Report Project 2 : XSLT

Rémy Detobel & Nathan Liccardo

May 10, 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) 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 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. To do this, we use the functions below :

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 name.

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 first name.
2. `replace(..., ' ','_')` : replace white spaces by underscore characters[1].
3. `removeSpecialChar(...)` : remove special characters.

Note that we cannot use the previous function to get last name (and then remove it) because the previous function convert some character (with function `removeSpecialChar`).

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. To reach this first character, we use below functions :

1. `substring(func:lastName($input), 0, 2)` : get 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. Finally, we concatenate these three texts as below :

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

Where each string is given by the corresponding function. Remark that some author values does not have any space. Therefore, first name and last name could not be separated using withe spaces. To solve this problem, we have decided to consider the given string as a last name. This is formally expressed by the following expression :

- ```
  <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 sets of rules. One project can be used to define multiple templates and then multiple sets of rules. Our project only contains one template which describe a set of rules to parse the XML database into HTML files. To realise this, we use both XPath (used to find specific nodes) and HTML (used to format output files). This section would be focused on the use of those elements in the context of this project.

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

3.1 Creation of HTML pages

An HTML page must be created for each author or editor. This is realised by iterate on each different author/editor using the `for-each` tag with a specific path (using XPath) :

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

Then, when we have a set with each different authors/editors, we can create each corresponding files. This is realised using the `result-doucment` tag (defined in xslt) :

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

3.2 Title

Each page start by the person name within `<h1> ... </h1>` tags. We can reach author/editor name using the path `"."` due to the fact that we are on an editor/author node. Finally, we obtain the below instruction :

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

3.3 Publications table

Each page must contains two tables : publication table and co-authors table (next section). The publication table contains multiple informations described by following sub-sections. Due to the fact that we will probably iterate on different elements, we have decided to save the current author/editor value in a specific `author` variable.

3.3.1 Obtaining publications

For each person, we must retrieve all of its publications. This is realised using the `publications` variable which contains all of the articles associated to the current `author` variable :

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

This variable will be used to make a loop but also in the "co-author" table (see point 3.4).

3.3.2 Sort of publications

All the publications must be sorted 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 using the `year` tag of the current publication :

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

Each row (which corresponds to a publication) in the table must have an index. As we are printing all the publications in the reverse sequence, we must also obtain indexes in the reverse sequence. Once again, this is realised using a variable :

- `<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 the `indexPub` variable using the `value-of` operator.

3.3.3 Year subsection

As publications are sorted by year, we must add specific rows to separate publications of different years. To realise this, we have decided to compare the year of the current node with the year of the previous node (due to the sorted aspect) :

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

Second part of the `or` condition is used to test if the current node is the first item. In that case, we do not need to apply the comparison and can directly add the specific row.

3.3.4 Link to the online version

The tag ee contain a link to the online version of the current document. In the case where this tag exists, we need to display a clicked image like this:

- ```
  <xsl:choose>
    <xsl:when test="ee">
      <td valign="top">
        <a href="{./ee}">
          <img alt="Electronic Edition" title="Electronic Edition"
              src="http://www.informatik.uni-trier.de/~ley/db/ee.gif"
              border="0" height="16" width="16"/>
        </a>
      </td>
    </xsl:when>
    <xsl:otherwise>
      <td />
    </xsl:otherwise>
  </xsl:choose>
  ```

3.3.5 Personal redirection

We must transform, for each author/editor, the name of the person into a link. This link is used to redirect to the personal author/editor page. The main difficulty is to avoid the transformation of the current author name. This is realised using below instructions :

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

3.3.6 Document information

We display all information about a document in the same order as the one one the dblp webiste. Thus for a lot of tag we just test if there exist and display like this:

- ```
  <xsl:if test="publisher" >
    <xsl:value-of select="publisher" /> 
  </xsl:if>
  ```

3.4 Co-author table

As indicated in the previous section, each page must contain two tables : publications table (previous section) and co-authors table. The co-authors table must contain all persons with whom P has jointly published. For this table, we have decided to use nested iterations. This is due to the fact that we must retrieve each co-author and then reach each article jointly published for each of the co-authors.

3.4.1 Co-authors iteration

The First iteration is used to find each person present on one of the current person publication. This is realised as below :

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

As the co-authors table is sorted by last names, we use (as following) the `sort` tag with the `lastName` function :

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

3.4.2 Publications loop

This second iteration (nested iteration) is used to reach each publications which are jointly published. Firstly, we use the `publications` variable to check if the current co-author is present as in the current publication :

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

If he is present, we can add the publication (linked to the previous table) :

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