

## **SEALS**

*Semantic Evaluation at Large Scale*

**FP7 – 238975**

---

# **D14.3 Results of the first evaluation of Semantic Web Service tools**

---

**Coordinator: Serge Tymaniuk**

**With contributions from: Liliana Cabral, Serge Tymaniuk,  
Daniel Winkler, Ioan Toma**

**Quality Controller: Francisco Martin-Recuerda**

**Quality Assurance Coordinator: Raúl García Castro**

Document Identifier:	SEALS/2010/D14.3/V1.0
Class Deliverable:	SEALS EU-IST-2009-238975
Version:	version 1.0
Date:	November 29, 2010
State:	final
Distribution:	public



## EXECUTIVE SUMMARY

The goal of workpackage 14 is to design and implement an approach for the automatic evaluation of Semantic Web Service (SWS) tools according to the SEALS infrastructure. This deliverable describes the results of the first SEALS Evaluation campaign for SWS tools, based on the approach described in deliverables D14.1 [1] and D14.2 [3] and also [2].



## DOCUMENT INFORMATION

<b>IST Project Number</b>	FP7 – 238975	<b>Acronym</b>	SEALS
<b>Full Title</b>	Semantic Evaluation at Large Scale		
<b>Project URL</b>	<a href="http://www.seals-project.eu/">http://www.seals-project.eu/</a>		
<b>Document URL</b>			
<b>EU Project Officer</b>	Carmela Asero		

<b>Deliverable</b>	<b>Number</b>	14.3	<b>Title</b>	Results of the first evaluation of Semantic Web Service tools
<b>Work Package</b>	<b>Number</b>	14	<b>Title</b>	Semantic Web Service Tools

<b>Date of Delivery</b>	<b>Contractual</b>	M18	<b>Actual</b>	30-Nov-10
<b>Status</b>	version 1.0		final	<input checked="" type="checkbox"/>
<b>Nature</b>	prototype <input type="checkbox"/> report <input checked="" type="checkbox"/> dissemination <input type="checkbox"/>			
<b>Dissemination level</b>	public <input checked="" type="checkbox"/> consortium <input type="checkbox"/>			

<b>Authors (Partner)</b>	Liliana Cabral (OU), Serge Tymaniuk (UIBK), Daniel Winkler (UIBK), Ioan Toma (UIBK)			
<b>Resp. Author</b>	<b>Name</b>	Serge Tymaniuk	<b>E-mail</b>	serge.tymaniuk@sti2.at
	<b>Partner</b>	UIBK	<b>Phone</b>	+43 512 507 6464

<b>Abstract (for dissemination)</b>	This deliverable presents a description of the first SEALS evaluation campaign over Semantic Web Service (SWS) tools and the results of the evaluation for the participating tools.
<b>Keywords</b>	SEALS services, SWS tools evaluation

Version Log			
Issue Date	Rev No.	Author	Change
11-Oct-2010	1	Ioan Toma	created TOC
27-Oct-2010	2	Liliana Cabral	Revised TOC and added tables with tools results
28-Oct-2010	3	Liliana Cabral	Added content to sections 1, 2, 3, 5, 7 and appendices A and C
6-Nov-2010	4	Serge Tymaniuk	Added content to sections 1, 4, 6 and appendix B
18-Nov-2010	5	Liliana Cabral	Updated all sections
26-Nov-2010	6	Serge Tymaniuk	Update after review
26-Nov-2010	7	Liliana Cabral	Update after review and final editing



## PROJECT CONSORTIUM INFORMATION

Participant's name	Partner	Contact
Universidad Politécnica de Madrid		Asunción Gómez-Pérez Email: asun@fi.upm.es
University of Sheffield	 The University Of Sheffield.	Fabio Ciravegna Email: fabio@dcs.shef.ac.uk
Forschungszentrum Informatik an der Universität Karlsruhe		Rudi Studer Email: studer@fzi.de
University of Innsbruck		Ioan Toma Email: ioan.toma@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	 The Open University	Liliana Cabral Email: l.s.cabral@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
LIST OF TABLES	7
1 INTRODUCTION	8
2 EVALUATION CAMPAIGN	9
3 EVALUATION SCENARIOS	10
3.1 SWS Discovery Evaluation Scenario . . . . .	10
3.1.1 The SWS Plugin . . . . .	10
3.1.2 Implemented Measures . . . . .	10
4 TESTDATA ACCESS	13
5 EVALUATION RESULTS	14
5.1 Participating Tools . . . . .	14
5.1.1 Tool Parameter Settings . . . . .	14
5.2 Testdata . . . . .	15
5.3 Evaluation Execution . . . . .	15
5.3.1 Tools Retrieval Performance by Goal . . . . .	15
5.4 Analysis of Evaluation Results . . . . .	17
5.5 Lessons Learned . . . . .	17
6 RESULTS ACCESS AND VISUALIZATION	19
6.1 Results Access . . . . .	19
6.2 Results Visualization . . . . .	19
7 CONCLUSIONS	21
REFERENCES	21
A SUBMITTING A TOOL	23
A.1 How to Submit Your Tool . . . . .	23
A.2 Discovery Interface Implementation Example . . . . .	23
B OLWS-TC METADATA	25
B.1 Example metadata of OWL-TC data set . . . . .	25
C OLWS-MX SWS PLUGIN IMPLEMENTATION	27
C.1 OLWS-M0 SWS Plugin implementation . . . . .	27
C.2 OLWS-M4 SWS Plugin implementation . . . . .	28



# LIST OF FIGURES

6.1	Sample Raw Results. . . . .	20
6.2	Sample Interpretation Results. . . . .	20



## LIST OF TABLES

3.1	Description of Metrics as implemented by Galago (reproduced for convenience). . . . .	12
5.1	Evaluation campaign participating tools. . . . .	14
5.2	Goals (service requests) in evaluation. . . . .	15
5.3	Comparative tool performance on the OWLS-TC4 dataset. . . . .	17



## 1. Introduction

The goal of workpackage 14 is to design and implement an approach for the automatic evaluation of Semantic Web Service (SWS) tools according to the SEALS infrastructure. This deliverable describes the results of the first SEALS Evaluation campaign for SWS tools, which is based on the approach described in deliverables D14.1 [1] and D14.2 [3] and also [2].

