HAL
open science

Iterative implementation of services for the automatic evaluation of matching tools (v2)

José Luis Aguirre, Christian Meilicke, Jérôme Euzenat

To cite this version:

José Luis Aguirre, Christian Meilicke, Jérôme Euzenat. Iterative implementation of services for the automatic evaluation of matching tools (v2). [Contract] 2012, pp.34. hal-00785742

HAL Id: hal-00785742

https://inria.hal.science/hal-00785742v1

Submitted on 6 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 - v2.0-FR

Coordinator: José Luis Aguirre Cervantes
With contributions from: Christian Meilicke, Jérôme Euzenat
Quality Controller: Heiner Stuckenschmidt
Quality Assurance Coordinator: Raúl García Castro

Document Identifier:	SEALS/2010/D12.5/Vv2.0-FR
Class Deliverable:	SEALS EU-IST-2009-238975
Version:	v2.0-FR
Date:	June 5, 2012
State:	final
Distribution:	public

Executive Summary

The aim of this deliverable is to report on the work that has been done in the last phase of task 12.5: iterative implementation of services for automatic evaluation of matching tools, and on the final status of those services. Since the previous iteration, reported in [12], we have been working on the following main tasks:

- extending the previous BPEL workflows in order to integrate the new custom services and fault handlers (§2);
- experimenting with the new release of SEALS RES environment (RES 1.2) (§3);
- defining suite metadata transformations for raw results and interpretations using SPARQL queries to be compliant with new SEALS ontologies specifications (§4);
- working with SEDL development team to adapt our SEALS Evaluation Description Document to the new version of the SEALS Evaluation Description Language (§5);
- extensions of the SEALS client for ontology matching evaluation (§6).

The first four tasks concern integration with the SEALS technology, while the last task concerns WP12 specific services that have been extended to advance the automation of evaluation campaigns and to sustain the OAEI 2011.5 campaign.

Regarding integration with the SEALS technology, our focus was on extending previous implementations of the WP12 services to conform with new versions/requirements of SEALS presented during the meeting held in Grenoble, France on February 2012. This comprehends a new distribution of RES (RES 1.2) and directives to transform the metadata of raw and interpretation results to a format compliant with new specifications of SEALS ontologies.

The new functionality and improvements introduced in the RES 1.2 version were added to our BPEL work-flow. This includes the use of fine-grained facilities for tool execution time and memory consumption measurements, and the standardization of Core Tool Services fault handling. The modified work-flows have been successfully tested with the RES 1.2 distribution, both on a local installation and on the SEALS Platform. These tests confirmed that significant differences arose when comparing the results obtained against results obtained using a runtime measurement custom service implementation present in previous versions.

SPARQL queries have been written to transform legacy metadata generated for raw results and interpretations in order to be compliant with the new SEALS ontologies specifications. This shows that even if the ontologies evolve, the legacy data can still be accessed and visualized using these or new transformations.

The SEALS client for ontology matching evaluation has been extended in two ways. First, a specific parameter setting is now possible to use the client to store the results in the Results Repository. Second, a flexible parameterization was introduced allowing the manipulation of the client through a command line interface where the users can specify the URL of a test repository, the ID of a test suite as well as the ID of a specific

version of that test suite. The SEALS client was extensively used for supporting the third SEALS ontology matching evaluation campaign.

Finally, this document reports on the whole set of software pieces that have been developed inside WP12 to support the automatic evaluation of matching tools.

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 - v2.0-FR
Work Package	**Number**	12	**Title**	Matching Tools

Date of Delivery	**Contractual**	M37	**Actual**	30-06-2012
Status	v2.0-FR		final ☒	
Nature	prototype ☒ report ☐ dissemination ☐			
Dissemination level	public ☐ consortium ☒			

Authors (Partner)	José Luis Aguirre Cervantes (INRIA), Christian Meilicke (University Mannheim), Jérôme Euzenat (INRIA)			
Resp. Author	**Name**	José Luis Aguirre Cervantes	**E-mail**	Jose-Luis.Aguirre@inria.fr
	Partner	INRIA	**Phone**	+33 (476) 615 476

Abstract (for dissemination)	This deliverable reports on the current status of the service implementation for the automatic evaluation of matching tools, and on the final status of those services. These services have been used in the third SEALS evaluation of matching systems, held in Spring 2012 in coordination with the OAEI 2011.5 campaign. We worked mainly on the tasks of modifying the WP12 BPEL work-flow to introduce new features introduced in the RES 1.2 version; testing the modified work-flows on a local installation and on the SEALS Platform; writing transformations of result data to be compliant with the new SEALS ontologies specifications; and finally, extending the SEALS client for ontology matching evaluation for better supporting the automation of WP12 evaluation campaigns and to advance in the integration with SEALS repositories. We report the results obtained while accomplishing these tasks.

Keywords	ontology matching, ontology alignment, evaluation, benchmarks, efficiency measure

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
23/04/2012	1	José Luis Aguirre	Set up structure deliverable.
18/05/2012	2	José Luis Aguirre	Wrote chapter introduction
21/05/2012	3	José Luis Aguirre	Wrote chapter BPEL
22/05/2012	4	José Luis Aguirre	Wrote chapters RES, SPARQL
23/05/2012	5	José Luis Aguirre	Wrote chapter SEDL
25/05/2012	6	Christian Meilicke	Wrote chapter on client
25/05/2012	7	José Luis Aguirre	Wrote executive summary and introduction
25/05/2012	8	José Luis Aguirre	Wrote conclusion chapter
30/05/2012	9	José Luis Aguirre	Addressed comments of QC
30/05/2012	10	Christian Meilicke	Final revision
31/05/2012	11	José Luis Aguirre	Addressed comments of QAC
01/06/2012	12	Christian Meilicke	Addressed final comment of QAC

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 8

1 Introduction 9

2 Updating BPEL evaluation to RES 1.2 distribution 10
2.1 Runtime measurement functionality 10
2.2 Standardization of Core Tool Service faults 14
2.3 Comments on the use of new functionality 16

3 Experimenting with the RES 1.2 environment 17
3.1 Testing the RES 1.2 distribution on a local installation 17
3.2 Testing the RES 1.2 distribution on the SEALS Platform 17

4 Metadata transformation of WP12 raw results and interpretations 20

5 New version of SEDL bundle 21

6 Extensions for running WP12 campaigns 22

7 Final status of services for the automatic evaluation of matching tools 24

8 Final remarks 25

References 26

A Sparql queries for transformation of raw results and interpretations metadata 29
A.1 Transformation of Raw Results 29
A.2 Transformation of Interpretations 30

B SEDL document 33

List of Figures

2.1 WP12 BPEL workflow. 11
2.2 Iteration over a test suite. 11
2.3 Iteration over a test suite without timestamp external service. 12
2.4 Exception handling for the alignment step in previous version. 15
2.5 Exception handling for alignment after standardization of Core Service Faults. 16

1. Introduction

Last versions of deliverable D12.5 were focused, on the one hand, on the integration of Work Package 12 developments (WP12) with SEALS components previously released, including Runtime Evaluation Service 1.1.x distributions (RES) and Test Data and Results Repositories (TDRS and RRS respectively). On the other hand, D12.5 deliverables were also focused on WP12 specific services needed to run the first and second SEALS ontology matching campaigns (OAEI 2010 and OAEI 2011). Those works have been reported in [12], [13] and [10]. This version focuses on the same subjects.

