Integrating Ontology Matching Tools into the SEALS Platform

Christian Meilicke, Cássia Trojahn, Daniel Faria
Jakob Huber and Jérôme Euzenat

May 20, 2016

Abstract

This tutorial explains how to prepare, package and zip an ontology matching tool to be integrated into the SEALS platform. It explains how to check that the generated zip-file is valid and can be executed on the platform. Further, it describes how the wrapped tool can be evaluated locally with one of the Benchmark, Conference, and Anatomy test suites used in the Ontology Alignment Evaluation Initiative.

1 Introduction

Wrapping a matching tool consists of mainly two steps. (1) You have to place all required libraries in a certain folder structure and (2) you need a simple java class that acts as a bridge between the signature of the align-method of your system and the signature expected by the SEALS platform.

The exptected folder structure looks like this:

```
descriptor.xml  * describes package content
bin/
   lib/
        * tool libs and other libs required by your tool
        * bridge class zipped as .jar
        * (X) some *.bat or *.sh scripts
conf/
        * files required by your tool to be accessed at runtime
lib/ (X)
        * (keep empty)
```

The SEALS platform is currently under development. To be compliant with future versions the package has to contain some elements that are currently not used. They are marked with an (X). The descriptor file is an XML file that describes the content of the package in simple and self-explaining

way. You can download a full example `demomatcher-package.zip(windows)` or `demomatcher-package.zip(linux)`[1].

The interface that has to be implemented in the bridge has the signature `URL align(URL sourceOnt, URL targetOnt)` where it is expected that the generated alignment is stored in a temporary file and the URL of this file is returned. Note also that the generated file is expected to be conform to the format of the Alignment API[2].

In the following we show how to generate such a package and its bridge. If you are a developer of a tool that implements the alignment API you can continue with §2 and skip over §3. If this is not the case you can skip over §2 and continue directly with §3.

2 Simplified procedure for systems implementing the Alignment API

For those matchers implementing the Alignment API, no programming is necessary. An ant script is able to generate the bridge, compile it, package it, and validate it. The procedure is the following:

1. get `package-template.zip`, unzip it and move to its content;
2. check that everything works by calling `ant pack`, `ant validate`, if all return a success, then perform `ant origin`;
3. move all the jars needed for running your matcher, including the Alignment API jars, to the `bin/lib` directory (subdirectory authorised);
4. edit `local.properties` (important properties are `classname`, the fully qualified name of the class to implementing `AlignmentProcess`, and `scriptext`, generating a Unix or Windows package);
5. run `ant pack`.

If your matcher does not rely on external resources besides jar-files, this should be done. You have a zip file available which is your package and you can progress to §5 (in fact, you can even execute the step of §5.1 by running `ant validate`).

On the other hand, if it needs some runtime configuration files, these could be put in the `conf` directory, before running again `ant pack`, but then it may be necessary to edit `src/bridge/Bridge.java` for telling your matcher where to get these configurations.

If it is more complex, it is unfortunately necessary to rely on the documentation and procedure of §3. However, you can start from what you have built: the `descriptor.xml` file and the `src/bridge/Bridge.java` Java file.

[1] All files can be downloaded from `http://oaei.ontologymatching.org/2011/tutorial/`.
[2] `http://oaei.ontologymatching.org/2011/align.html`

3 Standard procedure for other tools

In the following we assume that you are using eclipse. Note that this is not a requirement, but just helps to keep the explanation brief. These instructions require that you have downloaded `demomatcher-package.zip(windows)` or `demomatcher-package.zip(linux)` as well as `seals-omt-client.jar`. Our demomatcher example is described briefly in Appendix A.

1. Create a new eclipse project and ensure that all libraries required by your matcher are on the build path (for our example it was `demomatcher.jar`, `simmetrics.jar` and `owlapi.jar`; for your matcher it will be some other libraries).

2. Add also the jar file `seals-omt-client.jar` to your build path. It contains besides some other classes also the interfaces you have to implement.