In this deliverable we focus on the experimental evaluation of SWS tools with the objective of testing the SEALS infrastructure. Particularly, we focus on SWS discovery activity, which consists of finding Web Services based on their semantic descriptions.

An overview of the evaluation campaign is given in Section 2. Section 3 describes the evaluation scenario for SWS discovery (matchmaking) tools. We describe the SWS plugin API, which works as a wrapper for SWS tools, providing a common interface for evaluation. In addition, the evaluation metrics used in the current implementation are presented.

Section 4 describes the access to the testdata available in the SEALS repository, namely the OWLS-TC 4.0 test collection, which is used for the evaluation described in this deliverable.

In Section 5 we present the list of participating tools, tools parameter setting, test data used for the evaluation, evaluation execution and results.

SWS tools evaluation produces raw results and interpretations stored in the Result Repository Service. Section 6 describes how results can be accessed from the SEALS Results Repository and visualized using HTML.

Finally, in Section 7 we describe our conclusions and future work.



## 2. Evaluation Campaign

In the first SEALS evaluation campaign we execute the SWS discovery evaluation scenario<sup>1</sup>. Basically, participants register their tool via the Web interface provided in the SEALS website<sup>2</sup> and the organizers download and run their tools as part of the evaluation campaign scenario. Participants are also required to implement the SWS Tool plugin API according to the instructions provided in the website (also available in Appendix A). The organizers make instructions available for the participants about the scenario and perform the evaluation automatically by executing the evaluation workflow (see D14.2). The results become available in the Results Repository.

This evaluation campaign will run experimentally in order to test services of the SEALS platform. Note that not all capabilities are available at the moment. Therefore, the results are also experimental and comparable to the results obtained when participating for example in the S3 contest. The results and lessons learned from the S3 contest and this evaluation campaign are also to be discussed during the workshop for this campaign.

In the workshop we present results for the participating tools, show how to access the results of participating tools and how to perform an evaluation using the SEALS platform. Currently, we only perform automatic evaluation of tools implemented in Java.

For the results provided in this deliverable we used the OWLS-TC 4.0 test collection<sup>3</sup>. As this is a public test collection, participants can test the plugin implementation of their tools locally using the datasets provided.

---

<sup>1</sup><http://www.seals-project.eu/seals-evaluation-campaigns/semantic-web-services>

<sup>2</sup><http://www.seals-project.eu/registertool>

<sup>3</sup><http://projects.semwebcentral.org/projects/owl-tc/>



## 3. Evaluation Scenarios

### 3.1 SWS Discovery Evaluation Scenario

Currently, we focus on the SWS discovery activity, which consists of finding Web Services based on their semantic descriptions. Tools for SWS discovery or matchmaking can be evaluated on retrieval performance, where for a given goal, i.e. a semantic description of a service request, and a given set of service descriptions, i.e. semantic descriptions of service offers, the tool returns the match degree between the goal and each service, and the SEALS platform, through the provided services, measures the rate of matching correctness based on a number of metrics.

#### 3.1.1 The SWS Plugin

In SEALS we provide the SWS plugin API, available from the campaign website, that must be implemented by tool providers participating in the SWS tool evaluation (see the example in Appendix A). The SWS Plugin API has been derived from the SEE API (see also D14.1) and works as a wrapper for SWS tools, providing a common interface for evaluation.

The *Discovery* Interface has 3 methods (*init()*, *loadServices()*, *discover()*) and defines a class for returning discovery results. The methods are called in different steps of the evaluation workflow. The method *init()* is called once after the tool is deployed so that the tool can be initialized. The method *loadServices()* is called once for every dataset during the evaluation (loop) so that the list of services given as arguments can be loaded. The method *discover()* is called once for every goal in the dataset during the evaluation (loop) so that the tool can find the set of services that match the goal given as argument. The return type is defined by the class *DiscoveryResult*. The class *DiscoveryResult* contains the goal and the list of service matches (class *Match*). The class *Match* contains the service description URI, the order (rank), the match degree ('NONE', 'EXACT', 'PLUGIN', 'SUBSUMPTION') and the confidence value, which can be used as the score value. It is expected that the services that do not match are returned with match degree 'NONE'.

#### 3.1.2 Implemented Measures

Evaluation measures for SWS discovery will follow in general on the same principles and techniques from the more established Information Retrieval (IR) evaluation research area. Therefore we will use some common terminology and refer to common measures. For a survey of existing measures we refer the reader to existing published papers, for example [4] and [5].

In SEALS we make the measurement services available as a plugin (currently Java interfaces). That is, we can implement the interface using any publicly available metrics APIs. In the interface, *DiscoveryMeasurement* is the main class, which returns metrics results for a given Discovery Result and Reference Set corresponding to the same goal. This class also returns overall measures for a list of goals. The *Discovery*



*Result* class is part of the SWS plugin API (Section above). The *DiscoveryReferenceSet* class contains the list of service judgments (class *DiscoveryJudgement*) for a specific goal, which includes the service description URI, and the relevance value. The relevance value is measured against the match degree or confidence (score) value returned by a tool. The *MetricsResult* class will contain a list of computed measure values such as precision and recall and also some intermediate results such as the as number of returned relevant services for a goal.

In our current implementation of the *DiscoveryMeasurement* interface we use the evaluation metrics available from **Galago**<sup>1</sup> as described in Table 3.1. Galago is a open source toolkit for experimenting with text search. It is based on small, pluggable components that are easy to replace and change, both during indexing and during retrieval. In particular, we used the retrieval evaluator component, which computes a variety of standard information retrieval metrics commonly used in TREC. Galago is written in Java and works on any system with a JDK version 1.5 or later.

---

<sup>1</sup><http://www.galagosearch.org/galagosearch-core/apidocs/>



Table 3.1: Description of Metrics as implemented by Galago (reproduced for convenience).

