Lab exercises for theme 6 – Ontology Alignment

Aim
After completing this lab you should be familiar with a tool for performing ontology alignments, the *Alignment API*. In the first part of the lab you will go through a tutorial where the *Alignment API* is used as a tool by invoking it from the command line. In the second part of the lab you will extend the *Alignment API* itself by working with its source (which is written in Java) in an *Eclipse* project. We will be using version 4.0 of the *Alignment API*.

1. **Alignment API tutorial**
   The first task is to go through the *Alignment API* tutorial that can be found at:

   http://alignapi.gforge.inria.fr/tutorial/tutorial1/index.html

   Before you start you need to do the following preparations (and you should ignore the preparation step in the tutorial).
   The command-line environment we will use in the tutorial is *Windows PowerShell*. A tip that might be useful when following the instructions below is how to paste into a *PowerShell* window: that's done using the right mouse button. When you start *PowerShell*, its current directory should be in your home directory. If it's not, you can always change to your home directory with the following command:

   ```
   cd ~
   ```

   When you've made sure you're standing in your home directory, it's time to create a new directory named *aligntut* for holding files related to the tutorial:

   ```
   mkdir aligntut
   ```

   In that directory, create a directory called *results*:

   ```
   mkdir aligntut\results
   ```

   In the tutorial linked above, in the section labeled *The Data*, download the two OWL-files, *myOnto.owl* and *edu.mit.visus.bibtex.owl* and place them in the *aligntut* directory. Make sure you download the RDF/XML versions that have the .owl-extension and not the HTML versions. You might have to right-click the links and select something like "*Save Target As...*" to prevent the browser from rendering the OWL-files. This depends on the browser used and its settings. If you are using *Internet Explorer*, make sure you save the files with their original .owl extension and not .xml. Apart from the two OWL-files, you also need the file *data.xml* which you can find a link to in the section named *Output* and, finally, you need the file *refalign.rdf* which you can find under the section named *Evaluating*.
   To be able to follow the tutorial, some paths and such need to be set-up. We would have liked to provide a *PowerShell* script that could help with this, but due to security restrictions it's not possible to execute *PowerShell* scripts on the lab computers.
   java.exe should already be in the path on the lab computers (which you can confirm by issuing the command *java -version*), but if it isn't, you must add it to the path with the following command:

   ```
   $env:Path += ";C:\Program Files\Java\jre6\bin"
   ```

   The next step is to create a shell variable pointing out *procalign*:

   ```
   $procalign = "C:\Program Files\align-4.0\lib\procalign.jar"
   ```

   We also need a variable that holds the absolute path to the tutorial directory and that path must not contain any backslashes. First we need to make sure you are standing in the *aligntut* directory that you created in your home directory earlier. To change to that directory, no matter what the current directory is, you can issue the following command:

```
cd ~\aligntut
```

Now we can create the variable we need:

```
$cwd = $pwd.toString().Replace('\', '/')
```

If you did that correctly then the variable *$cwd* should now contain the absolute path to the *aligntut* directory (which should be directly under your home directory) and all backslashes should have been replaced with forward slashes. You can check the value of a variable by simply typing its name, like `$cwd`.
Now we are ready to start the actual tutorial. The commands we will enter will differ slightly from what is shown in the tutorial, so therefore we will show how to write the first command and from that you should be able to do the rest:

```
java -jar $procalign file:///$cwd/myOnto.owl file:///$cwd/edu.mit.visus.bibtex.owl
```

Notice that we use three forward slashes in the URI:s passed to *procalign* instead of two that is used in the tutorial.
Do the following sections: *Matching*, *Manipulating*, *Output*, and *Evaluating*, but skip the last section *Embedding*. In some sections there are parts marked as *More Work* than you can do if you want, but skip any parts labeled *Advanced* that involves *WordNet* since that's not installed on the lab computers.
In the last step in the *Output* section you have to add the directory where *xsltproc* is located to the path for the shell to find it and you do that by issuing the following command:

```
$env:Path += ";c:\Program Files\Oxygen XML Editor 10"
```

You don't have to demonstrate or write a report about the tutorial, but please go through it carefully anyway.

See the next page for part two of this lab.

2. **Alignment API matcher**

In this part of the lab you will extend the Alignment API with your own matcher. You will also write a test program for it. We have prepared an Eclipse project named Theme 6 Alignment API for you. Download it and import into Eclipse the same way you imported the Jena project in the previous excercise.
The matcher you will implement belongs to the family of string distance matchers, this one works on substrings. It compares two strings and calculates a normalized distance of their substrings. Here's how it should work:

- If the strings are equal, return 0 (yes, 0, because we are calculating distance not equality).
- If the strings are not equal, create two sets, one for each string. The sets should contain the 1-, 2-, and 3-grams for its respective string. When both sets have been constructed you must determine their intersection. For each intersecting element add its length to a counter, that we can call *totalLength*, which would contain the total length of all intersecting elements. When you have *totalLength*, you can calculate:

  ```
  abs(1.0 – totalLength / ((s1.length() + s2.length()) * 2))
  ```

  Here s1 and s2 denote the two strings being compared. The formula above is written in pseudocode, in the Java version you want to make sure that you cast any integers to doubles where appropriate.

Your test program should be contained in a class that inherits from *fr.inrialpes.exmo.align.impl.method.StringDistAlignment*. It should only contain a constructor that makes it use your particular matcher and a main method where you should place your test program. For ideas how the test program can be constructed you should consult the Alignment API documentation. You must figure out in which class your matcher belongs. You can create helper methods for matcher if you like, just make sure they are only visible inside the class.
For rendering the results you should use an *HTMLRendererVisitor*.
Basically you should contribute one new class (as described above) and extend another with a new matcher. If you find yourself needing to do more than that, you are not using the API as we intended. Don't worry about writing a fast implementation of the matcher, but do make sure you write clean and readable code.
Also, when performing alignments, it's important to understand that aligning A and B is not the same as aligning B and A. Say you start with the unmodified university ontology from the previous lab and align that with your slightly extended version. In this case, every class and instance from the unmodified university ontology can find a perfect match in your extended ontology. However, if you do the alignment the other way around, there will be objects in your extended ontology that have no perfect match in the unmodified university ontology.

Hand in a report with a print-out of all code you have written and the (rendered) HTML code of its output when run on the same ontologies you used in the tutorial.