>...</head>
            <body>...</body>
        </html>
    </xsl:result-document>
</xsl:for-each>
```

XSLT advanced. Copying / Input & output docs

- Example I
  - Collect the cast lists of Shakespeares „histories“ and copy them into one TEI file
  - Create an HTML version of each cast list (one HTML file for each cast list)
- Example II
  - Copy all of “The Discovery of Guiana“, thereby modernizing some words:
  - *riseth, paseth, lieth, runneth, abideth, quenheth, healeth,...*

→ <choice>
<orig>riseth</orig>
<reg>rises</reg>
</choice>

XSLT advanced. Templates, variables & parameters

- ***xsl:template***
  - templates may have names and modes; at least @match OR @name must be present
  - Format: <xsl:template match="*pattern*" name="*string*" mode="*string*">
- ***xsl:apply-templates***
  - further templates may be considered by mode
  - Format: <xsl:apply-templates select="*expr*" mode="*string*"/>
- ***xsl:call-template***
  - calls a named template
  - Format: <xsl:call-template name="*expr*">
  - optionally with parameters: <xsl:call-template name="*expr*">
    <xsl:with-param name="*string*" select="*expr*"/>
    </xsl:call-template>

XSLT advanced. Templates, variables & parameters

- ***xsl:param***
  - "parameters" give **additional context information**, e.g. in the context of the whole stylesheet, a template (or a function)
    - parameters have a "name" and a "value"
    - before they can be used in the template, they have to be "declared" with **<xsl:param>**
    - in the template call, the value of the parameter can be set with **<xsl:with-param>**
    - in the called template, the value of the parameter can be retrieved with: **$*pname***
  - Format:
    - <xsl:param name="*string*" select="*expr*" required="yes|no"/>
    - @select: supplies a default value, in case the parameter is not set; instead of using this attribute, a default value can also be defined as content of xsl:param
  - Example:

```
<xsl:template name="presentation">
    <xsl:param name="name">anonymous</xsl:param>
    Hello! My name is <xsl:value-of select="$name"/>
</xsl:template>
```

XSLT advanced. Templates, variables & parameters

- Example:
  - <xsl:template match="/">
    <xsl:call-template name="presentation">
    <xsl:with-param name="name">Ulrike Henny</xsl:with-param>
    <xsl:with-param name="age">32</xsl:with-param>
    </xsl:call-template>
    </xsl:template>
    <xsl:template name="presentation">
    <xsl:param name="name">anonymous</xsl:param>
    <xsl:param name="age">unknown</xsl:param>
    <xsl:text>Hello! My name is <xsl:value-of select="$name"/>. I am <xsl:value-of select="$age"/> years old.</xsl:text>
    </xsl:template>

XSLT advanced. Templates, variables & parameters

- ***xsl:variable***
  - variables are very similar to parameters
  - they can **hold information** which can be used in another place later
  - they have a “name“ and a “value“
  - they are “declared“ with **<xsl:variable>**
    their value is retrieved with **$vname**
  - they are only valid in their “context“
  - Format:
    - <xsl:variable name="*string*" select="*expr*"/>
    - <xsl:variable name="*string*">...</xsl:variable>
  - Example:
    - <xsl:variable name="title" select="//titleStmt/title"/>
    - <xsl:value-of select="$title"/>

XSLT advanced. Templates, variables & parameters

- Example: HTML output of Shakespeare's “A lover's complaint“
  - XML file: shakespeare/poetry/lov.xml
  - Question: Anything interesting about the structure of this poem that could be shown?
  - Characteristics (XPath!):
    - 47 stanzas; each with 7 verse lines → in total 329 verse lines
    - 2,571 “words“, 21,856 “characters“
    - different (string) length of verse lines
    - different (word) length of verse lines
    - average: 8 words → more than 8 words: “long lines“, less than 8 words: “short lines“
  - Approach:
    - Create an HTML page containing the whole text (all the stanzas and verses)
    - Color the verse lines differently, according to their length: short – average – long

XSLT advanced. Exercises

- (1) Cleaning: Copy all of “The Discovery of Guiana“, except references to places and persons (take out the <rs>, but leave the text!)