Measure	Description
Num Ret	Number of retrieved documents
Num Rel	Total number of documents judged relevant
Num Rel Ret	Number of retrieved documents that were judged relevant
Aver Prec	Average Precision. “Suppose the precision is evaluated once at the rank of each relevant document in the retrieval. If a document is not retrieved, we assume that it was retrieved at rank infinity. The mean of all these precision values is the average precision”.
NDCG @ N	Normalized Discounted Cumulative Gain at cutoff point N. “This measure was introduced in Jarvelin, Kekalainen, “IR Evaluation Methods for Retrieving Highly Relevant Documents” SIGIR 2001. The formula is copied from Vassilvitskii, “Using Web-Graph Distance for Relevance Feedback in Web Search”, SIGIR 2006. Score = $N \sum_i (2^{r(i)} - 1) / \log(1 + i)$ , where N is such that the score cannot be greater than 1. We compute this by computing the DCG (unnormalized) of a perfect ranking”.
NDCG	Normalized Discounted Cumulative Gain. “NDCG at (Math.max(retrieved.size(), judgments.size()))”.
R-prec	R-Precision. “Returns the precision at the rank equal to the total number of relevant documents retrieved. This method is equivalent to precision(relevantDocuments().size())”.
Bpref	Binary Preference. “The binary preference measure, as presented in Buckley, Voorhees “Retrieval Evaluation with Incomplete Information”, SIGIR 2004. The formula is: $1/R \sum_r 1 -  nrankedgreaterthanr /R$ where R is the number of relevant documents and n is a member of the set of first R judged irrelevant documents retrieved”.
Recip Rank	Reciprocal Rank. “Returns the reciprocal of the rank of the first relevant document retrieved, or zero if no relevant documents were retrieved”.
Precision @ N	Precision at cutoff point N. “Returns the precision of the retrieval at a given number of documents retrieved. The precision is the number of relevant documents retrieved divided by the total number of documents retrieved”.
Recall @ N	Recall at cutoff point N. “Returns the recall of the retrieval at a given number of documents retrieved. The recall is the number of relevant documents retrieved divided by the total number of relevant documents for the query”.



## 4. Testdata Access

For the results provided in this deliverable we used the OWLS-TC 4.0 test collection<sup>1</sup>.

The OWLS-TC4.0 version consists of 1083 semantic web services described with OWL-S 1.1, covering nine application domains (education, medical care, food, travel, communication, economy, weapons, geography and simulation). OWLS-TC4 provides 42 test queries associated with binary as well as graded relevance sets. 160 services and 18 queries contain Precondition and/or Effect as part of their service descriptions.

In our evaluation, the OWLS-TC4.0 test collection metadata was described with the *Suite* ontology and the *DiscoveryTestSuite* ontology as described in D5.4 [6] and D14.2 [3]. The test collection suite was encapsulated in a ZIP file, including the suite metadata, which has the file name *Metadata.rdf* and registered in the SEALS Testdata repository.

According to the *Suite* ontology, the suite metadata is described with the concepts *Suite*, *SuiteItem* and *DataItem*, where a Suite consist of multiple SuiteItems, which itself consists of various DataItems. In the metadata for OWLS-TC4.0, the Suite Items in *Metadata.rdf* consist of 42 discovery tests, which correspond to the number of reference sets in the test collection. Each Discovery Test Suite consists of a goal document and a list of judged service documents associated with it (DataItems). Each service has a relevance element, indicating the relevance value between the service socument and the goal document. An example of the metadata is shown in Appendix B.

The test collection can be accessed via the Test Data Repository Service:

<http://seals.sti2.at/tdrs-web/testdata/persistent/OWLS-TC/4.0/suite/>

Depending on the value of the HTTP Accept of the URL above, either the metadata or the data is retrieved. To retrieve the metadata of the test suite version, the HTTP Accept value should be set to "application/rdf+xml". To retrieve the actual data as a ZIP file, HTTP Header value should be set to "application/zip".

For example, in order to retrieve the goal document, which belongs to the particular DiscoveryTest, the following URL structure should be used:

<http://.../OWLS-TC/4.0/suite/DiscoveryTestSuite02/component/GoalDocument>

Otherwise, in order to retrieve the particular goal document, we should use the following URL structure:

<http://.../OWLS-TC/4.0/suite/DiscoveryTestSuite03/GoalDocument6>

---

<sup>1</sup><http://projects.semwebcentral.org/projects/owls-tc/>



## 5. Evaluation Results

### 5.1 Participating Tools

The list of participating tools is shown in Table 5.1. These tools are variants of OWLS-MX<sup>1</sup>, which is publicly available. Each OWLS-MX variant runs a different similarity measure algorithm and can be adjusted to run with certain parameters, which include type of sorting (Semantic (2); Syntactic(1); hybrid (0)) and syntactic similarity threshold (0.0 - 1.0) .

For our evaluation we packed each variant as a different tool, according to the instructions provided in the website<sup>2</sup> (also available in Appendix A). We downloaded the source code and created a jar of the OWLS-MX API not including the GUI APIs.

Regarding the SWS Plugin API, we implemented the required methods (*init()*, *loadServices()*, *discover()*) by calling the appropriate methods of the OWLS-MX API. The implementation of the plugin for OWLS-M0 and OWLS-M4 are given in Appendix C.

#### 5.1.1 Tool Parameter Settings

OWLS-M0 was configured in a way that only semantic based results (degree of match equals to 0 (*exact*), 1 (*plugin*), 2 (*subsumes*) or 3 (*subsumed-by*)) were included in the set of retrieved matches (raw results). Thus, it used the semantic type of sorting mentioned previously.

OWLS-M2 and OWLS-M3 were configured in a way that all results with syntactic similarity greater than zero were included in the set of retrieved matches (raw results). The degree of match was not taken into account. They used the syntactic type of sorting mentioned previously.

OWLS-M4 was configured in a way that only syntactic based results (degree of match equals 4 (*nearest neighbour*)) with syntactic similarity greater than 0.7 were included in the set of retrieved matches (raw results). It used the syntactic type of sorting mentioned previously.

The goals for these setting were: a) to compare the performance of a semantic based variant (OWLS-M0) with the performance of a syntactic based variant (OWLS-M4); and b) compare the performance of two syntactic based variants (OWLS-M2 and OWLS-M3) using different matching algorithms (similarity measures).

Table 5.1: Evaluation campaign participating tools.