Regarding integration with SEALS, during the last stage of the project our focus was on extending previous implementations of WP12 services to conform with new versions/requirements of SEALS presented during the meeting held in Grenoble, France on February 2012. This comprehends a new distribution of RES (RES 1.2) and directives to transform the metadata of raw and interpretation results to a format compliant with new specifications of SEALS ontologies, and that will be used to uniformize the way these data will be visualized through the SEALS portal.

WP12 specific services have been also extended to advance the automation of evaluation campaigns and to sustain the OAEI 2011.5 campaign, which was composed of 5 tracks with several tools evaluated over various test suites [9]. As it was stated in one of previous deliverables [10], BPEL evaluation work-flows compliant with the SEALS guidelines do not support iteration over tools and test suites. For this reason we considered to run just parts of the campaign using BPEL work-flows running on the SEALS infrastructure. The fall-back strategy followed in OAEI 2011 campaign has been retaken and used with an extended version of the client that was originally designed to support tool developers in testing their tools locally.

In this deliverable, we report first in §2 on the modifications of WP12 BPEL evaluation description work-flow for use of new features and improvements included in RES 1.2 distribution. These new features are fine-grained facilities for tool execution time and memory consumption measurements. Also, catching of exceptions was modified for standardize the handling of Core Tool Services faults. In §3, we report on our experiments installing and testing RES 1.2 in local mode, and testing our work-flows on the SEALS Platform. We present in §4 the work done inside WP12 to deal with transformation of raw results and interpretations through SPARQL queries in order to be compliant with new SEALS ontologies specifications. To conclude with the subject of SEALS integration, we deal in §5 with the adaptations of WP12 Seals Evaluation Description Language (SEDL) bundle to the last version released of SEDL [2]. Then we report in §6 on the client extensions done to support the third SEALS ontology matching campaign (OAEI 2011.5). We present in §7 the final status of the services implemented inside WP12 for the automatic evaluation of matching tools, and we end in §8 with some conclusions reflecting the experiences we got in the effort of making our evaluation descriptions work on SEALS infrastructure and the most relevant learning we had running the third evaluation campaign for ontology matching tools.

2. Updating BPEL evaluation to RES 1.2 distribution

The RES 1.2 distribution was introduced during the SEALS meeting held in Grenoble, France from February 6th to February 9th 2012. RES 1.2 added features to the last version RES 1.1.1, such as the support of fine-grained tool execution time measurement, the support of memory consumption measurement for a given tool execution, and the support of custom tools. Some improvements with respect to RES 1.1.1 were also presented. Those improvements have to do with the standardization of Core Tool Services faults handling, and the recovering of identifiers generated for raw results and interpretations, as this information was not included in the response of the `addRawResult` and `addInterpretations` services in RES 1.1.1.

Even if previous versions of BPEL evaluations developed for RES 1.1.1 can run with no modifications on RES 1.2, one of our goals was to modify our evaluation description to take advantage of features/improvements presented within the new distribution. From features and improvements listed above we worked only on the runtime and memory consumption measurements, and on the standardization of SEALS fault handling. Regarding the recovering of raw results' and interpretations' identifiers, for WP12 this information is directly passed as parameters via the SoapUI project, so we did not need to modify our work-flow neither our metadata generation custom service implementation. Next sub-sections detail the tasks that we had to do in order to achieve that goal.

Before this, it is worth to remember the general functioning of our BPEL work-flow, whose general structure is shown in Figure 2.1. After some initialization steps, the `while` step, whose details are hidden in the figure, iterates a tool over a test suite. After the iteration, results are processed and sent to SEALS Results Repository. Figure 2.2 shows the details of the iteration, where an alignment invocation consists in fact on a two-step process. First the message sent to the alignment request mut be populated with parameter data (`setParamsAlign` step); then the alignment process is invoked (`align` step).

2.1 Runtime measurement functionality

Deliverable [10] reported on significant variations on runtime measurements of tools when using the BPEL evaluation work-flow with respect to similar executions using our client version. The reason was due to the fact that runtime measurement with BPEL needed the invocation of an external timestamp service which revealed very time consuming. Taking this into consideration, RES 1.2 offers the possibility to ask for runtime measurement, and also memory consumption, each time a tool is executed.

For introducing the runtime and memory consumption measurement, we modified our BPEL file as indicated during the Grenoble meeting, and we deleted all code having to do with our ancient runtime measurement custom service implementation. All this implied:

- To suppress all variables and links used by the custom service.
- To delete from the iteration part of our work-flow, shown in Figure 2.2, all the steps needed to set and invoke the external timestamp custom service. Fig-

Figure 2.1: WP12 BPEL workflow.

Figure 2.2: Iteration over a test suite.

ure 2.3 shows the resulting iteration, where several steps have been deleted: `setParamsTimestamp`, `getTimestampBefore` and `setTimestampBefore` before the `align` step; `getTimestampAfter`, `setTimestampAfter` after the `align` step.

