Report Project 2 : XSLT

Rémy Detobel & Nathan Liccardo

April 12, 2018

1 Introduction

This document aims to detail all the choices that we made during our implementation. The main goal of this project was to create an xslt file which is able to transform a part of the dblp database into HTML files. Our xslt file has been separated into two parts (functions and template) which are described in this document.

2 Functions

Our project use five different functions to implement complex operations (`removeSpecialChar`, `lastName`, `firstName`, `firstLetter` and `nameToPath`). Each function and their complex operations would be described in this section.

2.1 Remove Special Char

This first function is used to replace non-alphanumeric characters in a given input string. It takes, as input, a person name and replace each characters which matches with the following regex :

- `[^0-9a-zA-Z_]`

If we have a match, then we substitue the character with the equal character (=). Finally, we obtain the below instruction, where we also use the `replace` function defined by xslt 2.0 :

- `<xsl:value-of select="replace($input, '[^0-9a-zA-Z_]' ,'=')"/>`

.

2.2 Last Name

This second function is used to extract the last name from a given author or editor string. We extract and format last names using the below functions :

1. `tokenize($input,' ')[last()]` : we split the input (using white spaces) and get the last element.
2. `removeSpecialChar(tokenize($input,' ')[last()])` : use the previous function to replace special characters in last names.

Finally, we obtain the following instruction :

- `<xsl:value-of select="func:removeSpecialChar(tokenize($input,' ')[last()])"/>`

.

2.3 First Name

This third function is used to extract first name from a given input string. Formally, first name is corresponding to all the input string minus the last name (see previous section). This element can be composed by multiple words (separated by withe spaces) and having special characters. Extracting first name was more complicated and has been implemented as below.

1. `substring-before($input, tokenize($input,' ')[last()])` : get last name.
2. `replace(..., ' ','_')` : remove white spaces[1].
3. `removeSpecialChar(...)` : replace special characters.

Finally, we obtain the following instruction :

- ```
  <xsl:value-of select="func:removeSpecialChar(
      replace(substring-before($input, tokenize($input,' ')[last()]), ' ' ,'_'))"/>
  ```

2.4 First Letter

This fourth function is used to extract the first letter of the last name. We call the previous function and exactract the first letter :

1. `substring(func:lastName($input), 0, 2)` : get the first letter of the last name.
2. `lower-case(...)` : transform upper-case to lower-case.

Finally, we obtain the following instruction :

- ```
  <xsl:value-of select="lower-case(substring(func:lastName($input), 0, 2))" />
  ```

2.5 Name To Path

This last function is used to properly format each HTML files name. Thanks to previous functions, we are now able to : return first letter (last name), return first name and return last name. Those three strings will be concatenated as follow :

- `"firstLetter"/"lastName"."firstName"`

Where each strings are given by corresponding functions. Note that in the dblp database, some values in author tag does not have any space. Thus we could not split the string to have first and last name. We have therefore arbitrarily decided to consider this as the last name and ignore the first name. To do that, we just added a condition. Formally, our code is structured as below :

- ```
  <xsl:value-of separator="">
      <xsl:value-of select="func:firstLetter($input)"/>
      <xsl:value-of select="'/'"/>
      <xsl:value-of select="func:lastName($input)"/>
      <xsl:if test="count(tokenize($input,' ')) > 1">
          <xsl:value-of select="'.'"/>
          <xsl:value-of select="func:firstName($input)"/>
      </xsl:if>
  </xsl:value-of>
  ```

3 Template

Templates are used to define set of rules. One project can use multiple templates to define different transformations rules. Our project contain only one template which is used to describe how to parse the XML database into HTML files. To realise this, our template use at the same time Xpath and HTML code. Xpath is used to find specific nodes inside the XML structure. Those nodes are then converted into HTML code using XSLT. This section describe how we use those elements to implement our project.