Tool	Description
OWLS-M0	OWLS-MX variant - Constraint Similarity Measure
OWLS-M2	OWLS-MX variant - Extended Jaccard Similarity Measure
OWLS-M3	OWLS-MX variant - Cosine Similarity Measure
OWLS-M4	OWLS-MX variant - Jensen Shannon Similarity Measure

<sup>1</sup><http://projects.semwebcentral.org/projects/owls-mx/>

<sup>2</sup><http://www.seals-project.eu/seals-evaluation-campaigns/semantic-web-services>



## 5.2 Testdata

For our evaluation we used the OWLS-TC 4.0 test collection<sup>3</sup> (see also Section 4).

We selected a number of queries and used the service descriptions (from the test collection) included in their reference sets.

The test collection was also deployed locally in a Glassfish Server and made available as prescribed by OWLS-TC. For example, service descriptions were available at <http://127.0.0.1/services/1.1/>.

We show the results for three individual Goals (service requests) within OWLS-TC 4.0 as shown in Table 5.2. We have not performed overall measures (over all goals).

Table 5.2: Goals (service requests) in evaluation.

Goal	Name	URI
request id="5"	car price service	<a href="http://127.0.0.1/queries/1.1/car_price_service.owl">http://127.0.0.1/queries/1.1/car_price_service.owl</a>
request id="8"	2 For 1 Price service	<a href="http://127.0.0.1/queries/1.1/dvd-playermp3player_price_service.owl">http://127.0.0.1/queries/1.1/dvd-playermp3player_price_service.owl</a>
request id="21"	Researcher address service	<a href="http://127.0.0.1/queries/1.1/researcher-in-academia_address_service.owl">http://127.0.0.1/queries/1.1/researcher-in-academia_address_service.owl</a>

## 5.3 Evaluation Execution

Results from our evaluation for the testdata and tools explained in the previous sections are given in Table 5.3. This table shows the comparative retrieval performance per Goal (service request) of OWLS-M0, OWLS-M2, OWLS-M3, OWLS-M4 over OWLS-TC4 datasets. The two first rows indicate the parameter settings of the tools as explained in subsection 5.1.1. The metrics have been explained in Section 3.1.2.

For reference purposes, we added a reference implementation (*Ref*), which accessed the dataset provided and retrieved all the relevant services from the reference set. Thus, the expected values for Precision and Recall of the *Ref* tool is one. We used the number of documents retrieved as the cutoff point for all tests. Loading time was calculated by measuring the time to execute the *loadService()* method in the SWS plugin.

We downloaded the test suites metadata from the SEALS repository and performed the evaluation in a local machine. The configuration of the local machine was an Intel Core 2 Duo CPU, 3 GHz processor, 64-bit OS, 4GB RAM, running Windows 7.

### 5.3.1 Tools Retrieval Performance by Goal

<sup>3</sup><http://projects.semwebcentral.org/projects/owl-tc/>



	Ref	OWLS-M0	OWLS-M2	OWLS-M3	OWLS-M4
Semantic Sorting	-	yes	no	no	no
Syntactic Threshold	-	-	0	0	0.70
<b>Request Id=05</b>					
Load time (ms)	243	22358	21713	21745	21592
Dataset size	176	176	176	176	176
Num Rel	93	93	93	93	93
Num Ret	93	21	174	174	5
Num Rel Ret	93	20	93	93	4
Aver Prec	47.00	2.26	47.00	47.00	0.11
NDCG	4.68	1.01	4.68	4.69	0.20
NDCG at Ret	4.68	2.75	4.68	4.68	1.36
R-prec	1.00	0	1.00	1.00	0
Bpref	0	0	0	0	0
Recip Rank	1.00	1.00	1.00	1.00	1.00
Precision at Ret	1.00	0.95	0.54	0.54	0.80
Recall at Ret	1.00	0.22	1.00	1.00	0.04
<b>Request Id=08</b>					
Load time (ms)	715	29910	29892	30448	30558
Dataset size	171	171	171	171	171
Num Rel	27	27	27	27	27
Num Ret	27	24	164	164	44
Num Rel Ret	27	20	27	27	6
Aver Prec	14.00	7.78	14.00	14.00	0.78
NDCG	3.16	2.34	3.16	3.16	0.70
NDCG at Ret	3.16	2.53	3.16	3.16	0.70
R-prec	1.00	0	1.00	1.00	0.22
Bpref	0	0	0	0	0
Recip Rank	1.00	1.00	1.00	1.00	1.00
Precision at Ret	1.00	0.95	0.17	0.17	0.14
Recall at Ret	1.00	0.74	1.00	1.00	0.22
<b>Request Id=21</b>					
Load time (ms)	756	144176	144421	144875	145062
Dataset size	283	283	283	283	283
Num Rel	72	72	72	72	72
Num Ret	72	19	76	76	5
Num Rel Ret	72	18	32	32	5
Aver Prec	36.50	2.38	7.33	7.33	0.21
NDCG	4.34	1.09	1.93	1.93	0.30
NDCG at Ret	4.34	2.64	1.93	1.93	1.70
R-prec	1.00	0	0.44	0.44	0
<i>continued on next page</i>					



<i>continued from previous page</i>					
	Ref	OWLS-M0	OWLS-M2	OWLS-M3	OWLS-M4
Bpref	0	0	0	0	0
Recip Rank	1.00	1.00	1.00	1.00	1.00
Precision at Ret	1.00	0.95	0.42	0.42	1.00
Recall at Ret	1.00	0.25	0.44	0.44	0.07

Table 5.3: Comparative tool performance on the OWLS-TC4 dataset.

## 5.4 Analysis of Evaluation Results

As mentioned before, the tools were packed by the organizers according to predetermined settings. These experimental settings were enough to raise very different results for most tools as well as unexpected results.

First, we found out that all variants performed exactly the same when using the same settings. This is shown in Table 5.3 for OWLS-M2 and OWLS-M3, but was tested for all variants. This result was unexpected because each variant is running a different matching algorithm (similarity measure).

Second, the recall performance of OWLS-M2 and OWLS-M3 for two queries is equal to 1 and the number retrieved close to the dataset size. This implies that these variants might have 'learned' the reference set.

Last, there is no conclusive results for the overall performance between OWLS-M0 OWLS-M4, thus we cannot say that the semantic based variant is better than the syntactic based variant (with 0.7 threshold).