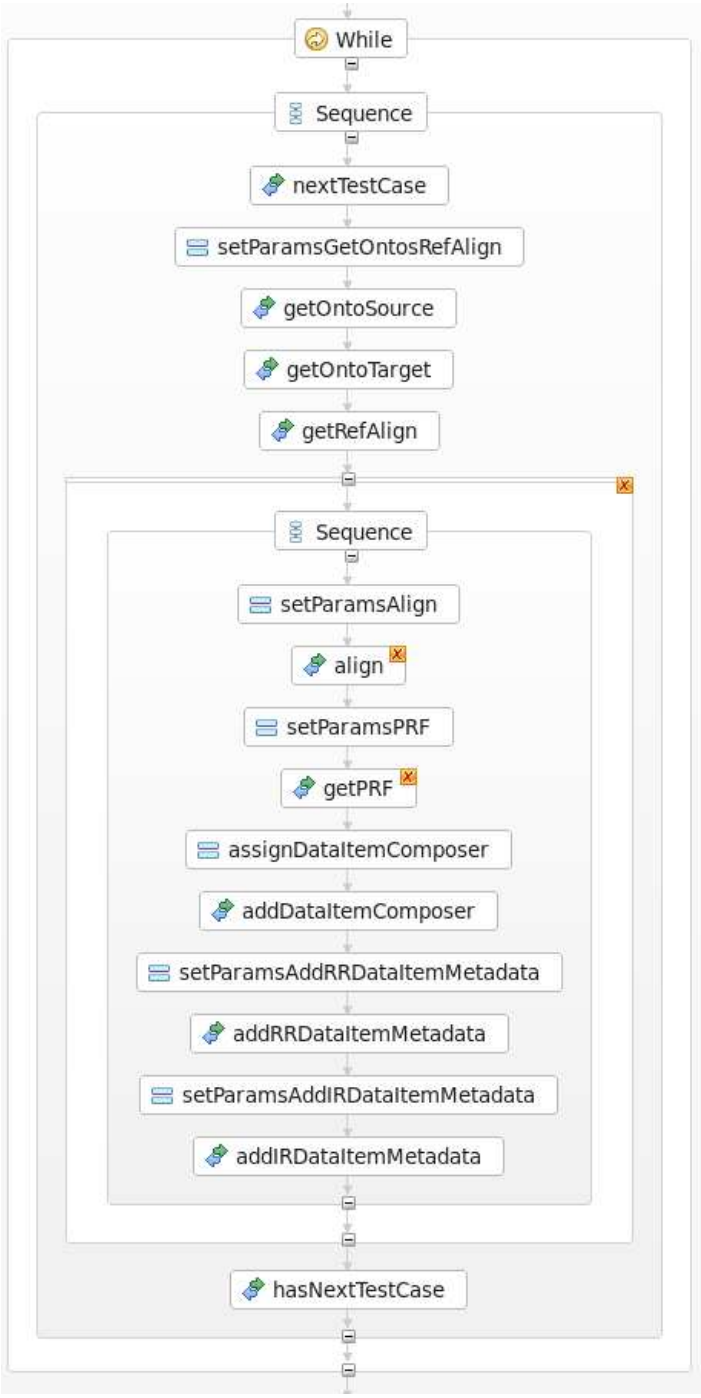

Figure 2.3: Iteration over a test suite without timestamp external service.

- To modify the reference of the ontology matching tools namespace to its new value in the files referring to it (BPEL and deployment files).

- To populate the tool-header part added to the message sent to the alignment request by including in the `setParamsAlign` step the assignment provided for that:

```
<bpel:copy>
  <bpel:from>
    <bpel:literal xml:space="preserve">
      <tools-schema:ToolHeader>
        <tools-schema:toolId>any-id</tools-schema:toolId>
        <tools-schema:options>
          <tools-schema:measure-time>true</tools-schema:measure-
             time>
          <tools-schema:measure-memory>
            <tools-schema:frequency>1000</tools-schema:frequency
               >
            <tools-schema:unit>B</tools-schema:unit>
          </tools-schema:measure-memory>
        </tools-schema:options>
      </tools-schema:ToolHeader>
    </bpel:literal>
  </bpel:from>
  <bpel:to part="toolHeader" variable="AlignRequest"> </bpel:to>
</bpel:copy>
```

  where the `AlignRequest` variable contains the message that will be passed to the `align` procedure.
- To recover and process the response after the alignment invocation. As runtime is one of the criteria added for interpretations in the same manner like precision, recall and F-measure, this is done before calling the service that adds those metadata (`addIRDataItemMetadata`), inside the `setParamsAddIRDataItemMetadata` step where the message `AddIRMetadataRequest` is populated:

```
<bpel:copy>
  <bpel:from>
    <![CDATA[$AlignResponse.toolHeaderResponse//*[local-name()='
       ellapsed'] div 1000]]>
  </bpel:from>
  <bpel:to part="payload" variable="AddIRMetadataRequest">
    <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:
       sublang:xpath1.0">
      <![CDATA[gen-types:runtime]]>
    </bpel:query>
  </bpel:to>
</bpel:copy>
```

2.2 Standardization of Core Tool Service faults

A result obtained during the execution of an evaluation is the number of problems detected. This information comes in terms of variables incremented each time an exception is caught. Exception catching in RES 1.1.1 was done through a fault type specific to each work package (ontology matching tools, ontology engineering tools and so on). This was redundant as three types of faults are detected independently of the type of tool: platform, tool and bridge faults. Besides, this implied that each time an exception was caught, we must include conditions to identify what was the cause of the problem. Figure 2.4 shows how this was accomplished for exceptions thrown during the alignment invocation. The code behind the steps shown in the figure is:

```
<bpel:catch faultName="omt:SEALSFault" faultVariable="Fault"
   faultMessageType="omt:SEALSFault">
  <bpel:sequence>
    <bpel:if>
     <bpel:condition><![CDATA[boolean($Fault.payload//*[local-name()
        ='ToolBridgeFault'])]]></bpel:condition>
     <bpel:sequence name="CatchToolBridgeFault">
      <bpel:assign validate="no" name="increaseNumToolBridgeFaults">
          <bpel:copy>
            <bpel:from><![CDATA[$NumToolBridgeFaults+1]]></bpel:from>
            <bpel:to variable="NumToolBridgeFaults"/>
            </bpel:copy>
        </bpel:assign>
      </bpel:sequence>
      <bpel:elseif>
        <bpel:condition><![CDATA[boolean($Fault.payload//*[local-name
           ()='ToolFault'])]]></bpel:condition>
        <bpel:sequence name="CatchToolFault">
          <bpel:assign validate="no" name="increaseNumToolFaults">
            <bpel:copy>
              <bpel:from><![CDATA[$NumToolFaults+1]]></bpel:from>
              <bpel:to variable="NumToolFaults"/>
            </bpel:copy>
          </bpel:assign>
        </bpel:sequence>
      </bpel:elseif>
    </bpel:if>
    <bpel:rethrow/>
  </bpel:sequence>
</bpel:catch>
```

One of the improvements in RES 1.2 is the standardization of Core Service Faults handling. Platform faults are now separated from tool and bridge faults, giving three types of SEALS faults defined like `UnexpectedPlatformFault`, `ToolFault` and `ToolBridgeFault`. To introduce the new way of handling SEALS faults, we modified the namespaces pointing to new fault types and the catching of exceptions where needed. Figure 2.5 shows exception handling for the alignment invocation. The resulting code is:

```
<bpel:catch faultName="tools-service:ToolBridgeFault" faultVariable="
   ToolBridgeFault" faultMessageType="tools-service:ToolBridgeFault">
  <bpel:sequence>
    <bpel:assign name="increaseNumToolBridgeFaults">
      <bpel:copy>
        <bpel:from><bpel:literal xml:space="preserve">true</bpel:
           literal></bpel:from>
        <bpel:to variable="haveToolFault"></bpel:to>
      </bpel:copy>
      <bpel:copy>
        <bpel:from><![CDATA[$NumToolBridgeFaults+1]]></bpel:from>
        <bpel:to variable="NumToolBridgeFaults"/>
      </bpel:copy>
    </bpel:assign>
    <bpel:rethrow/>
  </bpel:sequence>
</bpel:catch>
<bpel:catch faultName="tools-service:ToolFault" faultVariable="
   ToolFault" faultMessageType="tools-service:ToolFault">
  <bpel:sequence>
    <bpel:assign name="increaseNumToolFaults">
      <bpel:copy>
        <bpel:from><bpel:literal xml:space="preserve">true</bpel:
           literal></bpel:from>
        <bpel:to variable="haveToolFault"></bpel:to>
      </bpel:copy>
      <bpel:copy>
        <bpel:from><![CDATA[$NumToolFaults+1]]></bpel:from>
        <bpel:to variable="NumToolFaults"/>
      </bpel:copy>
    </bpel:assign>
    <bpel:rethrow/>
  </bpel:sequence>
</bpel:catch>
```

The full BPEL code for the last version of WP12 evaluation can be found at `https://svn.seals-project.eu/seals/deliverables/D12.5/M37/software/evals`.

