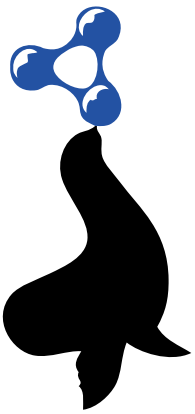
Services for the automatic evaluation of matching tools

Christian Meilicke, Cassia Trojahn dos Santos, Jérôme Euzenat



SEALS

Semantic Evaluation at Large Scale

FP7 – 238975

D12.2 Services for the automatic evaluation of matching tools v1

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

Document Identifier:	SEALS/2009/D12.2/V1.0
Class Deliverable:	SEALS EU-IST-2009-238975
Version:	version 1.0
Date:	July 29, 2010
State:	final
Distribution:	public

Executive Summary

In this deliverable, we describe a SEALS evaluation service that is based on the use of a web service interface wrapping the functionality of a tool to be evaluated. This interface allows to evaluate the tool without the need for a runtime environment. Contrary to this, the tool will be executed on the machine of the tool developer, while the evaluation takes place within the SEALS infrastructure. We refer to the proposed approach as a lightweight evaluation service, because the approach neither requires a Runtime Evaluation Service nor the powerful hardware infrastructure underlying this service.

We firstly discuss pros and cons of offering a lightweight evaluation service (§1). At the moment of writing this deliverable many required components for setting up a first prototype of an evaluation service are - due to the underlying complexity - still under development. For that reason we decided to implement an evaluation service that bypasses one of the main sources of complexity. Our approach is based on the idea of executing the tool on the machine of the tool vendor itself. In this scenario the functionality of the tool is made available via a web service, which is accessed by the SEALS platform during the execution of an evaluation workflow. This approach burdens the tool developer with the requirement to provide a working installation. However, the chosen design does not allow to measure metrics as runtime and memory consumption. The relevant system properties can only be controlled and measured appropriately if the tool runs on a machine or virtual host that is under control of the SEALS Runtime Evaluation Service.

Secondly, we describe the architecture of our system and explain its relation to SEALS components that will be available in future (§2). The entry point to our evaluation service is a graphical user interface implemented as a web-application. The web application invokes, based on a user request, a BPEL (Business Process Execution Language) workflow, which accesses several web services that provide functionalities such as retrieving test suites, evaluating a tool and storing the evaluation results. A detailed description of these web services is presented in §3, while the BPEL evaluation workflow is presented in detail in §4.

Third, we describe the methods for result visualization and manipulation (§5) developed as part of our evaluation service. These methods are supported by an OLAP (Online Analytical Process) application, which allows analysing large quantities of data in real-time.

Finally, we end the deliverable with some concluding remarks (§6) and present the future work. Our evaluation service will in near future progressively be extended and improved by using emerging components of the SEALS platform.

The contribution of our lightweight evaluation service is threefold. First, it requires a minimal effort from final users to make their tools available for evaluation, by implementing a simple web service interface. Second, it offers a web interface from what the users can start an evaluation and access the evaluation results, which are immediately available after starting the evaluation. Third, it offers a way for visualizing and manipulating the evaluation results, through which the final users can compare different submitted evaluations, select a subset of tests and a subset of evaluation metrics.

An important contribution of this deliverable is also related to the fact that our prototype enables a better understanding of the functional requirements regarding the platform components. The resulting experiences will help to develop and maintain a stable and powerful SEALS platform, that deals with the concrete needs of automized evaluation.

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**	12.2	**Title**	Services for the automatic evaluation of matching tools v1
Work Package	**Number**	12	**Title**	Matching Tools

Date of Delivery	**Contractual**	M14	**Actual**	12-11-09
Status	version 1.0		final ☒	
Nature	prototype ☒ report ☐ dissemination ☐			
Dissemination level	public ☒ consortium ☐			

Authors (Partner)	Christian Meilicke (University Mannheim), Cássia Trojahn dos Santos (INRIA), Jérôme Euzenat (INRIA)			
Resp. Author	**Name**	Christian Meilick	**E-mail**	christian@informatik.uni-mannheim.de
	Partner	University Mannheim	**Phone**	+49 621 181 2484

Abstract (for dissemination)	In this deliverable we describe a SEALS evaluation service for ontology matching that is based on the use of a web service interface to be implemented by the tool vendor. Following this approach we can offer an evaluation service before many components of the SEALS platform have been finished. We describe both the system architecture of the evaluation service from a general point of view as well as the specific components and their relation to the modules of the SEALS platform.
Keywords	ontology matching, ontology alignment, evaluation, benchmarks

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
25/06/2010	1	Christian Meilicke	Set up overall structure
02/07/2010	2	Christian Meilicke	Finished Appendix A
05/07/2010	3	Christian Meilicke	Finished Chapter 2
05/07/2010	4	Christian Meilicke	Finished Chapter 1
06/07/2010	5	Christian Meilicke	Converted online-tutorial into Appendix B
06/07/2010	6	Christian Meilicke	Finished Chapter 3
06/07/2010	7	Cassia Trojahn	Modified Chapter 3
06/07/2010	8	Cassia Trojahn	Added workflow section
06/07/2010	9	Christian Meilicke	Added summary/abstract
07/07/2010	10	Cassia Trojahn	Added visualization/manipulation results section
07/07/2010	11	Christian Meilicke	Added draft of Appendix C
07/07/2010	12	Cassia Trojahn	Modified Appendix C
13/07/2010	13	Cassia Trojahn	Addressed Points 3 and 5 of Reviewer
13/07/2010	14	Christian Meilicke	Addressed Points 1 and 2 of Reviewer
27/07/2010	15	Cassia Trojahn	Addressed QAC commments
27/07/2010	16	Christian Meilicke	Addressed QAC commments

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

1. Introduction

SEALS has an ambitious plan in terms of automation of benchmark processing. It plans to go beyond current evaluation software by providing a continuous evaluation platform that allows people to test their tools at anytime and publish the results that they want to be recorded. One of the most important aspects in the early stage of such a challenging project is the acceptance of the software provided. While SEALS will offer on the one hand a rich set of evaluation methods to tool vendors, it obliges them on the other hand to implement a specified interface. The willingness to implement such an interface is therefore a critical point for the overall success of the project.

For that reason, it is very important that the benefits of the SEALS platform are available and can be demonstrated in an early stage to justify the effort required on the side of the tool vendor. However, at the moment of writing this deliverable many required components for setting up a first prototype of an evaluation service are - due to the underlying complexity - still under development. For that reason we decided to implement an evaluation service that bypasses one of the main sources of complexity. Our approach is based on the idea of executing the tool on the machine of the tool vendor itself. In this scenario the functionality of the tool is made available via a web service, which is accessed by the SEALS platform during the execution of an evaluation workflow. Obviously, this approach burdens the tool developer with the requirement to provide a working installation. However, the tool developer itself can be expected to have no problems with installing and running his own tool on a well known infrastructure like e.g. his own laptop.

We have developed a set of services that allows to execute a complete evaluation experiment: validation service, test iterator service, redirect service, evaluation service and result service. The validation service ensures that the matcher web service is available and fullfills the minimal requirements to generate an alignment in the correct format. The set of test cases used for evaluating a tool is provided by the test iterator service, which is responsible for iterating each test case and providing a reference to the needed components (source ontology, target ontology, and reference alignment). Evaluation of the alignments provided by the tool is carried out via the evaluation service, which provides measures such as precision and recall. Finally, a result service is used for storing evaluation results, which are connected to a visualization module.

We refer to the proposed approach as a lightweight-evaluation service. We have chosen this terminology because the approach neither requires a Runtime Evaluation Service nor the powerful hardware infrastructure underlying this service. The most important advantages of such an approach can be summarized as follows.

- The lightweight evaluation service does not require the SEALS Runtime Evaluation Service and can be both developed and executed independently.
- The web service interface can be accessed directly from within a BPEL (Business Process Execution Language) workflow (see D8.2 [7]). Since the Runtime Evaluation Service will also wrap the functionality of the tools as web services, we can thus easily switch between both approaches once the Runtime Evaluation Service is available.

- The proposed tool interface can be implemented with minimal effort. Moreover, the conformance to the proposed interface can easily be validated and established by the tool vendor, since the tool itself is executed on the machine and under the control of the tool vendor.