In addition load time has been directly proportional to the dataset size. The factors that affect loading time are related to reading documents into memory and the size of documents.

As a note, it seems that the values for Average Precision and NDCG are not accurate (see results for Ref tool) when the number of relevant documents is equal to the number of relevant retrieved documents (maybe a bug within the Galago API).

## 5.5 Lessons Learned

With respect to the datasets we noticed that all variants of OWLS-MX could retrieve all the same relevant services for a given reference set, provided that they were set with the same parameters. Thus, either the algorithms are not being invoked properly or the measures seem not to account for the different matching algorithms (similarity measures) used by the different variants (e.g. execution time).

When recall at number retrieved is equal to 1 for a high number of queries, this indicates bias of the test suite against the tool. It would be important to check whether OWLS-TC has bias towards the OWLS-MX tools. It is important that the SWS community get more engaged in creating non-biased datasets.

In addition, given that small changes in parameters can produce different results, it is also important to make intermediate results available (via evaluation services) and provide alternative metrics.



Overall, it is not easy to say why some tools fail to certain queries. But, we have noticed that some ontologies could not be read. Thus, it is important to introduce some validation procedure.



## 6. Results Access and Visualization

SWS tools evaluation produces two types of results, stored in the Result Repository Service: *raw results*, which are generated by running a tool with specific testdata, and *interpretations*, which are obtained after the evaluation metrics are applied over the raw results. This section describes access to results in the SEALS repository, and their visualization in HTML.

### 6.1 Results Access

This section describes how results can be accessed from the SEALS Results Repository Service. The interactions with the Results Repository Service were refactored and updated and are currently handled by the *SingleResultManager* component. This component can be used to retrieve and extract the contents of a specified raw result or interpretation obtained from an specific evaluation execution, given the URI of the raw result or interpretation.

Currently, raw results can be published at:

<http://seals.sti2.at/rrs-web/results/>

Interpretations results can be published at:

<http://seals.sti2.at/rrs-web/interpretations/>

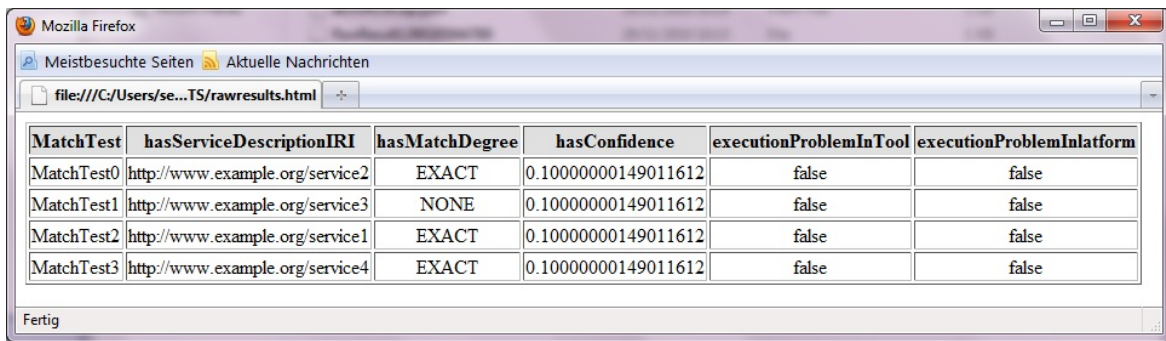
In order to retrieve the results three get methods, introduced in deliverable 14.2 [3], which contain relevant SPARQL queries, were implemented.

Firstly, the manager service creates a temporary directory, the metadata is retrieved and stored in the metadata file. Afterwards, the data file is created and the Accept header is set to "application/zip" in order to retrieve the ZIP file. Finally, the response from the HTTP request is copied to a local ZIP file and the content of the ZIP file is extracted.

### 6.2 Results Visualization

We describe below how results can be visualized using HTML in order to provide structured, user-friendly meaningful information to the user.

The visualization component (available under `swst/src/main/java/visualization`) handles the mapping of raw results and interpretations from RDF to HTML. Using SPARQL, the RDF data are queried from the Results Repository and are put to the HTML tables. The visualization consists of HTML tables that can be used for the analysis of the evaluation results by participants. In Figures 6.1 and 6.2 we show the visualization of sample raw results and interpretation results.

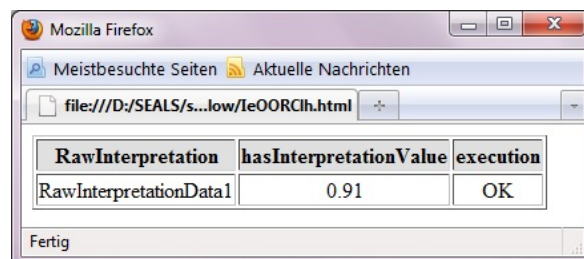


The screenshot shows a Mozilla Firefox browser window with the address bar set to `file:///C:/Users/se...TS/rawresults.html`. The main content area displays a table with the following data:

MatchTest	hasServiceDescriptionIRI	hasMatchDegree	hasConfidence	executionProblemInTool	executionProblemInlatform
MatchTest0	<a href="http://www.example.org/service2">http://www.example.org/service2</a>	EXACT	0.10000000149011612	false	false
MatchTest1	<a href="http://www.example.org/service3">http://www.example.org/service3</a>	NONE	0.10000000149011612	false	false
MatchTest2	<a href="http://www.example.org/service1">http://www.example.org/service1</a>	EXACT	0.10000000149011612	false	false
MatchTest3	<a href="http://www.example.org/service4">http://www.example.org/service4</a>	EXACT	0.10000000149011612	false	false

The status bar at the bottom of the browser window shows the word "Fertig".

Figure 6.1: Sample Raw Results.



The screenshot shows a Mozilla Firefox browser window with the address bar set to `file:///D:/SEALS/s...low/leOORClh.html`. The main content area displays a table with the following data:

RawInterpretation	hasInterpretationValue	execution
RawInterpretationData1	0.91	OK

The status bar at the bottom of the browser window shows the word "Fertig".

Figure 6.2: Sample Interpretation Results.