3. Create some package and a class that extends `AbstractPlugin` and implements `IOntologyMatchingToolBridge` as shown in Appendix A.4 (download `MatcherBridge.java`). This class acts as a bridge between the functionalility of your tool and the functionality required by SEALS. Do not change the methods `canExecute` and `getType`. The two `align` methods are specific for your tool.

4. Export your tool bridge as a jar-file. You do not need to include any of the libraries on your build path into that jar, only the class you have created in step 3.

5. Copy the folders of the packaged tool of our example, remove the jar-files of the example and replace them by the jar-files that are required by your matcher.

6. Add to the directory `conf` the resources required at runtime by your system. These files will at runtime be located at the current working directory.

7. Modify the `descriptor.xml` according to these changes. Also change the meta information in the first part of the `descriptor.xml` file. Specially, you need to specify the `class bridge` and the library `dependencies`.

8. Zip the content of the package (do not include the package folder itself in the zip-file).

Even if your system does not implement the Alignment API, you can use the ant facilities presented in §2 to package your tool instead of manually following all steps listed here. For that, you have to follow the steps indicated in §2 as well as you have to write the `src/bridge/Bridge.java` file (or modify the provided template example).

4 Using additional resources

Most matching systems require some additional resources to be available during execution in order to run properly. Examples are configuration files, lists of stopwords, dictionary files of Wordnet, and many other kind of resources. If such resources are required by a matching system, they have to be copied to the directory `conf`, see also Appendix §A.1. The SEALS platform will in the deployment phase of the tool copy the contents of this folder (recursively, if there are sub folders) into the working directory where the matching process will be executed later on. Note that the files that have to be stored can typically be found in the base folder of a matching system.

For instance, consider that *demomatcher* requires the file `threshold.txt` to be available in the directory `configuration/` relative to the working directory where *demomatcher* is executed.[3] For that reason we have placed it in the package of the matcher as shown in §A.1.

If your system requires some additional libraries/applications/servers that have to be installed prior to running the matcher, you have to inform us about this. We will take care that required software is installed on the machines where your system will finally by evaluated.

5 Test your packaged tool

There are three types of validation available for testing your tool-package. First, there is a validation of the structure and content of the zipped package. Second, there is a test in which you run the packaged tool in the same way as it will be run by the platform. Third, there is a test in which you run a complete evaluation including the computation of precision and recall.

5.1 Validating structure and content

The jar-file `seals-omt-validator.jar` can be used to check the structure and content (e.g., correctness of references) of the package. For that purpose you need to have your tool zipped as described in the last step of §3. If you have followed the steps in §2, `ant pack` generates such zip file (`mytool.zip` in the provided example, as indicated in the `local.properties` file).

In our example we called the resulting zip-file `demomatcher-package.zip`. Given that in the current working directory we have `seals-omt-validator.jar` and `demomatcher-package.zip`, we can check the zipped package with the following command.

```
A:\temp>java -cp seals-omt-validator.jar
eu.sealsproject.platform.res.tool.utils.clients.validation.PackageValidator
-v all -r demomatcher-package.zip
```

[3] In most cases the settings or configuration file(s) will be placed directly in `conf`. We have used this example to illustrate that it is possible to use subdirectories if required.

In case of a successful test, the following output is displayed.

```
log4j:WARN No appenders could be found for logger+
(eu.sealsproject.platform.res.tool.bundle.factory.impl.DirectoryNormalizer).
log4j:WARN Please initialize the log4j system properly.
Package 'A:\temp\demomatcher-package.zip' is valid
Package structure validation report:
- Tool package configuration: [Passed]
- Binary directory..........: [Passed]
- Configuration directory...: [Passed]
- Library directory.........: [Passed]

Package descriptor validation report:
- PackageConfiguration.............: [Passed]
- DeployCapabilityConfiguration....: [Passed]
  + Shell script configuration:
    - Script file 'deploy.bat' found.
- UndeployCapabilityConfiguration..: [Passed]
  + Shell script configuration:
    - Script file 'undeploy.bat' found.
- StartStopDependency..............: [Passed]
- StartCapabilityConfiguration.....: [Passed]
  + Shell script configuration:
    - Script file 'start.bat' found.
- StopCapabilityConfiguration......: [Passed]
  + Shell script configuration:
    - Script file 'stop.bat' found.
- InvokeCapabilityConfiguration....: [Passed]
  + Java application configuration:
    - Jar file 'demomatcher-bridge.jar' found.
    - Class 'de.unima.ki.demomatcher.seals.MatcherBridge' found.
    - Dependencies specified:
      + Dependency 'lib/demomatcher.jar' found and valid.
      + Dependency 'lib/owlapi.jar' found and valid.
      + Dependency 'lib/simmetrics.jar' found and valid.
- ShellScriptFileUniqueness........: [Passed]
```