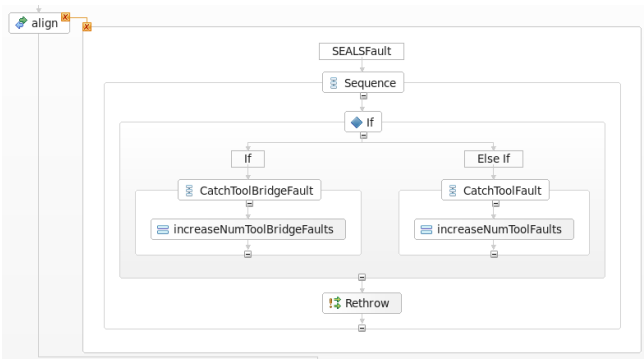

Figure 2.4: Exception handling for the alignment step in previous version.

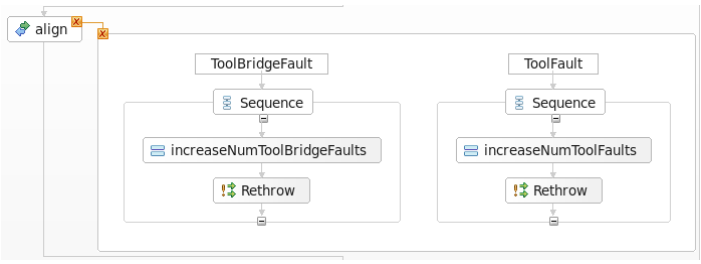

Figure 2.5: Exception handling for alignment after standardization of Core Service Faults.

2.3 Comments on the use of new functionality

The xsd type definition files describing how a request to invoke a tool must be populated specify that asking for runtime and/or memory consumption measurement is optional. However, we observed that if both of them are not included in the request, an exception is thrown.

An interesting fact was that our tests confirmed that runtime measurements are more fine grained that those we got when using the custom service implementation. Results obtained were more realistic diminishing in a 30-40% percent measurements obtained with custom service implementation.

3. Experimenting with the RES 1.2 environment

During the Grenoble meeting it was announced that RES 1.2 will be available on the SEALS Platform by the end of February. As the packages to install the RES 1.2 distribution were already available, we decided to test it first on a local installation to prepare our experiments on the SEALS Platform. We present in this chapter some reflections about these experiments.

3.1 Testing the RES 1.2 distribution on a local installation

The local installation of RES 1.2 is accomplished in the same manner than that of RES 1.1.1. It is long and tedious but easily done following the instructions contained in [6]. We proceeded successfully with this task, and we confirmed first that RES 1.1.1 evaluation descriptions could be run with no problem under the new distribution. We take advantage of this installation to test also the requirement stated in [6] for wrapping tools. A Python script was written to automate tool re-wrapping, liberating developers to re-compile their code in order to produce the new bridge tool implementation asked by RES. We observed however that tool re-wrapping is not necessary when running RES on a local installation, but it is an important issue as it is needed to execute tools on the SEALS Platform. Once we modify our BPEL work-flow as explained in §2, we also proceed to test it successfully with various tools.

3.2 Testing the RES 1.2 distribution on the SEALS Platform

The RES 1.2 distribution was released on the SEALS Platform on February 22th. As various differences were introduced in the way RES Worker instances are managed, we tried first to test evaluation versions for RES 1.1.1. In RES 1.2, the life cycle of RES Worker instances is controlled with a DOS command enabling four operations: stop, clean, start and deploy. We had several problems at the beginning due to the fact that the only documentation available to understand the effects of those operations is a one page diapositive in a hands-on session document.

The biggest problem is that before an evaluation execution, RES worker instances need to be stopped because one needs to inform the RES Worker of the location of the currently deployed tool that will be used for that evaluation. This is done manually by modifying the worker configuration file. Once the configuration file is set, the RES Worker instance can be restarted. However when we executed the SoapUI project to trigger the evaluation, the worker instance was never recognized. We searched for an explanation in the log files, but as they are very huge and contain a lot of non understandable information, we executed the clean operation to re-create them. We did this without knowing that the effect of this operation is also to undeploy the RES packages needed to run a RES Worker instance, so that was the reason the instance was never recognized. After several exchanges with people in charge of RES development, we finally knew that we needed to redeploy the instance.

At the end we could run and test our BPEL work-flow successfully; as we said before, in this context we need to use re-wrapped tools in order to make the resultant alignments available through the File publisher.

At one time we considered the possibility to run part of the evaluation campaign using the BPEL work-flow. However as it was also stated in previous deliverables, BPEL work-flows are unable to iterate over tools and test suites. In current conditions, we have to do a lot of manual work to run an evaluation campaign, and this represented a problem for us, as around 15 tools participated in each track of the ontology matching campaign, and several test suites were tried for each tool [9]. To support that affirmation, we present below the steps needed to iterate over tools and test suites on the SEALS Platform (all of them are manual). Before this, we recall that tool interfaces are encapsulated in a ZIP file with the following structure:

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

where the folder `bin/lib/` contains all required libraries for executing the tool, while the folders `conf/` and `lib/` contain configuration and distribution resources required by the tool at runtime, for example word dictionaries, background knowledge and so on.

Now, to iterate over tools and test suites we would need to proceed like follows:

1. Stop and clean RES Worker instance.
2. Modify the RES Worker configuration file to indicate the location of the currently deployed tool.
3. Deploy the tool if needed (install the conf and lib folders containing any resource needed by the tool to be executed).
4. Start and deploy RES Worker instance.
5. Start SoapUI if it is not started.
6. Edit SoapUI project to indicate which is the test suite that will be used and to assign identifiers to Raw Results and Interpretations.

7. Run SoapUI project.

Steps 6 and 7 must be repeated to iterate over test suites, and all the steps must be repeated to iterate over tools. Cleaning the RES worker is not really necessary but it is better to do it to be sure that RES Worker instances will be correctly identified.

Considering all the manual work needed to implement the iterations, we decided not to use BPEL evaluations for running the OAEI 2011.5 campaign. Instead, we use extensively the client version already introduced in other deliverables. We comment on extensions done to this version in §6.

Nevertheless, we conducted several short tests with tools that do not need step 3 of the iteration to be performed. The tools for which we ran BPEL work-flows on the SEALS Platform are Aroma [5], LogMapLt which is a light version of LogMap [7], MapSSS [4], MapEVO [8] and MapPSO [3], and the results of these tests were stored in the SEALS Results Repository. These results can be accessed via the identifiers `OMT-2011.5-biblio-SEALSbenchmarks-%SYSTEM%-rr`, which refers to the uninterpreted raw results, and `OMT-2011.5-biblio-SEALSbenchmarks-%SYSTEM%-ir` for interpretations where `%SYSTEM%` must be replaced by the name of a tool.

