HAL
open science

Iterative implementation of services for the automatic evaluation of matching tools

Cassia Trojahn dos Santos, Christian Meilicke, Jérôme Euzenat

To cite this version:

Cassia Trojahn dos Santos, Christian Meilicke, Jérôme Euzenat. Iterative implementation of services for the automatic evaluation of matching tools. [Contract] 2011, pp.21. hal-00793436

HAL Id: hal-00793436

https://inria.hal.science/hal-00793436v1

Submitted on 22 Feb 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

SEALS

Semantic Evaluation at Large Scale

FP7 – 238975

D12.5 Iterative implementation of services for the automatic evaluation of matching tools v1.0-beta

Coordinator: Cássia Trojahn dos Santos
With contributions from: Christian Meilicke, Jérôme Euzenat
Quality Controller: Heiner Stuckenschmidt
Quality Assurance Coordinator: Raúl García Castro

Document Identifier:	SEALS/2010/D12.5/V1.0-beta
Class Deliverable:	SEALS EU-IST-2009-238975
Version:	1.0-beta
Date:	February 25, 2011
State:	final
Distribution:	public

Executive Summary

The goal of this deliverable is to report the current status of the service implementation for the automatic evaluation of matching tools.

For the first evaluation campaign, as reported in [6], we have developed an evaluation service based on the use of a web service interface wrapping the functionality of a tool to be evaluated. This interface allowed to evaluate the tool without the need for a runtime environment. Contrary to this, the tools were executed on the machine of the tool developer. We have followed this approach mainly because some SEALS components were still under development in that time.

Since the end of the first campaign, several SEALS components have been finished. This allows to progressively migrate our previous implementation to the SEALS platform. Firstly, we have concentrated our efforts in using the Test Data and Results Repository by uploading and using test and results data (Chapter 2). Next, we have wrapped some tools in order to fulfill the requirements for deploying a tool into the SEALS platform (Chapter 3). Then, we have extended our BPEL workflow in order to use the new web service interfaces for accessing the repositories and invoking the tools (Chapter 4). In order to run integration tests, an express version of the SEALS Runtime Evaluation Service has been delivered. This express version allows to deploy locally some tools and to run a complete evaluation experiment (Chapter 5).

For the next development iteration (Chapter 6), we plan to improve the specification of our current raw results metadata. Further, we will modify the interpreter for generating interpretations to be integrated as custom services in the platform. Finally, we plan to extend the BPEL workflow in order to take into account fault handling.

Document Information

IST Project Number	FP7 – 238975	**Acronym**	SEALS
Full Title	Semantic Evaluation at Large Scale		
Project URL	`http://www.seals-project.eu/`		
Document URL			
EU Project Officer	Carmela Asero		

Deliverable	**Number**	12.5	**Title**	Iterative implementation of services for the automatic evaluation of matching tools v1.0-beta
Work Package	**Number**	12	**Title**	Matching Tools

Date of Delivery	**Contractual**	M21	**Actual**	28-02-2011
Status	1.0-beta		final ☒	
Nature	prototype ☒ report ☐ dissemination ☐			
Dissemination level	public ☒ consortium ☐			

Authors (Partner)	Jérôme Euzenat (INRIA), Cássia Trojahn dos Santos (INRIA), Heiner Stuckenschmidt (University Mannheim), Christian Meilicke (University Mannheim)			
Resp. Author	**Name**	Cássia Trojahn dos Santos	**E-mail**	cassia.trojahn@inrialpes.fr
	Partner	INRIA	**Phone**	+33 (476) 615 476

Abstract (for dissemination)	The implementation of the automatic services for evaluating matching tools follows an iterative model. The aim is to provide a way for continuously analysing and improving these services. In this deliverable, we report the first iteration of this process, i.e., current implementation status of the services. In this first iteration, we have extended our previous implementation in order to migrate our own services to the SEALS components, which have been finished since the end of the first evaluation campaign.
Keywords	ontology matching, ontology alignment, evaluation, benchmarks, efficiency measure

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
14/02/2011	1	Cássia Trojahn dos Santos	Set up document structure.
15/02/2011	2	Cássia Trojahn dos Santos	Filled the chapters.
16/02/2011	3	Christian Meilicke	Added Section 2.3 and revised document.
21/02/2011	4	Christian Meilicke	Addressed point 1 and 2 of reviewer.