In this example the validation has been passed successfully. In case of problems, they are reported.

5.2 Running the tool

In the SEALS platform your matcher will be executed in a certain folder. To simulate this behavior for the purpose of doing this test, you first have to set the system variable `SEALS_HOME` to some folder on your machine where you have full read, write and execution rights. Under Windows use the following command.

```
A:\temp>set SEALS_HOME=A:\seals\sealshome
```

If you are using a Unix system you have to type the following:

```
christian@arnheim:~$ export SEALS_HOME="/home/christian/sealshome"
```

Note that this variable is only valid in the same terminal. Please change now the current directory to the `SEALS_HOME` directory in order to execute your matcher! We have prepared a client which connects to the tool via its bridge and executes its `align` method as it will be the case when the tool is deployed in the SEALS platform. It is available in the jar-file `seals-omt-client.jar` that contains also the interfaces to be implemented. The client works directly on the unzipped tool package, so please unzip your tool package before proceeding (in our example we unzipped `demomatcher-package.zip` into `A:/temp/demomatcher-package`).

To test the SEALS client, you should run it with the following command (note that it is a single command without any line breaks).

```
A:\seals\sealshome>java -jar A:/temp/seals-omt-client.jar
A:/temp/demomatcher-package [OPTIONS]
```

Please ensure that you execute this command from the `SEALS_HOME` directory and ensure that the path to the jar-file and the unzipped tool-package is valid. In case of problems use absolute paths. If you run this command, it generates the following output. The options are the following:

```
> Predefined test: "<packageLocation> <-t>"
> Predefined test with input alignment: "<packageLocation> <-ti>"
> Parametrized test: "<packageLocation> <-o> <ontologyURL1> <ontologyURL2>
[<inputAlignURL>] [<-f> <outputAlignFile>] [<-i> <errorRate>] [<-z>]"
>>> The "-f" option allows the specification of the file name under which the
output alignment is to be saved
>>> The "-i" option activates interactive matching (with the given error rate);
it requires an "inputAlignURL"
>>> The "-z" option activates batch mode - no command line input will be required
to continue
> Run suite: "<packageLocation> <-x> <repUri> <suiteId> <versionId> <outputFolder>
[<-a>] [<-z>] [<-i> <errorRate>] [<-s> <resultsId> <toolName>]"
>>> The "-a" option causes all tests in the suite to be run, including pairs with
no reference alignment (which are ignored by default)
>>> The "-z" option activates batch mode - no command line input will be required
to continue
>>> The "-i" option activates interactive matching (with the given error rate, and
for tasks which have a reference alignment)
>>> The "-s" option activates store mode
```

- `-t` Two predefined ontologies from the Conference test suite in the SEALS test repository are used as input to your matching system. To run this simple test, a connection to the internet is required.
- `-ti` Same as `-t`, however, an alignment is added as input additionally. Requires that the tool implements this specific interface.
- `-o` You must specify as additional arguments the URLs of two ontologies, which can for example be locally stored on your machine. You can optionally provide an input alignment as well, which will be used as in the `-ti`

option, or you can provide both an input alignment and a interactive error rate with `-i`, in which case the input alignment will be used to simulate a user in interactive mode.

- `-x` You can run a full evaluation locally on your machine and store the evaluation results in a tab-separated output file. Again, a connection to the internet is required to retrieve the input ontologies. Do not forget the argument which specifies the folder, outside of the `SEALS_HOME` directory, in which results are stored. You can find the latest URLs and IDs pointing to available testsuites at `http://oaei.ontologymatching.org`. If you provide an interactive error rate with the `-i` option, the evaluation will be done in interactive mode.