- The most important evaluation metrics can be implemented on top of the chosen approach without any restriction. These measures are precision and recall. They are based on comparing the automatically generated alignment against a gold standard.

On top of the chosen approach we offer an evaluation service that allows any tool vendor, who implements the web service interface, to evaluate its tool via a simple web form. Moreover, feedback about the evaluation result is given on the fly for each test case successfully completed. However, the chosen design does not allow to measure runtime and memory consumption. The relevant system properties can only be controlled and measured appropriately if the tool runs on a machine or virtual host that is under control of the SEALS Runtime Evaluation Service. However, this loss in functionality pays off in an shortened time to market as we already argued.

Since we focussed our effort on the implementation of a system based on the proposed approach, we postponed development of the test data generator. Our decision is also based on the fact that for the first evaluation campaign we can use already a set of well-established datasets that cover most of our needs (see D12.1, Section 4.1-4.3 [3]). Compared to a working evaluation service, the availability of a test data generator is thus not on the critical path for the overall success of the evaluation campaign conducted by our workpackage.

The remainder of this deliverable is structured as follows. In Chapter 2 we present the overall architecture of our evaluation service. We describe the basic components and their relation to the SEALS platform. In Chapter 3 we describe the operations implemented as web services in detail. This chapter covers services that compute workpackage specific evaluation metrics as well as services that will finally be replaced by services of the SEALS Service Manager. We continue with a detailed description of a concrete BPEL workflow, which describes an evaluation experiment. Based on the experiences we made with practical implementation issues, we present an enhanced version of a BPEL workflow compared to the one described in D12.1. In Chapter 5 we focus on the visualization and manipulation of the stored results, presenting the visualization component implemented as part of our system. We end in Chapter 6 with some concluding remarks. We believe that a first prototype that offers concrete evaluation services will not only by a contribution on its own, but also helps to develop and maintain a robust and powerful SEALS platform, that deals with the concrete needs of automated evaluation.

2. Architecture

We start this chapter by roughly describing the main components of the architecture of the lightweight evaluation service followed by a detailed explanation of the data-flow for running an evaluation. Finally we list the components developed so far pointing to the corresponding components that will be developed as part of the SEALS platform.

The architecture of our evaluation service is depicted in Figure 2.1. The entry point to the application consists of a web interface, which makes use of the Wicket framework. This interface is deployed as a web application in a Tomcat application-server behind an Apache web server. It invokes the BPEL workflow, which is executed on the engine ODE. This engine runs as a web application inside the application server. The BPEL process accesses several web services that provide different functionalities. Some of them require to store and/or access permanent data. For the moment evaluation results are stored in a relational MySQL database and test data sets are stored and published as files on an Apache web server.[1]

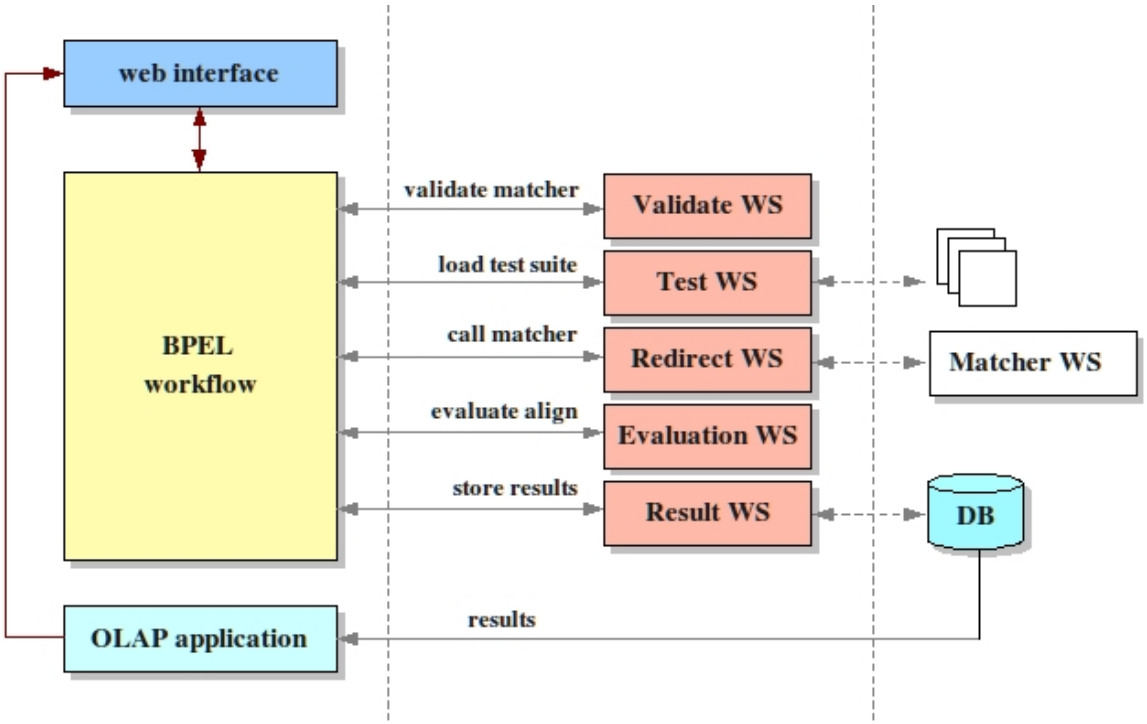


Figure 2.1: Architecture of the evaluation service.

We assume that matching systems have implemented the web service interface described in Appendix B. Once the web service interface has been deployed and published at a stable endpoint, the alignment method of these systems can be used by everybody who knows the address of the service endpoint. In particular, we can use it within a BPEL workflow or, more precisely, from one of the services used by a

[1] Apache web server `http://ode.apache.org/`, Tomcat Application Server `http://tomcat.apache.org/`, Wicket Framework `http://Wicket.apache.org/`, MySQL Database `http://www.mysql.com/`, and ODE BPEL Engine `http://ode.apache.org/`.

BPEL workflow. For that reason an evaluation starts by specifying the web service endpoint (URL and implementing class) via the web interface (Figure 2.1). This data is then forwarded to the BPEL as input parameter. The complete evaluation workflow is executed as a series of calls to different web service operations defined in the BPEL (details in Chapter 4). The specification of the web service endpoint becomes relevant for the invocation of the validation and redirect services. Both of them implement internally web service clients that connect to the URL specified in the web interface. It is not relevant that the submission of the evaluation request and the web service to be evaluated originate from the same machine.

Two of the services used in the BPEL process require to access additional data resources. The test web service iterates over the datasets of a test suite and the result web service stores evaluation results. The test datasets (ontologies, reference alignments and metadata) are published via an Apache web server. The test web service can access the metadata, extracts the relevant information and forwards the URLs of the required documents via the redirect service to the matcher, which is currently evaluated. These documents can then be accessed directly via a standard HTTP GET-request by the matching system. The result web service uses a connection to a MySQL database to store the results for each execution of an evaluation workflow. For visualizing and manipulating the stored results we access an OLAP application (details in Chapter 5), which access the database for retrieving the evaluation results. Since all of the results are associated with the evaluated endpoint, results can be re-accessed at any time (e.g. for comparing different tool versions against each other).

Several of these components will finally be replaced by components currently developed as parts of the SEALS platform. In particular, the following correspondences between system components exist:

- The redirect service, together with the overall approach of running the matching systems as web services, will be replaced by the SEALS runtime environment accessing the Tools Repository. The matching systems deployed on the machines of the matcher developer correspond to the matching systems deployed on the virtual machines set up by the SEALS runtime environment.

- The web services for accessing test data and results will be replaced by services of the SEALS services manager. The MySQL database and the web server used for the test data sets correspond to the results repository and the test data repository, respectively.

- The functionalities of the web interface correspond to the functionalities that will finally be provided as part of the SEALS portal.

During the subsequent development we will try to substitute the different components step by step by their SEALS counterparts adding functionality and stability to the proposed evaluation service. The advantages of using the platform are twofold. First, it allows having reproducible evaluations, that ensures that a same tool and test suite versions[2] are executed. Second, an important part of functionality that cannot

[2]Following the SEALS terminology, test suites are referred as test suite version, due the fact that a test suite can have multiple versions. For sake of simplicity, we refer test suite versions as test suite in this deliverable. The same is used for tool. We use 'tool' (matcher) to refer to the 'tool version'.

be offered on top of our architecture is related to the measurement of scalability and efficiency. Metrics like elapsed runtime or memory consumption can only be applied in a controlled environment available through the SEALS runtime environment. For that reason we have to leave out these measures for the moment and focus only on measures such as precision and recall.