## 7. Conclusions

The work presented in this deliverable has been used during the SWS Tools Evaluation Campaign 2010<sup>1</sup> corresponding to the SEALS Semantic Web Service Discovery Evaluation scenario. Instructions for participants have been made available, including the release of the SWS plugin API. In this first campaign, the organizers have packed the participating tools.

We have performed an experimental evaluation with the objective of testing the SEALS infrastructure. The evaluation results are not meant to be conclusive in terms of tool performance, but instead count as input to requirements for the SEALS infrastructure. Currently, we do not provide comparative results in graphical form. Instead, evaluation results are stored in RDF and can be visualized in HTML.

From our analysis, we find that public intermediate results and repeatability are important for studying the behaviour of the tools under different settings (not only best behaviour). In addition, the SEALS infrastructure can help in several steps of the evaluation process, including generating and accessing datasets and results via metadata.

With respect to measures, we have implemented for this evaluation the API provided by Galago as described in this deliverable. Galago provides several measures covering the ones provided by SME2 (S3 Contest), and also additional ones. The SEALS infrastructure allows to plugin different source APIs as metrics services.

In this deliverable we have shown how to use the ongoing approach and services for SWS tools evaluation using the SEALS platform. The SWS plugin API is currently very similar to SME2's matchmaker plugin in what concerns discovery (matchmaking). However, the former will be extended in order to account for other activities such as composition, mediation and invocation. There are also many similarities in purpose between our approach and existing initiatives in that they all intend to provide evaluation services and promote discussion on SWS technologies within the SWS community. In this case, the main difference is that SEALS is investigating the creation of common and sharable metadata specifications for testdata, tools and evaluation descriptions as well as respective public repositories. In addition, results are publicly available and can be used for alternative visualizations by users.

Currently, the metadata for OWLS-TC4.0 is stored in the dataset repository. For future campaigns we plan to release datasets described using other languages such as WSMO-Lite as well as datasets for other problem scenarios such as the ones in the SWS Challenge.

Future work includes developing the APIs as Web Services and implementing the evaluation workflow as a BPEL process.

---

<sup>1</sup><http://www.seals-project.eu/seals-evaluation-campaigns/semantic-web-services>



## REFERENCES

- [1] L. Cabral, M. Kerrigan, and B. Norton. D14.1. Evaluation Design and Collection of Test Data for Semantic Web Service Tools. Technical report, SEALS Project, March 2010.
- [2] L. Cabral and I. Toma. Evaluating Semantic Web Services Tools using the SEALS Platform. In *Proceedings of IWEST Workshop at ISWC 2010*, Shanghai, China, Nov 2010.
- [3] L. Cabral, I. Toma, and Adrian Marte. D14.2. Services for the Automatic Evaluation of Semantic Web Service Tools v1. Technical report, SEALS Project, August 2010.
- [4] G. Demartini and S. Mizzaro. A Classification of IR Effectiveness Metrics. In *Proceedings of ECIR 2006*. LNCS 3936, pp. 488 to 491, Springer, 2006.
- [5] U. Kuester and B. Koenig-Ries. Measures for Benchmarking Semantic Web Service Matchmaking Correctness. In *Proceedings of ESWC 2010*. LNCS 6089, Springer, June 2010.
- [6] A. Marte and D. Winkler. D5.4 Iterative evaluation and implementation of the Test Data Repository Service. Technical report, SEALS Project, November 2010.



## A. Submitting a Tool

### A.1 How to Submit Your Tool

When you unzip SEALS-SWST-Campaign-2010-Instructions.zip you will find the following content under the folder with the same name:

- SEALS-SWS-Plugin.jar - API (Interface) to be implemented by your tool.
- 'doc' folder - Java docs for the API above.
- Tool1DiscoveryImpl.java - Java source for sample tool implementation
- 'tool1' folder - Example of how your tool should be submitted
- SEALS-SWST-Campaign-2010-Instructions.pdf - this file.

Prepare your tool in a zip file as described below and send an e-mail to the contact person. See the sample implementation provided in folder tool1 in the instruction package. After implementation, zip your folder using the same name (e.g. tool1.zip).

Your tool folder (e.g. tool1) should contain:

1. The implementation for the Discovery Interface in a jar file
2. All other jar dependencies
3. A txt file with properties for execution (tool1.txt), including:

- full name of class implementing the Discovery interface
- test collection name - e.g. OWLS-TC or SAWSDL-TC
- test collection version - e.g 4.0
- test collection variant - e.g. OWLS-TC4-SWRL or SAWSDL-TC3\_WSDL11
- relevance set type - e.g. binary or graded

Format example (tool1.txt):

```
fullclassname=myCompany.sws.discovery.MyToolDiscoveryImpl
testcollectionname=OWLS-TC
testcollectionversion=4.0
testcollectionvariant=OWLS-TC4-SWRL
relevancesettype=binary
```

### A.2 Discovery Interface Implementation Example

```
package myCompany.sws.discovery;

import java.net.URI;
import java.util.List;
import eu.sealsproject.domain.swst.plugin.activities.Discovery;
import eu.sealsproject.domain.swst.plugin.results.DiscoveryResult;
import eu.sealsproject.domain.swst.plugin.results.Match;
```



```
import eu.sealsproject.domain.swst.plugin.results.DiscoveryResult.  
MatchDegree;  
  
public class MyToolDiscoveryImpl implements Discovery {  
private List<URI> services = null;  
  
@Override  
public DiscoveryResult discover(URI goalDescription) {  
// insert code to call tool's matchmaking algorithm  
// Dummy example  
  
DiscoveryResult dummyResult = new DiscoveryResult(goalDescription);  
  
dummyResult.add(new Match(URI  
.create("http://www.example.org/service2"), 1.0, MatchDegree.EXACT, 1));  
dummyResult.add(new Match(URI  
.create("http://www.example.org/service3"), 1.0, MatchDegree.NONE, 2));  
dummyResult.add(new Match(URI  
.create("http://www.example.org/service1"), 1.0, MatchDegree.EXACT, 3));  
dummyResult.add(new Match(URI  
.create("http://www.example.org/service4"), 1.0, MatchDegree.EXACT, 4));  
  
return dummyResult;  
}  
  
@Override  
public void init() {  
// insert code to initialise the tool  
}  
  
@Override  
public void loadServices(List<URI> services) {  
// insert code to load service descriptions for your tool  
this.services = services;  
}  
}
```

Listing A.1: Discovery Interface Implementation Example.