We recommend to start with `-t`. Here is the full command:

```
A:\seals\sealshome>java -jar A:/temp/seals-omt-client.jar
A:/temp/demomatcher-package -t
```

If everything is fine and you are connected to the internet, the following messages are printed to the screen.

```
>>> Preparing environment ...
>>> All files are copied to SEALS_HOME. Press y to start: y
... maybe some debugging/progress output generated by your system
>>> Result stored to URL: file:/C:/.../Temp/align438520766.rdf
>>> Matching finished. Press y to clear SEALS_HOME: y
>>> Cleaning up environment ...
```

Note that two times you are requested to type `y` to make the process continue. This allows you to inspect the content of SEALS_HOME as well as to check the automatically generated alignment. Once the process has finished, the contents of SEALS_HOME will be deleted again.

If you have no internet connection (or want to test with different ontologies), you can use the `-o` option and specify a second and third parameter, which have to be URLs pointing to the (locally stored) ontologies like `file:///C:/...`. Keep in mind that a valid URL is required.

5.3 Evaluating the tool

Once you have tested successfully the `-t` or the `-o` option, it is time to test if your system manages to run a complete evaluation. For doing so you can use the `-x` option and chose one of the available suites, which you can find at `http://oaei.ontologymatching.org/2011.5/suites.html`.

For example, you can run all test cases of the Conference test suite which was used for the OAEI 2010. The `repURI` is `http://seals.sti2.at/tdrs-web/`, the `suiteId` is `Conference+Testsuite` and the `versionId` is `2010`. So, you have to type this:

```
A:\seals\sealshome>java -jar ... ... -x  http://seals.sti2.at/tdrs-web/
Conference+Testsuite 2010 A:\results\
```

This runs all of the test cases subsequently and displays some progress information on the screen.

Note that a full evaluation can be run successfully while there might still be some problems related to the generated alignments. For that reason you should inspect the output files, which you can found in the specified output directory, carefully. The output directory contains all generated alignments and also a summary file `results.txt`, which might look like this:

```
101   1.0   1.0   file:/C:/.../align3100047406330015693.rdf
103   1.0   1.0   file:/C:/.../align6763189370432843972.rdf
104   1.0   1.0   file:/C:/.../align6804694726976981529.rdf
201   0.773 0.949 file:/C:/.../align6060859294027567944.rdf
201-2 0.969 1.0   file:/C:/.../align1927738950787331860.rdf
```

The file shows for each test case precision and recall of the alignment generated by your system as well as the path to the generated alignment.

If precision and recall values are missing, the generated alignment could not be parsed. In such a case there is an entry in an additional column that informs about possible problems. This will help you to get rid of the underlying problem. Incorrect or missing header information in the alignment is often the reason for this problem (e.g., filling the uri-tags with something that is not an URI).

Keep also track of lines that contain unexpected results for precision and recall (e.g. 0 lines) at places where you would not expect them. Typical problems may be related to one of the following issues:

- You system might have reversed the order of the input ontologies such that the matched concepts, properties, or instances appear in a reversed order.
- Your system might not be able to cope correctly with encoding issues. This has been the case for some systems with respect to some test cases in the Benchmark suite.
- Your system might not use namespaces that prefix the local names of concepts and properties in the appropriate way. Take care that something like `http://xyz#Person` appears in the alignment, and not just `Person` or `file://C:xyz.owl#Person`.

A A full example: demomatcher

The tool that we analyze as an example is called *DemoMatcher*[4]. Its functionality is encoded in a jar-file `demomatcher.jar`. It requires the additional libraries `simmetrics.jar` and `owlapi.jar` to be available on the classpath. Further, it uses a simple configuration file to specify a threshold. In some of the following sections of this Appendix we show how to package *DemoMatcher* and how to write the so called *ToolBridge*, which will be the interface through which the SEALS platform accesses its functionality. All the required files (jars, wrapped tool file) can be downloaded in a `windows` and `linux-variant`.