3. Web Services

In this section we describe the web services that are called from inside the BPEL workflow. In further development we will step by step substitute most of these web services by services provided by one of the SEALS subsystems. This is not the case for the validation and the evaluation service. They offer workpackage-specific operations and will finally be integrated in the SEALS platform. All web services explained in the following are implemented on top of the JAX-WS framework[1].

3.1 Validation service

The methods of this service are called prior to any evaluation. Thereby, we ensure that the matcher web service specified via its endpoint URL is available and fullfills the minimal requirements to generate an alignment in the correct format. If this is not the case, an informative error message is generated.

`String validate(String ns, String localName, String wsUri)` This method first checks if there exists a service `{ns}localName` at the URI `wsUri`. In case such a service is available it is tested by running a simple matching task, namely matching a toy ontology on itself. The format of the result is finally validated. If the validation passes, it returns a string which serves as an ID of the subsequent evaluation execution. This ID is a string that is concatenated from the endpoint URL of the evaluated matcher service and the current timestamp and it is used during the execution of a workflow as a key to store and access web service variables related to the current state of this evaluation experiment. By the use of this ID we support the execution of concurrent workflows.

`String errorMessage()` Returns an error message, which informs the user about the reason of the failure (incorrect web service URI, incorrect output format, etc.).

In case the first method returns *false*, the evaluation is stopped and the results of the second method are returned to the user. In particular, we check whether the format generated by the matching system is the format of the Alignment API [4]. In case the format is invalid, we display the output generated by the matching system to the user.

3.2 Test iterator service

An example for a service that has to keep track of an internal state is the Test Iterator service. It is used to iterate over the test cases of a test suite and to access the components of a test case. Each currently executed workflow requires a variable that points to the current test case. Thus, the first parameter for each method is the ID of the current evaluation run, which allows to re-access the relevant variables. This service offers three operations that follow a simple iterator style.

[1] `https://jax-ws.dev.java.net/`

`boolean loadTestSuite(String runId, String testSuiteId)` Loads a test suite with ID `testSuiteId`. This sets the internal pointer to the the first test case and makes it accessible to subsequent method calls via the key `runId`.

`String nextTestItem(String runId)` Returns the ID of the next test case and sets the internal pointer to the next test case.

`boolean hasNextTestItem(String runId)` Checks if there exists a next test case.

However, these three methods are not sufficient to access the relevant files that a standard ontology matching test case (= a bundle of files) consists of. These files are the source ontology, the target ontology, and the reference alignment. The following methods are used for that purpose.

`String getOntoSource(String runId, String testItemId)` Returns the URL of the source ontology of a test case that is specified via its ID.

`String getOntoTarget(String runId, String testItemId)` Returns the URL of the target ontology of a test case that is specified via its ID.

`String getRefAlign(String runId, String testItemId)` Returns the URL of the reference alignment of a test case specified via its ID.

All the operations of this service rely on a simple method for describing a test suite by a XML-based descriptor file. The implementation details and a concrete example can be found in Appendix A.

Notice that our current implementation is workpackage-specific. However, it can easily be modified to be more generic. We just have to define the generic method `getComponent` and add an additional argument that identifies the type of component we are interested in.

`String getComponent(String componentId, String runId, String testItemId)` Returns the URL of the component which is of type `componentId` for a test case specified via its ID.

Following this approach the finegrained access can be realized across the borders of different workpackages.

3.3 Evaluation service

For the first evaluation campaign we decided to implement the measures precision, recall and f-measure (see D12.1 [3] for details). The implemented web service comprises a method for computing these values and three methods for retrieving each of the measures. After computing the measures we store them internally. The session id is then used to retrieve them.

`void computePRF(String runId, String alignAsString, String referenceUrl)` Computes precision, recall and f-measure comparing a generated alignment represented as string and a reference alignment referred to by its URI. The computed values are stored internally for a subsequent retrieval.

`double getPrecision(String runId)` Returns the previously computed precision value stored for the current run of an evaluation workflow.

`double getRecall(String runId)` Returns the previously computed recall value stored for the current run of an evaluation workflow.

`double getFValue(String runId)` Returns the previously computed f-measure stored for the current run of an evaluation workflow.

3.4 Redirect service

The redirect service substitutes a complex component that is intended to offer the core functionality within the SEALS platform, namely the Runtime Evaluation Service (see D9.1 [6]). The Runtime Evaluation Service will be used to execute the tools to be evaluated. This component will set up the virtual hosts to deploy and execute the different tools taking into account all requirements and dependencies. Since this component is not yet available, we have chosen an approach that is based on a different assumption. We evaluate a tool that is running on the machine of the tool developer itself, while the evaluation takes place within the SEALS infrastructure. This burdens the responsibility of a correct tool installation, deployment and execution completely on the shoulder of the tool developer.

Since we support the evaluation of any matching tool, for which the web service wrapper has been set up and deployed correctly, we had to find a way to dynamically refer to different matcher endpoints within a BPEL workflow. For that reason we have implemented a service that redirects the request for running a matching task to a matcher service endpoint. This service consists of only one operation:

`String align(String namespace,String localName, URI web service, URI source, URI target)` Redirects the call of an align operation to the align operation of the web service endpoint specified by `namespace` and `localName` that is located at `web service`. Returns the result returned by this align operation.

3.5 Result services

The result service is used for storing evaluation results generated by running a BPEL workflow. The operations of this service are called directly from the BPEL. Contrary to this, the operations that access the database for retrieving stored results are not encapsulated as web service. For the purpose of results visualization we connect directly to the database. Details are presented in Chapter 5.

The web service used for storing abstracts from the implementation details of persisting the measured results. The details of the storage are hidden. This allows us to finally replace the implementation of this service easily by a service that connects to the SEALS Results Repository. The following three methods are sufficient for storing result data generated by a standard evaluation experiment.

`boolean insertEvaluation(String runId, String track, String tool)` Stores the information that a certain test suite `track` has been applied to the matching tool `tool`. This information is stored together with the sessionID `runId`.

`boolean insertRawResult(String runId, String test, String alignment)` This operation is used to store a raw result. For ontology matching this is a string representation of an alignment generated by the evaluated matcher. This alignment is parsed in order to extract the data that will be finally stored into the database: the alignment features (level, URI ontology source, URI ontology target, etc.) and the alignment cells (URI entity source, URI entity target, relation and measure). Each stored alignment is related to a certain run of an evaluation experiment (see above) and a test case with id `test`.

`boolean insertInterpretation(String runId, String test, String metric, double value)` This operation is used to store an interpretation as a double. Each interpretation is related to a certain run of an evaluation experiment and a test case with id `test`. The interpretation itself is always a concrete value `value` and an identifier `metric` of the metric used for generating the value (e.g, precision or recall).

Currently we only require a method for storing double values. However, we can add in a similar way operations for storing different types of values. Although we developed these interfaces for ontology matching, they should be applicable with minor modification for storing evaluation results in general.

4. BPEL Workflow

In this chapter we briefly explain the BPEL workflow, which uses the services we have described in Chapter 3. We have followed the general structure of the workflow presented in deliverable D12.1 and adjusted it based on the experience we made while developing the prototype. This workflow represents an evaluation scenario where one matcher is evaluated using a set of test cases and the evaluation criteria is based on the compliance of the generated alignments with respect to the reference alignments in such set (e.g. precision and recall).

Basically, the BPEL workflow interacts with the web services through partner links, which encapsulate the corresponding web service interfaces. The calls to web service methods are performed using "invoke" activities. The "evaluation" process starts by receiving from the client process (via the user interface, as explained in Appendix C) a set of input parameters: the matcher web service namespace, the matcher service name, the URI of the matcher web service, the name of the test suite to be used for that evaluation and the URI of the corresponding test repository. Firstly, the first three parameters are validated, by invoking the `validate` operation (§3.1), through the activity `validateWS` (Figure 4.1). If the parameters are valid, a message indicating that the evaluation is started is replied to the client (`ReplyOK`) and the evaluationID returned by this invoking is stored into a BPEL variable, which is used later in others calls. Otherwise, the corresponding error message is retrieved, `getMsgError` activity, by invoking `errorMessage` operation (§3.1) and this message is replied as response (`ReplyERROR`) to the client. Once the web service parameters are valid, the test suite descriptor is loaded through the activity `loadTestSuite`, which calls `loadTestSuite` in the test iterator web service (§3.2). The BPEL fragment representing these activities is illustrated in Figure 4.1.