4. Metadata transformation of WP12 raw results and interpretations

SEALS ontologies are used in the SEALS Platform for representing the resources managed by it, including evaluation descriptions, test data, results, etc. Data stored in the SEALS Result Repository concerns both raw results (i.e., alignments in WP12) and interpretations of these results (i.e., precision and recall measurement). All those data is described in a metadata format according to SEALS ontologies definitions[1]. One functionality to be added to the SEALS portal is visualization of those data.

SEALS suite ontologies have evolved over the duration of the project, and several updates have been done for describing raw results and interpretations in a generic way. Previous versions of the BPEL workflow generated descriptions which are not completely compliant with new SEALS ontologies and just a few work packages integrated already the new ontologies in their metadata results. Then, the problem of what to do with legacy data arose during the Grenoble meeting.

The proposed solution consists in writing some code to transform raw results and interpretations described with the old ontologies to the new ones, which permits on the one hand the visualization of old and new data, and avoids on the other hand rewriting the code that some work packages (including WP12) are currently using for metadata generation. In this way, results of the transformation will be compliant with new ontologies, in order to be processed for visualization purposes through the SEALS portal.

Based on examples put on the SEALS wiki by Raúl García Castro, we wrote two SPARQL queries to transform WP12 raw results and interpretations. The content of these queries can be consulted in Appendix A.

[1]See definitions in `http://www.seals-project.eu/wiki/index.php/Ontologies`.

5. New version of SEDL bundle

As reported in [13], an Evaluation Description Document specified in the SEALS Evaluation Description Language (SEDL) defines all required resources and dependencies among them for carrying out an evaluation in the SEALS Platform [1]. It comprises an Execution Contract, a WS-BPEL 2.0 workflow, custom services that are invoked from within the workflow, as well as ontologies that define categories used in the Execution Contract. The last one describes the conditions that need to hold for tools, test suites, and results that can be processed by the evaluation workflow. Those resources are bundled in a ZIP file that must be stored into the Evaluation Description repository (EDRS).

A new version of SEDL being released by January 2012 [2], our previous SEDL bundle had to suffer minor modifications. The SEALS team in charge of SEDL specification and implementation achieved the modifications resulting in the following structure:

```
ed-omt-test-sedl/
   evaluation.sedl.xml
   resources/
        bpel/
             EvaluationWP12.bpel
        impl/
             ed-omt-resources-1.0.0-SNAPSHOT-jar-with-dependencies.
                 jar
        ontologies/
             OMTOntology.owl
        wsdl/
             metadataGen.wsdl
             prf.wsdl
```

Listing 5.1: EDRS bundle structure

The SEDL document `evaluation.sedl.xml` can be found in Appendix B. We observed no big differences between the two versions, the only one being the use of the jar file `ed-omt-resources-1.0.0-SNAPSHOT-jar-with-dependencies.jar` instead of `ed-omt-resources-1.0.0-SNAPSHOT.jar`

At the time this deliverable is written, the SEALS platform does not seem able to run evaluations stored in the EDRS.

6. Extensions for running WP12 campaigns

As already explained, we have executed the majority of our evaluations in OAEI 2011.5 with the help of the SEALS client for ontology matching. This client has first been mentioned in deliverable D12.5-v2.0-beta [10], where we also report about its usage in the context of OAEI 2011 (the second SEALS evaluation campaign for ontology matching). The client is available as a single jar file. It can be used via a simple command line call to execute a complete evaluation run for one tool and one test suite. The tool needs to wrapped against the SEALS interface and the test suite needs to be stored in the SEALS Test Data Repository. The evaluation can take place on a standard laptop or on a powerful server machine. No specific infrastructure is required.

The client can be used for different purposes. Tool developers can use it to test their systems. They can check whether their tool correctly implements the required interface. This is a requirement for participating in the campaign. Moreover, they can directly compute precision and recall scores for the test suites stored in the TDRS (aside from any specific evaluation campaign). Evaluation campaign organizers can use it to execute basic evaluation workflows, as long as they follow the simple pattern that is common to each WP12 workflow (see Figure 2.1 and Figure 2.2). As explained on the previous pages, we have encountered several problems in running a complete campaign on top of the SEALS platform with a BPEL workflow. For that reason we decided to extend the client to integrate it more tightly with other SEALS components and to prepare it for future campaigns. In particular, we extended the client in the following way.

Storage of results The results generated by the usage of the previous version of the client were stored locally in an proprietary format. They have not been stored in the SEALS results repository. Meanwhile we have included the component for storing results and interpretation, which has been developed for usage within the BPEL workflow. Via a specific parameter setting it is now possible to use the client to store the results in the Results Repository.

Flexible Parameterization In the previous version of the client we supported to run three predefined test suites. These have been the test suites of OAI 2011. However, in OAEI 2011.5, we needed to execute the client on several new test suites, and the same can be expected to happen in any subsequent campaign. For that reason we modified the command line interface of the client and allowed to specify the URL of a test repository, the ID of a test suite as well as the ID of a specific version of that test suite.

We have also modified and improved the documentation of the client. The client, as well as its documentation, is available at `http://oaei.ontologymatching.org/2011.5/seals-eval.html`. This documentation helped the tool developers to solve the problem of wrapping their tool against the client on their own. Two or three developers had problems within this process that could finally be solved with our support. A list of available test suites (together with an explicit listing of relevant IDs) can be found at `http://oaei.ontologymatching.org/2011.5/suites.html`.

The client itself supports no functionality to iterate over tools or test suites. However, it supports a minimal mechanism to deploy the resources required for a tool to be executed correctly and it can easily be called from an external program. Thus, it is easy to implement the iteration in a simple script or in a short java program. This was the pragmatic approach that we have finally chosen to run most of our evaluations. Note also that we executed a large deal of our evaluation runs on the SEALS virtualization infrastructure, i.e., on virtual machines running on SEALS server machines. This was possible due to the flexibility of the client-based approach.

7. Final status of services for the automatic evaluation of matching tools

Three objectives were established in the document SEALS Description of Work (DOW) for WP12. One of them makes a clear reference to the technology that had to be developed to automate evaluations: **"developing automated tools that perform all aspects of evaluation: test generation, benchmark processing, result analysis, and publishing;".** With regard to this statement we present in the following the final status of WP12 implementations.

We have implemented a test generator [11], [13] which has been used to produce the tests used in two evaluation campaigns. The generator receives as input a reference ontology and a set of parameters describing the kind of modifications that will be applied to the reference ontology. It delivers as output a set of modified ontologies together with the corresponding reference alignments. The modifications can be done on three dimensions: removing (classes, properties, comments or restrictions), adding (classes or properties), and renaming (classes or properties).