A.1 Structure of the tool package

The main jar-file of *DemoMatcher* and the required libraries have to be packaged in a specific folder structure. Our example is based on the Windows variant. Please download `demomatcher-package.zip` and unzip it on your machine. The resulting directory structure will look like this:

```
bin/
   lib/
      demomatcher.jar
      owlapi.jar.jar
      simmetrics.jar
   demomatcher-bridge.jar
   deploy.bat (or *.sh on linux)
   start.bat
   stop.bat
   undeploy.bat
conf/
   configuration/
      threshold.txt
lib/
   (empty)
descriptor.xml
```

Besides the libraries required by the matcher, you find some additional files and folders. Some of these folders, such as `lib`, are not required by *DemoMatcher*. However, it is obligatory that these folders exist. Some matchers may need additional resources to the required Jar-files. These may be stored in the `conf` directory which is accessible at runtime. Our simple matcher has only one parameter, namely its threshold, which can be defined in a configuration file. It is stored in the file `threshold.txt`.

The four files `deploy.bat`, `start.bat`, `stop.bat`, and `undeploy.bat` (*.sh under Unix) have to be part of your tool package. The four files provided with

[4]The steps we present here are the same for using the example presented in §2

our example – which currently do not perform any action – should also be used for any other tool.

The two additional files that are relevant and specific for our matcher are the files `demomatcher-bridge.jar` and `descriptor.xml`. We will explain them in the subsequent sections.

A.2 Content of descriptor file

It is very easy to modify the descriptor file according to our needs. We explain it step by step.

```
<?xml version="1.0" encoding="UTF-8"?>
<ns:package
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:ns="http://www.seals-project.eu/resources/res/tools/bundle/v1"
   id="DemoMatcher"
   version="1.0">
   <ns:description>DemoMatcher is a matching tool developed
     for testpurpose.</ns:description>
   <ns:endorsement>
      <ns:copyright>Copyright information</ns:copyright>
      <ns:license>Specification of license</ns:license>
   </ns:endorsement>
...
```

In the first part you have to define the *id* of your matcher – chose its name or an abbreviation in case the name is too verbose – its *version* and a short *description*. You can also specify copyright and license information. While the first part contains some meta-information, the final part is about the wrapping of the tool itself. It describes where required libraries can be found inside the package.

```
<ns:wrapper>
   <ns:management>
      ...
      ...
      ...
   </ns:management>
   <ns:bridge>
      <ns:class>de.unima.ki.demomatcher.seals.MatcherBridge</ns:class>
      <ns:jar>demomatcher-bridge.jar</ns:jar>
      <ns:dependencies>
         <ns:lib>lib/demomatcher.jar</ns:lib>
         <ns:lib>lib/owlapi.jar</ns:lib>
         <ns:lib>lib/simmetrics.jar</ns:lib>
      </ns:dependencies>
   </ns:bridge>
```

```
</ns:wrapper>
```

As you can see, both the libraries and the main jar of the matcher have to be specified as dependencies. In addition, a class has to be implemented that acts as bridge between the functionality implemented in *DemoMatcher* and the interface required by the SEALS platform. The naming of this class and its package does not need to follow any pattern. In our example, it is the class `de.unima.ki.demomatcher.seals.MatcherBridge` that can be found in `demomatcher-bridge.jar`. Note also that the name of this jar can be chosen arbitrarily as long as it is correctly referenced in the file `descriptor.xml`. The file `demomatcher-bridge.jar` contains only this class and nothing else. In the following section we explain how to develop this single class and the file `demomatcher-bridge.jar`.

A.3 Development of the tool bridge

Different matching systems come with different interfaces. Some systems might use the following signature to do their job of matching two ontologies.

```
// matches to ontologies referred to via their filepath and
// return the File that has been generated as a result.
File match(String filepathSourceOnt, String filepathTargetOnt)
```

The interface we proposed for using the SEALS online evaluation service was based on the following signature.

```
// matches to ontologies referred to via their URI and
// returns the alignment generated directly as a string.
String match(URI filepathSourceOnt, URI filepathTargetOnt)
```

The interface that has to be implemented within this tutorial for running a tool under the SEALS platform, has the following signature.

```
// matches to ontologies referred to via their URL and
// returns the URL of the file generated locally
URL align(URL sourceOnt, URL targetOnt)
```

Note that the return value of this methods is a URL. You have to create a temporary file where you store the alignment generated by your system and return the URL of this alignment.