If the test suite descriptor has been successfully loaded (`SequenceTestOK`), an evaluation description is inserted into the database for the corresponding tool and test suite, via the activity `insertEvaluationDatabase`, which invokes in the result web service (§3.5) the operation `insertEvaluationDatabase`. For each test case in the test suite (`SequenceEachItem`), the ontology source (`getOntoSource`), the ontology target (`getOntoTarget`) and the reference alignment (`getRefAlign`) are retrieved by invoking the corresponding methods in the test iterator service (§3.2). These three elements are then stored into BPEL variables in order to be used in the next invokes (`setParamsAlign`). Having the source and target ontologies, the method `align` of the redirect service is invoked (§3.4). This method receives a set of parameters needed to invoke the web service that represents the real matcher (the matcher web service namespace, the matcher service name, web service URI and ontologies source and target). The BPEL fragment representing these activities is illustrated in Figure 4.2.

The resulting alignment provided by the matcher is then stored into the database, `insertAlignDatabase`, by invoking the method `insertRawResult` in the results service (§3.5). This alignment together with the reference alignment provided by the iterator service (as well as the evaluation ID) are used as parameter for invoking the method `computePRF` in the evaluator service (§3.3). This activity is represented by `eval` in Figure 4.1. After invoking `computePRF`, the precision and recall metrics can be retrieved from this service. For each metric, an interpretation is stored into the

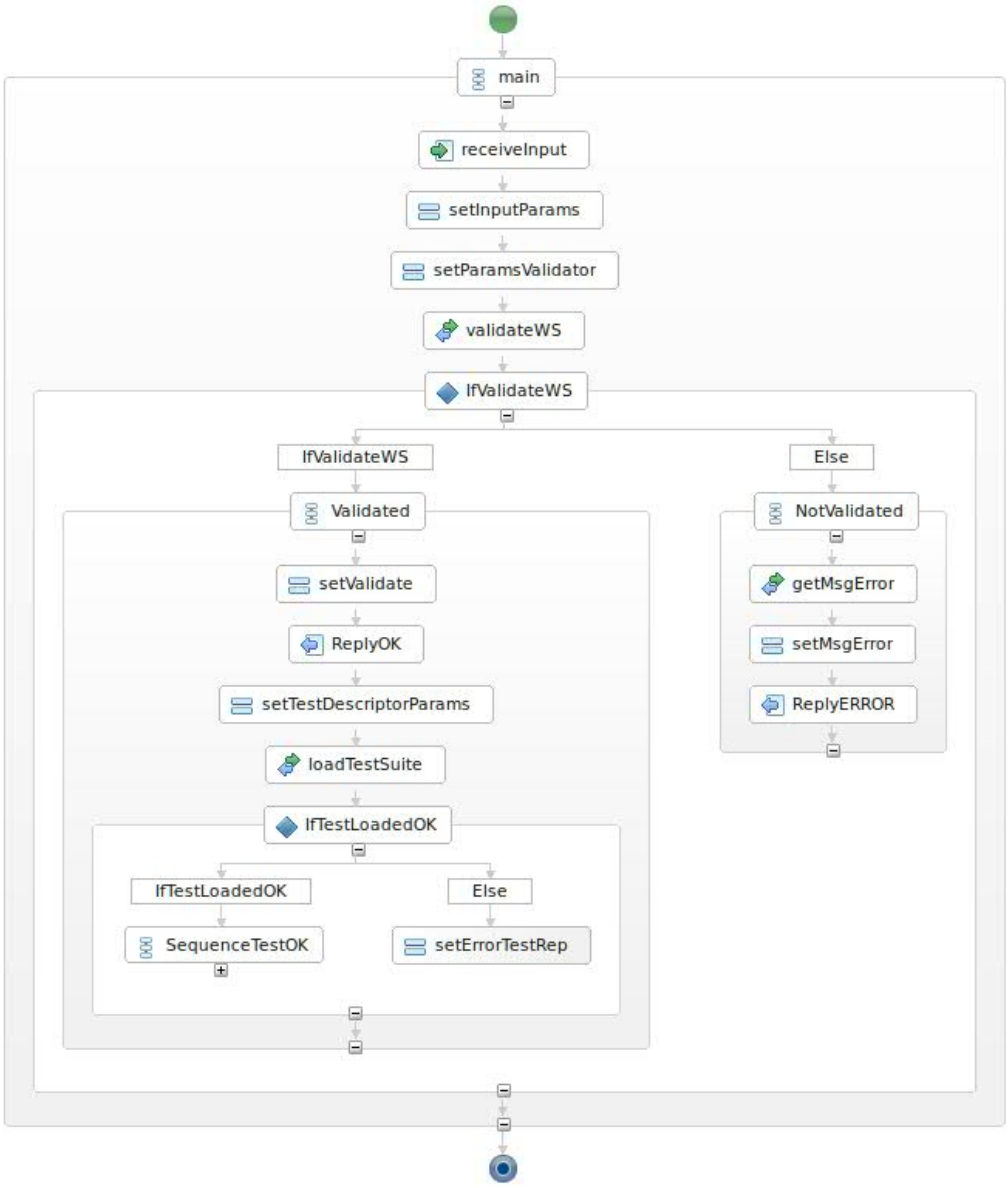


Figure 4.1: Validation fragment of the BPEL workflow.

database. It is done by invoking `getPrecision` and `getRecall` in the evaluation service (activities `getPrecision` and `getRecall` in Figure 4.2). Then, these values are stored into the database, by invoking `insertInterpretation` in the results service (activities `insertInterpretationPrecision` and `insertInterpretationRecall` in Figure 4.2). This process is repeated for each test case in the test suite. By storing results for each test case directly when they are obtained, we enable the user to see the progress of the process via the results user interface (compare Figure C in the Appendix).

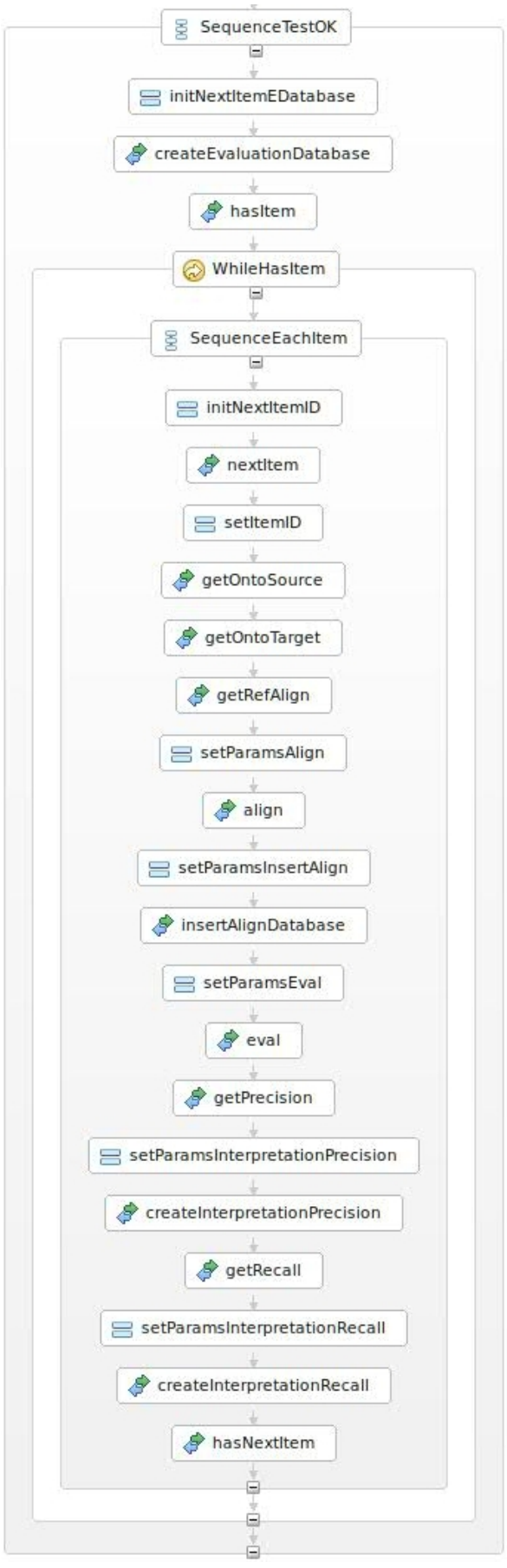


Figure 4.2: Iterating test suite and evaluating a matcher.