Project Consortium Information

Participant's name	Partner	Contact
Universidad Politécnica de Madrid		Asunción Gómez-Pérez Email: asun@fi.upm.es
University of Sheffield		Fabio Ciravegna Email: fabio@dcs.shef.ac.uk
Forschungszentrum Informatik		Rudi Studer Email: studer@aifb.uni-karlsruhe.de
University of Innsbruck		Barry Norton Email: barry.norton@sti2.at
Institut National de Recherche en Informatique et en Automatique		Jérôme Euzenat Email: Jerome.Euzenat@inrialpes.fr
University of Mannheim		Heiner Stuckenschmidt Email: heiner@informatik.uni-mannheim.de
University of Zurich		Abraham Bernstein Email: bernstein@ifi.uzh.ch
Open University		John Domingue Email: j.b.domingue@open.ac.uk
Semantic Technology Institute International		Alexander Wahler Email: alexander.wahler@sti2.org
University of Oxford		Ian Horrocks Email: ian.horrocks@comlab.oxford.ac.uk

Table of Contents

List of figures 6

1 Introduction 7

2 SEALS Repositories 8
2.1 Test Data Repository 8
2.2 Results Repository 9
2.3 Tools and Evaluation Description Repositories 10

3 Tool Wrappers 12

4 Evaluation Workflow 15

5 Integration test 17

6 Conclusion and Future Work 18

References 18

A SOAP-UI Test Suite 20

List of Figures

4.1 BPEL workflow. 16
4.2 Iteration over a test suite. 16

1. Introduction

The main aim of SEALS is to provide a platform for supporting automatic evaluations. It plans to go beyond current evaluation software by providing a continuous evaluation platform that allows people to test their tools at any time and publish the results that they want to be recorded.

For the first evaluation campaign, due the complexity of the SEALS platform, required SEALS components (e.g., Runtime Evaluation Service) were under development. For that reason, we have implemented an evaluation service that bypasses one of the main sources of complexity, i.e., the deployment of a tool. Our approach was based on the idea of executing the tool on the machine of the tool vendor itself. In this scenario, the functionality of the tool is made available via a web service, which is accessed by the evaluation service during the execution of an evaluation workflow. We have developed a set of services for running a complete evaluation experiment, i.e., for iterating the test suites, invoking a tool, interpreting raw results and storing results into a relational database.

For the second stage of the project, where the focus is the second evaluation campaign, we aim at extending our previous implementation to integrate the SEALS components that have been released since the first evaluation campaign. This deliverable reports the first iteration of this migration process. The first step was to replace our own test suite web service iterator by the service accessing the SEALS Test Data Repository. It involved to require the development of a finegrained access to the repository. Once this functionality had been implemented, we specified the test suite metadata for each of our test suites and uploaded them into the repository. This repository can now also be accessed via a web service interface implementation provided by the platform. We have also worked with the interface and its implementation for uploading raw results into the SEALS Results Repository.

Regarding tools, in order to deploy them into the SEALS platform, developers must extend an interface and wrap its implementation in a ZIP file that includes a bridge as well all required libraries for running the tool. We have tested the required structure of the ZIP file by using some of the tools that have participated in the first evaluation campaign. Furthermore, a web service interface has been defined for invoking the deployed tools from evaluation descriptions. We have also tested this interface.

For running a complete evaluation experiment, an express version of the SEALS Runtime Evaluation Service has been delivered, which allows for invoking the web service implementations for accessing the SEALS Test Data and Results repositories and invoking the deployed tools. This whole environment must be deployed locally in a test machine and allows researchers to run integration tests.