Several pieces of software have been developed to support benchmark processing in an automated way. A module allows for uploading complete data sets to the SEALS Test Data Repository; SEALS metadata compliant with SEALS ontologies is generated at the same time. A python script allows for downloading executable tools that have been uploaded and stored by the tool developers in the SEALS tool repository. Also, we have developed a client for ontology matching evaluation which has been used in two evaluation campaigns; the client can receive as parameters the location where a matching tool is deployed, the URL of a test repository, the ID of a test suite as well as the ID of a specific version of that test suite, and it is able to run the evaluation of that tool for the data set specified.

We have also produced a BPEL work-flow that can process the evaluation of a tool with a data set in the RES 1.2 distribution. However, we did not use the SEALS Evaluation Description Repository because work-flows stored there cannot yet be executed. For the reasons explained in this document §2, we can not run a full campaign via the BPEL work-flows. Instead of that, we wrote a simple Linux shell script as well as a Java program that iterates the client over tools and data sets.

Finally, related with results analysis and publishing, both the client and the BPEL work-flow allow for uploading the results of an evaluation to the SEALS Results Repository. These results data sets are freely available and can be donwloaded by everyone. We have not developed visualization components as other SEALS partners are working on this aspect, however, the ontology matching evaluation results stored in the Results Repository are compliant, through the use of SPARQL queries, with the SEALS ontologies. This will allow visualization tools to process those results as soon as these tools are available.

All these pieces of software can be found at `https://svn.seals-project.eu/seals/deliverables/D12.5/M37/software`.

8. Final remarks

This deliverable has reported about the work we have done in the last iteration of task 12.5: iterative implementation of services for automatic evaluation of matching tools, and on the final status of those services. We worked mainly on the task of modifying the WP12 BPEL work-flow to take advantage of the new functionality and improvements introduced in the RES 1.2 version during the SEALS meeting held in Grenoble on February 2012. The modified work-flows have been successfully tested both on a local installation and on the SEALS Platform. We worked also on the task of writing transformations of result data to be compliant with the new SEALS ontologies specifications. Finally, the SEALS client for ontology matching evaluation, used extensively in the third SEALS ontology matching campaign , has been extended for better supporting the automation of WP12 evaluation campaigns and to advance in the integration with important parts of SEALS technology, like the Tools, Test Data and Results repositories.

The lessons learned during this iterations can be resumed in the following points:

- Runtime measurement has been added to tools invocation confirming that significant differences resulted when comparing the results obtained against results obtained using a runtime measurement custom service implementation.
- Iteration over tools and data sets using the RES environment revealed a big time consuming task. A lot of manual work has to be done if we want to run a complete evaluation campaign with this environment. We hope that the release of the SEALS manager will help to solve this problem for future campaigns.
- Metadata generated for raw results and interpretations has been easily transformed through SPARQL queries to be compliant with the new SEALS ontologies specifications. It seems then that even though the ontologies continue to evolve, legacy data for results and interpretations can still be accessed and visualized using these transformations.
- We have seen that the client-based approach allows us to run a complete campaign in an automated way. The client is both a flexible solution and integrated with many other components of the platform. However, we cannot use it to measure, for example, memory consumption.

As we said before, we hope that the release of the SEALS manager will permit to fully exploit the hardware infrastructure and the software modules developed for the SEALS project.

From the very beginning of the project we decided to integrate SEALS campaigns with existing OAEI campaigns. Thus, we had a different starting point compared to the other research work packages. Our decision was based on the high acceptance of OAEI in the matching community. In doing so, we could benefit from a high number of participants and a partially predefined overall procedure. At the same, this resulted in a number of technical challenges. In particular, we had to use the software developed within the project from the very beginning in a complex scenario with a high number of participants that were used to a certain procedure following a tight schedule.

Finally, we can draw the conclusion, that the SEALS technology we have introduced to OAEI in an iterative way – from OAEI 2010 over OAEI 2011 to OAEI 2011.5 - has been very positively accepted. The main improvements are the reproducibility of results and the measurement of runtimes and scalability. As a side effect of asking tool developers to wrap their tools against the interfaces of the SEALS platform, many tools have become available in a more robust and executable version compared to the time before OAEI 2011. Overall, we conclude that the technology introduced by SEALS helped to improve OAEI evaluations significantly.

References

[1] Juergen Bock, Michael Schneider, and Carlos Moya. Language for describing evaluations – v2 (updated working specification). Technical Report D8.2-v2, SEALS Project, February 2011.

[2] Juergen Bock, Michael Schneider, and Stuart N. Wrigley. Language for describing evaluations – v3. Technical Report D8.2-v3, SEALS Project, January 2012.

[3] Jurgen Bock and Jan Hettenhausen. Discrete particle swarm optimisation for ontology alignment. *Information Sciences*, 192(0):152 – 173, 2012.

[4] M. Cheatham. MapSSS results for OAEI 2011. In *Proceedings of the ISWC 2011 Workshop on Ontology Matching*, Boston, USA, 2011.

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

[6] Miguel Esteban Gutiérrez, Matthias Pressnig, and Francisco Martín Recuerda. Iterative evaluation and implementation of the runtime evaluation service. Technical Report D9.3, V1.1, SEALS Project, October 2011.

[7] Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. Logmap: Logic-based and scalable ontology matching. In Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Noy, and Eva Blomqvist, editors, *The Semantic Web ISWC 2011*, volume 7031 of *Lecture Notes in Computer Science*, pages 273–288. Springer Berlin / Heidelberg, 2011.

[8] Carsten Danschel Jurgen Bock and Matthias Stumpp. MapPSO and MapEVO results for OAEI 2011. In *Proceedings of the ISWC 2011 Workshop on Ontology Matching*, Boston, USA, 2011.

[9] Christian Meilicke, José-Luis Aguirre-Cervantes, Jérôme Euzenat, Ondřej Šváb Zamazal, Ernesto Jiménez-Ruiz, Ian Horrocksa, and Cássia Trojahn. Results of the second evaluation of matching tools. Technical Report D12.6, SEALS Project, May 2012.

[10] Christian Meilicke, Cássia Trojahn, Jérôme Euzenat, and Heiner Stuckenschmidt. Iterative implementation of services for the automatic evaluation of matching tools. Technical Report D12.5, V2.0-beta, SEALS Project, December 2011.

[11] Maria Rosiou, Cássia Trojahn, and Jérôme Euzenat. Ontology matching benchmarks: generation and evaluation. In *Proceedings of the ISWC 2011 Workshop on Ontology Matching*, Bonn, DE, 2011.

[12] Cássia Trojahn, Christian Meilicke, and Jérôme Euzenat. Iterative implementation of services for the automatic evaluation of matching tools. Technical Report D12.5, V1.0-beta, SEALS Project, March 2011.

[13] Cássia Trojahn, Christian Meilicke, and Jérôme Euzenat. Iterative implementation of services for the automatic evaluation of matching tools. Technical Report D12.5, V1.0-FR, SEALS Project, July 2011.

A. Sparql queries for transformation of raw results and interpretations metadata

This section lists the queries written to transform WP12 metadata formats to new SEALS ontologies descriptions.