## B. OLWS-TC Metadata

### B.1 Example metadata of OWL-TC data set

```
<!-- metadata for one sample DiscoveryTest -->
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:xsd="http://www.w3.org/2001/
    XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:dc="http://purl.org/dc/terms
    /"
  xmlns:seals="http://www.seals-project.eu/ontologies/SEALSMetadata.owl#"
  xmlns:j.0="http://www.seals-project.eu/ontologies/DiscoveryTestSuite.owl#">

  <rdf:Description rdf:about="http://www.seals-project.eu/Discovery#DiscoveryTestSuite"
    >
  <rdf:type rdf:resource="http://www.seals-project.eu/ontologies/SEALSMetadata.owl#
    Suite"/>
  <seals:hasSuiteItem
    rdf:resource="http://www.seals-project.eu/discovery#DiscoveryTestSuite42" />
  </rdf:Description>

  <rdf:Description rdf:about="http://www.seals-project.eu/Discovery#
    DiscoveryTestSuite42">
  <rdf:type rdf:resource="http://www.seals-project.eu/ontologies/SEALSMetadata.owl#
    SuiteItem"/>
  <seals:hasDataItem rdf:resource="http://www.seals-project.eu/Discovery#
    GoalDocument_42"/>
  <seals:hasDataItem rdf:resource="http://www.seals-project.eu/Discovery#
    ServiceDocument_1068"/>
  <seals:hasDataItem rdf:resource="http://www.seals-project.eu/Discovery#
    ServiceDocument_1069"/>
  <seals:hasDataItem rdf:resource="http://www.seals-project.eu/Discovery#
    ServiceDocument_1070"/>
  <seals:hasDataItem rdf:resource="http://www.seals-project.eu/Discovery#
    ServiceDocument_1071"/>
  <seals:hasDataItem rdf:resource="http://www.seals-project.eu/Discovery#
    ServiceDocument_1072"/>
  <seals:hasDataItem rdf:resource="http://www.seals-project.eu/Discovery#
    ServiceDocument_1073"/>
  <seals:hasDataItem rdf:resource="http://www.seals-project.eu/Discovery#
    ServiceDocument_1074"/>
  <seals:hasDataItem rdf:resource="http://www.seals-project.eu/Discovery#
    ServiceDocument_1075"/>
  <seals:hasDataItem rdf:resource="http://www.seals-project.eu/Discovery#
    ServiceDocument_1076"/>
  <seals:hasDataItem rdf:resource="http://www.seals-project.eu/Discovery#
    ServiceDocument_1077"/>
  <seals:hasDataItem rdf:resource="http://www.seals-project.eu/Discovery#
    ServiceDocument_1078"/>
  <seals:hasDataItem rdf:resource="http://www.seals-project.eu/Discovery#
    ServiceDocument_1079"/>
  <seals:hasDataItem rdf:resource="http://www.seals-project.eu/Discovery#
    ServiceDocument_1080"/>
  <seals:hasDataItem rdf:resource="http://www.seals-project.eu/Discovery#
    ServiceDocument_1081"/>
  <seals:hasDataItem rdf:resource="http://www.seals-project.eu/Discovery#
    ServiceDocument_1082"/>
  <seals:hasDataItem rdf:resource="http://www.seals-project.eu/Discovery#
    ServiceDocument_1083"/>
  <dc:identifier>
    DiscoveryTestSuite42
  </dc:identifier>
  </rdf:Description>

  <rdf:Description rdf:about="http://www.seals-project.eu/Discovery#GoalDocument_42">
```



```
<rdf:type rdf:resource="http://www.seals-project.eu/ontologies/SEALSMetadata.owl#
  DataItem"/>
<seals:isLocatedAt>./OWLS-TC4_PDDL/htdocs/services/1.1/agent_movement.owl#
  seals:isLocatedAt>
<seals:hasComponentType>GoalDocument</seals:hasComponentType>
<dc:identifier>GoalDocument42</dc:identifier>
</rdf:Description>

<rdf:Description rdf:about="http://www.seals-project.eu/Discovery#
  ServiceDocument_1068">
<rdf:type rdf:resource="http://www.seals-project.eu/ontologies/SEALSMetadata.owl#
  DataItem"/>
<seals:isLocatedAt>./OWLS-TC4_PDDL/htdocs/services/1.1/agent_movement.owl#
  seals:isLocatedAt>
<j.0:hasRelevanceValue>0</j.0:hasRelevanceValue>
<dc:identifier>1068</dc:identifier>
</rdf:Description>

<rdf:Description rdf:about="http://www.seals-project.eu/Discovery#
  ServiceDocument_1069">
<rdf:type rdf:resource="http://www.seals-project.eu/ontologies/SEALSMetadata.owl#
  DataItem"/>
<seals:isLocatedAt>./OWLS-TC4_PDDL/htdocs/services/1.1/close_door.owl#
  seals:isLocatedAt>
<j.0:hasRelevanceValue>0</j.0:hasRelevanceValue>
<dc:identifier>1069</dc:identifier>
</rdf:Description>

<!-- the same for the rest ServiceDocuments -->

</rdf:RDF>
```

Listing B.1: Example metadata of OWL-TC data set.