The remainder of this deliverable is structured as follows. In Chapter 2, we discuss the usage of the SEALS repositories. Chapter 3 presents the required packaging format for deploying a tool into the platform. In Chapter 4, we present our extended evaluation workflow. Some notes about running an integration test are presented in Chapter 5. Finally, Chapter 6 concludes the deliverable and presents the perspectives for the next development iteration.

2. SEALS Repositories

2.1 Test Data Repository

In order to upload a test suite into the SEALS Test Data Repository, one must encapsulate the suite in a ZIP file and specify a test suite metadata, which describes a `Suite` and its `SuiteItem`(s) and `DataItem`(s). The ZIP file can be organized in an arbitrary way and contains the metadata encoded in a RDF file at the top level of the ZIP. For Work Package 12, a test suite is composed by a set of pairs of ontologies and their reference alignments:

```
Metadata.rdf
ont/
     101.rdf
     102.rdf
     103.rdf
ref/
     101-102.rdf
     101-103.rdf
```

For our evaluation workflows it was important to retrieve the `DataItem`s in a finegrained way. We participated in the specification of a `Suite` metadata definition that supports this finegrained access. At the same time we took into account that the approach has to be generic enough to be usable by the other research workpackages. According to this approach, a test suite is then described using the following metadata:

```
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:seals="http://www.seals-project.eu/ontologies/
        SEALSMetadata.owl#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:dc="http://purl.org/dc/terms/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<seals:Suite rdf:about="http://www.seals-project.eu/alignment/
   benchmark#BenchmarkSuite">
       <seals:hasSuiteItem rdf:resource="http://www.seals-project.eu/
          alignment/benchmark#101" />
       <seals:hasSuiteItem rdf:resource="http://www.seals-project.eu/
          alignment/benchmark#102" />
       <seals:hasSuiteItem rdf:resource="http://www.seals-project.eu/
          alignment/benchmark#103" />
</seals:Suite>

<seals:SuiteItem rdf:about="http://www.seals-project.eu/alignment/
   benchmark#102">
    <seals:hasDataItem rdf:resource="http://www.seals-project.eu/
       alignment/benchmark#101-source" />
```

```
    <seals:hasDataItem rdf:resource="http://www.seals-project.eu/
       alignment/benchmark#102-target" />
    <seals:hasDataItem rdf:resource="http://www.seals-project.eu/
       alignment/benchmark#ref101to102-ref" />
    <dc:identifier>102</dc:identifier>
</seals:SuiteItem>

<seals:DataItem rdf:about="http://www.seals-project.eu/alignment/
   benchmark#101-source">
    <seals:isLocatedAt>./ont/101.rdf</seals:isLocatedAt>
    <seals:hasComponentType>source</seals:hasComponentType>
    <dc:identifier>101-source</dc:identifier>
</seals:DataItem>

<seals:DataItem rdf:about="http://www.seals-project.eu/alignment/
   benchmark#102-target">
    <seals:isLocatedAt>./ont/102.rdf</seals:isLocatedAt>
    <seals:hasComponentType>target</seals:hasComponentType>
    <dc:identifier>102-target</dc:identifier>
</seals:DataItem>

<seals:DataItem rdf:about="http://www.seals-project.eu/alignment/
   benchmark#ref101to102-ref">
    <seals:isLocatedAt>./ref/101-102.rdf</seals:isLocatedAt>
    <seals:hasComponentType>reference</seals:hasComponentType>
    <dc:identifier>ref101to102-ref</dc:identifier>
</seals:DataItem>

</rdf:RDF>
```

We have uploaded three test suites into the SEALS repository, which correspond to the data sets used in the first evaluation campaign: Anatomy[1], Benchmark[2] and Conference[3].

2.2 Results Repository

The results repository stores raw results (i.e., alignments in Work Package 12) and interpretations of these results (i.e., precision and recall). In this deliverable, we focus on uploading raw results because custom interpreters, which are generating the interpretations, are not yet integrated into the express runtime version of the SEALS platform. For uploading raw results, metadata must be specified that describes the stored results.

In a similar way of what is done for a test suite, raw results are encapsulated in a ZIP file, together with the *suite metadata* file.