A.1 Transformation of Raw Results

```
PREFIX rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dc:     <http://purl.org/dc/terms/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

# v1
PREFIX sealsv1: <http://www.seals-project.eu/ontologies/SEALSMetadata.owl#>
PREFIX omt:    <http://www.seals-project.eu/ontologies/OMTEvaluations.owl#>

# v2
PREFIX sealsv2:  <http://www.seals-project.eu/ontologies/SEALSOntologies.owl#>
PREFIX c: <http://www.seals-project.eu/ConformanceOMT#>

CONSTRUCT {
# RawResultType

c:ConformanceRawResultType a sealsv2:ResultType ;
              dc:title "Conformance Raw Result" ;
              dc:description "Type for conformance raw results" ;
              sealsv2:hasComponent c:RRComponent1 ,
                                   c:RRComponent2 ,
                                   c:RRComponent3 .

c:RRComponent1 a sealsv2:SuiteTypeComponent ;
             sealsv2:requiresType c:Alignment .

c:Alignment a sealsv2:FileType ;
                dc:title "Alignment" ;
                dc:description "The alignment obtained" .

c:RRComponent2 a sealsv2:SuiteTypeComponent ;
               sealsv2:requiresType c:ExecutionProblemInTool .

c:ExecutionProblemInTool a sealsv2:RawType ;
                         dc:title "ExecutionProblemInTool" ;
                         dc:description "Whether there was any execution problem in
                             the tool" ;
                         sealsv2:hasLexicalRepresentation xsd:boolean  .

c:RRComponent3 a sealsv2:SuiteTypeComponent ;
               sealsv2:requiresType c:ExecutionProblemInPlatform .

c:ExecutionProblemInPlatform a sealsv2:RawType ;
                             dc:title "ExecutionProblemInPlatform" ;
                             dc:description "Whether there was any execution problem
                                 in the platform" ;
                             sealsv2:hasLexicalRepresentation xsd:boolean .

# RawResult
     ?suite rdf:type sealsv2:Suite .
     ?suite rdf:type sealsv2:RawResult .
     ?suite sealsv2:hasSuiteItem ?suiteItem .
     ?suite sealsv2:isTypedBy c:ConformanceRawResultType .

# SuiteItem
     ?suiteItem rdf:type sealsv2:SuiteItem .
     ?suiteItem sealsv2:belongsToSuite ?suite .
```

```
     ?suiteItem dc:identifier ?sid .
     ?suiteItem sealsv2:hasData [ rdf:type sealsv2:RawValue ;
              sealsv2:isTypedBy c:ExecutionProblemInTool ;
              sealsv2:hasValue ?executionProblemInTool  ;
              sealsv2:hasPosition "1"
                                 ] .
     ?suiteItem sealsv2:hasData [ rdf:type sealsv2:RawValue ;
              sealsv2:isTypedBy c:ExecutionProblemInPlatform ;
              sealsv2:hasValue ?executionProblemInPlatform ;
              sealsv2:hasPosition "2"
                                 ] .
     ?suiteItem sealsv2:hasData [ rdf:type sealsv2:File ;
              dc:identifier ?did ;
        sealsv2:isTypedBy c:Alignment ;
              sealsv2:isLocatedAt ?locatedAt ;
              sealsv2:hasPosition "3"
                                 ] .
}

WHERE {

     ?suite rdf:type sealsv1:Suite .
     ?suite rdf:type omt:RawResultSuite .
     ?suite sealsv1:hasSuiteItem ?suiteItem .

     ?suiteItem dc:identifier ?sid .
     ?suiteItem omt:executionProblemInTool ?executionProblemInTool .
     ?suiteItem omt:executionProblemInPlatform ?executionProblemInPlatform .
     ?suiteItem sealsv1:hasDataItem ?dataItem .

     ?dataItem dc:identifier ?did .
     ?dataItem sealsv1:isLocatedAt ?locatedAt.
}
```

A.2 Transformation of Interpretations

```
PREFIX rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dc:     <http://purl.org/dc/terms/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

# v1
PREFIX sealsv1: <http://www.seals-project.eu/ontologies/SEALSMetadata.owl#>
PREFIX omt:    <http://www.seals-project.eu/ontologies/OMTEvaluations.owl#>

# v2
PREFIX sealsv2:  <http://www.seals-project.eu/ontologies/SEALSOntologies.owl#>
PREFIX c: <http://www.seals-project.eu/ConformanceOMT#>

CONSTRUCT {
# RawResultType

c:ConformanceInterpretationType a sealsv2:ResultType ;
              dc:title "Conformance Interpretation" ;
              dc:description "Type for conformance interpretations" ;
              sealsv2:hasComponent c:IRComponent1 ,
                                   c:IRComponent2 ,
                                   c:IRComponent3 ,
                                   c:IRComponent4 .

c:IRComponent1 a sealsv2:SuiteTypeComponent ;
             sealsv2:requiresType c:Runtime .

c:Runtime a sealsv2:RawType ;
                dc:title "runtime" ;
```

```
                dc:description "The runtime measurement for the execution of the tool
                     on this alignment" ;
    sealsv2:hasLexicalRepresentation xsd:float .

c:IRComponent2 a sealsv2:SuiteTypeComponent ;
             sealsv2:requiresType c:Precision .

c:Precision a sealsv2:RawType ;
                dc:title "precision" ;
                dc:description "Precision measure of the alignment" ;
    sealsv2:hasLexicalRepresentation xsd:float .

c:IRComponent3 a sealsv2:SuiteTypeComponent ;
             sealsv2:requiresType c:Recall .

c:Recall a sealsv2:RawType ;
                dc:title "recall" ;
                dc:description "Recall measure of the alignment" ;
    sealsv2:hasLexicalRepresentation xsd:float .

c:IRComponent4 a sealsv2:SuiteTypeComponent ;
             sealsv2:requiresType c:FMeasure .

c:FMeasure a sealsv2:RawType ;
                dc:title "fmeasure" ;
                dc:description "fmeasure of the alignment" ;
    sealsv2:hasLexicalRepresentation xsd:float .

# Interpretation
     ?suite rdf:type sealsv2:Suite .
     ?suite rdf:type sealsv2:Interpretation .
     ?suite sealsv2:hasSuiteItem ?suiteItem .
     ?suite sealsv2:isTypedBy c:ConformanceInterpretationType .

# SuiteItem
     ?suiteItem rdf:type sealsv2:SuiteItem .
     ?suiteItem sealsv2:belongsToSuite ?suite .
     ?suiteItem dc:identifier ?sid .
     ?suiteItem sealsv2:hasData [ rdf:type sealsv2:RawValue ;
              sealsv2:isTypedBy c:Runtime ;
              sealsv2:hasValue ?runtimeValue  ;
              sealsv2:hasPosition "1"
                                  ] .
     ?suiteItem sealsv2:hasData [ rdf:type sealsv2:RawValue ;
              sealsv2:isTypedBy c:Precision ;
              sealsv2:hasValue ?precisionValue  ;
              sealsv2:hasPosition "2"
                                  ] .
     ?suiteItem sealsv2:hasData [ rdf:type sealsv2:RawValue ;
              sealsv2:isTypedBy c:Recall ;
              sealsv2:hasValue ?recallValue  ;
              sealsv2:hasPosition "3"
                                  ] .
     ?suiteItem sealsv2:hasData [ rdf:type sealsv2:RawValue ;
              sealsv2:isTypedBy c:FMeasure ;
              sealsv2:hasValue ?fmeasureValue  ;
              sealsv2:hasPosition "4"
                                  ] .
}

WHERE {

     ?suite rdf:type sealsv1:Suite .
     ?suite rdf:type omt:InterpretationSuite .
     ?suite sealsv1:hasSuiteItem ?suiteItem .

     ?suiteItem dc:identifier ?sid .
     ?suiteItem omt:hasCriterionItem ?runtimeCriterionItem .
     ?suiteItem omt:hasCriterionItem ?precisionCriterionItem .
     ?suiteItem omt:hasCriterionItem ?recallCriterionItem .
```

```
    ?suiteItem omt:hasCriterionItem ?fmeasureCriterionItem .

    ?runtimeCriterionItem omt:hasCriterion "runtime" .
    ?runtimeCriterionItem omt:hasValue ?runtimeValue .

    ?precisionCriterionItem omt:hasCriterion "precision" .
    ?precisionCriterionItem omt:hasValue ?precisionValue .

    ?recallCriterionItem omt:hasCriterion "recall" .
    ?recallCriterionItem omt:hasValue ?recallValue .

    ?fmeasureCriterionItem omt:hasCriterion "fmeasure" .
    ?fmeasureCriterionItem omt:hasValue ?fmeasureValue .
}
```