5. Result Visualization and Manipulation

The visualization and manipulation of evaluation results are supported by an OLAP (Online Analytical Process) application, as referred in the deliverable D12.1. Basically, OLAP allows to analyse large quantities of data in real-time, through a set of queries on this data. It employs a technique called multidimensional analysis, in which a multidimensional view (consisting of axes and cells) on a relational database is specified. Different MDX (multi-dimensional expressions) queries can be dynamically created in order to manipulate these axes and cells. We have used Mondrian[1], an open source OLAP server written in Java that allows to interpret MDX queries. This library offers also a JPivot library allowing to create a front end for visualising and manipulating the results of a MDX query.

Based on the relational database which stores evaluations data and interpretations (§3.5), different multidimensional views can be specified. The schema of the database is depicted in Figure 5.1.

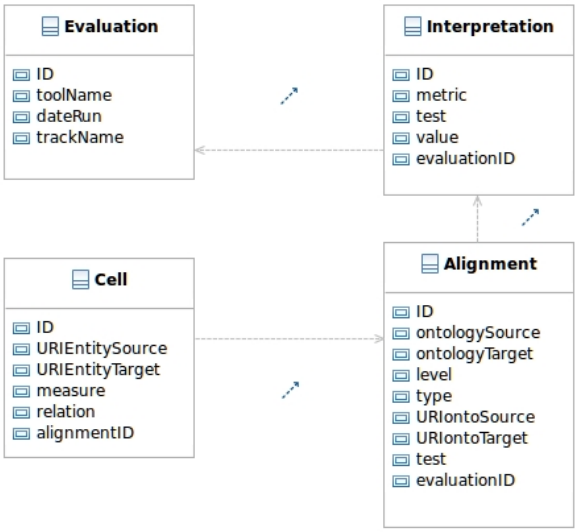


Figure 5.1: Result database schema.

For instance, the evaluation results to be visualized and manipulated by matcher developers are different from the results manipulated by the evaluation campaign organiser. For the first case, we specified a multidimensional view consisting of test cases as axis and interpretations as cells, and an OLAP application that provides the following main operations (Figure 5.2):

- to select a group or an individual test case that we want to consider (the results of a dataset);
- to order the evaluation data along a dimension according to a metric;
- to aggregate data of a group (that can be a unit) with a particular function (e.g., sum, average, harmonic means, variance, standard deviation);

[1] `http://sourceforge.net/projects/mondrian`

- to modify the MDX query to include dimensions.

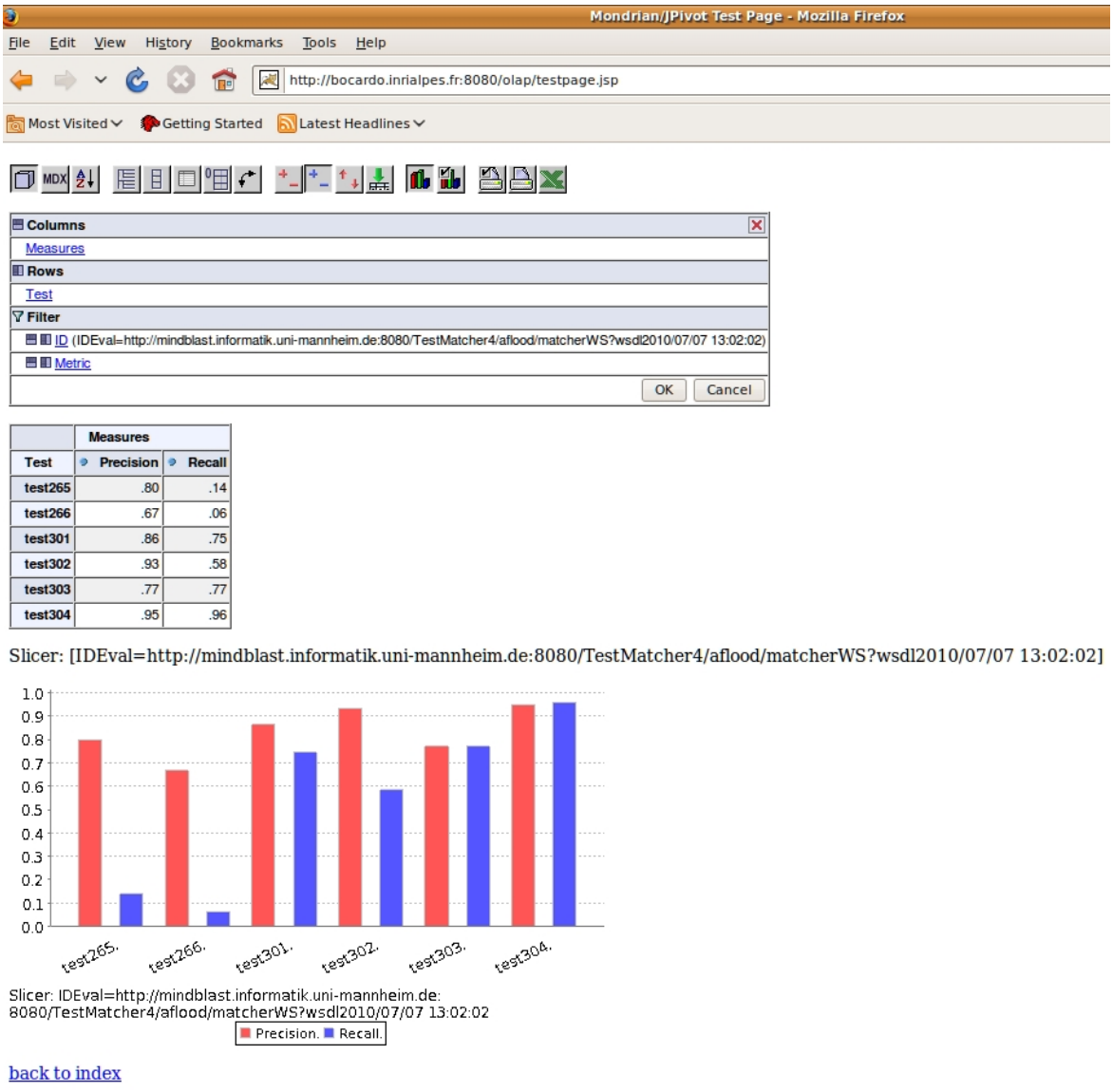


Figure 5.2: Front end for visualizing and manipulating evaluation results.

In addition, there are other available functions:

- Display the results as table (two or three dimensions);
- Plot the evaluation results;
- Export the results to an excel format.

From the administrator perspective, the available operations include:

- Selecting individually, or on criteria, the data that we want to consider (the results of a matcher, the results of a test suite, the results according to a particular metric);

- Projecting data on more dimensions according to particular split and aggregation functions;
- Ordering the data along a dimension according to a particular criterion (the name of the matcher, etc.);
- Grouping the data along a dimension according to a particular criterion (all the test cases of that matcher, etc.);
- Aggregating the data of a group (that can be a unit) with a particular function (e.g., sum, average, harmonic means, variance, standard deviation).

6. Conclusions

This deliverable presented an online available service for the automatic evaluation of matching tools. By means of this service, SEALS enables ontology matching systems to be evaluated via a web service interface. Participants will thus have to implement a very simple web service, provide its URL to the platform, and are able to directly access evaluation results. We have described pros and cons of the approach, explained the underlying architecture and described how to run a BPEL process in the context of this architectural design.

We have registered three test suites, which correspond to the OAEI (Ontology Alignment Evaluation Initiative)[1] datasets anatomy, benchmark and conference. These test suites include 133 test cases and 129 ontologies. Two tools are already connected, Anchor-flood and Aroma. We are ready to execute the first SEALS evaluation campaign what has been conducted together the OAEI 2010 campaign. Our evaluation service has been used to ensure that the tools correctly implement the proposed web service interface. Moreover, the users will have the possibility of running several evaluation experiments and store the most suitable one for being officially considered in the OAEI campaign.