[1] http://seals.sti2.at/tdrs-web/testdata/persistent/Anatomy+Testsuite/2010/suite/
[2] http://seals.sti2.at/tdrs-web/testdata/persistent/Benchmark+Testsuite/2010/suite
[3] http://seals.sti2.at/tdrs-web/testdata/persistent/Conference+Testsuite/2010/suite/

```
Metadata.rdf
results/
   align102.rdf
   align103.rdf
   align104.rdf
```

In the current implementation, we have used the *results composer service* provided by the platform and a minimal version of the *suite metadata*. This service bundles the raw result of each test iteration (i.e., each resulting alignment) and encapsulates all the results in the ZIP file.

A minimal suite metadata[4] has been specified for describing the raw results:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:seals="http://www.seals-project.eu/ontologies/SEALSMetadata.
     owl#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dcterms="http://purl.org/dc/terms/">
<seals:RawResult rdf:about="http://www.seals-project.eu/metadata/
   anyURI /">
  <seals:hasName rdf:datatype="http:/www.w3.org/XMLSchema#string">
     Benchmark-Aroma-Raw-Results</seals:hasName>
</seals:RawResult>
</rdf:RDF>
```

For the next iteration, we have to extend the required metadata in order to fulfill the specifications in [5]. Furthermore, as we have reported below, at the time of writing this deliverable, the details about how to include interpreters in the current version of the Runtime Evaluation Service have not been provided. As discussed in the last Seals plenary meeting[5], each WP will be responsible for defining their own interpreters (as WSDL interfaces) and integrating them into the platform. We have already defined an interface for measuring precision and recall and are waiting for guidelines for integrating it into the platform in a way that it will be available for access from the BPEL workflow (as a partner link).

2.3 Tools and Evaluation Description Repositories

Currently we do not use the other two repositories, because they are only partially integrated into the platform. In particular, the following reasons prevent us from using the services of these repositories.

- The SEALS Tools Repository is to our knowledge currently not connected to the upload functionality of the web portal. Even more important is the problem

[4]One of our uploaded raw results can be found at `http://seals.sti2.at/rrs-web/results/Benchmark-Aroma+raw+results/`

[5]Sheffield, 31/01/2011–03/02/2011

that there is no connection between storing a tool in the repository and executing it on the platform. Currently tools have to be deployed manually in order to evaluate them. There is no automatism that retrieves a tool from the repository and deploys and executes it automatically.

- The situation of the SEALS Evaluation Description Repository is similar. Currently only the core element of the evaluation description – the BPEL itself – is required to run an evaluation. This BPEL workflow has to be deployed and executed manually. There exists no automatism that retrieves an evaluation description and executes it automatically.

To benefit from the functionality of the SEALS Tools Repository, it is first of all required to continue with the development of the automatic tool deployment. Currently it is not supported that a tool is deployed automatically. Thus, it is unclear why tool developers should upload a tool to the platform that cannot be deployed and executed. For that reason it is highly problematic that this functionality is currently set to the status unplanned according to the relevant Redmine issues.[6]

The use of the SEALS Evaluation Description Repository depends on the availability of the functionality to execute a SEDL document (or at least to execute the BPEL encapsulated in the SEDL document). To enable this, a tight collaboration between the developer of the SEDL specification and the people responsible for providing the execution environment of the BPEL workflow is required.

[6]See `http://www.development.seals-project.eu/redmine/issues/344`.

3. Tool Wrappers

In order to deploy a tool into the SEALS platform, developers must implement a tool interface (i.e., *IOntologyMatchingToolBridge* in the case of Work Package 12). This interface specifies the methods that will be called for invoking the tool from the platform. The interface implementation must be encapsulated together with all required libraries and additional files in a ZIP file, following the structure presented in the following example.

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

The folder `bin/lib/` contains all required libraries for executing the tool. In the specific example, `demomatcher-bridge.jar` contains the implementation of the tool interface. Besides the required libraries, some additional files and few empty folders can be found. The empty folders can contain additional resources that are not required with respect to our simple example. The four files `deploy.bat`, `start.bat`, `stop.bat`, and `undeploy.bat` have to be part of the package. They are currently not used, but will be used in the next version of the platform to automatically prepare the environment required by the tools.

One important component in this package is the `descriptor.xml` file, which provides the description of the tool, the list of required libraries and the specification of the class implementing the tool interface:

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