B. SEDL document

This section lists the SEDL document that specifies all the requirements needed to execute an evaluation workflow.

```
<?xml version="1.0"?>

<!-- ******************** -->
<!-- OMT SEDL Document    -->
<!-- ******************** -->

<sedl:EvaluationDescription
    xmlns:sedl="http://www.seals-project.eu/resources/sedl/v2/"
    xmlns:seals="http://www.seals-project.eu/ontologies/SEALSMetadata.owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tdcat="http://www.seals-project.eu/sedl/example/testData"
    xmlns:rcat="http://www.seals-project.eu/sedl/example/Result"
    name="OMT Evaluation"
    creator="Cassia Trojahn"
    created="2011-07-18T23:00:00+01:00">

  <!-- ~~~~~~~~~~~~~ -->
  <!-- Documentation -->
  <!-- ~~~~~~~~~~~~~ -->
  <sedl:Documentation>OMT evaluation description.</sedl:Documentation>

  <!-- ~~~~~~~~~~~~~~~~~~~~~~~ -->
  <!-- Execution Contract Part -->
  <!-- ~~~~~~~~~~~~~~~~~~~~~~~ -->
  <sedl:ExecutionContract>
    <sedl:Parameters>
      <sedl:Parameter name="testSuite"        type="tdcat:OMTTestSuite"
          resourceRef="OMTOntology"/>
      <sedl:Parameter name="testSuiteVersion" type="tdcat:OMTTestSuiteVersion"
          resourceRef="OMTOntology"/>
      <sedl:Parameter name="tool"             type="seals:OntologyMappingTool"
          resourceRef="OMTOntology"/>
      <sedl:Parameter name="toolVersion"      type="seals:OntologyMappingToolVersion"
           resourceRef="OMTOntology"/>
    </sedl:Parameters>
    <sedl:Outputs>
      <sedl:Output name="rawResult"           type="rcat:OMTRawResult" resourceRef="
          OMTOntology"/>
    </sedl:Outputs>
  </sedl:ExecutionContract>

  <!-- ~~~~~~~~~~~~~~~~~~~ -->
  <!-- Custom Services Part -->
  <!-- ~~~~~~~~~~~~~~~~~~~ -->
  <sedl:CustomServices>
    <sedl:CustomService name="PRFService">
      <sedl:Documentation>Service that computes precision, recall and f-measure.</
          sedl:Documentation>
      <sedl:WSDL portType="service:PRF"
     resourceRef="PRFServiceWSDL"
     xmlns:service="http://www.seals-project.eu/resources/omt/custom/prf/wsdl/v1"/>

      <sedl:Implementation className="eu.sealsproject.domain.omt.custom.prf.v1.
          PRFImpl"
         resourceRef="CustomServicesJAR"/>
    </sedl:CustomService>

    <sedl:CustomService name="MetadataGenService">
      <sedl:Documentation>Service that generate the required raw result and
          interpretation suite metadata.</sedl:Documentation>
      <sedl:WSDL portType="service:MetadataGen" resourceRef="MetadataGenServiceWSDL"
```

```
                xmlns:service="http://www.seals-project.eu/resources/omt/custom/
                    metadataGen/wsdl/v1"/>

      <sedl:Implementation className="eu.sealsproject.domain.omt.custom.metadataGen.
          v1.MetadataGenImpl"
         resourceRef="CustomServicesJAR"/>
    </sedl:CustomService>

  </sedl:CustomServices>

  <!-- ~~~~~~~~~~~~~~~~~~~~~~~~ -->
  <!-- Evaluation Workflow Part -->
  <!-- ~~~~~~~~~~~~~~~~~~~~~~~~ -->
  <sedl:EvaluationWorkflow processName="wp12-evaluation" resourceRef="EvaluationBPEL
      "/>

  <!-- ~~~~~~~~~~~~~~ -->
  <!-- Resources Part -->
  <!-- ~~~~~~~~~~~~~~ -->
  <sedl:Resources>

   <sedl:Resource xsi:type="sedl:XMLResourceType"
      name="EvaluationBPEL"
      xmlType="BPEL"
      location="resources/bpel/EvaluationWP12.bpel">
        <sedl:Dependency resourceRef="PRFServiceWSDL"/>
        <sedl:Dependency resourceRef="MetadataGenServiceWSDL"/>
   </sedl:Resource>

   <sedl:Resource xsi:type="sedl:XMLResourceType"
      name="PRFServiceWSDL"
      xmlType="WSDL"
      location="resources/wsdl/prf.wsdl"/>

   <sedl:Resource xsi:type="sedl:OntologyResourceType"
      name="OMTOntology"
      ontologyLanguage="OWL"
      location="resources/ontologies/OMTOntology.owl"/>

   <sedl:Resource xsi:type="sedl:XMLResourceType"
      name="MetadataGenServiceWSDL"
      xmlType="WSDL"
      location="resources/wsdl/metadataGen.wsdl"/>

   <sedl:Resource xsi:type="sedl:BinaryResourceType"
                  name="CustomServicesJAR"
      location="resources/impl/ed-omt-impl-1.0.0-SNAPSHOT-jar-with-dependencies.jar
          "/>
  </sedl:Resources>

</sedl:EvaluationDescription>
```