[1] ... : reference to the previous point.
  ```

3.1 Creation of HTML pages

Each author or editor must have an HTML page. This means that we must iterate each different editor or author nodes in the dblp database. To realise this, we use the `for-each` tag with the following path :

- `<xsl:for-each select="distinct-values(//author|//editor)">`

Then, for each element we create an HTML file. This is created using the `result-document` tag as below :

- `<xsl:result-document href="{func:nameToPath(.)}.html">`

We use the `nameToPath` function previously defined to format the HTML file name.

3.2 Title

Each pages start by the person name within `<h1>` tags. This title can be given using the operator `"."`. This is due to the fact that we are currently looping on a list of authors/editors. Below is the final instruction used to print the person name :

- `<h1> <xsl:value-of select="."/> </h1>`

3.3 Publications table

For each person, we must have a table which contain each publications of this person. For each publication we must realised multiple operations. Those operations are going to be detailed below. Remark also that we have created a global variable called `author` which is used to retain the current author name.

3.3.1 Obtaining publications

For each person, we must retrieve all of its publications. This is realised using a global variable called `publications`. To initialise this variable, we must retrieve each publications where one of the author is the same as the one contained in the `author` variable. The is realised as follow :

- `<xsl:variable name="publications" select="$root//*[./author=$author]" />`

Where the path contained in the `select` argument is used to compare `author` variable with authors of the current publication.

3.3.2 Ordering publications

We must ordering all publications of the current person in the descending order. This means that we will start with the oldest publication and finish with the newest one. This is realised using the `sort` xslt command coupled with the `descending` argument. This argument is referring to the `year` tag (inside each publication).

- `<xsl:sort select="year" order="descending"/>`

Each row (corresponding to a publication) in the table must have an index. As we are printing publications in the reverse order, we must also obtain indexes in the reverse order. Once again, this is realised using a variable defined as follow :

- `<xsl:variable name="indexPub" select="last()-position()+1" />`

Where we compare the current and last position. Finally, below in the code, we simply add the value of (using `value-of` operator) the `indexPub` variable.

3.3.3 Year subsection

As publications are ordered by year of publication, we must add (inside the table) specific rows used to separate publications of different years. To realise this, we have decided to compare year of the current node with year of the previous node (due to the ordered aspect). Condition is implemented using the `if` tag with the following path :

- `<xsl:if test="not(preceding-sibling::*[1]/year=./year) or position()=1">`

Remark that we also test if the current node is the first one. If it is, we do not need to apply the comparison and we can directly add the specific row (year).

3.3.4 Personal redirection

For each author/editor of the current publication, we should link his name with his personal page. The tricky part is that the we do not need to link the current person to his personal page (which is the current page). This is realised using following instructions :

- ```
  <xsl:if test="not(.=$author)" >
      <a href="../{func:nameToPath(.)}.html">
          <xsl:value-of select="." />
      </a>
  </xsl:if>
  <xsl:if test=".=$author" >
      <xsl:value-of select="." />
  </xsl:if>
  ```

3.4 Co-author table

For each page, we must have a table which contain all persons with whom P has jointly published. This new table is realised using two nested loops.

3.4.1 People loop

First loop iterate on authors/editors which are present on same publications than the current person. This first iteration is realised as follow :

- `<xsl:for-each select="$root//*[./author=$author]/author[not(.=$author)]">`

This table is order by last names. This is done by using the `sort` tag with the `lastName` function.

- `<xsl:sort select="func:lastName(.)"/>`

3.4.2 Publications loop

For each co-author, we must retrieve each publications where the co-author and the author are present. The nested loop use the `publications` variable defined before and check if the current co-author is present as follow :

- `<xsl:if test="$coAuthor=./author" >`

Then if it is the case, we can add the publication which is linked to the corresponding item in the previous table :

- ```
  <xsl:variable name="linkPublication" select="last()-position()+1" />
  [<a href="#p{$linkPublication}">
  <xsl:value-of select="$linkPublication" />
  </a>]
  ```