The web interface from which an evaluation can be executed is available at `http://seals.inrialpes.fr/platform/`. All code is under the SVN repository of the project at `https://svn.seals-project.eu/seals-dev/omt/`.

Our evaluation service will in near future progressively be extended and improved by using emerging components of the SEALS platform. We already explained the relations between our architecture and the components and services of the SEALS platform. First we will successively attach our lightweight evaluation service to the different repositories for storing test data, results and evaluation workflows. As a next step we have to replace the web services we developed by the services offered by the SEALS service manger. The most important component is finally the SEALS Evaluation Runtime. Due to the specifics of our architectural design, we could bypass the complexity of this component. In the future it might make sense to offer both the evaluation as web service and the evaluation of a tool deployed inside the SEALS platform. The first approach has its benefits in the non-binding and uncomplicated access to the evaluation service, while the latter approach allows for more expressive evaluations taking efficiency and scalability into account.

However, using the components of the SEALS platform in their current state requires to solve several problems. First of all, the repositories offer their interfaces only as RESTful services. These interfaces cannot be used from within a BPEL workflow directly. Instead of that, standard webservice interfaces are required. Additional problems exist with respect to the required functionality. The Test Data Repository, for example, should support an iterator interface similar to the one described in Section 3.2. Unfortunately, the repositories support only very generic interfaces. The expected metadata description has not yet been specified or has only been specified partially. However, a metadata definition that is common for all research workpackages is relevant for many important operations, like e.g. accessing directly a component of a data

[1] http://oaei.ontologymatching.org/2010/

item. To sum up, a higher degree of integration with the SEALS platform depends on additional functionality, which can only be implemented on top of a generally accepted and supported metadata description, and a webservice interface to the services of the repository provided by the SEALS service manager.

Our focus in development was on the benefit for the final user. However, we also managed to run the first practically used evaluation workflow as BPEL process, which can be seen as a milestone in the development of SEALS. Thus, we managed to give a proof of concept which finally also resulted in a beneficial SEALS service.

References

[1] Caterina Caracciolo, Jérôme Euzenat, Laura Hollink, Ryutaro Ichise, Antoine Isaac, Véronique Malaisé, Christian Meilicke, Juan Pane, Pavel Shvaiko, Heiner Stuckenschmidt, Ondrej Sváb-Zamazal, and Vojtech Svátek. Results of the ontology alignment evaluation initiative 2008. In *Ontology Matching Workshop*, 2008.

[2] Jérôme David. AROMA results for OAEI 2009. In *Proceedings of the ISWC 2009 workshop on ontology matching*, Washington DC, USA, 2009.

[3] Cássia Trojahn dos Santos, Jérôme Euzenat, Christian Meilicke, and Heiner Stuckenschmidt. D12.1 Evaluation Design and Collection of Test Data for Matching Tools. Technical report, SEALS Project `<http://sealsproject.eu>`, November 2009.

[4] Jérôme Euzenat. An API for ontology alignment. In *Proceedings of the 3rd International Semantic Web Conference (ISWC)*, pages 698–712, Hiroshima (JP), 2004.

[5] Jérôme Euzenat, Alfio Ferrara, Laura Hollink, Véronique Malaisé, Christian Meilicke, Andriy Nikolov, Juan Pane, Francois Scharffe, Pavel Shvaiko, Vassilis Spiliopoulos, Heiner Stuckenschmidt, Ondrej Sváb-Zamazal, Vojtech Svátek, Cássia Trojahn dos Santos, and George Vouros. Results of the ontology alignment evaluation initiative 2009. In *Ontology Matching Workshop*, 2009.

[6] Miguel Esteban Gutiérrez. D9.1 Design of the Architecture and Interfaces of the Runtime Evaluation Service. Technical report, SEALS Project `<http://sealsproject.eu>`, November 2009.

[7] Michael Schneider, Stephan Grimm, Andreas Abecker, and Andrej Titov. D8.2 Language for describing evaluations. Technical report, SEALS Project `<http://sealsproject.eu>`, January 2010.

[8] Hanif Seddiqui and Masaki Aono. Anchor-flood: results for OAEI 2009. In *Proceedings of the ISWC 2009 workshop on ontology matching*, Washington DC, USA, 2009.

A. Defining a testsuite

We used a simple approach to iterate over the test cases in a test suite and to access its components. Since the web services defined in Chapter 3 require to access the files that make up a testcase by a URI, we published the test suite via a web server. An example for a small test suite is given in the following.

```
http://somehost/sometestsuite/
    ontologies/
        ont101.owl
        ont102.owl
        ont103.owl
    refalign/
        ref101-102.rdf
        ref101-103.rdf
```

This test suite consists of three ontologies and two reference alignments. It can be used e.g. to compute precision and recall for two matching tasks (matching `ont101.owl` on `ont102.owl` and matching `ont101.owl` on `ont103.owl`). We have developed a simple XML-format to describe the combination of ontologies and alignments that results in the corresponding test cases. For our example, we have to make the following content accessible at `http://somehost/sometestsuite/descriptor.xml`.

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuite>
    <testdataitems>
        <testdataitem id="test102">
            <source file="ontologies/ont101.rdf"></source>
            <target file="ontologies/ont102.rdf"></target>
            <reference file="refalign/ref101-102.rdf"></reference>
        </testdataitem>
        <testdataitem id="test103">
            <source file="ontologies/ont101.rdf"></source>
            <target file="ontologies/ont103.rdf"></target>
            <reference file="refalign/ref101-103.rdf"></reference>
        </testdataitem>
    </testdataitems>
</testsuite>
```

If the URL of the descriptor file is known, the URL of a specific component (source ontology, target ontology or reference alignment) of a specific test case can easily be generated by string concatenation. This is what we have implemented in the web service described in Section 3.2. The load-method parses the `descriptor.xml` file, keeps the extracted structure of the test suite in memory, and returns the appropriate URIs depending on the current request.

Since we support the concurrent execution of several BPEL workflows, the iterator method requires to track additionally which workflow execution currently points to which of the test cases. For that reason we store each of the pointers currently used in a hashmap under the corresponding sessionId. The stored pairs of sessionId and pointer are themselves stored in a kind of ringbuffer to avoid possible memory leaks.

B. Implementing the Interface

In this part of the appendix we explain the minimal interface by comparing it to a typical interface of a state of the art matching system. We show how to wrap this interface as a web service. We give detailed instructions how to implement the web service interface as part of an existing matching system. This part of the appendix is mainly based on a tutorial that we published online at `http://alignapi.gforge.inria.fr/tutorial/tutorial5/`, integrated in the Alignment API tutorials.

B.1 The minimal web service interface

The tools we consider so far implement a method that is very similar or can be easily transformed into the one we propose in the following. Typical examples can be found in the matching system Aroma [2] and Anchor-Flood [8]. Their interface can be described as follows.

```
void align(String ontFilePath1, String ontFilePath2, String outPath);
```

This method expects that the ontologies are accessible as files located on the machine where the matching system is executed. The generated alignment is not returned to the caller of the method, but written to a file specified via the third parameter. Instead of that we decided to go for an interface which works with input URIs and returns the generated alignment directly as string. This method signature fits better with an interface that is encapsulated as web a service.

```
String align(URI uriOnto1, URI uriOnto2);
```

It is obvious that the difference between the first and the second signature can be bridged with minimal effort. Even more, we ensured that only few lines of Java code are required to extend the first interface to the second interface. Therefore, we expect that most matching systems can easily be modified/extended to implement this interface.

Different ways for creating web services can be used. Here we propose to follow the Java API for XML Web Services (JAX-WS). Following this approach, a service endpoint is a Java class that specifies the methods that a client can invoke on the service. The development of JAX-WS web services is based on a set of annotations. The `@web service` annotation defines the class as a web service endpoint while `@WebMethod` defines the operations of this web service. We can determine the encoding style for messages send to and from the Web Service through the annotation `@SOAPBinding`. The code of our minimal web service interface AlignmentWS.java is defined as follows.

```
package eu.sealsproject.omt.ws.matcher;

import java.net.URI;

import javax.jws.WebMethod;
import javax.jws.web service;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;

@web service
@SOAPBinding(style = Style.RPC)
public interface AlignmentWS {
```

```
    @WebMethod
    public String align(URI source, URI target);
}
```

This interface defines a service endpoint, `AlignmentWS` and its operation, `align`, which takes as parameters the URIs of the two ontologies to be aligned, and returns a String representing the alignment.

B.2 Extending the minimal interface

Extending the web service interface requires the definition of the corresponding interface by adding the `endpointInterface` element to the `@web service` annotation in the implementation class. An example is given in the following listing.