```
    <ns:management>
      <ns:deploy>
        <ns:executable xsi:type="ns:ShellScript">
          <ns:script>deploy.bat</ns:script>
          <ns:error-log>deploy-error.log</ns:error-log>
        </ns:executable>
      </ns:deploy>
      <ns:start>
        <ns:executable xsi:type="ns:ShellScript">
          <ns:script>start.bat</ns:script>
          <ns:error-log>start-error.log</ns:error-log>
        </ns:executable>
      </ns:start>
      <ns:stop>
        <ns:executable xsi:type="ns:ShellScript">
          <ns:script>stop.bat</ns:script>
          <ns:error-log>stop-error.log</ns:error-log>
        </ns:executable>
      </ns:stop>
      <ns:undeploy>
        <ns:executable xsi:type="ns:ShellScript">
          <ns:script>undeploy.bat</ns:script>
          <ns:error-log>undeploy-error.log</ns:error-log>
        </ns:executable>
      </ns:undeploy>
    </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>
</ns:package>
```

We have contacted several tool developers to check if the wrapping works with real tools. For some of these tools we used the version that is available via the official webpage of the tool, while for other we contacted the developer to retrieve access to the tool. The tools we have tried to wrap so far are the tools AnchorFlood [7], Aroma [2], Eff2Match [1], Falcon-AO [4], Lily [8], and Taxomap [3]. While doing this we noticed that many tools specify internally a relative path from the current working directory to a required configuration file (or a similar kind of resource). We have to clarify if this results in a problem or requires some additional agreements regarding tool deployment and execution. In parallel we are writing a tutorial that allows the tool developer to wrap the tool on his own.

Our first experiences indicate that some problems relevant in the deployment process can only be solved by the use of manual work-arounds. Even more, it is not

clear whether such an approach will always be applicable given a high number of participants. This is also related to the fact that the automized deployment is not yet available. In particular, is is currently not the case that the start, stop, deploy, and undeploy-scripts are executed by the platform, nor it is planned to make this functionality available in near future.[1]

[1]See for example `http://www.development.seals-project.eu/redmine/issues/344`.

4. Evaluation Workflow

We have modified the BPEL workflow described in [6] in order to replace our previous partner link definitions (external web services) by the new SEALS services:

- test repository service: provides access to the Test Data Repository, in order to retrieve a test suite and its test items to iterate over the test suite;
- results repository service: provides access to the Results Repository, in order to add a raw result and its interpretations;
- results composer service: a utility service that provides operations for bundling the raw results generated during the evaluation execution in a ZIP file;
- tool service: provides access to a deployed tool;
- evaluation service: describes the common service interface for the workflow entry point and the callback service interface (asynchronous process).

In the current implementation, the workflow iterates over a test suite and invokes a tool that generates an alignment between the source and target ontologies for each test item of a test suite. As custom interpreters are not yet integrated in the express version of the Runtime Evaluation Service, we can not generate interpretations for the raw results.

Basically, the BPEL workflow (Figure 4.1) interacts with the services through partner links, which encapsulate the corresponding web service interfaces. The "evaluation" process starts by receiving from the client process two input parameters: the test suite name and version. Due to an internal platform requirement, some information (e.g., ExecutionRequestId) is passed to the BPEL process in the header part of each message exchanged with partner links. One assigns activity (`setHeaderForInvocations`) is used to initialize the header part of all messages.

As we reported in §2.2, for uploading results into the Results Repository one must specify their metadata. In our workflow, we firstly initialize the metadata for the results composer (activities `assignMetadata` and `addMetadaResults`). This metadata will be encapsulated later, by the results composer service, into the ZIP file containing all raw results. Next, the parameters for loading a test suite are initialized (`setParamsToLoadTestSuite`) and the suite is loaded (`loadTestSuite`). Figure 4.2 shows the iteration over a test suite. For each test case in the test suite, the ontology source (`getOntoSource`) and the ontology target (`getOntoTarget`) are retrieved from the Test Data Repository. These two elements are then used as input parameter (`setParamsAlign`) for invoking the tool (`align`). The output of this invoke is the URL of the file containing the generated alignment. This URL is then sent to the results composer (`addDataItem`), which uses this information later for generating the ZIP file containing all generated raw results.

