XML and Web Technologies

INFO-H509

XSLT

30 April 2021

BAKKALI Yahya : 000445166
HAUWAERT Maxime : 000461714

Université Libre de Bruxelles (ULB)

Contents

1 Introduction

DBLP is an online bibliographical database for computer science containing around 1 million references. Its content is publicly available in XML format. The goal of this project is to write an XSLT program that will transform a small excerpt of this data into several HTML files that emulates a part of the DBLP website, in each file a specific person's publications will be displayed as well as other useful information. In this report, all the hypotheses that have been made as well as some implementation choices.

2 Templates

Several templates have been used to improve the readability of the code.

2.1 Base

It is the base template; it creates all the pages of the authors.

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

It loops on all the people : `distinct-values(/dblp/*/(author | editor))` and call the construct_author_page template with the name of the author and its publications : `/dblp/*[(author | editor)=$person]`

2.2 construct_author_page

It takes the name of the author as well as their publications and creates their page in the right directory.

```
<xsl:template name="construct_author_page">
       <xsl:param name="author"/>
       <xsl:param name="author_publications"/>
       ...
</xsl:template>
```

It first sorts the publications by descending year then by ascending title if they have the same year, using a perform-sort.

```
<xsl:variable name="author_publications" as="element()+">
       <xsl:perform-sort select="$author_publications">
              <xsl:sort select="year" data-type="number" order="descending"/>
```

```
            <xsl:sort select="title" order="ascending"/>
        </xsl:perform-sort>
</xsl:variable>
```

Then it calls the construct_table and construct_index templates.

2.3 construct_table

It takes the name of the author as well as their publications and creates a table that contains all their publications grouped by year sorted by titles.

```
<xsl:template name="construct_table">
        <xsl:param name="author"/>
        <xsl:param name="author_publications"/>
        ...
</xsl:template>
```

It loops on the publications and calculates their publication number : `last()-position()+1`, then verifies if it is a new year : `position()=1 or not(preceding-sibling::*[1]/year=year)` and indicates it if it is, finally it puts the publication number, the link to the online version (if it exists) and calls the construct_bibliographic_reference template.

2.4 construct_index

It takes the name of the author as well as their publications and creates a co-author index where for each co-author all the joint publications are referenced.

```
<xsl:template name="construct_index">
        <xsl:param name="author"/>
        <xsl:param name="author_publications"/>
        ...
</xsl:template>
```

It loops on all the co-authors : `distinct-values($author_publications/(author | editor)[not(.=$author)])` sorted by their last names, then loops on the publications `$author_publications` and put its publication number if the co-author was the author or editor of the publication : `./(author | editor) = $coAuthor`.

2.5 construct_bibliographic_reference

It takes a publication and creates its bibliographic reference.

```
<xsl:template name="construct_bibliographic_reference">
       <xsl:param name="publication"/>
       ...
</xsl:template>
```

It goes through all the fields of the publication and put all of them in a format that looks like the one of the DBLP website.

3 Functions

Several functions had to be created to reduce code duplication as well as to improve readability.

3.1 getFirstname

It takes a full name and return only the first name. The first name has been defined as the part of the full name before the first space.

```
<xsl:function name="f:getFirstname">
       <xsl:param name="fullname"/>
       <xsl:value-of select="substring-before($fullname, ' ')"/>
</xsl:function>
```

`Example` : f:getFirstname(”David Maier”) returns ”David”

3.2 getLastname

It takes a full name and return only the last name. The last name has been defined as the part of the full name after the first space. If the last name was defined by the part of the full name after the last space, the middle name would have been excluded but it would have created some collisions, two different people having the same first and last names but a different middle name.

```
<xsl:function name="f:getLastname">
       <xsl:param name="fullname"/>
       <xsl:value-of select="substring-after($fullname, ' ')"/>
</xsl:function>
```

`Example` : f:getLastname(”David Maier”) returns ”Maier”

3.3 getFirstLetterOfLastname

It takes a last name and return only the first letter and set it to lowercase.

```
<xsl:function name="f:getFirstLetterOfLastname">
       <xsl:param name="lastname"/>
       <xsl:value-of select="lower-case(substring($lastname,1,1))"/>
</xsl:function>
```

`Example` : f:getFirstLetterOfLastname(”Maier”) returns ”m”

3.4 getPath

It takes a full name and return the path as described in the assignment.
/first-letter-of-lastname/lastname.firstname.html

```
<xsl:function name="f:getPath">
       <xsl:param name="fullname"/>
       <xsl:variable name="fullname" select="f:toAlphaNumeric($fullname)"/>
       <xsl:variable name="lastname" select="f:getLastname($fullname)"/>
       <xsl:variable name="firstname" select="f:getFirstname($fullname)"/>
       <xsl:variable name="flol" select="f:getFirstLetterOfLastname($lastname)"/>
       <xsl:value-of select="concat('/', $flol, '/', replace($lastname, ' ',
           '_'), '.', replace($firstname, ' ', '_'), '.html')"/>
</xsl:function>
```

`Example` : f:getPath(”David Maier”) returns ”/m/Maier.David.html”

3.5 toAlphaNumeric

It takes a string and return it with all its non alpha-numeric letters, except spaces and underscores, replaced by an equal sign.

```
<xsl:function name="f:toAlphaNumeric">
       <xsl:param name="s"/>
       <xsl:value-of select="replace($s, '[^a-zA-Z0-9_ ]' ,'=')"/>
</xsl:function>
```

`Example` : f:toAlphaNumeric(”José”) returns ”Jos=”

4 Character map

In the given excerpt of the DBLP database the character, decimal 150, was present in the names of some the people. The problem is that this character is illegal in HTML, so to tackle this problem a character map was used. It replaces this character with an underscore.

```
<xsl:character-map name="illegalCharacters">
       <xsl:output-character character="&#150;" string="_"/>
</xsl:character-map>
```

There is obviously more than one character that are illegal in HTML but to keep it simple, it has been decided to only take care of the decimal 150 character as it is the only present in the excerpt. In the future if more of these characters are added, they just have to be added in this map.

5 Program manual

To launch the program, the following command should be executed :

```
$ java -jar xslt-tool.jar generate-author-pages.xslt input_file null
```

It will create all the requested files in the dblp directory.

`input_file` should be replaced by the DBLP XML filename path that you want.

`Note` : the program requires an output_file argument but the XSLT will not write anything on it, it can be replaced by anything but must nevertheless be present.

`Example` :

```
$ java -jar xslt-tool.jar generate-author-pages.xslt dblp-exceprt.xml null
```