## C. OLWS-MX SWS Plugin Implementation

### C.1 OLWS-M0 SWS Plugin implementation

```
package seals.swst.plugin.impl;

import java.io.IOException;
import java.net.URI;
import java.util.Iterator;
import java.util.List;
import java.util.SortedSet;

import de.dfki.owlsmx.SimilarityMatchingEngine;
import de.dfki.owlsmx.data.MatchedService;
import de.dfki.owlsmx.similaritymeasures.ConstraintSimilarity;
import de.dfki.owlsmx.utils.MatchmakerUtils;
import eu.sealsproject.domain.swst.plugin.activities.Discovery;
import eu.sealsproject.domain.swst.plugin.results.DiscoveryResult;
import eu.sealsproject.domain.swst.plugin.results.Match;
import eu.sealsproject.domain.swst.plugin.results.DiscoveryResult.MatchDegree;

/**
 * @author Liliana Cabral
 * This is a OWLS-M0 implementation of the SEALS SWS plugin
 * ConstraintSimilarity
 * Set syntactic similarity threshold to => 0.0
 * Set minimum hybrid degree of match to FAIL (5)
 * Set type of sorting: Semantic (2);
 */
public class owlsm4 implements Discovery {

    private List<URI> services = null;
    private SimilarityMatchingEngine engine;
    /*
     * @see eu.sealsproject.domain.swst.plugin.activities.
     * Discovery#discover(java.net.URI)
     */
    @Override
    public DiscoveryResult discover(URI goalDescription) {

        DiscoveryResult discoveryResult =
            new DiscoveryResult(goalDescription);
        MatchedService matchedService;
        int degreeOfMatch = -1;
        URI serviceURI = null;
        double syntactSim = -0.1;
        String degreeOfMatchStr = null;
        int order = -1;
        MatchDegree matchDegreeSEALS;

        try {
            SortedSet <MatchedService> result =
                engine.matchRequest(goalDescription, 5, 0.0, 2);
            Iterator <MatchedService> iter = result.iterator();
            while(iter.hasNext()) {
                matchedService = (MatchedService) iter.next();
                degreeOfMatch = matchedService.getDegreeOfMatch();
                serviceURI = matchedService.getServiceURI();
                syntactSim = matchedService.getSyntacticSimilarity();
                degreeOfMatchStr =
                    MatchmakerUtils.degreeOfMatchIntToString(degreeOfMatch);

                switch(degreeOfMatch) {
                    case (0): // "EXACT";
                        matchDegreeSEALS = MatchDegree.EXACT; break;
                }
            }
        }
    }
}
```



```
        case(1): // "PLUG-IN";
            matchDegreeSEALS = MatchDegree.PLUGIN; break;
        case(2): // "SUBSUMES";
            matchDegreeSEALS = MatchDegree.SUBSUMPTION; break;
        case(3): // "SUBSUMED-BY"
            matchDegreeSEALS = MatchDegree.PLUGIN; break;
        case(4): // "NEAREST NEIGHBOUR" // only semantic match
            matchDegreeSEALS = MatchDegree.NONE; break;
        case(5): // "FAIL"
            matchDegreeSEALS = MatchDegree.NONE; break;
        default:
            matchDegreeSEALS = MatchDegree.NONE; break;
    }
    // confidence => syntactSim
    if (! (matchDegreeSEALS == MatchDegree.NONE) )
        discoveryResult.add(
            new Match(serviceURI, syntactSim, matchDegreeSEALS, order+=1));
    }
    } catch (Exception e){
        e.printStackTrace();
    }
    }
    return discoveryResult;
}

@Override
public void init() {
    engine = new SimilarityMatchingEngine(new ConstraintSimilarity());
}

@Override
public void loadServices(List<URI> services) {
    try {
        for (URI service : services) {
            engine.addService(service.toURL().openConnection().
                getInputStream(), service); }
        } catch (IOException e) {
            e.printStackTrace();}
    }
}
```

Listing C.1: OLWS-M0 SWS Plugin implementation.

## C.2 OLWS-M4 SWS Plugin implementation

```
package seals.swst.plugin.impl;

import java.io.IOException;
import java.net.URI;
import java.util.Iterator;
import java.util.List;
import java.util.SortedSet;

import de.dfki.owlsmx.SimilarityMatchingEngine;
import de.dfki.owlsmx.data.MatchedService;
import de.dfki.owlsmx.similaritymeasures.JensenShannonMeasure;
import de.dfki.owlsmx.utils.MatchmakerUtils;
import eu.sealsproject.domain.swst.plugin.activities.Discovery;
import eu.sealsproject.domain.swst.plugin.results.DiscoveryResult;
import eu.sealsproject.domain.swst.plugin.results.Match;
import eu.sealsproject.domain.swst.plugin.results.DiscoveryResult.MatchDegree;

/**
 * @author Liliana Cabral
 * This is a OWLS-M4 implementation of the SEALS SWS plugin
 * JensenShannonMeasure
 * Set syntactic similarity threshold to => 0.7
 * Set minimum hybrid degree of match to FAIL (5)
 */
```



```
* Set type of sorting: Syntactic (1);
*/
public class owlsm4 implements Discovery {

    private List<URI> services = null;
    private SimilarityMatchingEngine engine;
    /*
    * @see eu.sealsproject.domain.swst.plugin.activities.
    *         Discovery#discover(java.net.URI)
    */
    @Override
    public DiscoveryResult discover(URI goalDescription) {

        DiscoveryResult discoveryResult =
            new DiscoveryResult(goalDescription);
        MatchedService matchedService;
        int degreeOfMatch = -1;
        URI serviceURI = null;
        double syntactSim = -0.1;
        String degreeOfMatchStr = null;
        int order = -1;
        MatchDegree matchDegreeSEALS;

        try {
            SortedSet <MatchedService> result =
                engine.matchRequest(goalDescription,5,0.7,1);
            Iterator <MatchedService> iter = result.iterator();
            while(iter.hasNext()) {
                matchedService = (MatchedService) iter.next();
                degreeOfMatch = matchedService.getDegreeOfMatch();
                serviceURI = matchedService.getServiceURI();
                syntactSim = matchedService.getSyntacticSimilarity();
                // case(4): // "NEAREST NEIGHBOUR"
                if (degreeOfMatch == 4) matchDegreeSEALS = MatchDegree.EXACT;
                else matchDegreeSEALS = MatchDegree.NONE;
                // confidence => syntactSim
                if (! (matchDegreeSEALS == MatchDegree.NONE) )
                    discoveryResult.add(
                        new Match(serviceURI, syntactSim, matchDegreeSEALS, order+=1));
            }
        } catch(Exception e){
            e.printStackTrace();
        }
        return discoveryResult;
    }

    @Override
    public void init() {
        engine = new SimilarityMatchingEngine(new JensenShannonMeasure());
    }

    @Override
    public void loadServices(List<URI> services) {
        try {
            for (URI service : services) {
                engine.addService(service.toURL().openConnection().
                    getInputStream(),service); }
            } catch (IOException e) {
                e.printStackTrace();}
    }
}
```

Listing C.2: OLWS-M4 SWS Plugin implementation.