After iterating over all test cases, the ZIP file, containing all raw results (alignment files) and the corresponding metadata, is generated by the results composer service (`createBundle`). Next, this bundle is submitted to the Results Repository by invoking the `addRawResults` method of the results repository service.

The workflow is asynchronous, i.e., the client does not wait for the complete execution of the evaluation. For that reason a notification must be sent when the process finishes (`callbackClient`).

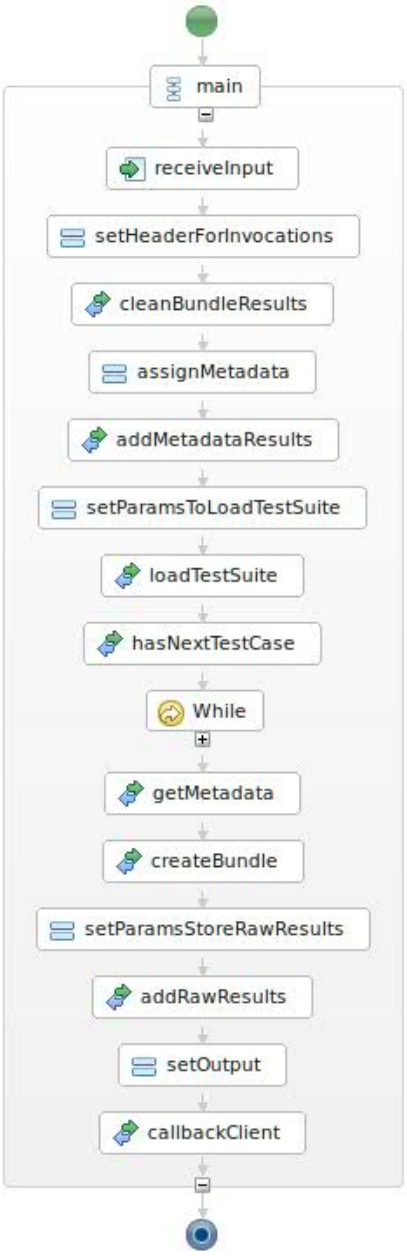

Figure 4.1: BPEL workflow.

Figure 4.2: Iteration over a test suite.

The full BPEL code can be found at `https://svn.seals-project.eu/seals/tags/deliverables/D12.5/M21/software/evals`.

5. Integration test

For running an integration test, i.e., executing a BPEL workflow that accesses repository services and deployed tools, an express version of the SEALS Runtime Evaluation Service has been delivered for local deployment. This version is composed of two main sub-modules:

- `res-core-sa`: takes care of the complete orchestration of the activities involved in the evaluation of a tool. From this module, an evaluation description (BPEL workflow) is executed;
- `res-worker-sa`: is responsible for the launching of tools. From this sub-module, tools are deployed, started and executed.

These two modules are deployed locally in different instances of a servicemix container. The BPEL workflow must be deployed in the same instance as `res-core-sa`, while the path for the tool package (ZIP file presented in §3) must be configured in the same instance as `res-worker-sa`. Both of these modules communicate with each other and share service endpoints. This allows to execute an integrated test.

We have successfully deployed `res-core-sa`, `res-worker-sa`, the BPEL workflow and one tool. Based on this we could successfully execute an integration test. In our integration test we have used the tool Anchor-Flood [7], which is – wrapped in the appropriate way – available at `https://svn.seals-project.eu/seals-dev/omt/tools/aflood`. For invoking the deployed BPEL we have used an `EvaluationBinding` test suite, created using the SOAP-UI tool. In Appendix A, the code for the test suite can be found.

6. Conclusion and Future Work

This deliverable has reported the current implementation status of Work Package 12. We have migrated our previous implementation in order to take into account the new delivered SEALS components. We have successfully wrapped some tools and started to write a tutorial to help tool developers. We successfully achieved to run an integration test which involves the execution of a BPEL workflow that accesses the SEALS repositories and invokes a deployed tool.

For the next iteration, we plan to extend the definitions of metadata for raw results, in order to fulfill the requirements specified in [5]. We already implemented the most important interpreters, however, we need to integrate them as custom interpreter services into the express version of the SEALS Runtime Evaluation Service in the next step. We plan as well to extend the BPEL workflow in order to take into account fault handling. Further, we have to gain more experiences in wrapping tools, to anticipate possible problems in running a fully automized evaluation on top of the SEALS platform.

References

[1] Watson Wei Khong Chua and Jung-Jae Kim. Eff2Match results for OAEI 2010. In *Proceedings of the ISWC 2010 Workshop on Ontology Matching*, Shanghai, China, 2010.

[2] Jérôme David. AROMA results for OAEI 2009. In *Proceedings of the ISWC 2009 Workshop on Ontology Matching*, Washington DC, USA, 2009.

[3] Faycal Hamdi, Brigitte Safar, Nobal B. Niraula, and Chantal Reynaud. TaxoMap alignment and refinement modules: Results for OAEI 2010. In *Proceedings of the ISWC 2010 Workshop on Ontology Matching*, Shanghai, China, 2010.

[4] Wei Hu and Yuzhong Qu. Falcon-AO: A practical ontology matching system. *Journal of Web Semantics*, 6:237–239, 2008.

[5] Adrian Marte and Daniel Winkler. Iterative evaluation and implementation of the results repository service v1.1-beta. Technical Report D7.4, SEALS Project, November 2010.

[6] Christian Meilicke, Cássia Trojahn, Jérôme Euzenat, and Heiner Stuckenschmidt. Services for the automatic evaluation of matching tools. Technical Report D12.2, SEALS Project, July 2010.

[7] Md. Hanif Seddiqui and Masaki Aono. Anchor-Flood: results for OAEI 2009. In *Proceedings of the ISWC 2009 Workshop on Ontology Matching*, Washington DC, USA, 2009.

[8] Peng Wang and Baowen Xu. Lily: ontology alignment results for OAEI 2009. In *Proceedings of the ISWC 2009 Workshop on Ontology Matching*, Washington DC, USA, 2009.

A. SOAP-UI Test Suite

This section lists the xml file of the SOAP-UI project. It corresponds to a test suite for invoking the BPEL workflow.