```
package example.ws.matcher;

import javax.jws.web service;

@web service(endpointInterface="eu.sealsproject.omt.ws.matcher.AlignmentWS")
public class AlignmentWSImpl implements AlignmentWS {

    public String align(URI source, URI target) {

        //  your implementation
        return alignment;
    }
}
```

The method `align` must implement the matching process. The easiest way to do this is to implement the Alignment API (see details in how to extend the Alignment API with a new matcher at `http://alignapi.gforge.inria.fr/tutorial/tutorial3/`). Basically, you need to implement the `AlignmentProcess` interface (method `align`) and extend the `URIAlignment` class as depicted in the listing of MyAlignment.java.

```
package example.ws.matcher;

import java.net.URI;
import java.util.Properties;

import org.semanticweb.owl.align.Alignment;
import org.semanticweb.owl.align.AlignmentException;
import org.semanticweb.owl.align.AlignmentProcess;

import fr.inrialpes.exmo.align.impl.URIAlignment;

public class MyAlignment extends URIAlignment implements AlignmentProcess {

    public void align(Alignment alignment, Properties params) throws AlignmentException {

        //  matcher code
    }
}
```

Then, it is simple to expose MyAlignment as a web service as follows.

```
package example.ws.matcher;

import java.io.*;
import java.net.URI;
import java.util.Properties;
```

```
import javax.jws.web service;

import org.semanticweb.owl.align.*;

import fr.inrialpes.exmo.align.impl.renderer.RDFRendererVisitor;

@web service(endpointInterface="eu.sealsproject.omt.ws.matcher.AlignmentWS")
public class MyAlignmentWS extends MyAlignment implements AlignmentWS {

    @Override
    public String align(URI source, URI target) {
        try {
            init(source,target);
            align((Alignment)null, new Properties());
            SBWriter sbWriter = null;
            try {
                sbWriter = new SBWriter(new BufferedWriter(
                    new OutputStreamWriter( System.out, "UTF-8" )), true);
                AlignmentVisitor renderer = new RDFRendererVisitor(sbWriter);
                render(renderer);
                String alignment = sbWriter.toString();
                return alignment;
            } catch(Exception e) { }
        } catch (AlignmentException e) {
            e.printStackTrace();
        }
        return null;
    }

}
```

B.3 Publishing the web service

In order to be available for external access, the web service must be published at an endpoint, at which point it starts accepting incoming requests. This could be done by creating a WAR file to be deployed in a Tomcat server, or by creating a publisher. In the second case, the publisher can use the method `publish` to publish its address to the Endpoint.

```
package example.ws.matcher;

import javax.xml.ws.Endpoint;
import eu.sealsproject.omt.ws.matcher.AlignmentWS;
import eu.sealsproject.omt.ws.matcher.AlignmentWSImpl;