  - Use the copy-all-but-template
  - Create another template matching the <rs> elements
- (2) Sampling: Copy just the 1st and the last paragraph of “The Discovery of Guiana“
  - Use xsl:copy-of
- (3) Collection: Collect the titles and 1st speeches of Shakespeare's comedies in an HTML list
  - Use collection()
- (4) Split: Create a single HTML file for each stanza of “A lover's complaint“
  - XML file: shakespeare/poetry/lov.xml
  - Use xsl:result-document
- (5) Variable: Copy all verse lines of “A lover's complaint“ into a variable; Loop through the variable (<xsl:for-each>), sort the verse lines alphabetically (<xsl:sort>) and write them into the result document (<xsl:value-of>)

XSLT in use

- Scenarios for using XSLT
- Visualization
  - Analysis: Charts

XSLT in use. Scenarios for using XSLT

- Data conversion
  - XML 1 to XML 2
  - XML to text
  - text to XML
  - (text to text)
- Data selection
- Data enrichment
- Publishing
- Analysis
- Visualization

XSLT in use. Visualization

- Aims:
  - Presentation of analysis results
  - Heuristic approach to data
- Possible tools:
  - Google Chart API: https://google-developers.appspot.com/chart/
  - D3 - Data-Driven Documents: http://d3js.org/
  - vis.js: http://visjs.org
  - … and many more …
  - Characteristics:
    - Web-based
    - JavaScript
    - Provide many standard chart types (Pie Charts, Bar Charts, Scatter Plots, …) and some advanced ones (e.g. Circle Packing, Treemap, Networks, …)
    - Easy to use
    - Data input: JSON

XSLT in use. Visualization

- JSON – JavaScript Object Notation
  - Open standard
  - text-based
  - „alternative to XML“
  - → can be created with XSLT!
  - Data objects consisting of attribute-value pairs
- Example:

```
{"cities":
    [{"name" : "Graz",
    "country" : "Austria",
    "districts" : ["Innere Stadt", "St. Leonhard", "Geidorf", "Lend", "Gries",...],
    "inhabitants" : 269,997},
    {"name" : "Paris",
    "country" : "France"}]
}
```

XSLT in use. Visualization. Analysis: Charts

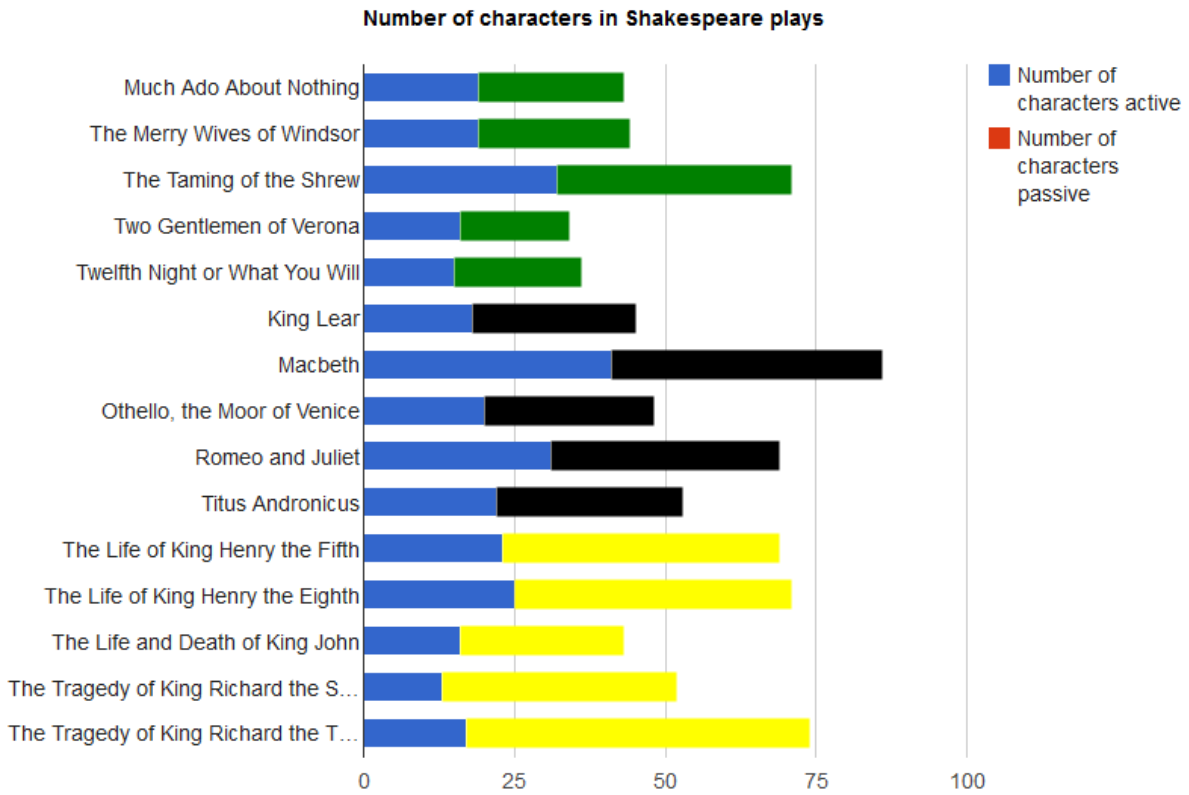
- Example: Shakespeare Plays – Types of plays
  - Question: Is it possible to differentiate between different types of plays (tragedies, comedies, „histories“) based on criteria like the number of characters/scenes/words etc.?