```
<?xml version="1.0" encoding="UTF-8"?>
<con:testSuite xmlns:con="http://eviware.com/soapui/config" name="BPEL-TestSuite">
  <con:settings/>
  <con:runType>SEQUENTIAL</con:runType>
  <con:testCase failOnError="true" failTestCaseOnErrors="true" keepSession="false"
      maxResults="0" name="evaluationBinding TestSuite" searchProperties="true" id="4
      fd1db7e-8ddd-4665-913a-604e01e3c010">
    <con:settings/>
    <con:testStep type="properties" name="Initializing">
      <con:settings/>
      <con:config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="con
          :PropertiesStep" saveFirst="true">
        <con:properties>
          <con:property>
            <con:name>header.destination.httpURI</con:name>
            <con:value>http://localhost</con:value>
          </con:property>
          <con:property>
            <con:name>header.destination.executionRequestId</con:name>
            <con:value>urn:erq:6ba7b810-9dad-11d1-80b4-00c04fd430c8</con:value>
          </con:property>
        </con:properties>
      </con:config>
    </con:testStep>
    <con:testStep type="transfer" name="Transfer variables">
      <con:settings/>
      <con:config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="con
          :PropertyTransfersStep">
        <con:transfers setNullOnMissingSource="true" transferTextContent="true"
            failOnError="true" ignoreEmpty="false" transferToAll="false" useXQuery="
            false" entitize="false" transferChildNodes="false">
          <con:name>ToHttpURI</con:name>
          <con:sourceType>header.destination.httpURI</con:sourceType>
          <con:sourceStep>Initializing</con:sourceStep>
          <con:targetType>Request</con:targetType>
          <con:targetStep>evaluation</con:targetStep>
          <con:targetPath>//*[local-name()='HttpURI']</con:targetPath>
        </con:transfers>
        <con:transfers setNullOnMissingSource="true" transferTextContent="true"
            failOnError="true" transferChildNodes="false">
          <con:name>ToExecutionRequest</con:name>
          <con:sourceType>header.destination.executionRequestId</con:sourceType>
          <con:sourceStep>Initializing</con:sourceStep>
          <con:targetType>Request</con:targetType>
          <con:targetStep>evaluation</con:targetStep>
          <con:targetPath>//*[local-name()='ExecutionRequestId']</con:targetPath>
        </con:transfers>
      </con:config>
    </con:testStep>
    <con:testStep type="request" name="evaluation">
      <con:settings/>
      <con:config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="con
          :RequestStep">
        <con:interface>evaluationBinding</con:interface>
        <con:operation>initiate</con:operation>
        <con:request name="evaluation">
          <con:settings>
            <con:setting id="com.eviware.soapui.impl.wsdl.WsdlRequest@request-headers
                ">&lt;xml-fragment/&gt;</con:setting>
          </con:settings>
          <con:encoding>UTF-8</con:encoding>
          <con:endpoint>http://localhost:8092/evaluation</con:endpoint>
```

