Prerequisites:

1. Install Apache Tomcat and set the CATALINA\_HOME variable in your .bash\_profile.
2. Install Apache Axis2.

Steps for OWL-S Match Making:

1. Create a new Dynamic Web Project in Eclipse for the ‘services’.
2. Set Target runtime environment to Apache Tomcat v9.0
3. Modify the configuration to include Axis2 Web Services in Project Facets. This requires changing the Dynamic Web Module to version 2.5. Also, change Java from 10 to 1.8.
4. It might say that the Axis2 Runtime is not set.
5. Go to Eclipse, Preferences, Web Services, Axis2 Preferences, Set the runtime location to the directory where you have installed Axis2.
6. Right click on the project ‘services’, select Properties, ensure the following:
   1. Targeted Runtimes is set to the Apache Tomcat v9.0
   2. Project Facets: Axis 2 Web Services is checked.
7. There may be JSP problems such as this:
   1. Download and copy jstl-1.2.jar to WebContent/WEB-INF/lib and refresh project.
8. Ensure that the web.xml’s web-app, welcome-file-list, welcome-file points to axis2-web.
9. Do the following in WebContent/WEB-INF/conf/axis2.xml file:
   1. Set disableSOAP12 parameter to true.
   2. Set useGeneratedWSDLinJAXWS parameter to true.
   3. Set clustering tag to true.
   4. Modify transportReceivers under the “Transport Ins” block.
10. Either create new services in the src, or copy the ones from the composition/src dir.
11. The third-party libraries would be missing. For simplicity, we can copy the jars to the WebContent/WEB-INF/lib directory. Download from Maven and copy the listed jars. Then refresh the project, and you may see some warnings but no errors.

javax.ws.rs-api-2.1.jar

jaxp-ri-1.4.5.jar

xmlschema-core-2.2.3.jar

jstl-1.2.jar

stanford-ner-resources.jar

stanford-ner-3.9.1.jar

muf-3.0.53-all.jar

reflections-0.9.11.jar

guava-20.0.jar

javassist-3.21.0-GA.jar

easy-rules-mvel-3.2.0.jar

easy-rules-core-3.2.0.jar

slf4j-nop-1.8.0-beta2.jar

jollyday-0.4.9.jar

joda-time-2.4.jar

xalan-2.7.2.jar

serializer-2.7.2.jar

1. Create a WSDL file for a service: say, YahooNewsService: File, New, Other, Web Services, Web Service, type: Bottom up Java bean Web Service, select the service impl class. Under configuration, change web service runtime to Axis2 and Tomcat v9.0 server. Click next, Generate a default services.xml file, click next, Start the server, you may see that several ports are already in use. This happens because you may have a Tomcat instance running with the same ports, so, we could modify the ports for this instance of the server. Go to Servers, Tomcat v9.0, server.xml, do the following:
   1. locate the Connector tag, modify the port from 8080 to something else, say 9090, and redirectPort from 8443 to 9443.
   2. locate the Connector tag for AJP 1.3 Connector, modify the port from 8009 to something else, say 9009, and redirectPort from 8443 to 9443.
2. Refresh the server, and create the WSDL again as mentioned in Step 12. The server starts this time, then, click Finish. The web service is now created. You’d now see the newly created service in WebContent/WEB-INF/services dir.
3. Deploy the service to the Tomcat instance on your machine. That is, copy the WebContent/WEB-INF/services/YahooNewsService dir to Axis2’s services directory. That is: axis2-1.7.8/repository/services dir.
4. Start the Axis2 server: axis2-1.7.8/bin/axis2server.sh and you’d notice that the service is deployed. Since, we’re installing it to the Axis2’s home dir, we’d want to check it on the 8080 port of Tomcat. Go to: <http://localhost:8080/axis2/services/> and you’d see your service deployed. When you click on the service link: say, YahooNewsService, you’d see that it opens the WSDL file for the service.
5. Create a wsdl dir in the project’s WebContent dir, and save the above WSDL file as YahooNewsService.wsdl, modify the Format to “All Files”.
6. Clone the OWL-S Composer: <https://github.com/FORMAS/OWL-S-Composer> on the system, then, in the api dir, run the wsdl2owls.sh file.
7. Enter the URL for the above saved YahooNewsService.wsdl file, you’d see that there may be an exception on the terminal where you’ve run the wsdl2owls script, we can ignore it as long as we see the operations of the service listed on the left pane.
8. Click on the operation for which we’d like to generate an OWL-S file. It then populates the Service description on the right pane. Under the Namespaces, select OWL and click add, then enter, a namespace URL, say: <http://services.composition.inmind.cmu.edu>, and click ok. You’d notice that on the script console, the service namespace is printed.
9. Then, to generate the OWL-S file, click on Generate OWL-S button, and store it on the user’s home dir to start with, it will then generate the OWL-S file and print it to the console, and write it to the file system, and you’d get a confirmation dialog.
10. We can then copy the OWL-S file to an ‘owls’ dir in WebContent dir by first creating a dir for the service (because same grounded services will generate a same name OWL-S file), and then, within that dir.
11. If your service has more than one methods for which you’d like to generate an OWL-S file, then, repeat the steps 18-21 above for a new service.
12. Then, clone the owl-s repository that has the MatchMaker program, open it in Eclipse. You’d see a red exclamation mark on the project. Right-click and go to ‘Build Path’, select ‘Projects’, and you’d notice that weka is missing. Remove weka’s entry from the classpath. However, download the weka.jar and place it in libs. Refresh the project.
13. You may also need some Eclipse plugins, which you can download from this repo’s “owlspace/.metadata” dir.
14. Go to the MatchMaker.java in src/examples, and modify the run() method by providing as many OWL-S files against which you’d like to compare. And, run the main() method.