In our *DemoMatcher* there exists a method with the signature that fitted with the SEALS online evaluation service. Please take now a look at the code depicted in §A.4 to see how this method can be called from within `URL align(URL source, URL target)`, the method that has to be implemented. For our example this method is quite simple. It converts the input URLs into URIs, calls the `align`-method of *DemoMatcher*, stores the result in a temporary file, and returns the URL of this file. During this process several Exceptions might be caught. Whenever this happens a `ToolBridgeException` is thrown. This means that the wrapping has not been done correctly, by opposition to an

error of the matcher. For example, it is not possible to create a temporary file. It is also possible to throw a `ToolException` to indicate that the matcher itself had some problems. Both types of exceptions will help to debug the tool within the SEALS platform whenever problems occur.

In addition to the method that wraps the main functionality of the matcher, there are three more methods. One of them is an `align`-method that has three input parameters to point to an additional input alignment. If a matcher supports exploiting an additional input alignment (e.g., subtask #4 of the anatomy track), this method should be implemented. Otherwise follow the example in the appendix. Furthermore, there are two additional methods that should not be modified.

A.4 Example for a bridge

The following code is also available in the file `MatcherBridge.java`, inside the `demomatcher-package.zip` file.

```
package de.unima.ki.demomatcher.seals;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.net.URISyntaxException;
import java.net.URL;

import de.uma.alignment.matcher.demo.DemoMatcher;

import eu.sealsproject.platform.res.domain.omt.IOntologyMatchingToolBridge;
import eu.sealsproject.platform.res.tool.api.ToolBridgeException;
import eu.sealsproject.platform.res.tool.api.ToolException;
import eu.sealsproject.platform.res.tool.api.ToolType;
import eu.sealsproject.platform.res.tool.impl.AbstractPlugin;


public class MatcherBridge extends AbstractPlugin implements
                                        IOntologyMatchingToolBridge {

   /**
   * Aligns to ontologies specified via their URL and retunrs the
   * URL of the resulting alignment, which should be stored locally.
   *
   */
   public URL align(URL source, URL target)
                                  throws ToolBridgeException, ToolException {
     DemoMatcher demoMatcher;
     try {
        demoMatcher = new DemoMatcher();
        try {
           String alignmentString = demoMatcher.align(source.toURI(), target.toURI());
           try {
              File alignmentFile = File.createTempFile("alignment", ".rdf");
              FileWriter fw = new FileWriter(alignmentFile);
              fw.write(alignmentString);
              fw.flush();
```

```
            fw.close();
            return alignmentFile.toURI().toURL();
         }
         catch (IOException e) {
            throw new ToolBridgeException("cannot create file for results", e);
         }
      }
      catch (URISyntaxException e1) {
         throw new ToolBridgeException("cannot convert the input param to URI");
      }
    }
    catch (NumberFormatException e2) {
       throw new ToolBridgeException("cannot read from configuration file", e2);
    }
    catch (IOException e3) {
       throw new ToolBridgeException("cannot access configuration file", e3);
    }
  }


  /**
  * This functionality is not supported by the tool. In case
  * it is invoked a ToolException is thrown.
  */
  public URL align(URL source, URL target, URL inputAlignment)
                             throws ToolBridgeException, ToolException {
     throw new ToolException("functionaility of called method is not supported");
  }



  /**
  * In our case the DemoMatcher can be executed on the fly. In case
  * prerequesites are required it can be checked here.
  */
  public boolean canExecute() {
     return true;
  }

  /**
  * The DemoMatcher is an ontology matching tool. SEALS supports the
  * evaluation of different tool types like e.g., reasoner and storage systems.
  */
  public ToolType getType() {
     return ToolType.OntologyMatchingTool;
  }

}
```