```
          <con:request><![CDATA[<soapenv:Envelope xmlns:soapenv="http://schemas.
              xmlsoap.org/soap/envelope/" xmlns:v1="http://www.seals-project.eu/
              resources/res/common/header/xsd/v1" xmlns:v11="http://www.seals-project
              .eu/resources/res/engine/evaluation/wsdl/v1" xmlns:v12="http://www.
              seals-project.eu/resources/res/common/types/xsd/v1">
   <soapenv:Header>
      <v1:Header>
         <!--Optional:-->
         <v1:Destination>
            <v1:HttpURI>http://localhost</v1:HttpURI>
         </v1:Destination>
         <v1:ExecutionRequestId>urn:erq:6ba7b810-9dad-11d1-80b4-00c04fd430c8</v1:
             ExecutionRequestId>
      </v1:Header>
   </soapenv:Header>
   <soapenv:Body>
    <v11:initiate>
         <v11:argument>
            <v12:name>Testsuite</v12:name>
            <v12:value>Benchmark+Testsuite</v12:value>
         </v11:argument>

         <v11:argument>
            <v12:name>Version</v12:name>
            <v12:value>2010</v12:value>
         </v11:argument>

    </v11:initiate>
   </soapenv:Body>
</soapenv:Envelope>]]></con:request>
          <con:jmsConfig JMSDeliveryMode="PERSISTENT"/>
          <con:jmsPropertyConfig/>
          <con:wsaConfig mustUnderstand="NONE" version="200508"/>
          <con:wsrmConfig version="1.2"/>
        </con:request>
      </con:config>
    </con:testStep>
    <con:testStep type="mockresponse" name="evaluationCallback">
      <con:settings/>
      <con:config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="con
          :MockResponseStep">
        <con:interface>evaluationCallbackBinding</con:interface>
        <con:operation>onResult</con:operation>
        <con:path>/evaluation/callback</con:path>
        <con:port>8088</con:port>
        <con:response>
          <con:settings/>
          <con:responseContent xsi:nil="true"/>
          <con:wsaConfig mustUnderstand="NONE" version="200508"/>
        </con:response>
      </con:config>
    </con:testStep>
    <con:properties/>
  </con:testCase>
  <con:properties/>
</con:testSuite>
```