public class AlignmentWSPublisher {

    public static void main(String args[]) {

        // Publish matcher service web service
        AlignmentWS serverMatcher = new AlignmentWSImpl();
        Endpoint.publish("http://134.155.86.66:8080/matcherWS", serverMatcher);
        System.out.println("Matcher service published ... ");
    }
}
```

The endpoint has to be started on a publicly available machine (a machine that is accessible from the internet). In the example above we specified the IP 134.155.86.66 as part of the address, because it is the IP of the machine that we used for testing the example. If you make your matcher available as a web service based on this example, you have to additionally take into account the following:

- Run the web service on a machine that is accessible from the internet.

- Replace the IP of the example by the IP (or hostname) of your machine.
- Specify a port that is not blocked by a firewall of your machine or by your network/provider.

To use the seals infrastructure you have to specify the class including its package specification (e.g. `example.ws.matcher.AlignmentWSImpl` or `example.ws.matcher.MyAlignmentWS`) and you have to specify the URL of the service endpoint (e.g. `http://134.155.86.66:8080/matcherWS` or `http://134.155.86.66:8080/matcherWS?wsdl`).

Implementing and publishing the interface requires minimal effort from final users, when compared with the process of deploying the tool in a runtime environment. For implementing the interface, it requires 1-2 working hours. Publishing the tool may require more time (2-3 hours) because it involves to configure the network for receiving external requests.

C. Evaluation of a matching tool

In the following we briefly demonstrate the usage of our lightweight evaluation service by means of an example. For that reason we extended the matching system Anchor-Flood [8] with the web service interface and evaluated it. Anchor-Flood has been participating in OAEI 2008 [1] and OAEI 2009 [5] and is thus a typical evaluation target. The current version of the web application that we describe in the following is available at `http://seals.inrialpes.fr/platform/`.

For testing our web-application, we deployed the Anchor-Flood web service on a Tomcat and made it available via `http://mindblast.informatik.uni-mannheim.de:8080/TestMatcher/AFlood?wsdl`. The entry point to our web based evaluation service is a web page where you can specify the URL of the matcher endpoint, the class implementing the required interface, and the name of the matching system to be evaluated. We filled the form for our test matcher as depicted in Figure C.1. In this specific scenario we decided to apply the matcher to the conference test suite (see D12.1 for a detailed description of the corresponding dataset [3]). It is possible to choose between different test suites by using the drop-down menu.

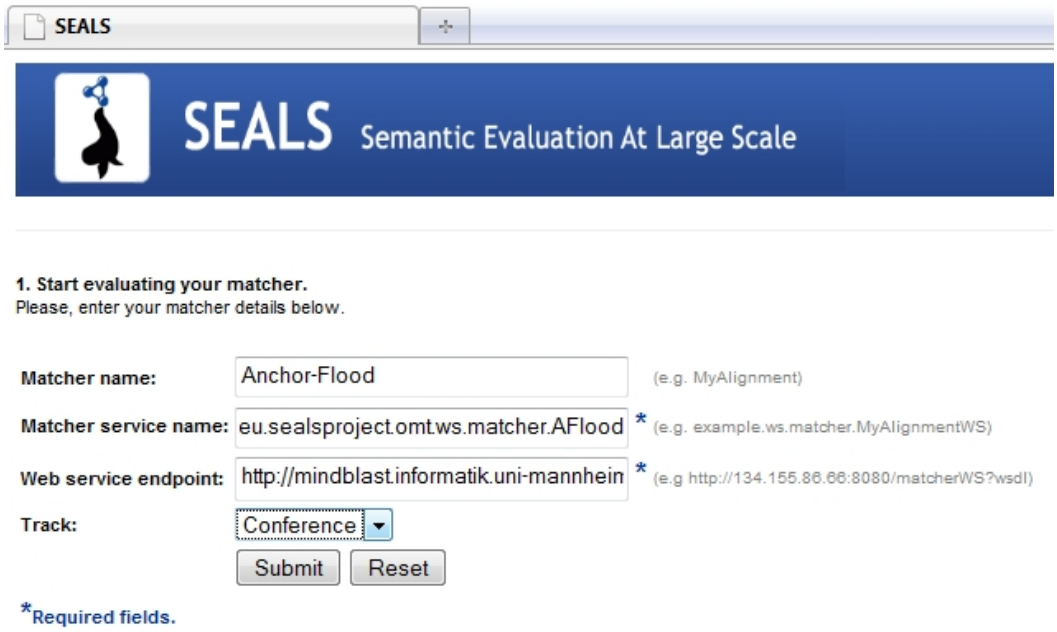


Figure C.1: Specifying a matcher endpoint as evaluation target.

Submitting the form results in an invocation of the BPEL evaluation workflow. As already described in Chapter 4, the workflow first validates the specified web service as well as its output format based on a toy example matching task. In case of a problem, the concrete validation error is displayed to the user as direct feedback. Figure C.2 is an example where we modified Anchor-Flood to generate an invalid output alignment. In this concrete example we removed a closing tag for one of the elements in the generated XML.

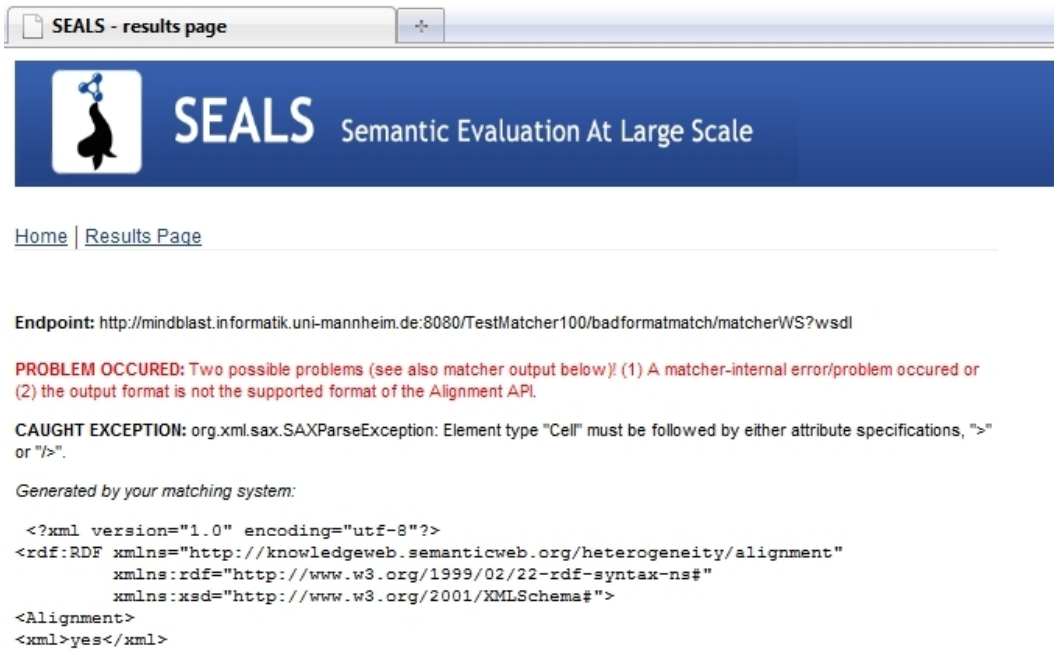


Figure C.2: Incorrect output format.

In case of a successfully completed validation the BPEL returns a confirmation message and continues with the evaluation itself. Every time an evaluation is conducted the results are stored under the enpoint address of the deployed matcher. For our small usage example we performed three evaluations listed in Figure C.3. First we started with the conference test suite, and then evaluated Anchor-Flood against the anatomy test suite, and finally again applied the conference test suite. Notice that this might be a typical situation, because some configuration parameters might have changed meanwhile. A second evaluation using the conference test suite might have been conducted by the tool developer to analyze and understand the effects of his parameter modifications.

The results itself can be displayed as a table, when clicking on one of the three evaluation IDs (endpoint+timestamp). An example is depicted in Figure C.4. The complete set of results might not yet be available. However, we decided that the results table is already (partially) available while the evaluation itself is still running. Only those rows are filled with results that correspond to test cases that have already been applied to the tool. By reloading the page from time to time, the user can see the progress of an evaluation that is still running. This is indicated by the arrow in Figure C.4.

In the table, we display, for each of the three test suite, precision and recall for each test case. In future we will add additional metrics to the results table. These results show already at which test cases the matcher fails to detect correspondences (low recall) or generates many incorrect correspondences (low precision). However, this information might not be detailed enough. The matcher developer might like to know which correspondences are the incorrect ones in order to improve the system based on this information. For that reason we offer a detailed view on the alignment

Figure C.3: Listing of available evaluation results.

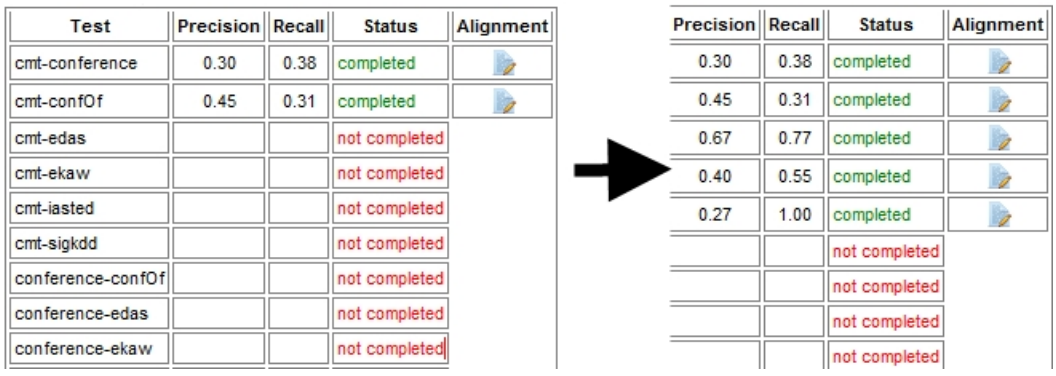


Figure C.4: Display results of an evaluation.

results. The complete alignment is displayed when clicking on the alignment icon in Figure C.4. In Figure C.5 a screenshot of the resulting page is shown. The generated

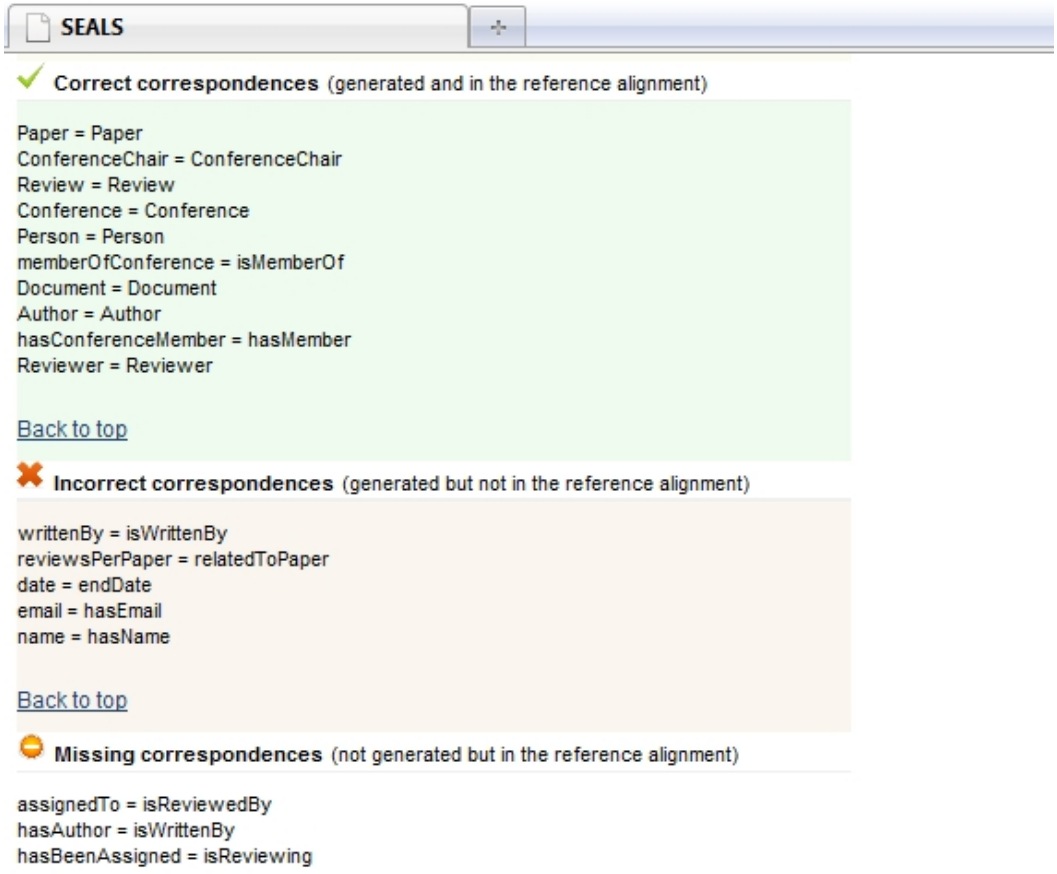


Figure C.5: Detailed view on a generated alignment.

The information is depicted in a user-friendly way and allows to analyse in detail the reasons for low precision. For the example depicted in C.5 there are five incorrect correspondences. Obviously, the matching system uses an approach that is strongly affected by label similarity. However, these correspondences are nevertheless incorrect. The matched concepts and properties can then be analyzed by the tool developer in order to find and use some information in the ontology that allows to filter out these incorrect correspondences. Once the tool has been modified, the SEALS evaluation platform can be used again to ensure the effects of the modifications.