  - Approach: compare numbered characteristics of play types with the help of simple charts
  - Tools: XSLT & Google Chart API

XSLT in use. Visualization. Analysis: Charts

- Example: Number of characters in tragedies, comedies and histories
  - Goal: a (horizontal) bar chart showing the number of characters in the different plays/play types, distinguishing between 'active' and 'passive' characters (without speech)
  - Step 1: How to create a bar chart?
    - check out the Google Chart API
    - https://google-developers.appspot.com/chart/interactive/docs/quick_start
  - Step 2: How to get our own data into the chart?
    - create an XSLT-file producing the chart
    - replace example data with XSLT expression

XSLT in use. Visualization. Analysis: Charts

- Example: Number of characters in tragedies, comedies and histories
  - Result:



XSLT in use. Visualization. Analysis: Charts

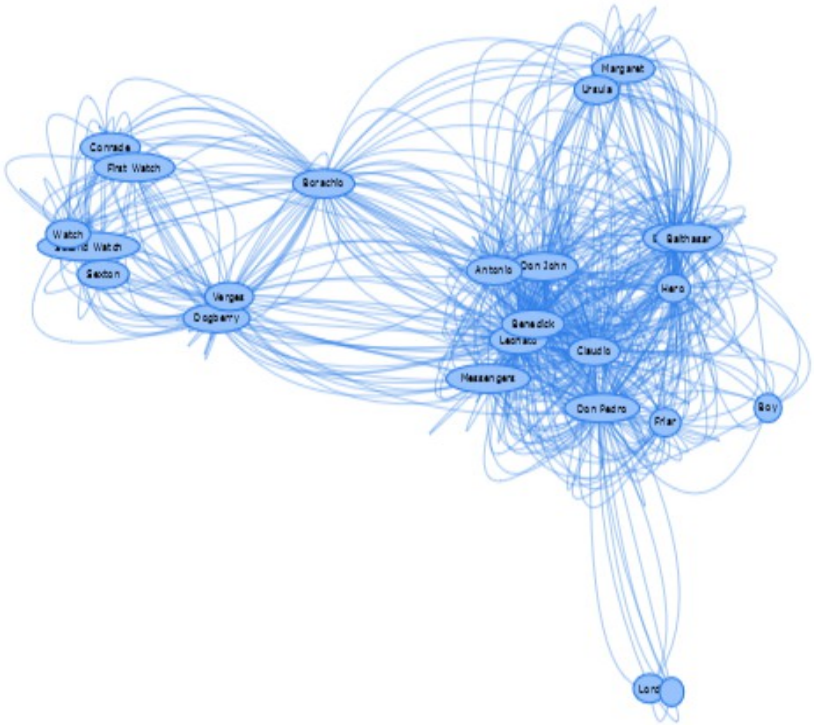
- Exercise: Number of scenes in tragedies, comedies, histories
  - Use the Shakespeare data to create a bar chart visualizing the number of scenes in each play
    - differentiated by play type (bars in different colours)
    - optionally as stacked bar chart: number of scenes per act

XSLT in use. Visualization. Analysis: Charts

- Example: Shakespeare Plays - Network
  - Question: Which characters do interact with each other?
  - Approach: who speaks in the same scene? create a network of characters
  - Tools: XSLT & vis.js
  - Step 1: How to create a network?
    - check out networks in vis.js
    - http://visjs.org/network_examples.html
  - Step 2: How to get our own data into the network?
    - create a new XSLT file which produces the network
    - replace example data with Shakespeare data

XSLT in use. Visualization. Analysis: Charts

- Example: Shakespeare Plays – Network
  - Character interaction in „Much Ado About Nothing“



XSLT in use. Visualization. Analysis: Charts

- Exercise: Character interaction in Shakespeare plays
  - Create speaker networks for other plays and compare the results

Thank you for your attention!

- References:
  - Short W3C Tutorial: http://www.w3schools.com/xsl/default.asp
  - W3C Recommendation for XSLT 2.0: http://www.w3.org/TR/xslt20
  - Kay, Michael, *XSLT 2.0 and XPath 2.0 Programmer's Reference (Programmer to Programmer), Wiley Publishing* 2008[4].